熱環境計算戯法

富樫 英介　著

熱環境計算戯法

The art of thermal environmental computing

version 2.2.0

富樫 英介著

熱環境計算戯法

富樫英介著

序 言

本書は建築の熱環境の計算に関する解説書である。

本分野の基礎理論に関しては従来より数多くの書籍が出版されており、今日では誰でも意欲さえあれば容易に情報にたどりつくことができる環境にある。しかし、書籍に記された基礎理論や基礎式があるだけでは、必ずしも現実の建物で生じる様々な問題には即応できない。建築設備設計やコンサルティングなどの実務をこなすためには、知識としての基礎理論だけでは役に立たず、これらを応用する術を知らねばならないからである。一方で、建築設備は自然・機械・人間が関連する巨大なシステムであるために非常に多くの基礎式が作用しており、簡易な手計算で解ける問題は限定的である。

従来、技術者が鍛えた「熟練と勘（技能）」はこのような複雑なシステムを捉えるための唯一の手段であったが、今日ではこれに加えてコンピュータの利用という選択肢が存在する。即ち、自動計算により複雑なシステムを直接的に模擬して挙動を把握するという方法である。本書の大きな特徴は、基礎理論や基礎式の解説に留まらず、コンピュータを用いてこれらを具体的に解く方法を例示している点にある。ある問題に対して、基礎式のどの変数を入力値と捉えるのか、どのような数値計算を用いて解を得るのか、どこまでの抽象化と簡略化を許容するのか、結果をどのように解釈するのか、など、単純な基礎式の向う側にある世界へたどり着く方法について記したつもりである。

このような技術の習得は、建築熱環境の研究者にとって必要であることは当然であるが、現に実務に身を置いている設備設計者にとってもその設計内容に強い根拠を得るという価値を持つものである。また、筆者の経験から言えば、設計段階は抽象度が高いために解くべき問題も類型化しやすく、基礎理論をそのまま適用することができる場面が多い。一方、施工段階や管理運用段階で現出する問題は具体的かつ個別的である。この意味ではむしろ設計段階よりも現場段階においてこそ、基礎理論の高い応用力が求められる。

本書で取り扱う計算の範囲を次頁の図に示す。本書は「基礎編」「エネルギー評価編」「室内環境評価編」の 3 編で構成した。「エネルギー評価編」と「室内環境評価編」において共通して必要となる基礎的な知識と技術は第 1 章~第 8 章の「基礎編」にまとめた。「エネルギー評価編」では第 9 章~第 19 章において個別の設備機器の計算法を示した後、これらを組み合わせて熱源システムとして解く方法を第 21 章に示した。また、第 20 章では環境性と経済性を評価する方法を示し、熱源設備システムをエネルギーの観点から評価する方法を示した。「室内環境評価編」では第 22 章~第 26 章において建物の熱負荷と空調機の計算法を示した後、これらを組み合わせて空調システムとして解く方法を第 28 章に示した。また、第 27 章では人体モデルを解くことで熱的快適性を評価する方法を示し、空調設備システムを熱的快適性の観点から評価する方法を示した。第 29 章では第 21 章の熱源設備システムモデルと第 28 章の空調設備システムモデルを組み合わせ、建築熱環境システムを総合的に評価する方法を示した。従来、本分野の教科書は内容が熱負荷計算または設備システムのいずれかに偏る傾向があったと思うが、両者を等しく解説して最終的に横断的な計算ができるようにした点は本書の大きな特徴の 1 つである。また、最終章では建築熱環境計算の歴史を振り返り、未来展望を記した。

それぞれの章は基本的に「概要」「理論」「計算法」の 3 節で構成した。「概要」では、各章で取り扱う内容と作成するモデルのあらましを 1 ページ程度でまとめた。「理論」では、モデル化を行うために必要となる基礎式や知識をまとめた。「計算法」が本書の核であり、「理論」において記した基礎式にもとづいて具体的に問題を解くための計算処理（ソースコード）を記すとともに、その内容を解説した。読みながら具体的に確認できるようにソースコードの全体を記載するようつとめたが、紙面の都合上、省略した箇所もある。後述の通り、本書に記載のプログラムはすべて Web site において公開したため、必要に応じてソースコードを入手して確認して頂ければと思う。

建築熱環境分野のシミュレーションに関する書籍としては、宇田川光弘教授による「パソコンによる空気調和計算法」と宿谷昌則教授による「数値計算で学ぶ光と熱の建築環境学」という名著が存在するが、いずれも絶版になって久しい。本書では両書で取り扱っていない冷媒や水の物性、気象データの生成、個別の設備機器（熱源、冷却塔、流体機械、蓄熱槽など）、人体、経済性なども加えた。

熱負荷計算、設備システムの決定、運用管理、快適性と経済性の評価という一連の熱環境設計行為の幅広い範囲に対応することで、設計者が直面するであろう各種の問題において活用可能な技術を提供するという狙いによるものである。多くの話題を広く提供した反面、個別の項目については必ずしも最新の研究成果が反映しきれておらず、また各分野の専門家からみると記述が厳密ではない可能性がある。著者の不勉強によるものであるが、各章においてできる限り多くの参考文献を挙げたため、さらなる技術力向上を目指す読者の方々はこれらの情報にあたっていただきたい。また、本書で修正が必要な点に気が付かれたら是非ご指摘いただければと思う。

本書で例示したプログラムのソースコードは基本的にすべて General Public License で公開した。また、本書自体も GNU Free Documentation License により公開情報としており、電子データを PDF で提供した。紙媒体に関しては出版社を通さずに Print On Demand とすることで原価による提供を可能とした。技術の習得は、まずは模倣から始まると著者は考えている。ソースコードをただ眺めて理解したつもりになるだけではなく、実際に書いて動作させ、さらに必要な修正や改良を自由に加えて試行錯誤する中で研鑽を積んで頂ければと期待している。著者が設備のシミュレーションを学び始めた 2002 年頃は、参考とすべき名著の多くはすでに絶版となり、また、主要なソフトウェアの殆どはソースコードを公開しておらず、初学者にとっての学習環境は優れたものではなかった。学ぶ意欲がある者に対して学習環境が提供できていないという状況に、当時の著者はひどく憤慨し、そして失望した記憶がある。そのような中で唯一、ソースコードと解説書が容易に入手可能な状態で公開されていたソフトウェアが名古屋大学の中原信生教授による HVACSIM+(J)である。修士課程の 2 年間、著者は HVACSIM+(J)のソースコードを一行一行読み解くことで設備のシミュレーションを学ぶことができた。本書とソースコードを公開する 1 つの理由は、自由の理念を重視する中原研究室の伝統に対する感謝と共感によるものである。

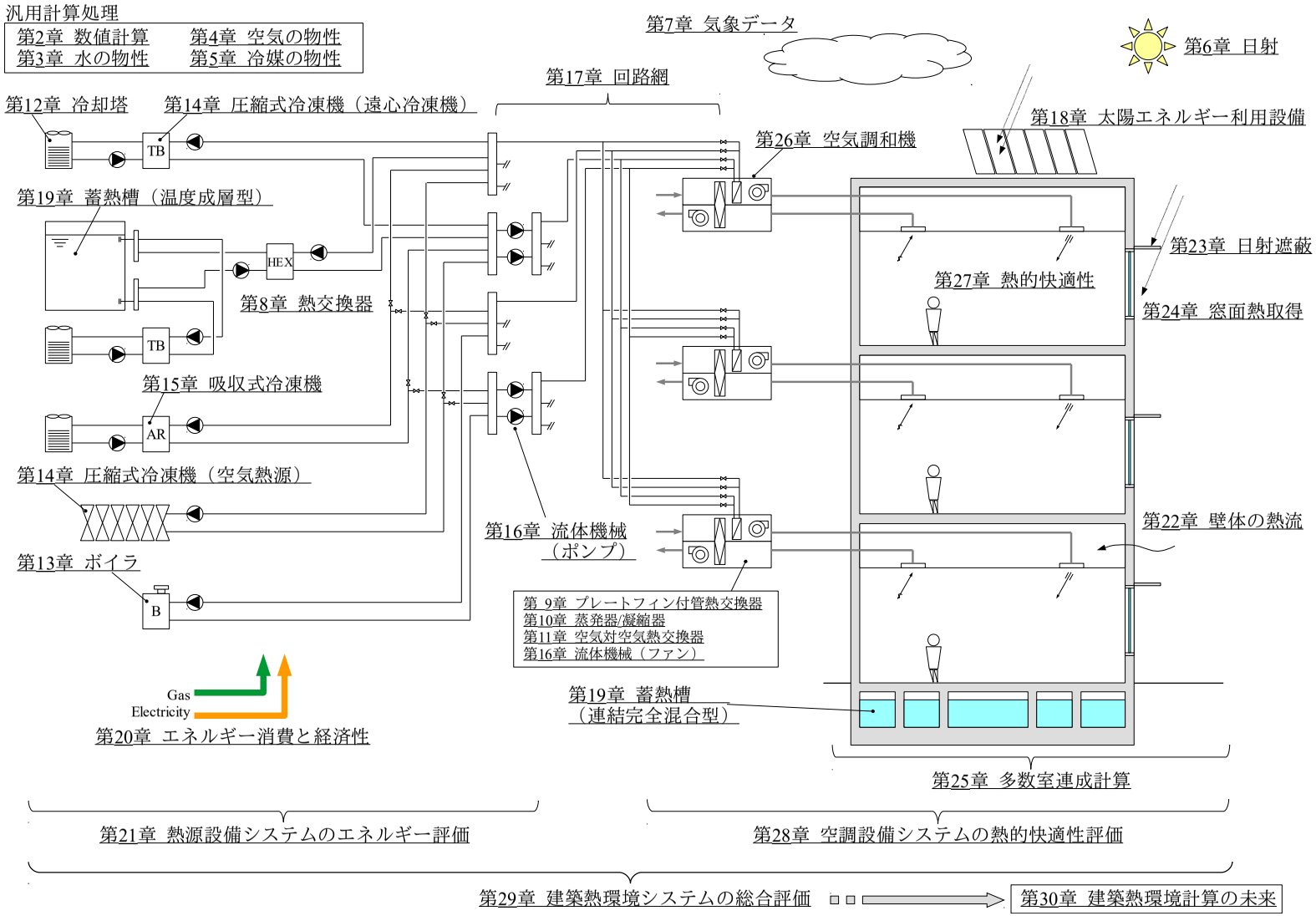
筆者は 2010 年から 2015 年に株式会社日建設計に勤め、設備設計の実務に身を置いた。僅か 5 年間の経験であり、実務設計の一部を垣間見たに過ぎないが、それでも本書には実務を経なければ得られなかったであろう視点にもとづく記述が多くあり、乖離しがちな建築設計実務と研究との橋渡しの役割を多少は担えるのではないかと期待している。博士研究員を 5 年間に渡って雇用するということは、短期的な経済性の観点からは全く合理的ではなく、それを受け入れてくれた日建設計には感謝しきれない。現時点において本書の社会的評価は全く不明であるが、仮に本書が建築熱環境分野に身をおく人々にとって理論と応用を学ぶための一助となり、日建設計が建築分野に果たした社会貢献の 1 つであると受け止められることがあれば、筆者にとって非常に大きな喜びである。

本書の記述の多くは大学時代と実務期間を通じて知り合えた諸先輩と友人らから学んだものである。筆者の出身研究室の田辺新一先生は、研究室外での活動と交流に関して極めて寛容であり、むしろ筆者が無邪気に知らぬところで積極的に後押しをしてくださっていたのだろうと推測する。今、振り返ってみれば、筆者の建築環境分野における人間交流のほぼすべての起点は学部 4 年生で田辺研究室を選択したときに遡る。仮想の話であるからその良し悪しを軽々に評価することはできないが、異なる選択をしたならば今とは全く異なる人生が待っていたことは確実であり、この意味では、もはや感謝を超えて人の繋がりの不思議さを畏れるくらいしかできない。

学ぶという行為は何かの理由があって行われるものではない。完全に自由・崇高で、それだけに不安を覚える行為である。学ぼうとする人間には孤独と闘う意思が求められる。近年では、幸か不幸か、地球環境・エネルギーという大きな問題が表れ、建築熱環境学に関わる者はその行為に簡単に目的を付与することができるようになった。しかし、一時の安心を得るために学ぶことの目的を安易に取り繕うことがあってはならないと筆者は思う。人間は必ずしも地球環境ごときに奉仕するために学ぶのではない。欲動のままに、ただ、学ぶだけである。後ろ盾は期待せず、信念をもち、学ぶことに独りで戯れなければならない。

2016 年 12 月

著　者 **e.togashi@gmail.com**


汎用計算処理
第2章 数値計算
第3章 水の物性
第4章 空気の物性
第5章 冷媒の物性
第7章 気象データ
第6章 日射
第17章 回路網
第12章 冷却塔
第14章 圧縮式冷凍機（遠心冷凍機）
第18章 太陽エネルギー利用設備
第26章 空気調和機
第19章 蓄熱槽（温度成層型）
第23章 日射遮蔽
第27章 熱的快適性
第8章 熱交換器
第24章 窓面熱取得
第15章 吸収式冷凍機
第14章 圧縮式冷凍機（空気熱源）
第22章 壁体の熱流
第16章 流体機械（ポンプ）
第13章 ボイラ
第 9章 プレートフィン付管熱交換器
第10章 蒸発器/凝縮器
第11章 空気対空気熱交換器
第16章 流体機械（ファン）
第19章 蓄熱槽（連結完全混合型）
Gas
Electricity
第20章 エネルギー消費と経済性
第25章 多数室連成計算
第21章 熱源設備システムのエネルギー評価
第28章 空調設備システムの熱的快適性評価
第29章 建築熱環境システムの総合評価
第30章 建築熱環境計算の未来
TB
HEX
TB
AR
B

目次

第I編　基 礎 編

空調設備システムはエネルギーを投入することで快適な温熱・空気質環境を得るシステムである。このために冷温熱を製造し、水や空気でこれを搬送し、最終的に人間が滞在する居住空間に供給する。システムを構成する要素は相互に関連しあうため、いずれも熱環境とエネルギー消費の双方に影響を持つものであるが、その影響の度合いは異なる。本書では、エネルギー消費に大きな影響を持つ要素群の計算法を第II編に、熱環境や快適性に大きな影響を持つ要素群の計算法を第III編にまとめた。

第I編の基礎編では両者に共通して必要となる基礎的な技術について解説する。第1章では熱環境計算という観点からプログラムの基礎について解説する。第2章では多くのモデルで必要となる数値計算について解説する。ただし、後章で開発するモデルで実際に必要となった際に参照すればよく、すべての数値計算法を順序通りに学ぶ必要はない。第3章~第5章は熱媒の物性計算である。特に第3章と第4章の水と湿り空気の計算は重要であり、空気調和を専門に扱う者にとっては必須の素養である。空調設備に関するプログラミングの初学者であるならば、まずは肩慣らしに湿り空気の物性計算を習得し、自分専用のライブラリを作成すると良いだろう。空調分野にいるかぎり、生涯にわたって役に立つ財産となるはずである。第6章および第7章は建物を取り巻く外界条件の計算に関する章である。第6章で解説する日射も応用範囲の広い計算技術であり、自分のライブラリを作成する価値は大きい。最後の第8章では一般的な熱交換の計算法を解説する。第II編で取り扱うほぼすべての機器では熱交換が行われるため、ここで熱交換の基礎を習得しておけば後章での理解が早まるだろう。

第1章　建築熱環境計算の基礎
(Basis of Thermal Environmental Computing)

1.1　概要

本章では、建築熱環境に関するシミュレーションプログラムを作成する際に必要となる、情報の入手源、コーディングルール、デバッグの方法などの基礎的情報を整理する。

1.2　既往のシミュレーションプログラム

シミュレーションを行うにあたって使用する各種の基礎式は、検討の対象とする事象に関わる状態変数の関係性を示したものである。応用の段階では、ある状態変数について基礎式を解くことで、状態変数の値を特定する処理が必要となる。この処理の流れを記したものをプログラムのソースコードと呼ぶ。

才能あふれる人間ならば、基礎式が与えられただけでも検討対象とする事象の特徴を正確に理解できるのであろうが、実際には、プログラムを作成し、試行錯誤する中で事象の正しい理解へたどり着くことも多いだろう。基礎式の正しい理解のもとでしか、正しいプログラムは作成できないため、意図しないプログラムの出力値を目の前にして初めて、自分の理解の誤りに気付かされることは非常に多く、これはプログラムを開発することの大きな価値である。

ソースコードから基礎式へ遡ることで理解が深まることを考慮すると、事象の理解にあたっては、他者の作成したソースコードを読み解くという方法も有力であると言える。そこで、以下にソースコードが入手可能な建築熱環境関連のプログラムの例を挙げる。1)～4)に関しては、無償で公開されており、ソースコードの改変等に関して権利上の大きな制約もないため、プログラム開発にあたっては利便性が高い。なお、建築熱環境のどの範囲を計算対象とするかはプログラムによって異なる。本書では、便宜的に、熱負荷および室内環境を対象とする場合には「熱負荷シミュレーション」、設備システムを対象とする場合には「設備システムシミュレーション」、制御系を対象とする場合には「制御系シミュレーション」と呼ぶこととし、これらをすべて合わせた概念を「熱環境シミュレーション」とする。ただし、多くのプログラムはこれらの概念を横断する領域を計算対象としており、上記で完全な分類ができるとは言えないことには留意する必要がある。

1）HVACSIM+(J)

HVACSIM+(J)は設備システムの非定常[†1]な計算が可能な制御系シミュレーションプログラムである。プログラムには、個別の機器の挙動を表現した多くのサブルーチンが定義されており、予め用意された非線形連立代数方程式（NLEs : Non Linear Equations）や固い連立常微分方程式（ODEs : Ordinary Differential Equations）のソルバによって解くことができる。これらのサブルーチンは TYPE と呼ばれており、一定の入出力仕様に適合させれば、使用者がオリジナルの TYPE を追加することも

†1　実際には定常と非定常プログラムの境界は曖昧である。設備システムの定常計算プログラムの代表格である HASP/ACLD/ACSS も建物負荷や蓄熱槽に関しては非定常計算を行うし、逆に、非定常計算プログラムとして認知されている HVACSIM+(J)においても流量圧力計算など、静的な計算のみを行う部分も多くある。

可能である†1)。

使用者は既定義の TYPE や自らが定義した TYPE の入出力変数を相互に接続し、プログラム上でシステムモデルを構築する。従って、当然、上記の NLEs および ODEs ソルバは、特定の関数形状に特化したものではなく、汎用的なソルバである。この意味では、構築したシステムモデルが解き得る（解がある）問題であるか否かについては、使用者側で判断することが必須である。モデルの設定の妥当性自体が使用者に委ねられるという意味では、プログラム使用者に求められる能力は相対的に高く、設備シミュレーションの初心者にとっては使いこなすことが難しい。少なくとも、1 つや 2 つは自前でプログラムを開発した経験があって初めて、HVACSIM+(J)が持つ汎用性の高さの価値を享受できるだろう。

HVACSIM+(J)のオリジナルは、米国 NBS（現 NIST）にて 1985 年に開発された HVACSIM+（HVAC SIMulation PLUS other systems）である。名古屋大学教授の中原と京都大学教授の吉田が NBS の楠田と Park を経由して入手した。1995~1999 年には空気調和・衛生工学会にて蓄熱最適化委員会が組織され、300 ページを超える日本語版マニュアルが作り上げられた。2000~2003 年にかけてコミッショニングツール開発小委員会において改良が続けられ、現在の HVACSIM+(J)が完成した[1.14)]。public domain のソフトウェアであり、ソースコードを含めて、Web サイトから無償で入手可能である。上記の通り、全体のプログラムを使用するという意味では高い能力が求められるが、個別の機器が小さなサブルーチンとして定義されているということと、そのソースコードが容易に入手でき、解説マニュアルも付属しているという意味では、初学者にとっても大きな価値がある。なお、プログラムの言語は FORTRAN である。

2）HASP/ACLD/ACSS

建物の動的な熱負荷計算を行うプログラムである HASP/ACLD（Air Conditioning LoaD）と、設備システムの計算を行うプログラムである HASP/ACSS（Air Conditioning System Simulation）とで構成されたプログラムである[1.8)]。我が国において最もよく知られ、論文等においても多くの引用が行われている。その意味では、自らプログラムを開発するにあたっては、計算しようとしている問題を HASP/ACLD/ACSS ではどのように扱っているかという対比を意識する必要がある†2)。

2012 年 4 月よりソースコード（FORTRAN 言語）が一般に公開され、現在は建築設備技術者協会の Web サイトから入手することが可能である。設備システムの解法については、HASP/ACSS に付属のマニュアル[1.7)]で詳細に解説されている。BECS/CEC/AC[1.9)]、d-BECS、Micro ACSS、FACES[1.10)]、建物の原単位管理ツール[1.11)]等のプログラムが HASP/ACLD/ACSS から派生して開発されているが、解法に関しては同マニュアルの記述が最も詳細である†3)。HASP/ACLD もマニュアルが付属しているが、解法に関しては日本建築設備士協会が発行する「空調設備の動的熱負荷計算入門」が詳しい[1.13)]。

3）ESP-r

ESP-r は University of Strathclyde において 1970 年代から開発が継続されている熱負荷と設備システムのシミュレーションプログラムである[1.12)]。ライセンスとしては GPL（General Public License）が採

†1　後述の TRNSYS も同様の仕様であり、モジュール型のプログラムと呼ばれる。
†2　「既存のプログラム、HASP/ACSS などとの計算結果の比較は？」という問いは、学会のシミュレーションに関するセッションで最もよく聞く質問である。
†3　HASP/ACSS シリーズのすべての開発に関わった猪岡達夫教授へのヒアリングによる。

用されており、ソースコードの再頒布および改変が認められている。Subversion を用いてソースコードの管理が行われているため、インターネットを用いて最新のソースコードが常時入手可能である。言語としては FORTRAN と C++が併用されている。

4）LCEM のためのシミュレーションツール

LCEM のためのシミュレーションツールは国土交通省大臣官房官庁営繕部において開発が進められている設備システムのシミュレーションプログラムである。[1.18) 1.19)]。設備機器ごとにまとまりを持たせたモジュール型のプログラムであるが、汎用の表計算プログラム上で動作させている点が特徴である。表計算プログラムのいくつかのセル群として設備機器を表現し（オブジェクト化セルズ法）、このモジュール相互の連成計算は、表計算プログラムが標準で備えている循環参照の収束計算機能によって解くという方式である。汎用の表計算プログラムであるため、セル内の数式を直接に見ることが可能であり、計算処理の内容を学ぶことが容易である。また、個別の機器の計算内容を説明したマニュアルも整備されている。

5）TRNSYS

TRNSYS は欧州と米国の共同で開発が進められているシミュレーションプログラムである[1.17)]。微分方程式のソルバを備えているため、HVACSIM+と同様に制御系の計算が可能であるが、実態としては（熱負荷計算を除き）静的な計算にとどまる応用例が多いようである。システムをサブルーチンに分割して表現し、各サブルーチンを連結して解くというプログラムの構成となっており、HVACSIM+と同様の方式である。HVACSIM+と同様に各サブルーチンは TYPE と呼ばれており、多くのサードパーティから TYPE が開発され供給されている。プログラムに関するメーリングが活発であり、プログラム購入者以外も参加することが可能である。また、プログラム自体は有償であるが、個々のユーザーやメーカー等が TRNSYS 対応の TYPE を開発して公開していることがあり、プログラムの作成にあたっては参考になる。

6）Energy Plus

Energy Plus は米国において 1998 年から開発が進められている建物熱負荷およびエネルギー消費量のシミュレーションソフトウェアである[1.15) 1.16)]。1980 年頃から開発が始められた BLAST および DOE–2 が前身のソフトウェアであり、1 時間以下の計算時間間隔に対応させるなど、多くの改良が加えられている。米国エネルギー省の主導のもと、ローレンス・バークレー国立研究所、イリノイ大学、カリフォルニア大学が中心となって開発が行われている。かつては、いくつかのライセンスの形態を持っていたが、現在では BSD ライセンスに統一され、ソースコードも入手できる。

7）その他の情報入手源

書籍（プログラムコード自体が解説とともに記載されているもの）による情報としては、以下が挙げられる。また、古い論文に関しては、論文自体にプログラムコードが記載されていることも多い。

・パソコンによる空気調和計算法[1.2)]

工学院大学の宇田川光弘先生による著書である。湿り空気の状態値、日射量、気象データ、窓の熱取得、壁体の非定常熱伝導などの基本的な計算方法に加え、熱負荷、室温計算、放射冷暖房の計算方法などが記されている。また、BASIC 言語を用いた具体的なプログラムコードも記載されている。

・数値計算で学ぶ光と熱の建築環境学[1.4)]

武蔵工業大学の宿谷昌則先生による著書である。タイトルにある通り、特に光と熱にスポットライトがあてられており、日射量や照度の計算、窓やブラインドの計算、室の熱負荷等の計算方法が記されている。また、FORTRAN 言語による具体的なプログラムコードも記載されている。

・Methodologies for the Design and Control of Central Cooling Plants[1.6)]

James Edward Braun 氏の Wisconsin 大学における博士論文であり、Wisconsin 大学の web サイトより入手することが可能である。中央熱源に加え、凝縮器側の設備機器である冷却塔、蒸発器側の機器であるコイルなどについて理論およびプログラムソースが提示されている。

・建築家のための熱環境解析入門[1.5)]

北海道大学の荒谷登先生と北海道工業大学の鈴木憲三先生による著書である。逐次積分法を用いた熱負荷計算法についての解説書であり、BASIC 言語を用いたプログラムコードが掲載されている。

1.3　変数の名称

モデルを作成するための理論が記された文献には、変数を用いて基礎式が記載されることが一般的であり、本書においても多くの変数を用いて理論式や実験式を記載している。これらの変数名称の命名法は文献著者の自由であるため一概には言えないが、実際にはある程度の暗黙のルールがあり、これを認識することで文献の解読は容易になる。表 1.1 に良く用いられる変数名称、意味、英文表記の対応を示した[1.22)]。

表 1.1　変数名称と意味の対応

変数名称	意味	変数名称	意味	変数名称	意味	変数名称	意味
A	面積	I	日射	S	表面積	γ	比重量
a	熱拡散率	K	熱貫流率	T	絶対温度	η	効率
c_p	定圧比熱	L	水量	t	時間, 温度	λ	熱伝導率
d	直径	l	長さ	U	熱貫流率	μ	粘性係数
E	エネルギー	m	質量流量	V	体積	ρ	密度
F	形態係数	P	圧力	v	速度	τ	透過率
G	流量	Q	熱量, 体積流量	x	絶対湿度	-	-
H	熱流	R	抵抗	α	熱伝達率	-	-

1.4　プログラム言語の基礎

1.4.1　コーディングルール

既に例示したように、建築環境シミュレーション分野の既存の多くのプログラムは FORTRAN 言語で開発されている。歴史が古いために転用できる既存のプログラム資産が多く、また、ユーザー数が多いということも理由だろう。応用の実績が積まれており、大方のバグが解決されているという点は長所であるが、一方で、初学者にとってのプログラムの読みやすさという点では、近年のプログラム言語に劣るように思う。特に、計算機資源が乏しかった時代のプログラムは、如何にメモリの消費を抑え、計算機負荷を減らすかという点を重視してプログラムが書かれていることが多く、読みやすさと再利用性という観点が乏しい場合が多い。例えば、我が国の代表的エネルギー計算プログラムである HASP/ACLD/ACSS の開発の契機となった調査報告である「国内の空調装置シミュレーションプログラムの比較[1.21)]」は 1981 年の資料であるが、同資料においては、概略処理速度、概略処理ステートメント数、使用メモリサイズなどが主要な性能比較項目として挙げられている。データの通信・保存・演算に必要なコストが相対的に大きく、計算機という資源を如何に効率的に使用するかが、ソフ

トウェアにとって主要な価値であった時代の視点である。近年の計算機能力の増大速度を前提とするならば、多少迂遠な処理となり、計算速度が落ちようとも、プログラムの可読性を向上させて人的資源の節約を狙った方が、全体としては効率が良いと筆者は考える†1)。

プログラムをコンパイルするためには最低限、プログラム言語自体の文法に従う必要があるが、その他にも、上記のような観点から、可読性を向上させるための規約の提案が行われている[1.1) 1.20)]。これから新規のプログラム開発を行うのであれば、これらの文献などを参照して一定の規約に従った、読みやすいコードとすることが望ましい。コーディングルールに関する資料をいくつか読めば、様々なルールにはそれぞれ一長一短があり、絶対的な正解は無いということがわかる。重要な事は一連のプログラムコードをできる限り同じルールに従って書くということにあり、基準が無いままに場当たり的に書き散らかしたソースコードは読み手に混乱を与える†2)。本書で使用する C#言語について、いくつかの基本的なコーディングルールを以下に記す。

1）名称に略称の使用は避ける

FORTRAN77 では、変数および関数の名称は 6 文字までとなっていたが、近年のプログラム言語にはこのような制約はない。主要な変数や関数の名称には略称を用いず、読むだけで機能が類推できるような名称とする。

× GetAlphaR

○ GetRadiativeHeatTransferCoefficient

○ GetRadiativeCoefficient

ただし、意味内容が理解できることにこだわりすぎると長過ぎる変数だらけになり、かえって可読性が低下する危険性もある。プロパティのようにプログラムの様々な行で繰り返し出現するような場合にはわかりやすい名称とし、10 行程度の繰り返し文の中でしか登場しないような局所的な変数の場合には略称の使用も認めると良いだろう。

2）大文字と小文字を使い分ける

クラスや変数の命名の方法には、Pascal 形式と Camel 形式がある。Pascal 形式は、名称を構成する単語の 1 文字目を大文字とする方式である。Camel 形式は名称を構成する最初の単語は全て小文字とし、以降の単語の 1 文字目を大文字とする方式である†3)。本書ではクラス名称を含め public メンバについては Pascal 形式を用い、インスタンス変数やパラメータに関しては Camel 形式を用いる。また、定数に関しては全てを大文字で表現する。

プログラム 1.1　命名規則の例

```
1 public class MoistAir{    //←クラス名称は Pascal 形式
2   const double VAPORIZATION_LATENT_HEAT = 2501;    //←定数は大文字
3
4   //↓public メソッドは Pascal 形式、パラメータは Camel 形式
5   public double GetDryBulbTemperature(double enthalpy, double relativeHumidity){
6     ・・・・・
7   }
8 }
```

3）名称は英語とする

プログラム言語はアルファベットであるため、ローマ字綴りによる日本語の名称は非常に読みづら

†1　人的資源の節約に関連するもう一つの話題としては、仮想マシンを用いたクロスプラットフォーム開発がある。例えば C#言語で開発を行うと、PC と携帯端末でコア部のコードを再利用できる。

†2　このあたりは建築設計図書に通ずるものがある。

†3　名称の途中に挟まれる大文字が、あたかもラクダのコブのように見えることが語源らしい。

い。名称は英語とする。

× NetsuKoukanKi

○ HeatExchanger

表 1.2 に主要な用語の変数名称の例を示す。建築躯体の計算に関する用語については『JIS A0202 断熱用語[1.25)]』を参照した。設備に関する用語については『空気調和・衛生用語辞典[1.26)]』を参照した。なお、変数名称は Camel 形式で記したが、上記のルールに従い、必要に応じて Pascal 形式に変更して使用する。

表 1.2　主要な用語の変数名称例

用語	単位 (例)	変数名称	出典	用語	単位 (例)	変数名称	出典
熱流	W	heatFlowRate	A	外表面	-	exteriorSurface	A
熱伝導率	W/(m·K)	thermalConductivity	A	対流	-	convection	A
熱コンダクタンス	W(m^2·K)	thermalConductance	A	放射	-	radiation	A
熱伝達率	W(m^2·K)	surfaceCoefficientOfHeatTransfer	A	接触	-	contact	A
熱貫流率・熱通過率	W(m^2 K)	thermalTransmittance	A	空間	-	space	A
熱容量	J/K	heatCapacity	A	周囲	-	ambient	A
比熱	J/(kg·K)	specificHeat	A	冷水	-	chilledWater	B
容積比熱	J/(m^3·K)	volumetricSpecificHeat	A	冷却水	-	coolingWater	B
換気量	m^3/h	ventilationAirVolume	A	温水	-	hotWater	B
換気回数	回/h	airChangesRate	A	ブライン	-	brine	B
密度	kg/m^3	density	A	電力	W	electricPower	B
湿り空気	-	moistAir	A	電力量	Wh	electricEnergy	B
相対湿度	%	relativeHumidity	A	ガス量	m^3	gasVolume	-
絶対湿度	kg/kg	humidityRatio	-	冷却塔	-	coolingTower	B
比エンタルピー	J/kg	specificEnthalpy	A	ポンプ	-	pump	B
空気流量	m^3/s	airFlowRate	A	熱源	-	heatSource	B
放射熱流量	W	radiantHeatFlowRate	A	ヘッダ	-	header	B
黒体	-	blackBody	A	コイル	-	coil	B
入射量	W/m^2	irradiance	A	空気調和機	-	airHandlingUnit	B
放射量	W/m^2	radiosity	A	FCU	-	fanCoilUnit	B
吸収率	-	absorptance	A	加湿器	-	humidifier	B
反射率	-	reflectance	A	加湿負荷	W	humidifyingLoad	B
透過率	-	transmittance	A	水蓄熱	-	waterThermalStorage	B
摂氏温度	ºC	temperature	-	氷蓄熱	-	iceStorageSystem	B
絶対温度	K	thermodynamicTemperature	A	特性曲線	-	characteristicCurve	B
長さ	m	length	A	定格~~	-	rated~, nominal~	B
幅	m	width	A	揚程	kPa	pumpHead	B
面積	m^2	area	A	水量（質量）	kg/s	waterMassFlowRate	-
容積	m^3	volume	A	水量（体積）	L/min	waterFlowRate	-
内側	-	interior	A	潜熱負荷	W	latentHeatLoad	B
外側	-	exterior	A	全熱負荷	W	totalHeatLoad	B
表面	-	surface	A	顕熱負荷	W	sensibleHeatLoad	B
内表面	-	interiorSurface	A	-	-	-	-

出典　A: JIS A 0202, B: 空気調和・衛生用語辞典

4）コメントを使い単位を明記する

設備シミュレーションで行われる計算の多くは物理式に基づいたものであり、入力変数の単位の組み合わせが出力の単位と合致していることが重要である。例えば、体積流量であるのか質量流量であるのか、あるいは同じ質量流量であっても、kg/sec であるか g/hour であるかを間違えると、全く異なった結果が出力される。そもそもの単位の換算ミスが無いことを確認するという意味と、プログラ

ムの使用者が正しく入力条件を設定できるようにするという意味で、コメントを使って単位を明記することが望ましい。

単位に関連して特に意識すべき概念は量と流束の違いである。例えば熱に関しては、量としての単位である熱量と、時間あたりの量である熱流（または熱流束）があるが、初学者はこれらを混同した間違いを起こしやすい。前者の単位は MJ や kWh などであり、後者の単位は MJ/h や kW（=kJ/s）である。本書では日本語としても両者の違いを表現する（消費電力量、熱量に対して消費電力、熱流、など）ように心がけたが、慣例的に流束であっても●●量と表現することが多い場面もあり、必ずしも貫徹した表現とはできていない。必要に応じて章末の記号表の単位を参照していただきたい。

1.4.2 デバッグ

プログラムの記述に欠陥があると、プログラムがコンパイルできなかったり、開発者の意図とは異なる計算結果が得られる。このような欠陥をバグ[†1]と呼び、このバグを取り除く作業を「デバッグ」と呼ぶ。

プログラミングの初学者の多くは、比較的長いプログラムを書き上げてから一気に実行させようとするが、多くのバグが発生するために思うような結果が得られずに挫折する。開発を行う際には、一挙に巨大なプログラムを書くのではなく、迂遠であっても細かな機能に分割して要素要素でデバッグを積み重ねることが効率的である[†2]。例えば室温変動の計算であれば、まずは壁体の熱流、窓の熱流、室内顕熱発生、日射、換気などの要素に分割し、それぞれの熱流計算が正常に動作することを確認する。また、動作確認の際には、コンピュータを用いずにも予想できるような単純な境界条件から始めると良い。例えば壁体熱流計算であれば、最初から両側の空気温度が揺れ動くような境界条件で動作検証をするのではなく、温度一定の条件から始める。こうすれば手計算による定常計算と比較が可能であるため、誤りを発見しやすい。また、複層ガラスの日射透過の計算であれば透過率を 100 %とした場合（→総合透過率も 100 %となる）、反射率を 100 %とした場合（→総合反射率も 100 %となる）で動作検証をする。

Visual Studio や Eclipse など、近年のプログラム開発環境ではデバッガと呼ばれるデバッグ作業を効率化させる仕組みが備わっている。デバッガを用いるとプログラムの動作中に特定の位置で一時停止させ、プログラムを一行ずつ実行させたり、変数に代入されている値を確認したり、変数値を強制的に書き換えたりができる。プログラムの開発速度が大きく向上するため、使いこなす必要がある[†3]。

1.5 オブジェクト指向プログラミング

1.5.1 概要と効果

本書ではオブジェクト指向プログラミング言語（OOP: Object Oriented Programming）である C#を用いる。OOP では「データ」と「そのデータを処理する関数」とをまとめた概念を「オブジェクト」と呼び、オブジェクトとオブジェクトとの相互通信によって計算を進める。これに対してオブジェクト指向ではない言語、例えば FORTRAN では、データと関数はパッケージ化されていないた

†1 1980 年頃まではプログラムは穴を開けた紙（パンチカードと呼ばれる）で保存することが主流であった。この紙が虫に食われるとプログラムの誤動作に繋がり、これがバグという言葉の由来であるらしい。

†2 中部大学の猪岡教授に HASP/ACSS 開発時の話を聞いたことがある。開発時間の半分以上は個別の関数の入出力仕様などのプログラムの構想を練るために費やされ、その後、半年程度を使って小さなサブルーチン集が開発された。サブルーチンが揃った後に全体のプログラムを組んだが、仕様が明確でパーツとしては動作確認が行われていたため、全体としても大きな問題もなく速やかに動作まで漕ぎ着けたという。

†3 筆者の知る限り、建築環境分野で現在も主流の FORTRAN 言語のコンパイラは、総じてデバッガの性能が比較的弱い。幾人かの知人は WRITE 文を用いて要所で値を書き出すことでデバッグを行っている（「デバッグ WRITE」と呼ばれる）ようで、明らかにプログラム開発に必要な時間が拡大してしまっている。

め、関数を操作する上位の関数によってデータを管理する必要がある。OOP のこのような特徴は、特に個別の機器を組み合わせてシステムとしての設備を計算する際に有益である。 C#と FORTRAN でそれぞれ熱源一次システムのモデルを作成した場合の、データ管理の概念を図 1.1 に示す。

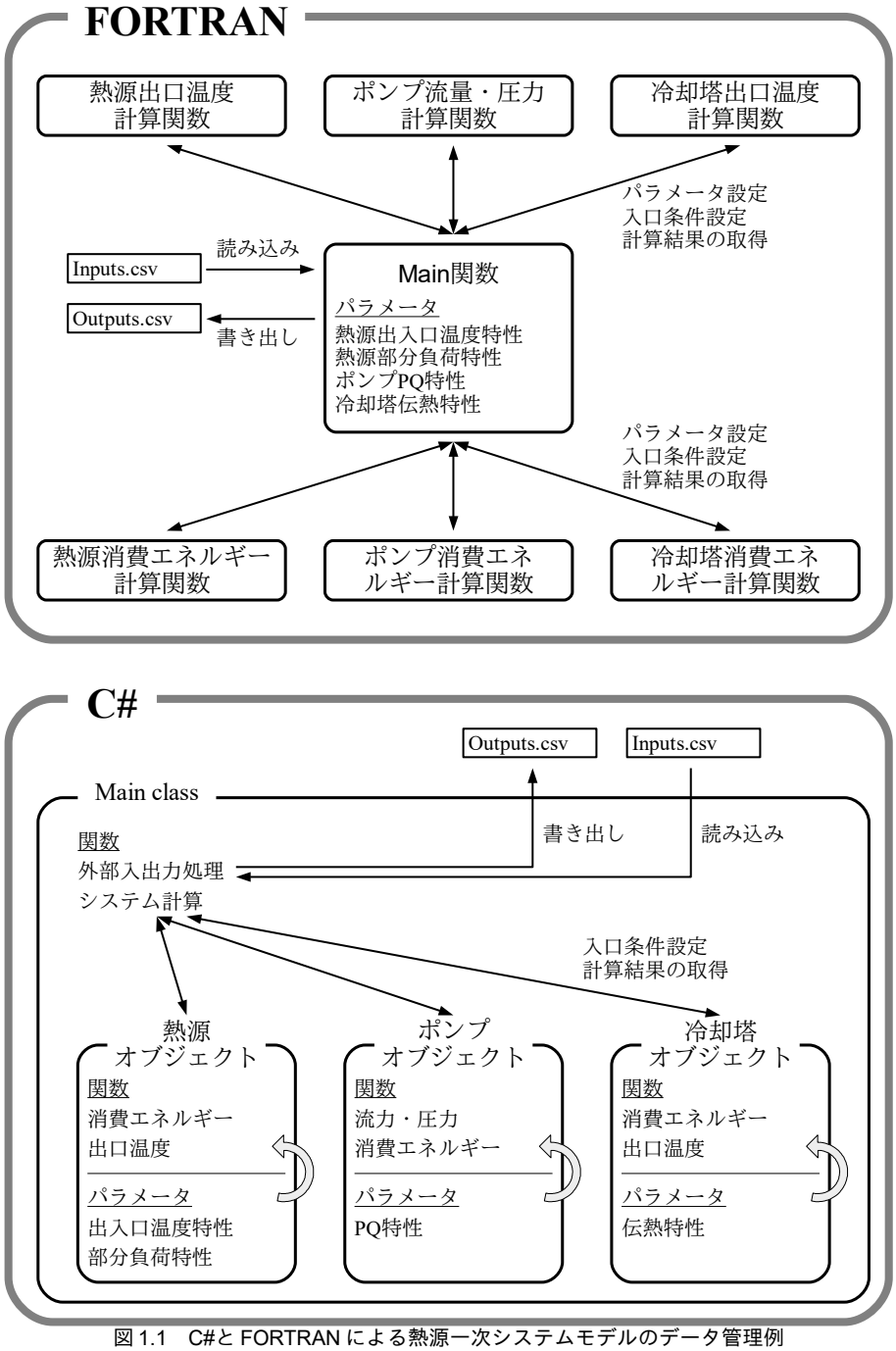


図 1.1　C#と FORTRAN による熱源一次システムモデルのデータ管理例

FORTRAN でシステムモデルを計算する場合には、熱源や冷却塔などの機器の情報（特性曲線や幾何学的形状）は、個別の機器の計算処理とは切り離されて、上位の関数が一元的に管理する。個別の機器の出力を知るためには、上位の関数が適切なデータを渡して計算を行う必要がある。例えば熱源のエネルギー消費量を知りたい場合には Inputs.csv から読み込んだ境界条件と Main 関数で管理してい

る熱源パラメータを合わせて熱源出口温度計算関数に渡して計算を行い、得られた出口温度と熱源パラメータを熱源消費エネルギー関数に渡して計算を行う。一方で、C#でシステムモデルを計算する場合には、個別の機器を表現するオブジェクトがそれぞれ自分自身の情報を保持している。従って、上位のオブジェクトは下位のオブジェクトの内部の詳細な情報を把握する必要はなく、必要な時に必要な出力を求めるだけで良い。前記の熱源エネルギー消費の例では、Main 関数は Inputs.csv から読み込んだ境界条件を熱源オブジェクトに渡すだけでよく、熱源オブジェクトは自身で管理している熱源パラメータにもとづいて出口温度とエネルギー消費量を更新する。

システムが複雑であるほど OOP の利点は大きい[†1)]。図 1.1 は熱源、ポンプ、冷却塔がそれぞれ 1 台しかない場合であるが、一般の熱源システムはこれよりも遥かに多くの機器からなる集合体であり、これらの全ての機器特性を上位の関数が一元的に管理することは非常に煩雑である。参考に HASP/ACLD のプログラムの冒頭を図 1.2 に示す。COMMON 文を用いて巨大な広域実数値配列が定義されており、プログラムの可読性、堅牢性に問題があることがわかる[†2)]。

```
BLOCK DATA BLOCK
 COMMON /CN1/TCM1(94),TCM2(90) /CN2/TCW(2,30),IGS(2,90)
1 /CN3/RL(54),RM(9),RE(27),RG(18),AM(27),RZ1(9,9),RZ2(8,9)
2 /CN4/IOR,IOW,IIW,ISC,ISE,ISP,IOUT,PI,RADI,TPR,TV1,AHR,BL(7),
3 WX(3),WXB(2)
 DATA TCM1/.019,.516,1.892,.052,38.7,180.6,332.,.0  ,.0  ,.0  ,2.7,
* 1.2,1.3,.8,.9,.4,.53,.0  ,.0  ,.0  ,1.3,1.2,.67,.15,.95,.46,1.3,
* 1.03,.0  ,.0  ,.68,.15,.64,.59,.67,1.1,.55,.86,.0  ,.0  ,.163,
* .224,.095,.18,.13,.06,.069,.0  ,.0  ,.0  ,.16,.15,.12,.16,.0  ,
* .0  ,.0  ,.0  ,.0  ,.0  ,.048,.052,.12,.19,.15,.16,.038,.07 ,
* .0  ,.0  ,.036,.034,.07 ,.036,.044,.055,.0  ,.0  ,.0  ,.0  ,.040,
* .032,.022,.024,.025,.043,.038,.031,.0  ,.0  ,.18,.08,.0 ,.0 /
 DATA TCM2/.31,998.,449.,43.,865.,567.,824.,.0  ,.0  ,.0  ,571.,
* 399.,744.,468.,798.,428.,370.,.0  ,.0  ,.0  ,456.,462.,384.,156.,
* 430.,375.,380.,435.,.0  ,.0  ,390.,246.,330.,269.,457.,480.,332.,
* 360.,.0  ,.0  ,350.,448.,220.,217.,69.,62.,76.,.0  ,.0  ,.0  ,
* 186.,155.,124.,171.,.0  ,.0  ,.0  ,.0  ,.0  ,78.,93.,234.,
* 326.,171.,226.,9.3,248.,.0  ,.0  ,4.8,6.4,2.4,20. ,240.,60.,.0  ,
* .0  ,.0  ,.0  ,5.4,8.4,12.,11.3,11.3,9.,15.,12.,.0  ,.0  /
 DATA TCW/.180,.230,.183,.233,.184,.234,.187,.237,.193,.243,.183,.2
1 33,.183,.233,.183,.233,.180,.230,.183,.233,.184,.234,.187,.237,.1
2 93,.243,.0  ,.0  ,.0  ,.0  ,.0  ,.0  ,.344,.394,.350,.400,.353,.4
3 03,.359,.409,.0  ,.0  ,.0  ,.0  ,.344,.394,.350,.400,.353,.403,.3
4 59,.409,.0  ,.0  ,.0  ,.0  ,.0  ,.0  ,.0  ,.0  /
 DATA IGS/99,1,96,2,95,2,93,2,90,2,96,2,96,2,96,2,79,7,67,10,63,10,
154,13,41,16, 0, 0, 0, 0, 0, 0,90,1,84,4,81,5,77,6, 0,0, 0, 0,69,5,
256,7,50,8,40,10, 0, 0, 0, 0, 0, 0, 0, 0, 0,27,26,27,26,26,27,26,27,25
3,28, 0, 0, 0, 0, 0, 0,25,28,24,28,23,28,21,28,20,27, 0, 0, 0, 0, 0
4, 0,24,28,23,29,24,29,23,30, 0, 0, 0, 0,20,25,17,24,17,22,15,24, 0
5, 0, 0, 0, 0, 0, 0, 0,25,39,25,39,25,39,24,39,24,38,27,47,33,57,19
6,16,23,38,22,36,22,34,20,34,18,31, 0, 0, 0, 0, 0, 0,22,39,22,39,22
7,39,21,39, 0, 0, 0, 0,18,34,16,30,15,29,13,27, 0, 0, 0, 0, 0, 0, 0
8, 0/
 DATA RL,RM/.3993,.0398,.9337,.4023,.0516,.9137,.4095,.0824,.8605,
1 .5002,.0369,.9261,.5024,.0457,.9081,.4999,.0709,.8581,.6901,
2 .0290,.9064,.6911,.0328,.8937,.6760,.0478,.8525,.5261,.0392,
3 .9173,.5279,.0464,.9016,.5160,.0699,.8556,.5694,.0377,.9124,
4 .5708,.0437,.8981,.5555,.0648,.8542,.0  ,.0  ,.0  ,.0  ,
5 .0  ,.0  ,.0  ,.0  ,.0  ,.6554,.0290,.9158,.6566,.0342,
6 .9005,.6469,.0511,.8552/
```

広域定数宣言

壁素材 熱伝導比抵抗

壁素材 容積比熱

ガラス 熱貫流抵抗

ガラス 遮蔽係数

照明人体 重み係数

図 1.2　HASP/ACLD における広域実数値配列定義

†1　空気調和・衛生工学会でコンパスと呼ばれるプログラムライブラリの開発が行われたことがあった[1.34)]。本ライブラリには FORTRAN によって記述された建築環境分野の基礎的な関数が多く含まれていたが、複雑な設備機器のモデルは無かった。ライブラリの方針ということもあるだろうが、FORTRAN 言語では多くのパラメータを含む設備機器・設備システムのライブラリは作りづらいという実態はあったと筆者は推測する。

†2　HASP/ACLD の記憶領域は複雑だが、松尾による「プログラミングメモ」を参照すると、その管理は非常に厳格である。各パラメータを保存する領域のサイズと位置が明確に定められており、ポインタを使って入出力処理が行われるため、オブジェクト指向言語を使って場当たり的にオブジェクトを量産するようなプログラムに比較すると、遥かに高速で必要な記憶領域も小さいだろう。一方で、このような美しい最適化は、ソフトウェアを松尾先生という天才と不可分の関係にしてしまうという大きな弱点を持つ。計算資源を節約するか人的資源を節約するかの問題である。

1.5.2　クラスとインスタンス

実際にオブジェクトを作成する際には、クラスとインスタンスという概念が必要になる。例えば熱交換器のオブジェクトを作成するとする。設備システムには数十の熱交換器が含まれるため、毎回 0 からオブジェクトを作成することは手間である。そこで「処理」と「パラメータ」を分離する。処理とは、水を流す、出口状態を推定するなどの計算である。パラメータとは、伝熱面積や伝熱係数などである。パラメータはそれぞれの熱交換器を特徴づける値であるため、熱交換器ごとに異なる値を取るが、処理は物理式などにもとづくため、すべての熱交換器で同じである。そこで、すべての熱交換器オブジェクトで使い回すことができる処理を「型」としてまとめ、予め定義しておく。これを「クラス」と呼ぶ。実際にオブジェクトを作成するときには、型に熱交換器ごとの具体的なパラメータを付与し、「処理」と「パラメータ」の両者を持たせることで実体化する。この実体を「インスタンス」と呼ぶ。

クラスは他のクラスから派生して開発することもできる。例えば熱交換器クラスには流体間の熱交換を行う処理が定義されているが、冷温水コイルクラス、冷却塔クラス、蒸発器クラスなど、2 種類の流体の熱交換を行う機器は多い。そこで、熱交換器クラスから派生させることで熱交換に関する類似の処理を使いまわす[†1)]。多くの派生クラスの生成を意図し、抽象度を高めたクラスを特に「抽象クラス」と呼ぶ。また、派生したクラスを子クラス、派生元を親クラスとも呼ぶ。クラスの派生とインスタンスの生成の概念を図 1.3 に示す。

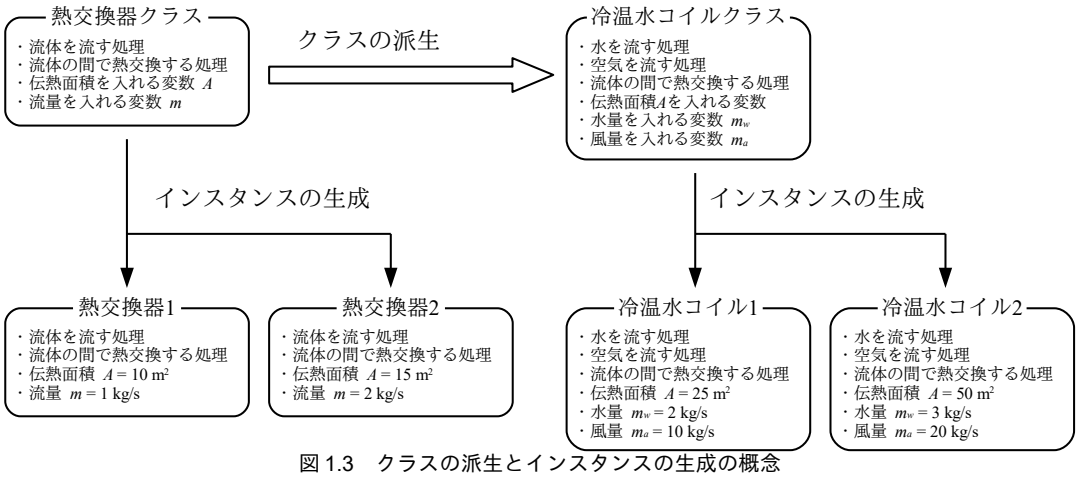


図 1.3　クラスの派生とインスタンスの生成の概念

1.5.3　C#の言語仕様とクラスの開発例

本節では配管のクラスを例示し、具体的な設備シミュレーションプログラムへの応用を通じて C#の基本的な言語仕様を解説する。C#の言語仕様一般に関しては既に解説書が豊富にあるため[1.27) 1.28) 1.29)]、個別機能の詳細についてはそれらを参照されたい[†2)]。

プログラム 1.2 に配管クラスを示す。プログラム内で使用する物理式を式 1.1 に示す[1.32)]。K_L [W/(m·K)]は内部の水から外部の空気までの線熱通過率である。変数は図 1.4 の通りであり、d は直径[m]、λ は熱伝導率[W/(m·K)]、$\alpha_{(c)}$は水と空気の対流熱伝達率[W/(m²·K)]である。

†1　クラスの派生はオブジェクト指向言語の特徴であるが、十分にクラス間のつながりについて設計を行わないで無闇に派生を重ねると、実体がどこにあるのかわからなくなり、むしろプログラムの可読性を損ねる。本書においてもクラスの派生は必要最小限度にとどめている。

†2　本節の主目的はオブジェクト指向言語で記述した具体的な設備クラスの例を提供することである。C#の言語仕様の解説としては不足しており、直感的な理解を促すために厳密には不正確な表現も行っている。専門書を片手に読み進めることを強く推奨する。

$$K_L=\frac{\pi}{\frac{1}{\alpha_{(c)w}d_i}+\frac{2}{\lambda_p}\ln\frac{d_b}{d_i}+\frac{2}{\lambda_I}\ln\frac{d_o}{d_b}+\frac{1}{\alpha_{(c)a}d_o}} \tag{1.1}$$

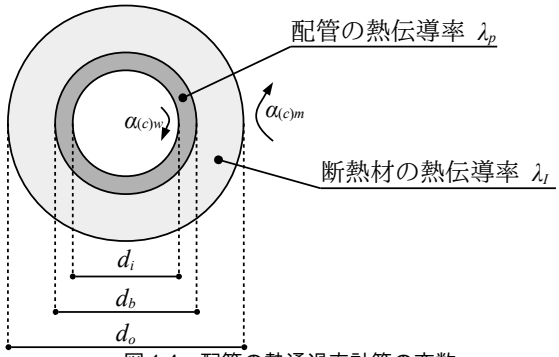


図 1.4　配管の熱通過率計算の変数

プログラム **1.2**　**C#**によるクラスの定義例（配管クラス）

Popolo.HVAC.Circuit.WaterPipe class

```
 1 /* WaterPipe.cs
 2  *
 3  * Copyright (C) 2014 E.Togashi
 4  *
 5  * This program is free software; you can redistribute it and/or modify it under the terms of the GNU General
 6  * Public License as published by the Free Software Foundation; either version 3 of the License, or (at your
 7  * option) any later version.
 8  *
 9  * This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even
10  * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
11  * License for more details.
12  *
13  * You should have received a copy of the GNU General Public License along with this program; if not, write
14  * to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
15  */
16
17 using System;
18
19 using Popolo.ThermophysicalProperty;
20 using Popolo.Numerics;
21
22 namespace Popolo.HVAC.Circuit
23 {
24   /// <summary>水配管</summary>
25   public class WaterPipe : ImmutableWaterPipe, ICircuitBranch
26   {
27
28     /// <summary>大気圧は 101.325kPa（海抜 0m）で一定とする</summary>
29     private const double ATMOSPHERIC_PRESSURE = 101.325;
30
31     /// <summary>配管材料</summary>
32     public enum Material
33     {
34       /// <summary>炭素鋼管</summary>
35       CarbonSteel = 0,
36       /// <summary>ビニル管</summary>
37       Plastic = 1,
38       /// <summary>ステンレス</summary>
39       StainlessSteel = 2
40     }
41
42     /// <summary>断熱材料</summary>
43     public enum Insulator
44     {
45       /// <summary>無し</summary>
46       None = 0,
47       /// <summary>ロックウール</summary>
48       RockWool = 1,
49       /// <summary>グラスウール</summary>
50       GlassWool = 2,
51       /// <summary>ポリスチレン</summary>
52       Polystyrene = 3
53     }
```

```
54
55      /// <summary>線熱通過率（水と空気の対流熱伝達率は含まない）[W/(mK)]</summary>
56      private double linearThermalTransmittance = 0;
57
58      /// <summary>配管長[m]</summary>
59      private double length = 1;
60
61      /// <summary>管内表面表面の粗度[m]</summary>
62      private double roughness;
63
64      /// <summary>内径[m]を取得する</summary>
65      public double InnerDiameter { get; private set; }
66
67      /// <summary>外径[m]を取得する</summary>
68      public double OuterDiameter { get; private set; }
69
70      /// <summary>入口水温[C]を取得する</summary>
71      public double InletWaterTemperature { get; private set; }
72
73      /// <summary>周囲の乾球温度[C]を取得する</summary>
74      public double AmbientTemperature { get; private set; }
75
76      /// <summary>周囲の絶対湿度[kg/kg]を取得する</summary>
77      public double AmbientHumidityRatio { get; private set; }
78
79      /// <summary>線熱通過率[W/(mK)]を取得する</summary>
80      public double LinearThermalTransmittance { get; private set; }
81
82      /// <summary>配管長[m]を設定・取得する</summary>
83      public double Length
84      {
85        get { return length; }
86        set { if (0 < value) length = value; }
87      }
88
89      /// <summary>管内表面の粗度[m]を設定・取得する</summary>
90      public double Roughness
91      {
92        get { return roughness; }
93        set { if (0 < value) roughness = value; }
94      }
95
96      /// <summary>熱損失[kW]を取得する</summary>
97      public double HeatLoss { get; private set; }
98
99      /// <summary>出口水温[C]を取得する</summary>
100     public double OutletWaterTemperauture { get; private set; }
101
102     /// <summary>配管の熱伝導率[W/(mK)]を取得する</summary>
103     public double PipeThermalConductivity { get; private set; }
104
105     /// <summary>断熱材の熱伝導率[W/(mK)]を取得する</summary>
106     public double InsulatorThermalConductivity { get; private set; }
107
108     /// <summary>断熱材厚[m]を取得する</summary>
109     public double InsulatorThickness { get; private set; }
110
111     /// <summary>インスタンスを初期化する</summary>
112     /// <param name="length">長さ[m]</param>
113     /// <param name="innerDiameter">内径[m]</param>
114     /// <param name="roughness">配管粗度[m]</param>
115     /// <param name="thickness">配管厚み[m]</param>
116     public WaterPipe(double length, double innerDiameter, double roughness, double thickness)
117     {
118       InnerDiameter = innerDiameter;
119       Length = length;
120       Roughness = roughness;
121       OuterDiameter = InnerDiameter + 2 * thickness;
122
123       //断熱材無しとする
124       RemoveInsulator();
125     }
126
127     /// <summary>インスタンスを初期化する</summary>
128     /// <param name="length">長さ[m]</param>
129     /// <param name="innerDiameter">内径[m]</param>
130     /// <param name="material">配管材料</param>
131     public WaterPipe(double length, double innerDiameter, Material material)
132     {
133       InnerDiameter = innerDiameter;
```

```
134       Length = length;
135       double dth;
136       switch (material)
137       {
138         case Material.CarbonSteel:
139           Roughness = 0.03e-3;
140           dth = 1.0796e-3 * Math.Pow(1000 * innerDiameter, 0.3181);
141           break;
142         case Material.Plastic:
143           Roughness = 0.03e-3;
144           dth = 0.5196e-3 * Math.Pow(1000 * innerDiameter, 0.5746);
145           break;
146         default://SUS
147           Roughness = 0.015e-3;
148           dth = 0.7250e-3 * Math.Pow(1000 * innerDiameter, 0.4581);
149           break;
150       }
151       OuterDiameter = InnerDiameter + 2 * dth;
152
153       //断熱材無しとする
154       RemoveInsulator();
155     }
156
157     /// <summary>初期化処理</summary>
158     private void initialize()
159     {
160       //線熱通過率[W/(mK)]を初期化する
161       linearThermalTransmittance = GetPipeLinearThermalTransmittance
162         (InnerDiameter, InsulatorThermalConductivity, InsulatorThickness, PipeThermalConductivity,
163         0.5 * (OuterDiameter - InnerDiameter));
164
165       //熱流を初期化する//水量は水速2.0m/s相当
166       UpdateHeatFlow(7, InnerDiameter * InnerDiameter / 4d * Math.PI * 2 * 1000, 25, 0.02);
167     }
168
169     /// <summary>熱流を更新する</summary>
170     /// <param name="inletWaterTemperature">入口水温[C]</param>
171     /// <param name="waterFlowRate">水量[m3/s]</param>
172     /// <param name="ambientTemperature">周囲の乾球温度[C]</param>
173     /// <param name="ambientHumidityRatio">周囲の絶対湿度[C]</param>
174     public void UpdateHeatFlow(double inletWaterTemperature,
175       double waterFlowRate, double ambientTemperature, double ambientHumidityRatio)
176     {
177       VolumetricFlowRate = waterFlowRate;
178       InletWaterTemperature = inletWaterTemperature;
179       AmbientTemperature = ambientTemperature;
180       AmbientHumidityRatio = ambientHumidityRatio;
181
182       //体積流量[m3/s]を質量流量[kg/s]に換算
183       double mw = waterFlowRate / Water.GetLiquidDensity(inletWaterTemperature);
184
185       //水側の対流熱伝達率[W/(m2K)]を計算
186       double aw = GetInsideHeatTransferCoefficient(InletWaterTemperature, InnerDiameter, mw);
187
188       //配管表面温度は水温と空気温度の平均と仮定
189       double ts = 0.5 * (InletWaterTemperature + AmbientTemperature);
190
191       double err = 1;
192       int iterNum = 0;
193       while (0.01 < err)
194       {
195         if (20 < iterNum) throw new Exception("WaterPipe iteration error");
196
197         //空気側の対流熱伝達率[W/(m2K)]を計算
198         double aa = GetOutSideHeatTransferCoefficient
199           (AmbientTemperature, AmbientHumidityRatio, OuterDiameter, ts);
201
202         //線熱通過率[W/(mK)]を更新する
203         LinearThermalTransmittance = GetAirToWaterLinearThermalTransmittance
204           (aw, aa, InnerDiameter, OuterDiameter, linearThermalTransmittance);
205
206         //線表面温度[C]を更新して誤差を評価
207         double tsOld = ts;
208         ts = AmbientTemperature - LinearThermalTransmittance
209           * (AmbientTemperature - InletWaterTemperature) / aa;
210         err = Math.Abs(ts - tsOld);
211         iterNum++;
212       }
213
214       //熱損失と出口水温を更新
```

```
215       HeatLoss = LinearThermalTransmittance * Length
216         * (AmbientTemperature - InletWaterTemperature) / 1000d;
217       double cpw = Water.GetLiquidIsobaricSpecificHeat(InletWaterTemperature);
218       OutletWaterTemperauture = HeatLoss / (mw * cpw) + InletWaterTemperature;
219     }
220
221     /// <summary>断熱材を設定する</summary>
222     /// <param name="thickness">断熱材厚み[m]</param>
223     /// <param name="insulator">断熱材種別</param>
224     public void SetInsulator(double thickness, Insulator insulator)
225     {
226       InsulatorThickness = thickness;
227       switch (insulator)
228       {
229         case Insulator.RockWool:
230           SetInsulator(thickness, 0.044);
231           break;
232         case Insulator.GlassWool:
233           SetInsulator(thickness, 0.043);
234           break;
235         case Insulator.Polystyrene:
236           SetInsulator(thickness, 0.043);
237           break;
238         default:
239           throw new Exception("Invalid Insulator");
240       }
241     }
242
243     /// <summary>断熱材を設定する</summary>
244     /// <param name="thickness">断熱材厚み[m]</param>
245     /// <param name="thermalConductivity">断熱材熱伝導率[W/(mK)]</param>
246     public void SetInsulator(double thickness, double thermalConductivity)
247     {
248       InsulatorThickness = thickness;
249       InsulatorThermalConductivity = thermalConductivity;
250       initialize();
251     }
252
253     /// <summary>断熱材を取り外す</summary>
254     public void RemoveInsulator()
255     {
256       InsulatorThickness = 0.0d;
257       InsulatorThermalConductivity = 0.04d;
258       initialize();
259     }
260
261     /// <summary>配管サイズを変更する</summary>
262     /// <param name="innerDiameter">内径[m]</param>
263     /// <param name="thickness">厚み[m]</param>
264     public void ChangeSize(double innerDiameter, double thickness)
265     {
266       if ((0 < innerDiameter) && (0 < thickness))
267       {
268         InnerDiameter = innerDiameter;
269         OuterDiameter = InnerDiameter + 2 * thickness;
270         initialize();
271       }
272     }
273
274     /// <summary>配管熱伝導率を設定する</summary>
275     /// <param name="thermalConductivity">熱伝導率[W/(mK)]</param>
276     public void SetPipeThermalConductivity(double thermalConductivity)
277     {
278       if (0 < thermalConductivity) PipeThermalConductivity = thermalConductivity;
279       initialize();
280     }
281
282     /// <summary>配管内表面の対流熱伝達率[W/(m2･K)]を計算する</summary>
283     /// <param name="waterTemperature">水温[C]</param>
284     /// <param name="diameter">直径[m]</param>
285     /// <param name="waterFlowRate">水量[kg/s]</param>
286     /// <returns>対流熱伝達率[W/(m2･K)]</returns>
287     /// <remarks>乱流かつ発達した流れの場合</remarks>
288     public static double GetInsideHeatTransferCoefficient
289       (double waterTemperature, double diameter, double waterFlowRate)
290     {
291       //水の物性を計算
292       //動粘性係数[m2/s], 熱拡散率[m2/s], 熱伝導率[W/(m･K)], 密度[kg/m3]
293       double v = Water.GetLiquidDynamicViscosity(waterTemperature);
294       double a = Water.GetLiquidThermalDiffusivity(waterTemperature);
```

```
295       double lam = Water.GetLiquidThermalConductivity(waterTemperature);
296       double rho = Water.GetLiquidDensity(waterTemperature);
297
298       //配管内流速[m/s]からヌセルト数を計算
299       double u = waterFlowRate / (rho * diameter * diameter / 4 * Math.PI);
301       double reNumber = u * diameter / v;
302       double prNumber = v / a;
303       double nuNumber = 0.023 * Math.Pow(reNumber, 0.8) * Math.Pow(prNumber, 0.4);
304
305       //ヌセルト数から対流熱伝達率[W/m2K]を計算
306       return nuNumber * lam / diameter;
307     }
308
309     /// <summary>配管外表面の自然対流熱伝達率[W/(m2･K)]を計算する</summary>
310     /// <param name="drybulbTemperature">湿り空気の乾球温度[C]</param>
311     /// <param name="humidityRatio">湿り空気の絶対湿度[kg/kg]</param>
312     /// <param name="diameter">直径[m]</param>
313     /// <param name="surfaceTemperature">配管表面温度[C]</param>
314     /// <returns>配管外表面の自然対流熱伝達率[W/(m2･K)]</returns>
315     public static double GetOutSideHeatTransferCoefficient
316       (double drybulbTemperature, double humidityRatio, double diameter, double surfaceTemperature)
317     {
318       //湿り空気の物性を計算
319       //動粘性係数[m2/s], 熱拡散率[m2/s], 膨張率[1/K], 熱伝導率[W/(m･K)]
320       double v = MoistAir.GetDynamicViscosity(drybulbTemperature, humidityRatio, ATMOSPHERIC_PRESSURE);
321       double a = MoistAir.GetThermalDiffusivity(drybulbTemperature, humidityRatio, ATMOSPHERIC_PRESSURE);
322       double beta = MoistAir.GetExpansionCoefficient(drybulbTemperature);
323       double lam = MoistAir.GetThermalConductivity(drybulbTemperature);
324
325       //グラフホフ数の計算
326       double grNumber = 9.8 * Math.Pow(diameter, 3) * beta *
327         Math.Abs(surfaceTemperature - drybulbTemperature) / (v * v);
328
329       //プラントル数の計算
330       double prNumber = v / a;
331
332       //ヌセルト数の計算
333       double nuNumber;
334       double grpr = grNumber * prNumber;
335       if (grpr < 1e10) nuNumber = 0.53 * Math.Pow(grpr, 0.25);
336       else nuNumber = 0.13 * Math.Pow(grpr, 0.333);
337
338       //ヌセルト数から対流熱伝達率[W/(m2K)]を計算
339       return nuNumber * lam / diameter;
340     }
341
342     /// <summary>水から空気までの線熱通過率[W/(mK)]を計算する</summary>
343     /// <param name="insideHeatTransferCoef">水側対流熱伝達率[W/(m2K)]</param>
344     /// <param name="outsideHeatTransferCoef">空気側対流熱伝達率[W/(m2K)]</param>
345     /// <param name="innerDiameter">内径[m]</param>
346     /// <param name="outerDiameter">外径[m]</param>
347     /// <param name="thermalConductance">内表面から外表面までの線熱通過率[W/(mK)]</param>
348     /// <returns>水から空気までの線熱通過率[W/(mK)]</returns>
349     public static double GetAirToWaterLinearThermalTransmittance
350       (double insideHeatTransferCoef, double outsideHeatTransferCoef,
351       double innerDiameter, double outerDiameter, double thermalConductance)
352     {
353       double p2 = 2 * Math.PI;
354       double kl = 1 / (insideHeatTransferCoef * innerDiameter) +
355         1 / (outsideHeatTransferCoef * outerDiameter) + p2 / thermalConductance;
356       return p2 / kl;
357     }
358
359     /// <summary>配管の内外表面間の線熱通過率[W/(mK)]を計算する</summary>
360     /// <param name="innerDiameter">配管の内径[m]</param>
361     /// <param name="insulatorThermalConductivity">断熱材の熱伝導率[W/(mK)]</param>
362     /// <param name="insulatorThickness">断熱材の厚み[m]</param>
363     /// <param name="pipeThermalConductivity">配管の熱伝導率[W/(mK)]</param>
364     /// <param name="pipeThickness">配管の厚み[m]</param>
365     /// <returns>配管の内外表面間の線熱通過率[W/(mK)]</returns>
366     public static double GetPipeLinearThermalTransmittance
367       (double innerDiameter, double insulatorThermalConductivity, double insulatorThickness,
368       double pipeThermalConductivity, double pipeThickness)
369     {
370       double d_b = innerDiameter + pipeThickness;
371       double d_o = d_b + insulatorThickness;
372       double klc = 1 / pipeThermalConductivity * Math.Log(d_b / innerDiameter)
373         + 1 / insulatorThermalConductivity * Math.Log(d_o / d_b);
374       return Math.PI / klc;
375     }
```

```
376   }
377
378   /// <summary>読み取り専用の配管</summary>
379   public interface ImmutableWaterPipe
380   {
381     /// <summary>内径[m]を取得する</summary>
382     double InnerDiameter { get; }
383
384     /// <summary>外径[m]を取得する</summary>
385     double OuterDiameter { get; }
386
387     /// <summary>入口水温[C]を取得する</summary>
388     double InletWaterTemperature { get; }
389
390     /// <summary>周囲の乾球温度[C]を取得する</summary>
391     double AmbientTemperature { get; }
392
393     /// <summary>周囲の絶対湿度[kg/kg]を取得する</summary>
394     double AmbientHumidityRatio { get; }
395
396     /// <summary>線熱通過率[W/(mK)]を取得する</summary>
397     double LinearThermalTransmittance { get; }
398
399     /// <summary>配管長[m]を取得する</summary>
400     double Length { get; }
401
402     /// <summary>水量[m3/s]を取得する</summary>
403     double VolumetricFlowRate { get; }
404
405     /// <summary>熱損失[W]を取得する</summary>
406     double HeatLoss { get; }
407
408     /// <summary>出口水温[C]を取得する</summary>
409     double OutletWaterTemperauture { get; }
410   }
411 }
```

1）ライセンス

1~15 行はプログラム本体ではなく、ライセンスである。本書に記載のプログラムコードは基本的に全て GPL: General Public License を適用する。即ち、読者を含めた全ての人に対して営利・非営利の目的を問わず、無償でその使用と再頒布が認められる。

2）名前空間

無数のクラスをすべて水平に並べてしまうと整理がつかない。そこで関連のあるクラスをまとめてグルーピングを行う方法が「名前空間」である。本例では 22 行で名前空間を宣言しており、中括弧 { }で囲まれた 23~411 行の範囲で定義された WaterPipe クラスと ImmutableWaterPipe インターフェースが Popolo.HVAC 名前空間に属することになる。以降、本書で提示するプログラムコードは、そのプログラムコードが属する名前空間とクラス名称をプログラムの右上に記す。ただし、例題の解答のプログラムはユーザーが自由な名前空間とクラスで記述すれば良いため記載を省くことにする。

別の名前空間のクラスを呼び出すためには、そのクラスが属する名前空間の名称に「.（ドット）」を付け、続けてクラス名称を記述する。例えば、湿り空気を計算する「MoistAir クラス」が「 Popolo.ThermophysicalProperty 」 名 前 空 間 に 属 し て い た と す る 。 こ の 場 合 に は「Popolo.ThermophysicalProperty.MoistAir」と記述すれば良い。この他に using キーワードを用いる方法もある。22 行では Popolo.ThermophysicalProperty 名前空間を参照しているため、以降では単純に名前空間を書かなくてもクラスの呼び出しができる。本例では、配管内外の対流熱伝達率を計算するために水と空気の熱物性が必要であるため、 using キーワードで「Popolo.ThermophysicalProperty」名前空間を参照し、水を表す Water クラスと湿り空気を表す MoistAir クラスが簡単に呼び出せるようにしている。

3）クラスの定義

25 行では class キーワードを用いて Pipe クラスを定義している（public キーワードと ImmutablePipe の意味に関しては後述する）。25~376 行の中括弧{ }で囲まれた範囲が Pipe クラスの中身である。クラスが持つメソッド（関数）や変数などはこの中に記述する。なお、クラスに属するメソッドや変数などを、そのクラスの「メンバ」と呼ぶ。

4）コメント

プログラムの実際の処理には関係のない開発者の覚書をプログラム中に書くことができる。1 つ目の方法はダブルスラッシュ「//」を用いる方法で、同じ行に属するダブルスラッシュ以降の文字は無視される。例えば 160 行ではこの方法を使うことで「線熱伝導率[W/(mK)]を初期化する」というプログラム処理の説明を行っている。2 つ目の方法は開始記号「/*」と終了記号「*/」を用いる方法で、これらの記号で挟まれた範囲はすべてがコメントとみなされる。例えば冒頭の 1~15 行では、この方法を用いてプログラムのライセンスについての記載をしている。3 つ目の方法は XML ドキュメントコメントと呼ばれるもので、トリプルスラッシュ「///」に続いて XML 形式で記載する。例えば 282~287 行では、メソッドの前に XML ドキュメントコメントを付し、メソッドと引数の概要を説明している。このような一定の形式でコメントを記述すると、C#のコンパイラを使用して XML 形式でコメントファイルを作成することができる。表 1.3 は書き出しを行った XML ドキュメントコメントの例である。C#では Sandcastle というソフトウェアが開発されており、この XML ドキュメントコメントを読み込ませて、自動的に API リファレンスを作成することが可能である。Java の場合には同様に javadoc と呼ばれるソフトウェアがある。図 1.5 に Sandcastle で自動生成した API リファレンスの例を示す。Web サイトでの HTML 形式のリファレンス作成にも対応しているため、自作のクラスライブラリを公開する場合などには便利である。

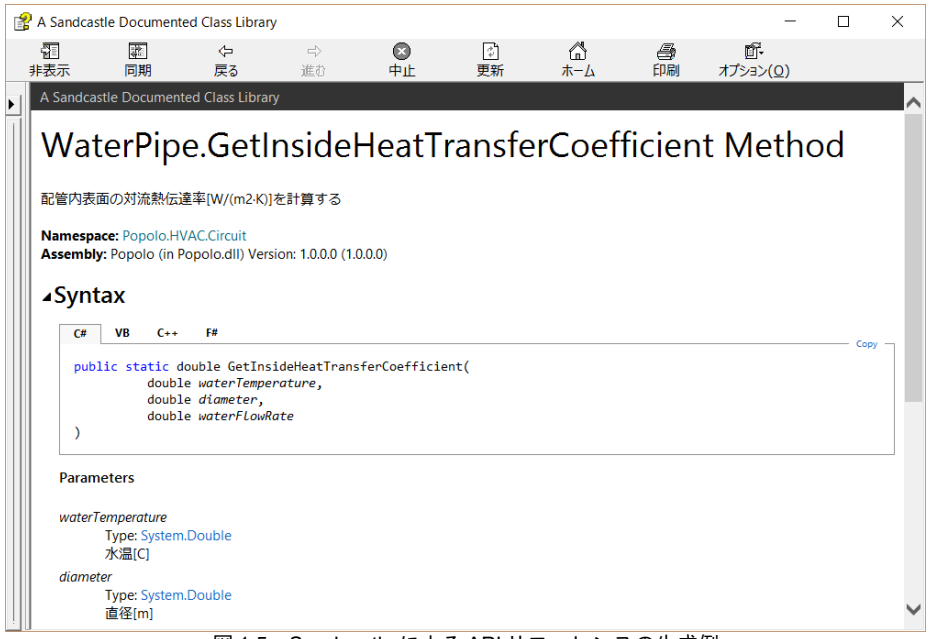


図 1.5　Sandcastle による API リファレンスの生成例

表 1.3　XML ドキュメントコメントの例

```
1 <member name="M:Popolo.HVAC.WaterPipe.GetInsideConvectiveHeatTransferCoefficient
2 (System.Double,System.Double,System.Double)">
3 <summary>配管内表面対流熱伝達率[W/(m2･K)]を計算する</summary>
4 <param name="waterTemperature">水温[C]</param>
5 <param name="diameter">直径[m]</param>
6 <param name="waterFlowRate">水量[kg/s]</param>
7 <returns>対流熱伝達率[W/(m2･K)]</returns>
8 <remarks>乱流かつ発達した流れの場合</remarks>
9 </member>
```

5）定数

通常、プログラムでは変数に値を代入しながら計算を重ねていくが、中には値を変更したくない変数もある[†1)]。このような場合には const キーワードを用いて定数とすることができる。例えば、29 行では大気圧を示す「ATMOSPHERIC_PRESSURE」という変数に海抜 0 m での大気圧である 101.325 kPa を割り当てている。const キーワードの効果により、以降、ATMOSPHERIC_PRESSURE に数値を代入しようとするとコンパイルエラーとなるため、誤って変数を上書きすることを回避することができる。const と同様の働きをするキーワードとして readonly がある。値が実行時に割り当てられ、const による定数と比較して計算速度は僅かに遅い。

この他に、定数の一種として列挙型を挙げることができる。列挙型はいくつかの代表的な選択肢がある場合、あるいは有限の選択肢の中から択一を迫りたい場合に有用である。42~53 行では列挙型を使用して配管の断熱材を定義している。断熱材無しを意味する「None」、ロックウールを意味する「RockWool」、グラウウールを意味する「GlassWool」、ポリスチレンを意味する「Polystyrene」についてそれぞれ整数値である 0, 1, 2, 3 を割り当てている。なお整数値は省略することもでき、その場合にはコンパイラが自動的に整数値を割り当てる。

本書で使用する主な定数を表 1.4 に示す。紙面の節約のため、以降のプログラム例では、これらの定数の初期化処理は記載を省略する。ただし、問題の性質に応じて kJ と J、あるいは kW と W を切り替えるため値については注意されたい。

表 1.4　本書で用いる主な定数

定数名称	意味	値
ATMOSPHERIC_PRESSURE	大気圧 [kPa]	101.325
AIR_DENSITY	湿り空気の密度 [kg/m^3]	1.2
AIR_SPECIFICHEAT	湿り空気の定圧比熱 [J/(kg･K)]	1,022
G_FORCES	重力加速度 [m^2/s]	9.8
WATER_DENSITY	水の密度 [kg/m^3]	1,000
WATER_SPECIFICHEAT	水の定圧比熱 [J/(kg･K)]	4,186
BLACK_CONSTANT	黒体の放射定数 [W/(m^2･K^4)]	5.67×10^{-8}
VAPORIZATION_LATENT_HEAT	0 ºC の水の蒸発潜熱 [kJ/kg]	2,500

6）変数

55~62 行では配管の線熱通過率 [W/(m･K)]、粗度 [m]、配管長 [m]を表す変数を定義している（private 修飾子に関しては後述）。変数には種類（「型」と呼ぶ）があり「double 型」は実数を表現できる変数である。整数を表現する場合には「int 型」、真偽を示す 2 値を表現する場合には「bool 型」、文字列を表現する場合には「string 型」を用いる[†2)]。

変数は 55~62 行のようにクラスの直下に定義してもよいし、183 行の質量流量 mw のようにメソッ

†1　例えば、大気圧である 101.325 kPa、水の比熱である 4.186 kJ/kg、黒体の放射定数である 5.67×10^{-8} W/(m^2･K^4)などである。しかし標高の高い場合には大気圧を可変としたいこともあるし、高温水を取り扱う場合には水の比熱を都度、計算したい。結局のところ、何を変数とし何を定数とするかは開発者の判断である。

†2　この他、float 型、byte 型、long 型など、多くの型があるが、とりあえずは上記の 4 種を使えればほとんどの問題に対応できる。

ドの中で定義することもできる。メソッドの内部で定義した変数は別のメソッドから参照できないが、クラスの直下に定義した場合には、クラスに属するすべてのメソッドから参照することができる。従って、プログラムの可読性を高めて堅牢にするためにはできるだけメソッド内部での変数定義とすべきである。なお、クラスの直下に定義した変数を「クラス変数」または「インスタンス変数」（両者の違いは後述）と呼ぶ。

変数には「値型」と「参照型」の2種類がある。メソッドの「引数」（後述）として変数を与えた場合の処理が異なるため、その違いを意識する必要がある。詳細は参考文献を参照してもらいたいが、当面は上記の4種の変数は「値型」、クラスのインスタンスは「参照型」と覚えておけば良い。

7）アクセス修飾子

変数やメソッドなどのメンバがクラス外から参照できるか否かを管理するためにはアクセス修飾子を用いる。例えば29行の定数宣言や56行のインスタンス変数の宣言では「private」という修飾子が付されており、この場合にはクラスの外部からこの変数を参照することはできない。逆に、32行と43行の列挙型の定義では「public」という修飾子が付されており、この場合にはクラスの外部からもこの列挙型を参照することができる。例えば配管の線熱伝導率の値はクラス内部での中間計算結果を保存するための変数であるが、これがクラスの外部から勝手に変更されると意図しない計算結果が算出される危険性がある。このような場合に private 修飾子を用いれば、外部からのメンバの操作を禁止することができる。オブジェクトの正常な動作に支障をきたすような、外部からの操作を防ぐことをカプセル化と呼び、オブジェクト指向プログラミングの主要な特徴である。

この他のアクセス修飾子として protected と internal がある。protected は、子クラスから親クラスのメンバの参照を許可する場合のアクセス修飾子である。internal は同じアセンブリ内のクラスからの参照を許可する場合のアクセス修飾子である。自作のライブラリを dll 形式などで公開する場合に、データ破損を防ぐために用いると便利である。

8）メソッド

111~376行ではメソッド（関数）を定義している。

冒頭にキーワードの static を付しているか否かで、インスタンスメソッドかクラスメソッドかに大きく分けられるが、static キーワードに関しては後述する。

アクセス修飾子 private、public の意味は前述のとおりである。public であればクラスの外からも使用可能なメソッドであり、private であればクラスの内部においてのみ使用可能なメソッドとなる。例えば、外部から水や空気の情報を受け取って状態を更新するメソッドである 174 行の UpdateHeatFlow メソッドは public としている一方で、必要に応じてクラス内部の初期化処理を行うメソッドである 158 行の initialize メソッドは private としている。

アクセス修飾子に続く「void」や「double」はメソッドの戻り値を意味している（後述するが 116 行と 131 行はコンストラクタと呼ばれる特別なメソッドであり、戻り値を持たない）。void であればメソッドを実行した場合の出力は無し、double であれば実数型の出力を返す。戻り値の次はメソッド名称であり、その後の両括弧( )の中身はメソッドが受け取る入力値（引数と呼ぶ）である。即ち $y=f(x_1, x_2)$と表される関数は下記のようなメソッドとして記述できる。

```
private double f(x1, x2){
  //処理内容
}
```

例えば 282~307 行で定義された GetInideHeatTransferCoefficient メソッドは double 型の変数である waterTemperature（水温）、diameter（直径）、waterFlowRate（水量）を引数として受け、double 型の出力（配管内表面の対流熱伝達率）を返すメソッドである。

引数に続く中括弧{ }の中にはメソッドの具体的な処理を記述する。

9）コンストラクタ

プログラム 1.2 はクラスを定義しただけであるため、実際に計算を行うにはパラメータに具体的な数値を代入してインスタンスを生成する必要がある。Pipe クラスのインスタンスを生成するプログラムは下記のとおりである。

```
WaterPipe pipe;
pipe = new WaterPipe (引数);
```

1 行目では Pipe という型の定義にもとづき、pipe という名称の変数（参照型）を用意している。2 行目では Pipe 型のインスタンスを生成し、1 行目で用意した pipe に代入している。もちろん、

```
WaterPipe pipe = new WaterPipe(引数);
```

と 1 行で表記してもよい。「new Pipe(引数)」がインスタンスの生成処理であるが、このようなインスタンス生成時に実行される特別なメソッドをコンストラクタと呼ぶ。本例では 116 行にコンストラクタの定義がある。クラス名称とメソッド名称が同じ場合にコンストラクタとみなされるが、戻り値を持たないこと以外は一般のメソッドと同様の文法である。116 行では配管長、内径、粗度、配管厚を引数として受けている。

131 行でコンストラクタを再び定義しているが、引数が 116 行のものとは異なる。C#ではこのように同じ名称で引数の異なるメソッドを定義することが可能であり、このような多重定義を「オーバーロード」と呼ぶ。オーバーロードされたメソッドが呼び出された場合には、与えられた引数に整合するメソッドが自動的に選択されて実行される。

具体的に 116 行と 131 行のコンストラクタの引数の違いを確認すると、前者は配管粗度と配管厚が実数として与えられていることに対し、後者では先に定義した列挙型の変数が与えられている。内部の処理を読むと、136~150 行で列挙型の値に応じてデフォルトの粗度と厚みが設定されていることがわかる。断熱材の製品仕様を考慮した詳細な計算を行いたい場合には 116 行のコンストラクタを用い、一般的な数値[†1]に基づいた概略検討を行いたい場合には 131 行のコンストラクタを用いるという使い分けを想定している。

10）プロパティ

クラスの変数を外部から参照・操作する方法の一つは、変数のアクセス修飾子を public とすることであるが、異常な値に設定されて意図しない計算が生じる危険性があることを考慮すると、あまり推奨されない。通常は 64~109 行に定義したようなプロパティを用いて変数操作を行う。82~87 行に示

†1　ここでは JISA9511 ポリスチレンフォーム、JISA9504 ロックウール・グラスウールを参照した。詳細に関しては第 17 章の回路網を参照。

すLengthプロパティが基本形である。配管長を示す変数であるlength変数を参照・操作するためにLengthという名称のプロパティを定義している。外部からは下記のように参照・操作する。

```
WaterPipe pipe = new WaterPipe(引数);
pipe.Length = 10;              //Length プロパティに 10 m を設定
double pl = pipe.Length;   //Length プロパティを取得して pl に代入
```

get{ }はLengthプロパティが参照された際の処理を記載しており、lengthの値をそのまま出力している。一方、set{ }はLengthプロパティへの代入操作が行われた場合の処理であり、入力値（value）が0未満の異常値の場合には0をlengthに設定している。プロパティを用いると、このように簡単な処理を挟むことができるため、クラス内部の変数の破壊を防ぐことができる。同様にgetアクセッサで処理を行うこともできる。

64~80行に示すようにgetアクセッサとsetアクセッサの具体的な処理を省略することもできる。また、各々に別々のアクセス修飾子を設定することもできる。結果として64~80行のプロパティは、クラス内部からは設定可能で、クラス外部からは読み取り専用（set不可）となる変数として扱うことができる。

11）静的メソッド

メソッドの中にはインスタンス変数の値に依存せず、引数のみで処理が完結するものもある。このようなメソッドを使うためにわざわざインスタンスを生成することは煩雑であるため、staticキーワードを用いてクラス名称から直接メソッドを呼び出せるようにする。staticキーワードの付されたメソッドをクラスメソッド（または静的メソッド、staticメソッド）と呼ぶ。逆にインスタンス変数に依存し、インスタンスを経由してしか呼び出せないメソッドをインスタンスメソッドと呼ぶ。具体的には111~280行で定義されたメソッドがインスタンスメソッド、282~375行で定義されたメソッドがクラスメソッドである。クラスメソッドの呼び出しは下記のように行う。

```
WaterPipe.GetInsideHeatTransferCoefficient(引数);
```

クラスメソッドはメソッドの引数のみに依存するため[†1)]、プログラムを読解すべき範囲が狭く、書籍での解説に向いている。またクラスメソッドであればFORTRAN 77のようなオブジェクトの概念を持たない言語との互換性も高くなるため、本書では、できるだけクラスメソッドによるプログラムの実装を中心に解説を行う。静的メソッドさえ作成できれば、インスタンスメソッドから適宜それを呼び出す構成とすることでクラスの定義も難しくない。実際にPipeクラスではそのような構成をとっており、汎用性の高い水および空気の対流熱伝達率計算処理と配管の熱通過率計算処理を静的メソッドで定義し、これらをインスタンスメソッドから使用している。

変数に関してもstaticキーワードを付して宣言することが可能であり、このような変数はクラス変数と呼ばれ、全てのインスタンスから参照可能な共有の変数となる。逆にstaticキーワードのない変数をインスタンス変数と呼び、各々のインスタンスごとに値を管理する変数となる。55~62行の変数はすべてインスタンス変数である[†2)]。

†1　実際にはクラス変数や定数にも依存するため正確ではない。

†2　本書では殆ど触れないが、近年、マルチコアCPUによる分散計算処理が注目を集めている。このような計算法をとる際には複数のCPUからの同時アクセスによってデータが壊れないスレッドセーフなプログラムとする必要がある。static変数が頻繁に書き換えられてすべてのインスタンスがその影響を受けるような構成にすると、スレッドセーフなプログラムとはなりづ

12）Immutable インターフェース

インスタンス自体を他のインスタンスに渡したい場合にはデータの破壊を防ぐ手立てが必要である。イメージしづらいため、以下に具体例を示す。

例えば、床を表す Floor クラスが定義されており、床暖房の計算を行うために、メンバとして上記の WaterPipe 型のインスタンス pipe を持っていたとする。温熱源の計算を行うために床暖房配管の戻り温度を知りたければ、Floor クラスのインスタンスである floor から、メンバである pipe を取得し、そのプロパティである OutletWaterTemperauture を参照すれば良いと思われるだろう。具体的には Floor クラスに下記のようにプロパティを定義する。

プログラム 1.3　プロパティによるインスタンスの受け渡し 1

```
1 public class Floor{
2     private WaterPipe pipe;
3     //中略
4     public WaterPipe HeatingPipe{
5         get{ return pipe;}
6     }
7     //中略
8 }
```

しかしこれは非常に危険であり、例えば第 3 者である Boiler クラスで下記のようなコードが記載されてしまうと Floor クラスが意図しない計算結果が出力される。

プログラム 1.4　意図しないインスタンスの書き換え

```
1 WaterPipe pipe = floor.HeatingPipe;
2 double oTemp = pipe.OutletWaterTemperauture;
3 pipe.Length = 10;
```

1 行目で Floor クラスの WaterPipe インスタンスを受け取り、2 行目でその出口水温を取得するまでは問題ない。しかし、3 行目では配管の長さを上書きしてしまっている。床暖房の配管の長さを熱源が書き換えるという処理であり、Floor クラスとしては正常な動作が保証できなくなる。このような問題を回避するために Immutable インターフェースを用いる方法がある[1.33)]。

インターフェースとはオブジェクト指向プログラミングの特徴である多態性を実現するための仕様である。378~410 行では ImmutablePipe インターフェースが定義されている。読めばわかるように、インターフェースには処理そのものは記載せず、プロパティやメソッドの名称のみを記述する。本例では各種のプロパティについて get アクセッサのみが定義されている。インターフェースは、オブジェクトをある側面から見た場合の見え方を記述しており、その意味では ImmutablePipe インターフェースは、WaterPipe オブジェクトを「読み取る機能のみ」という側面から見た場合の見えがかりを表現したものである。Upadate 関数や各種の set アクセッサなど、WaterPipe オブジェクトの中身を変更する機能は隠蔽されている。このインターフェースを具体的に実装するためには 25 行に示す通り、クラス定義の際に、コロン「:」に続けてインターフェースの名称を記載する。この実装により、すべての WaterPipe インスタンスは ImmutablePipe として扱うことも可能となる。そこでプログラム 1.3 の Floor クラスのプロパティを下記の通り変更する。

プログラム 1.5　プロパティによるインスタンスの受け渡し 2

```
1 public class Floor{
2     private WaterPipe pipe;
3     //中略
4     public ImmutablePipe HeatingPipe{
```

らい。多少のメモリ領域を犠牲にしてもインスタンス変数を多用した方が良いように思う。

```
5         get{ return pipe;}
6     }
7     //中略
8 }
```

Floor クラスが渡すインスタンスは、実体としては従前と同じ pipe であるが、4 行目を見ればわかるように、ImmutablePipe としてプロパティが定義されている。従って、プログラム 1.4 を実行しようとすると 3 行目でコンパイルエラーが生じる。ImmutablePipe は読み取り機能という側面から見た WaterPipe オブジェクトであり、オブジェクトに変更を加える処理である Length プロパティの set アクセッサは定義されていないからである。即ち、Immutable インターフェースにより、オブジェクトの破壊がコンパイラレベルで防止される。

本書では開発するクラスの多くに Immutable インターフェースを定義するが、紙面に限りがあるため、すべてのソースコードは記載しない。Immutable インターフェースを実装するクラスのプロパティと同じ名称の get アクセッサが定義されているものと捉えていただきたい[†1]。

13）まとめ

以上により、プログラム 1.2 を理解するために必要な個別の言語仕様については確認できた。最後に、プログラム 1.2 が全体としてどのように振る舞うのかを確認して本節を終える。

まず、Pipe クラスがインスタンス化される際には、111~155 行で定義されたコンストラクタが呼び出される。2 種類のコンストラクタが定義されているため、引数に応じて自動的に切り分けが行われる。両者の違いは粗度と配管厚の扱いであり、116 行のコンストラクタは具体的な粗度と配管厚の数値を受け取る一方で、131 行のコンストラクタは列挙型変数の値に応じてデフォルト値を設定する。両者とも最終的には initialize メソッドを呼び出して初期化を行う。配管や断熱材のサイズは空気と水に比較するとプログラム中に変化する可能性は低い。そこで、式 1.1 を式 1.2 と 1.3 に分け、initialize メソッドを用いて K_{Ls} までを計算してインスタンス変数として保存しておく。K_{Ls} は配管内表面から外表面までの熱通過率であり、配管と断熱材の物性に依存する項である。このような構成とすれば、水と空気の条件が変わる度に式 1.1 を全て再計算する時間を節約することができる。

$$K_{Ls}=\frac{2\pi}{\frac{1}{\lambda_p}\ln\frac{d_b}{d_i}+\frac{1}{\lambda_I}\ln\frac{d_o}{d_b}} \tag{1.2}$$

$$K_L=\frac{\pi}{\frac{1}{\alpha_{(c)w}d_i}+\frac{2\pi}{K_{Ls}}+\frac{1}{\alpha_{(c)a}d_o}} \tag{1.3}$$

initialize メソッドは外部から参照ができない private メソッドであり、配管仕様と断熱材仕様にもとづいて内表面から外表面までの線熱通過率を計算する。この際には GetPipeLinearThermalTransmittance メソッド（式 1.2 の実装）を使用する。計算結果はインスタンス変数である pipeLinearThermalTransmittance に格納される。インスタンス変数であるため、initialize メソッドを含む全てのメンバから参照が可能である。

水と空気の条件を変更する場合には UpdateHeatFlow メソッドを用いる。新たな水条件および空気条件にもとづいて、GetInsideHeatTransferCoefficient メソッドと GetOutsideHeatTransferCoefficient メ

†1 Immutable インターフェースのように、オブジェクト指向プログラミングを行う上で踏襲すべき典型的な技法が「デザインパターン」としてまとめられている[1.30) 1.31)]。中でも本節で紹介した Immutable インターフェースは、複雑な設備システムをモデル化する上で非常に有益なパターンであると筆者は捉えている。なお、Immutable とは「不変」を意味する形容詞である。

ソッドを用いて対流熱伝達率を計算する。なお、これら両メソッドは静的メソッドであり、その中身に関しては後章の水および湿り空気の物性計算で説明する。対流熱伝達率が計算できれば、予め保存していた pipeLinearThermalTransmittance と合わせて GetAirToWaterLinearThermalTransmittance メソッド（式 1.3 の実装）に代入し、水から空気までの熱通過率を計算する。計算結果は 64~109 行に定義されたプロパティを用いて取得することができる。

1.6　熱環境システムと制御

熱環境システムの概念を図 1.6 に示す。熱環境システムはいくつかのサブシステムによって構成され、これらのサブシステムは相互に熱交換を行う。この階層の深さは熱環境システムによって様々である。例えば図 1.6 は 4 層構造であるが、右から順に室、空調機、熱源、冷却塔を表していると仮定すれば理解しやすい。便宜的に左側をシステムの上流、右側をシステムの下流と呼ぶことにする。各サブシステムは設定値 T_{SS} を持ち、外乱や下流からの熱要求に影響を受けつつ、当該設定値を維持しようとする。上記の例に即して言えば、T_{SS3} は室温、T_{SS2} は空調機吹出温度、T_{SS1} は冷水温度、T_{SS0} は冷却水温度となろう[†1)]。このため、サブシステムは必要に応じて外部からガスや電力などのエネルギー供給を受けたり、上流から熱供給を受けたりする。また、各サブシステムは熱媒によって接続され、この熱媒を搬送するための機械に対してもエネルギー供給が必要となる。

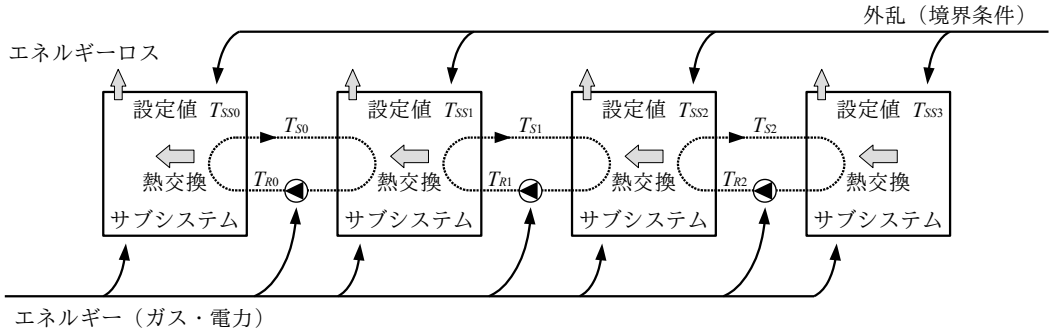


図 1.6　熱環境システムの概念

検討の目的に応じて図 1.6 を解く方法は 2 つに大別される。1 つは制御の検討を目的とするものであり、制御系のシミュレーションと呼ばれる。もう 1 つの目的はエネルギー消費予測であり、エネルギーシミュレーションと呼ばれることが多い。本書では主にエネルギーシミュレーションに即した計算法を解説する。

制御系のシミュレーションは、システムの状態を設定値に到達させるための方法を検討するものであり、熱媒流量制御のためのバルブやダンパの開度や機器の起動停止を判定する制御などの設定値が入力条件となる。一方、エネルギーシミュレーションは、基本的には状態値は設定値に一致するという前提で計算を行うものであり、状態値と設定値が一致するまでの過程は問題にならない。制御系シミュレーションではこの過程こそが主たる検討対象であるから、その様子を再現できるように、多くの機器は時間的な変動が表現できる動的なモデルとして表現する。エネルギーシミュレーションの場合には一部の要素（蓄熱槽や建築躯体など）を除き、静的なモデルを用いる。

エネルギーシミュレーションとして図 1.6 を解くためには、下流から順に上流に遡って計算を行う。前記の例に即して言えば、室温設定値である T_{SS3} を実現するために必要な熱交換量を計算し、上

†1　概念図であるため顕熱流のみを対象に乾球温度で表現したが、潜熱流に関しても同じである。

流からの熱媒温度 T_{S2} にこれを加える事で上流への熱媒温度 T_{R2} を出力する。上流からの熱媒温度 T_{S2} は上流への熱媒温度 T_{R2} に影響を受けるため、計算が循環するように見えるが、エネルギーシミュレーションであるため T_{R2} の如何によらず、T_{S2} は設定値である T_{SS2} に一致するとみなすのである。この計算を上流のサブシステムに向けて連続して行うことで各サブシステムが必要とするエネルギー消費が計算できる。

通常は上記の計算で良いが、計算結果が循環して反復計算が必要となる場合がある。サブシステムの出力が不足して過負荷となり、設定値を維持できない場合である。このような場合には過負荷となったサブシステムから下流のシステムに熱媒の温度を戻し、熱媒温度が設定値からどのように外れて均衡するのかを反復計算で特定しなければならない。

以上により、各サブシステムに求められる計算は基本的には 2 つであることがわかる。1 つは状態値が設定値に一致するという前提のもとにシステムの状態を特定する計算処理であり、もう 1 つは過負荷となった場合にシステムの状態を特定する計算処理である。後者の計算は俗に「成り行き状態の計算」と呼ばれる。制御系のシミュレーションでは制御のためのモデルを機器とは別に用意するため、通常は機器モデルとしては成り行きの計算処理のみを実装することが多い。制御結果を入力条件とするというモデルの構造はエネルギーシミュレーション特有のものであるという認識は大切である。機械分野などで単体の機械を計算対象とする研究者はこのようなモデル構造は取らないことが多いようで、議論がかみ合わないことが多々ある。

過負荷の場合には下流に対して実現した温度を戻して反復計算を行うと記したが、計算負荷を削減するために未処理負荷という概念を用いる方法もある。これは設定値と実現値とのずれにもとづいて算出した不足分の熱量を未処理負荷と定義し、これを積算して別に出力するというものである。そもそもエネルギーシミュレーションの主たる目的は期間的なエネルギーを予測することにあり、積算した未処理負荷を適当に割り戻すなどの方法で最終の積算エネルギー消費量に加える事ができればこの目的は達成される。過負荷状態において均衡する温度を正しく予測するということにあまり意味が無い場面においては、このような方法をとることも妥当である。例えば、省エネルギー法において用いられていた BECS/CEC/AC では、未処理負荷を用いることで過負荷状態もエネルギー消費に還元して評価を行っていた。過負荷状態で温度が設定値から乖離するという現象を解釈するためには熱的快適性という観点が必要であるが、エネルギーと熱的快適性という 2 つの軸での複合的な評価を行うことは簡単ではなく、画一的で公平性が強く求められる法制度にはなじまないためであろう。本書では第 29 章でこの問題に踏み込む。

【第 1 章 参考文献】

1.1）　超図解 C#ルールブック, エクスメディア, 株式会社 電通国際情報サービス, 2004
1.2）　宇田川光弘: パソコンによる空気調和計算法, オーム社, 1986
1.3）　William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling : Numerical Recipes in C, 株式会社技術評論社, 1993
1.4）　宿谷昌則: 数値計算で学ぶ光と熱の建築環境学, 丸善株式会社, 1993
1.5）　荒谷登, 鈴木憲三: 建築家のための熱環境解析入門, 北海道大学図書刊行会, 1993
1.6）　Braun, J.E.: Methodologies for Design and Control of Central Cooling Plants, Ph.D. Thesis, University of Wisconsin-Madison
1.7）　標準シミュレーションプログラム開発委員会 ：空調システム標準シミュレーションプログラム HASP/ACSS/8502 プログラム解説書, 1986, 日本建築設備士協会
1.8）　松尾陽, 猪岡達夫, 横山浩一 ：空調システムのエネルギーシミュレーションプログラム（第 1 報~第 3 報）, 空気調和・衛生工学会学術論文集, pp.425-pp.436 1985
1.9）　猪岡達夫 ：省エネルギー法に基づく CEC/AC のシミュレーションプログラム BECS, 日本建築学会環境工学委員会　熱環境小委員会第 29 回熱シンポジウム, pp.75-pp.84
1.10）　Fumio Sakurai, Tatsuo Inooka, et al. : Faces (Forecasts of air-conditioning system's energy, environmental, and economical performance by simulation), Proceedings: Building Simulation 2007 Beijing, pp.1661-pp.1668, 2007
1.11）　早川智, 小峯裕己, 猪岡達夫, 渡辺健一郎, 石黒邦道, 佐藤孝輔, 事務所ビル用エネルギー消費原単位管理ツール, 日本建築学会環境系論文集, 第 616 号, 91-98, 2007.06
1.12）　Joe Clarke : A review of ESP-r's flexible solution approach and its application to prospective technical domain developments , Advances in Building Energy Research, 2007.4
1.13）　松尾陽, 横山浩一, 石野久彌, 川元昭吾 ：空調設備の動的熱負荷計算入門, 日本建築設備士協会, 1980
1.14）　朱穎心, 楊靖, 中原信生 ：空気調和システムの動的シミュレーションとフォルト検知に関する研究その 1 改良 HVACSIM+を用いた VAV 制御システムのシミュレーション, 建築学会学術大会学術講演会講概集, pp.1581-1582, 1993.9
1.15）　Crawley Drury B. : Energy Plus ; creating a new generation building energy simulation program, Energy and Buildings, Vol.33, No.Issue 4, pp.319-331, 2001
1.16）　V. Bazjanac, T. Maile : IFC HVAC interface to energy plus – a case of expanded interoperability for energy simulation, SimBuild 2004, IBPSA-USA National Conference Boulder, CO, August 4-6, 2004
1.17）　A. Fiksel, J. W. Thornton, S. A. Klein, and W. A. Beckman : Developments to the TRNSYS Simulation Program, Journal of Solar Energy Engineering , pp.123~127, 1995
1.18）　時田繁, 松縄堅, 丹羽英治, 杉原善文, 岡崎徳臣 ：ライフサイクルエネルギーマネージメントのための空調システムシミュレーション開発 第 1 報, 空気調和・衛生工学会大会学術講演論文集, pp.195, 2005
1.19）　渡邊剛, 丹羽英治, 西谷義彦 ：建築設備のライフサイクルマネジメントにおけるシステムシミュレーションの活用に関する研究, 空気調和・衛生工学会論文集, No.128, pp.25~34, 2007.11
1.20）　Dustin Boswell, Trevor Foucher : The art of readable code, 訳: 角征典, 株式会社オライリー・ジャパン, 2012.6
1.21）　日本建築設備士協会空気調和衛生設備シミュレーション研究会 ：国内の空調装置シミュレーションプログラムの比較, 空気調和・衛生工学 第 58 巻, 第 7 号, pp.65-pp.78 (pp.683-pp.696)
1.22）　木村健一: 建築設備基礎, pp.7~10, 2009, 国際人間環境研究所
1.23）　窪田英樹, 林徹夫: 規格・標準化の検討課題（その 1）用語（量）記号について, 日本建築学会 環境工学委員会, 熱環境小委員会 第 30 回シンポジウム, pp.51-58
1.24）　林徹夫, 窪田英樹: 規格・標準化の検討課題（その 2）用語・単位・定数の現状, 日本建築学会 環境工学委員会, 熱環境小委員会 第 30 回シンポジウム, pp.59-64
1.25）　JIS A 0202 断熱用語, 日本規格協会, 2008
1.26）　空気調和・衛生用語辞典, 空気調和・衛生工学会, オーム社, 2006
1.27）　Jesse Liberty 著, 鈴木幸敏, 首藤一幸 訳: プログラミング C# 第 4 版, オライリー・ジャパン, 2006
1.28）　Jay Hilyard 著, 鈴木幸敏 訳: C#クックブック 第 3 版, オライリー・ジャパン, 2008
1.29）　ビル・ワグナー 著, 鈴木幸敏 訳: Effective C# 4.0, 翔泳社, 2011
1.30）　結城浩: Java 言語で学ぶデザインパターン入門, ソフトバンクパブリッシング, 2001
1.31）　Steven John Metsker: Design Patterns in C#, Addison-wesley, 2004
1.32）　西川兼康, 藤田恭伸 ：伝熱学, 第 3 章 p.22, 理工学社, 1982
1.33）　富樫英介, 田辺新一: Immutable interface を利用した汎用建築熱負荷計算クラスライブラリの設計法, 空気調和・衛生工学会 学術講演会論文集, pp.1995-1998, 2009
1.34）　空気調和・衛生工学会 空気調和設備委員会 熱負荷算法小委員会シンポジウム(1989) 資料, 3. 空調ソフトウェア・コンパス, p.26

第2章　数値計算
(Numerical Computing)

2.1　概要

プログラム作成の基礎となる数式の中には、手計算による加減乗除で代数的に解を算出できない問題がある。この場合にはコンピュータによる自動計算が有効であり、膨大な反復計算によって近似的に解にたどり着く手法が提案されている。このような手法を総称して数値計算と呼ぶ。数値計算はそれ自体が一つの学問分野であり、数ページの記述で内容を網羅することはできない。本章では他の章で用いることになる基本的な数値計算法について解説を行う。特に 1 変数の非線形方程式の解法と行列演算は工学の基礎であり、その応用範囲は建築熱環境分野にとどまらない。理解し、習得すべき技術である。

どのような問題にも適用できる万能な数値計算法というものはなく、計算対象とする問題の性質や必要とする精度などを踏まえてソフトウェア開発者が主体的に選択すべきものである。様々な数値計算手法がどのような方法によって解を求めようとしているのかについて理解することは、そもそもどのように問題をたてるべきかという発想にも良い影響を与える。

一方、数値計算ソフトウェアに関しては、ソースコードが公開されたライブラリも多い[2.17) 2.18)]。このようなライブラリと自分のプログラムをリンクさせて計算を行うということも一つの手段であるが、さらに手軽な方法として MATLAB や Octave などの汎用の数値計算ソフトウェアを用いるという手段もある。近年では多くの海外の研究者はこのようなソフトウェアを用いて数値計算という作業を外出ししているようである。しかし、数値計算が収束して求根できた時の喜びはシミュレーションを行う一つの醍醐味である。是非体験することを推奨する。

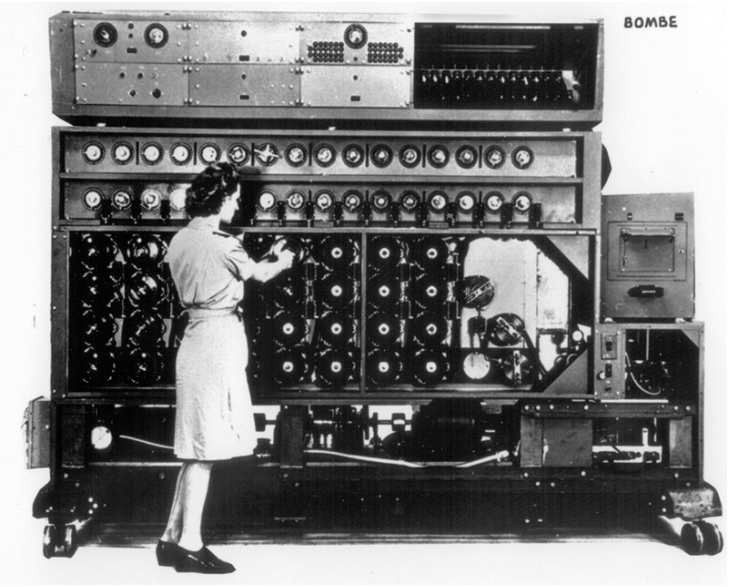


写真 2.1　史上初の計算機 Turing Bombe[2.19)]

2.2　連立一次方程式の解法

2.2.1　行列とベクトルの表現

連立一次方程式は、一般的に式 2.1 で表現でき、行列で表現すると式 2.2 となる。ただし、行列[A]およびベクトル[x], [b]は各々式 2.3 である。

$$\begin{aligned} a_{11}x_1+a_{12}x_2+a_{13}x_3+\cdots+a_{1N}x_N &= b_1 \\ a_{21}x_1+a_{22}x_2+a_{23}x_3+\cdots+a_{2N}x_N &= b_2 \\ a_{31}x_1+a_{32}x_2+a_{33}x_3+\cdots+a_{3N}x_N &= b_3 \\ &\vdots \\ a_{M1}x_1+a_{M2}x_2+a_{M3}x_3+\cdots+a_{MN}x_N &= b_M \end{aligned} \tag{2.1}$$

$$[A][x]=[b] \tag{2.2}$$

$$[A]=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_2 & \cdots & a_{2N} \\ & & \vdots & \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{bmatrix}, \quad [x]=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}, \quad [b]=\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix} \tag{2.3}$$

連立一次方程式を扱う際には上記のような行列表現を行うことが多いため、具体的な解法に入る前に、行列およびベクトルを表わすクラスを定義しておく。もちろん、通常のプログラム言語であれば多段階配列や多次元配列が用意されているため、これらを単純に使うという方法もある。しかし、一口に行列と言っても密行列、疎行列、帯行列、三角行列、など、様々な種類があり、それぞれの行列の特徴に応じた解法が存在する。従って、単純な多段階配列ではなく、それぞれを別個のクラスとして表現しておけば、その特徴に応じた処理が可能となり、便利である。

1）行列およびベクトルのインターフェース

行列およびベクトルのインターフェースである IMatrix および IVector の定義をプログラム 2.1 に示す。以降、一般行列（ベクトル）クラス、部分行列（ベクトル）クラス、疎行列クラスを定義するが、このインターフェースを実装することで等しく取り扱うことが可能になる。

プログラム 2.1　行列およびベクトルインターフェースの定義

Popolo.Numerics.MatrixOperation.IMatrix, IVector interface

```
/// <summary>行列インターフェース</summary>
public interface IMatrix
{
  /// <summary>要素の値を設定・取得する</summary>
  /// <param name="row">行番号</param>
  /// <param name="column">列番号</param>
  /// <returns>要素の値値</returns>
  new double this[int row, int column] { get; set; }

  /// <summary>初期化する</summary>
  /// <param name="val">初期化する値</param>
  void Initialize(double val);
}

/// <summary>ベクトルインターフェース</summary>
public interface IVector
{
  /// <summary>要素の値を設定・取得する</summary>
  /// <param name="index">要素番号</param>
  /// <returns>要素の値</returns>
  new double this[int index] { get; set; }

  /// <summary>初期化する</summary>
  /// <param name="val">初期化する値</param>
  void Initialize(double val);
}
```

2）一般の行列・ベクトルクラス

一般の行列を表わすクラスである Matrix の定義をプログラム 2.2 に示す。2 次元配列を用いて行列要素の取得設定処理を行う。ベクトルを表わすクラスである Vector の定義は省略するが処理はほぼ同様である。

プログラム 2.2　一般の行列クラスの定義

Popolo.Numerics.MatrixOperation.Matrix class

```
/// <summary>行列</summary>
public class Matrix : IMatrix
{
  /// <summary>行列</summary>
  private double[][] matrix;

  /// <summary>行数を取得する</summary>
  public int Rows { get; private set; }

  /// <summary>列数を取得する</summary>
  public int Columns { get; private set; }

  /// <summary>要素の値を設定・取得する</summary>
  /// <param name="row">行番号</param>
  /// <param name="column">列番号</param>
  /// <returns>要素の値値</returns>
  public double this[int row, int column]
  {
    get { return matrix[row][column]; }
    set { matrix[row][column] = value; }
  }

  /// <summary>コンストラクタ</summary>
  /// <param name="rowSize">行数</param>
  /// <param name="columnSize">列数</param>
  public Matrix(int rowSize, int columnSize)
  {
    Rows = rowSize;
    Columns = columnSize;
    matrix = new double[Rows][];
    for (int i = 0; i < Rows; i++) matrix[i] = new double[Columns];
  }

  /// <summary>初期化する</summary>
  /// <param name="val">初期化する値</param>
  public void Initialize(double val)
  {
    for (int i = 0; i < Rows; i++)
      for (int j = 0; j < Columns; j++)
        matrix[i][j] = val;
  }
}
```

3）部分行列クラス

問題によっては、行列の一部のみを集中的に操作するため、その他の要素には関心が無い場合がある。このような場合には図 2.1 に示すように、操作対象の範囲を部分行列として新たな行列とみなすと操作が楽になる。

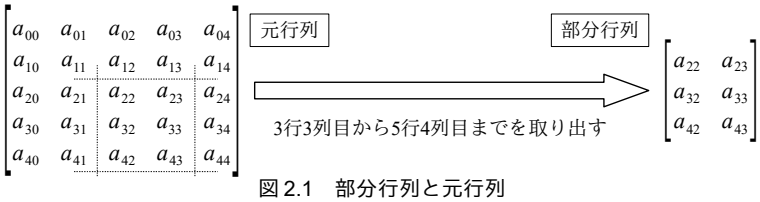


図 2.1　部分行列と元行列

部分行列を表わすクラスである MatrixView の定義をプログラム 2.3 に示す。元行列のインスタンス（5 行）と部分行列開始位置（7~11 行）を保持する。要素へのアクセスがあった場合には、部分行列開始位置に合わせてシフトさせた元行列インスタンスの要素に接続させる（19~35 行）。このようにすることで部分行列クラスを用いる主体は、自分が元行列からみてどの位置の要素を操作しているの

かを気にせずに計算を進めることができるようになる。37~50行はコンストラクタであり、元行列のインスタンス、部分行列開始位置、行列数を引数とする。元行列を表わすために IMatrix インターフェースを用いる点は重要であり、このような仕組みとすれば、 MatrixView のインスタンスを MatrixView のコンストラクタの引数にすることが可能となる。即ち、入れ子状の部分行列の定義が可能になる。部分ベクトルを表わすクラスである VectorView の定義は省略するが処理は同様である。

プログラム 2.3　部分行列クラスの定義

Popolo.Numerics.MatrixOperation.MatrixView class

```
 1 /// <summary>部分行列</summary>
 2 public class MatrixView : IMatrix
 3 {
 4   /// <summary>もとの行列</summary>
 5   private IMatrix matrix;
 6
 7   /// <summary>部分行列開始行番号</summary>
 8   private int rowStartNumber;
 9
10   /// <summary>部分行列開始列番号</summary>
11   private int columnStartNumber;
12
13   /// <summary>行数を取得する</summary>
14   public int Rows { get; private set; }
15
16   /// <summary>列数を取得する</summary>
17   public int Columns { get; private set; }
18
19   /// <summary>要素の値を設定・取得する</summary>
20   /// <param name="row">行番号</param>
21   /// <param name="column">列番号</param>
22   /// <returns>要素の値値</returns>
23   public double this[int row, int column]
24   {
25     get
26     {
27       if (Rows < row + 1 || Columns < column + 1) throw new Exception("Matrix out of range");
28       return matrix[row + rowStartNumber, column + columnStartNumber];
29     }
30     set
31     {
32       if (Rows < row + 1 || Columns < column + 1) throw new Exception("Matrix out of range");
33       matrix[row + rowStartNumber, column + columnStartNumber] = value;
34     }
35   }
36
37   /// <summary>インスタンスを初期化する</summary>
38   /// <param name="matrix">もとの行列</param>
39   /// <param name="rowSize">行数</param>
40   /// <param name="columnSize">列数</param>
41   /// <param name="rowStartNumber">部分行列開始行番号</param>
42   /// <param name="columnStartNumber">部分行列開始列番号</param>
43   public MatrixView(IMatrix matrix, int rowSize, int columnSize, int rowStartNumber, int columnStartNumber)
44   {
45     this.matrix = matrix;
46     this.rowStartNumber = rowStartNumber;
47     this.columnStartNumber = columnStartNumber;
48     this.Rows = rowSize;
49     this.Columns = columnSize;
50   }
51
52   /// <summary>初期化する</summary>
53   /// <param name="val">初期化する値</param>
54   public void Initialize(double val)
55   {
56     for (int i = 0; i < Rows; i++)
57       for (int j = 0; j < Columns; j++)
58         matrix[i + rowStartNumber, j + columnStartNumber] = val;
59   }
60 }
```

4）疎行列クラス

詳細は後章において述べるが、モデルを集中定数系で表現して連立方程式をたてると、行列要素の殆どが0となる場合が多い。このような要素の殆どが0である行列を疎行列（スパースマトリクス）

と呼ぶ[†1]。疎行列の場合には 0 を含むすべての要素について記憶領域を確保せず、0 以外の有効な要素の位置（行列番号）とその値のみを記憶しておけば良い。

疎行列を表わすクラスの定義をプログラム 2.4 に示す。2 次元配列ではなく、5 行に示すように連想配列で非 0 となる要素の値を保存する。13~27 行が行列要素の取得処理である。23, 24 行で連想配列の検索を行い、キーがあれば値を返し、要素が存在しなければ 0 を返す（24 行）。28~35 行は行列要素への値代入処理である。0 が設定される場合には連想配列から要素を削除する。38~47 行はコンストラクタであり、行数分の空の連想配列を生成する。59~70 行はベクトルとの積の計算処理である。行列要素の全てを計算せず、非 0 要素のみを取り出すことで計算を高速化する。

プログラム 2.4　疎行列クラスの定義

Popolo.Numerics.MatrixOperation.SparseMatrix class

```
 1 /// <summary>疎行列</summary>
 2 public class SparseMatrix : IMatrix
 3 {
 4   /// <summary>非 0 要素を格納する連想配列</summary>
 5   private Dictionary<int, double>[] elem;
 6
 7   /// <summary>行数を取得する</summary>
 8   public int Rows { get; private set; }
 9
10   /// <summary>列数を取得する</summary>
11   public int Columns { get; private set; }
12
13   /// <summary>要素の値を設定・取得する</summary>
14   /// <param name="row">行番号</param>
15   /// <param name="column">列番号</param>
16   /// <returns>要素の値値</returns>
17   public double this[int row, int column]
18   {
19     get
20     {
21       if (row < Rows)
22       {
23         if (column < Columns && elem[row].ContainsKey(column)) return elem[row][column];
24         else return 0;
25       }
26       else return 0;
27     }
28     set
29     {
30       if (row < Rows && column < Columns)
31       {
32         if (value == 0.0) elem[row].Remove(column);
33         else elem[row][column] = value;
34       }
35     }
36   }
37
38   /// <summary>インスタンスを初期化する</summary>
39   /// <param name="rows">行数</param>
40   /// <param name="columns">列数</param>
41   public SparseMatrix(int rows, int columns)
42   {
43     Rows = rows;
44     Columns = columns;
45     elem = new Dictionary<int, double>[Rows];
46     for (int i = 0; i < Rows; i++) elem[i] = new Dictionary<int, double>();
47   }
48
49   /// <summary>初期化する</summary>
50   /// <param name="val">初期化する値</param>
51   /// <remarks>疎行列で 0 以外での初期化は無意味</remarks>
52   public void Initialize(double val)
53   {
54     if (val == 0.0)
```

†1　「殆ど」とは随分定性的な表現であるが、定量的な閾値を与えることは難しい。後述するが、そもそも連立方程式を解く際に、比較的 0 が多い行列の場合には高速に解く方法が存在するということが疎行列を一般行列と分けて扱う理由である。しかし、0 の数がいくつ以上ならば有利ということが確定的に言えるものではなく、行列の性質に依存するため、疎行列の定義もまた定性的に 0 が多いという表現にならざるを得ない。

```
55       for (int i = 0; i < Rows; i++) elem[i].Clear();
56     else throw new Exception("Not Implemented");
57   }
58
59   /// <summary>入力ベクトルとの積を計算して出力ベクトルに格納する</summary>
60   /// <param name="vec1">入力ベクトル</param>
61   /// <param name="vec2">出力：出力ベクトル</param>
62   public void Multiplicate(IVector vec1, ref IVector vec2)
63   {
64     vec2.Initialize(0);
65     for (int i = 0; i < Rows; i++)
66     {
67       Dictionary<int, double> row = elem[i];
68       foreach (int col in row.Keys) vec2[i] += row[col] * vec1[col];
69     }
70   }
71 }
```

2.2.2　直接法と反復法

連立一次方程式の解法は直接法と反復法に大きく分けられる。直接法は有限の計算回数で解にたどり着くことが保証されており、行列の性質に関わりなく安定的に解を得ることができる。反復法は適当な初期値から出発して逐次計算により解に漸近していく手法である。直接法とは異なり、行列の性質によっては収束に非常に時間がかかるようなこともあるが、非 0 要素についてのみ反復式を適用すれば良いため、0 要素が多い疎行列の場合には直接法よりも遥かに短い時間で解が得られる。本書では直接法の例として LU 分解と Thomas Algorithm を、反復法の例として双共役勾配法を解説する[†1)]。

1）ベクトル・行列演算

連立一次方程式の計算に直接は用いないが、計算結果を利用する際にベクトルと行列の演算が必要になることが多い。行列の演算は基礎的な処理であるため、プログラムの中での呼び出し回数が非常に大きくなりがちである。従ってこれらのプログラムが高速に動作できるか否かは、全体プログラムの速度に大きな影響を与える。このために、コンピュータが配列をどのようにメモリ上で管理しているかを考慮して繰り返し処理の方法を調整したり、1 回の計算ループあたりの処理を増やしてループに必要な計算時間を削減する方法（ループ・アンローリング）などが考案されている。本書の処理は可読性を上げるためにこのような工夫は行っていないため、さらに計算速度を向上したい場合には、文献等を参照の上、プログラムを修正されたい。

プログラム 2.5 にベクトル・行列演算を示す。1~18 行は行列同士の積である。12 行で A 行列の(i,j)要素の値を変数 smA に代入しており、これにより 14 行のループでは A 行列への繰り返しアクセスが不要となる。細かな工夫であるがこれだけでも計算速度が約 20 %向上する。

19~33 行と 35~45 行はそれぞれ、行列とベクトルの積和、行列同士の和の計算処理である。

プログラム 2.5　ベクトル・行列演算プログラム

Popolo.Numerics.MatrixOperation.LinearAlgebra class

```
 1 /// <summary>行列の積(C=AB)を計算する</summary>
 2 /// <param name="mA">A 行列</param>
 3 /// <param name="mB">B 行列</param>
 4 /// <param name="mC">C 行列</param>
 5 public static void Multiplicate(IMatrix mA, IMatrix mB, ref IMatrix mC)
 6 {
 7   mC.Initialize(0);
 8   for (int i = 0; i < mA.Rows; i++)
 9   {
10     for (int j = 0; j < mA.Columns; j++)
11     {
12       double smA = mA[i, j];
```

†1　直接法の主要な解法としては本節で取り上げる LU 分解法の他に QR 法が挙げられる。計算速度は LU 分解法の方が早いが、計算の規模が非常に大きい場合における数値計算の安定性に関しては QR 分解が優れる。双共役勾配法は反復計算により近似解を更新する手法であるが、一方で、有限回の計算で収束することが証明されているため、直接法と反復法のいずれにも属さないとみなすこともある。

```
      if (smA != 0)
        for (int k = 0; k < mB.Columns; k++) mC[i, k] += smA * mB[j, k];
    }
  }
}

/// <summary>行列とベクトルの積和を計算する (vC = α mA vB + β vC) </summary>
/// <param name="mA">行列 A</param>
/// <param name="vB">ベクトル B</param>
/// <param name="vC">ベクトル C (解が上書きされる) </param>
/// <param name="alpha">第一項の係数</param>
/// <param name="beta">第二項の係数</param>
public static void Multiplicate(IMatrix mA, IVector vB, ref IVector vC, double alpha, double beta)
{
  for (int i = 0; i < mA.Rows; i++)
  {
    double buff = 0;
    for (int j = 0; j < mA.Columns; j++) buff += mA[i, j] * vB[j];
    vC[i] = vC[i] * beta + alpha * buff;
  }
}

/// <summary>行列の和を計算する (mB = cA*mA + cB*mB) </summary>
/// <param name="mA">A 行列</param>
/// <param name="mB">B 行列 (解が上書きされる) </param>
/// <param name="cA">第一項の係数</param>
/// <param name="cB">第二項の係数</param>
public static void Add(IMatrix mA, ref IMatrix mB, double cA, double cB)
{
  for (int i = 0; i < mA.Rows; i++)
    for (int j = 0; j < mA.Columns; j++)
      mB[i, j] = mA[i, j] * cA + mB[i, j] * cB;
}
```

2）LU 分解

行列[*A*]を式 2.4 および式 2.5 で示される二つの行列[*L*]と[*U*]の積に分解することを考える。ここで、行列[*L*]は対角要素とその下側のみが 0 ではない行列で、下三角行列と呼ぶ。同様に、行列[*U*]は対角要素とその上側のみが 0 ではない行列で、上三角行列と呼ぶ。下三角行列と上三角行列を総称して三角行列と呼ぶ。

$$[L][U]=[A] \tag{2.4}$$

$$[L]=\begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ l_{N1} & \cdots & \cdots & l_{N,N-1} & l_{NN} \end{bmatrix} \quad [U]=\begin{bmatrix} u_{11} & u_{12} & \cdots & \cdots & u_{1N} \\ 0 & u_{22} & \ddots & & \vdots \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & u_{N-1,N} \\ 0 & 0 & \cdots & 0 & u_{NN} \end{bmatrix} \tag{2.5}$$

上記の分解を行うと、式 2.2 は式 2.6 と式 2.7 の 2 段階の計算で表現できることがわかる。

$$[L][y]=[b] \tag{2.6}$$

$$[U][x]=[y] \tag{2.7}$$

三角行列の場合に、式 2.6 や式 2.7 で示される連立方程式を解くことは簡単であり、式 2.6 は式 2.8 に示す前進代入処理で解け、式 2.7 は式 2.9 に示す後退代入処理で解ける。一旦、行列[*L*]と[*U*]を求めてしまえば係数行列[*A*]が変化しないかぎり、式 2.8 と式 2.9 を適用して高速に問題を解くことができる。これが、行列[*A*]を三角行列に分解する目的である。

$$y_i=\begin{cases} \dfrac{b_i}{l_{ii}} & (i=1) \\ \dfrac{1}{l_{ii}}\left[b_i-\displaystyle\sum_{j=1}^{i-1} l_{ij}y_j\right] & (i=2,3,\cdots,N) \end{cases} \tag{2.8}$$

$$x_i = \begin{cases} x_i = \dfrac{y_i}{u_{ii}} & (i = N) \\ \dfrac{1}{u_{ii}}\left[y_i - \sum_{j=i+1}^{N} u_{ij} y_j \right] & (i = N-1, N-2, \cdots, 1) \end{cases} \tag{2.9}$$

以下に LU 分解の一手法である Crout 法を示す。

1）$l_{ii} = 1$ （$i = 1, 2, 3, \ldots, N$）とする。

2）$j = 1, 2, 3, \ldots, N$ について下記の処理を行う。

2-1）$i = 1, 2, 3, \ldots, j$ について、式 2.10 に従い、u_{ij} を計算する。

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \tag{2.10}$$

2-2）$i = j+1, j+2, \ldots, N$ について、式 2.11 に従い、l_{ij} を計算する。

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right) \tag{2.11}$$

ここで、式 2.11 右辺には u_{jj} の逆数があるが、u_{jj} が 0 に近い値をとった場合には l_{ij} の値が極端に大きくなるため、精度が悪化する。計算精度に大きな影響を与えるカナメの要素であるということにちなみ、対角要素 u_{jj} を pivot（軸）と呼ぶ。ところで、式 2.1~式 2.3 を見ればわかるように、行列[A]、ベクトル[b]、ベクトル[x]のそれぞれの行を同じように入れ替えたとしても計算結果に影響は無い。そこで、計算精度の向上を目的に、pivot が大きな値となるように行を入れ替えるという工夫を行う。これを pivot 選択と呼ぶ。

pivot は大きな値であるほど良いが、同一の列に所属する各要素の絶対値の大小関係の比較のみでは不十分である。式 2.1 に示す各一次方程式の右辺と左辺の両方にどれだけ大きい数字を乗じても、連立方程式の解は変化せず、絶対値としてはいくらでも大きな数字を取りうるからである。そこで、「それぞれの一次方程式の係数群の中における、当該係数の相対的な大きさ」を pivot の大きさの評価尺度とする。具体的には、第 i 行の係数 a_{ij} を a_{i1}~a_{iN} の中の最大値で除し、各行の係数値を-1~1 の範囲に正規化した上で、大小関係を比較する。これを陰的 pivot 選択と呼ぶ。

LU 分解および前進/後退代入計算を行うプログラムを 2.6 に示す。

プログラム 2.6　LU 分解および前進/後退代入計算プログラム

Popolo.Numerics.MatrixOperation.LinearAlgebra class

```
/// <summary>Crout 法により LU 分解 (A=LU) を行う</summary>
/// <param name="matrix">
/// 入力：LU 分解を行う正方行列
/// 出力：上三角+対角成分-U 行列、下三角成分-L 行列</param>
/// <param name="perm">pivot 選択による行置換ベクトル</param>
/// <param name="wArray">作業用記憶領域（行数）</param>
/// <remarks>Newmerical Recipies より移植</remarks>
public static void LUDecompose(ref IMatrix matrix, ref int[] perm, ref IVector wArray)
{
  //行列の行数・列数を取得
  int num = matrix.Rows;

  for (int i = 0; i < num; i++)
  {
    double big = 0.0;
    for (int j = 0; j < num; j++)
      big = Math.Max(big, Math.Abs(matrix[i, j]));
    if (big == 0.0) throw new Exception("Singular matrix error");
    wArray[i] = 1.0 / big;
```

```
  }

  for (int j = 0; j < num; j++)
  {
    double sum = 0;
    double big = 0.0d;
    int imax = 0;

    //Crout法を適用
    //i~jまでの繰り返し計算
    for (int i = 0; i < j; i++)
    {
      sum = -matrix[i, j];
      for (int k = 0; k < i; k++) sum += matrix[i, k] * matrix[k, j];
      matrix[i, j] = -sum;
    }

    //j~Nまでの繰り返し計算
    for (int i = j; i < num; i++)
    {
      sum = -matrix[i, j];
      for (int k = 0; k < j; k++) sum += matrix[i, k] * matrix[k, j];
      matrix[i, j] = -sum;

      //スケーリングを考慮して最大のpivot選択を行う
      double dum = wArray[i] * Math.Abs(sum);
      if (big <= dum)
      {
        big = dum;
        imax = i;
      }
    }

    //行交換の必要判定
    if (j != imax)
    {
      //行の交換
      for (int k = 0; k < num; k++)
      {
        double dum = matrix[imax, k];
        matrix[imax, k] = matrix[j, k];
        matrix[j, k] = dum;
      }
      wArray[imax] = wArray[j];   //スケーリング係数の交換
    }
    //行の置換を記憶
    perm[j] = imax;

    if (matrix[j, j] == 0.0) matrix[j, j] = double.MinValue;

    if (j != num)
    {
      double bf = 1d / matrix[j, j];
      for (int i = j + 1; i < num; i++) matrix[i, j] *= bf;
    }
  }
}

/// <summary>LU行列にもとづき前進・後退代入処理を行う</summary>
/// <param name="luMatrix">LU分解済の行列</param>
/// <param name="perm">置換ベクトル</param>
/// <param name="b">bベクトル：解が上書きされる</param>
public static void FAndBSubstitute(IMatrix luMatrix, int[] perm, ref IVector b)
{
  //行列の行数・列数を取得
  int num = luMatrix.Rows;

  //ベクトルbが0以外の数値をとる位置
  int ii = 0;

  for (int i = 0; i < num; i++)
  {
    //置換ベクトルに従ってbベクトルを入替え
    int ip = perm[i];
    double sum = b[ip];
    b[ip] = b[i];

    //前進代入処理
    if (ii != 0)
      for (int j = ii - 1; j < i; j++) sum -= luMatrix[i, j] * b[j];
```

```
100     else
101       if (sum != 0) ii = i + 1;
102     b[i] = sum;
103   }
104   //後退代入処理
105   for (int i = num - 1; 0 <= i; i--)
106   {
107     double sum = b[i];
108     for (int j = i + 1; j < num; j++) sum -= luMatrix[i, j] * b[j];
109     b[i] = sum / luMatrix[i, i];
110   }
111 }
112
113 /// <summary>mA の逆行列を計算する</summary>
114 /// <param name="mA">逆行列を求める行列</param>
115 /// <param name="mB">出力:逆行列</param>
116 public static void GetInverse(ref IMatrix mA, ref IMatrix mB)
117 {
118   if (mA.Columns == 1)
119   {
120     mB[0, 0] = 1d / mA[0, 0];
121     return;
122   }
123
124   int[] wA1 = new int[mA.Rows];
125   IVector wA2 = new Vector(mA.Rows);
126   LUDecompose(ref mA, ref wA1, ref wA2);
127   for (int i = 0; i < mA.Rows; i++)
128   {
129     for (int j = 0; j < wA2.Length; j++) wA2[j] = 0;
130     wA2[i] = 1;
131     FAndBSubstitute(mA, wA1, ref wA2);
132     for (int j = 0; j < wA2.Length; j++) mB[j, i] = wA2[j];
133   }
134 }
```

【例題 2.1】

下記の多層壁の内部の温度分布を把握するために、各層の境界温度を行列表記せよ。また、 LU 分解を利用して実際に温度分布を求めよ。ただし、外気温度 t_o は 35 ℃、室内温度 t_i は 26 ℃、内表面総合熱伝達率 α_i と外表面総合熱伝達率 α_o はそれぞれ 9 W/(m²·K)と 20 W/(m²·K)とする。また、各材料の並び順、厚み l [mm]、熱伝導率 λ [W/(m·K)]は下記の通りとする（中空層については熱抵抗 R_a [(m²·K)/W]）。

1：コンクリート　l_1：180 mm　　λ_1：1.6 W/(m·K)
2：断熱材　　l_2：25 mm　　λ_2：0.034 W/(m·K)
3：非密閉中空層　l_3：　-　　R_a：0.07 (m²·K) / W
4：石膏ボード　l_4：12 mm　　λ_4：0.22 W/(m·K)

【解】

壁層の間に計算点を取り、境界の温度を t_0~t_4 とする。ただし、t_0は屋外側表面温度、t_4は屋内側表面温度である。各計算点での熱収支式は下記の通りである。

$\alpha_o (t_o - t_0) - \lambda_1 / l_1 (t_0 - t_1) = 0$　　（外表面）
$\lambda_1 / l_1 (t_0 - t_1) - \lambda_2 / l_2 (t_1 - t_2) = 0$　　（コンクリート・断熱材の境界）
$\lambda_2 / l_2 (t_1 - t_2) - (t_2 - t_3) / R_a = 0$　　（断熱材・中空層の境界）
$(t_2 - t_3) / R_a - \lambda_4 / l_4 (t_3 - t_4) = 0$　　（中空層・石膏ボードの境界）
$\lambda_4 / l_4 (t_3 - t_4) - \alpha_i (t_4 - t_i) = 0$　　（内表面）

上記を整理すると、

$$
\begin{aligned}
-\left(\frac{\lambda_1}{l_1}+\alpha_o\right)t_0 + \frac{\lambda_1}{l_1}t_1 &= -\alpha_o t_o \\
\frac{\lambda_1}{l_1}t_0 - \left(\frac{\lambda_1}{l_1}+\frac{\lambda_2}{l_2}\right)t_1 + \frac{\lambda_2}{l_2}t_2 &= 0 \\
\frac{\lambda_2}{l_2}t_1 - \left(\frac{\lambda_2}{l_2}+\frac{1}{R_a}\right)t_2 + \frac{1}{R_a}t_3 &= 0 \\
\frac{1}{R_a}t_2 - \left(\frac{1}{R_a}+\frac{\lambda_4}{l_4}\right)t_3 + \frac{\lambda_4}{l_4}t_4 &= 0 \\
\frac{\lambda_4}{l_4}t_3 - \left(\frac{\lambda_4}{l_4}+\alpha_i\right)t_4 &= -\alpha_i t_i
\end{aligned}
$$

となる。数値を代入して行列表記すると下記の通りとなる。

$$\begin{bmatrix} -28.9 & 8.9 & & & \\ 8.9 & -10.3 & 1.4 & & \\ & 1.4 & -15.7 & 14.3 & \\ & & 14.3 & -32.6 & 18.3 \\ & & & 18.3 & -27.3 \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = \begin{bmatrix} -700 \\ 0 \\ 0 \\ 0 \\ -234 \end{bmatrix}$$

LU 分解を使用して上記行列を解くプログラムを 2.7 に示す。ただし、15 行目および 17 行目の「LinearAlgebra」は、プログラム 2.6 で記した LU 分解および前進および後退代入処理関数が属する static クラスである。

プログラム 2.7　LU 分解による連立一次方程式の求解の例

```
 1 private static void LUDecompositionTest()
 2 {
 3     double[,] matrix = new double[5, 5];    //行列
 4     double[] b = new double[5];             //解ベクトル
 5     int[] perm = new int[5];                //置換ベクトル
 6
 7     //値を代入
 8     matrix[0, 0] = -28.9; matrix[0, 1] = 8.9;   b[0] = -700;
 9     matrix[1, 0] = 8.9;   matrix[1, 1] = -10.3; matrix[1, 2] = 1.4;
10     matrix[2, 1] = 1.4;   matrix[2, 2] = -15.7; matrix[2, 3] = 14.3;
11     matrix[3, 2] = 14.3;  matrix[3, 3] = -32.6; matrix[3, 4] = 18.3;
12     matrix[4, 3] = 18.3;  matrix[4, 4] = -27.3; b[4] = -234;
13
14     //LU分解を実行
15     LinearAlgebra.LUDecomposition(matrix, perm);
16     //前進代入・後退代入処理
17     LinearAlgebra.FAndBSubstitute(matrix, perm, b);
18
19     for (int i = 0; i < b.Length; i++)
20         Console.WriteLine("t" + i.ToString() + " = " + b[i].ToString("F1"));
21 }
```

プログラムを実行すると t_0 = 34.6°C、t_1=33.7°C、t_2=27.9°C、t_3=27.3°C、t_4=26.9°C、が得られる。本例題は数値計算を使わずにも解けるため、手計算を行い、結果を比較すると良い。

3）Thomas Algorithm による三重対角行列連立一次方程式の解法

例題 2.1 の例に示される通り、問題を行列で表現すると対角要素の両隣以外が 0 となることは多い。このような行列を三重対角行列連立一次方程式と呼ぶ。式 2.12~2.15は Thomas algorithm と呼ばれる解法であり、前述の LU 分解に比較して計算速度が速い。実装結果をプログラム 2.8 に示す。

$$c_n' = \begin{cases} \dfrac{c_n}{b_n} & (n=0) \\ \dfrac{c_n}{b_n - a_n c_{n-1}'} & (n=1,2,3\ldots,N-1) \end{cases} \tag{2.12}$$

$$d_n' = \begin{cases} \dfrac{d_n}{b_n} & (n=0) \\ \dfrac{d_n - a_n d_{n-1}'}{b_n - a_n c_{n-1}'} & (n=1,2,3\ldots,N-1,N) \end{cases} \tag{2.13}$$

$$x_N = d_N' \tag{2.14}$$

$$x_n = d_n' - c_n' x_{n+1} \tag{2.15}$$

プログラム 2.8　Thomas algorithm による三重対角行列連立一次方程式の解法

Popolo.Numerics.MatrixOperation.LinearAlgebra class

```
1 /// <summary>
2 /// Thomas algorithmで三重対角行列連立一次方程式を解く
3 /// abc(0,i)*nx(i-1)+abc(1,i)*nx(i)+abc(2,i)*nx(i+1)=x(i)
4 /// </summary>
5 /// <param name="abc">係数行列</param>
6 /// <param name="x">解で上書きされる</param>
```

```
 7 public static void SolveTridiagonalMatrix(IMatrix abc, ref IVector x)
 8 {
 9   int num = abc.Columns - 1;
10   abc[2, 0] /= abc[1, 0];
11   x[0] /= abc[1, 0];
12
13   for (int i = 1; i < num; i++)
14   {
15     abc[2, i] /= abc[1, i] - abc[0, i] * abc[2, i - 1];
16     x[i] = (x[i] - abc[0, i] * x[i - 1]) / (abc[1, i] - abc[0, i] * abc[2, i - 1]);
17   }
18
19   x[num] = (x[num] - abc[0, num] * x[num - 1]) / (abc[1, num] - abc[0, num] * abc[2, num - 1]);
20   for (int i = num - 1; 0 <= i; i--) x[i] -= abc[2, i] * x[i + 1];
21 }
```

プログラム 2.8 を用いて例題 2.1 を解く例をプログラム 2.9 に示す。

プログラム 2.9　Thomas algorithm による三重対角連立一次方程式の求解の例

```
 1 private static void TDMATest()
 2 {
 3     double[,] abc = new double[3, 5];   //係数ベクトル
 4     double[] x = new double[5];         //解ベクトル
 5
 6     abc[0, 0] = 0; abc[1, 0] = -28.9; abc[2, 0] = 8.9;
 7     abc[0, 1] = 8.9; abc[1, 1] = -10.3; abc[2, 1] = 1.4;
 8     abc[0, 2] = 1.4; abc[1, 2] = -15.7; abc[2, 2] = 14.3;
 9     abc[0, 3] = 14.3; abc[1, 3] = -32.6; abc[2, 3] = 18.3;
10     abc[0, 4] = 18.3; abc[1, 4] = -27.3; abc[2, 4] = 0;
11     x[0] = -700; x[1] = 0; x[2] = 0; x[3] = 0; x[4] = -234;
12
13     LinearAlgebra.SolveTridiagonalMatrix(abc, ref x);
14     for (int i = 0; i < x.Length; i++)
15         Console.WriteLine("t" + i.ToString() + " = " + x[i].ToString("F1"));
16 }
```

4）双共役勾配法

計算対象が疎行列の場合には、非 0 の要素のみを計算対象とする反復法が有効である[†1)]。共役勾配法はその代表であり、本節の双共役勾配法（BCG: Biconjugate Gradient Method）はこれを非対称行列に対応させた手法である[†2)]。

適当な解ベクトルから出発して、これを修正しながら誤差を低減させていくということが反復法の基本方針である。問題は修正を行う方向であるが、各ステップでの修正を無駄にしないためには、修正方向が互いに直交していることが望ましい。このような方向を共役勾配方向と呼ぶ。

共役勾配方向と修正量を連続的に求めるためには下記の手順で計算を行う。j は反復回数である。

1）解ベクトルに適当な初期値 $\boldsymbol{x}_0$ を設定する

2）残差ベクトル $\boldsymbol{r}_0$ を計算する

$$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0$$

3）$\boldsymbol{p}^*_0 = \boldsymbol{p}_0 = \boldsymbol{r}^*_0 = \boldsymbol{r}_0$ とする。

4）式 2.16 で係数 α_j を計算する。ただし、(·)はベクトルの内積を表わす。

$$\alpha_j = \frac{(\boldsymbol{r}_j \cdot \boldsymbol{r}^*_j)}{(\boldsymbol{A}\,\boldsymbol{p}_j \cdot \boldsymbol{p}^*_j)} \tag{2.16}$$

5）式 2.17 で $\boldsymbol{x}_j$、$\boldsymbol{r}_j$、$\boldsymbol{r}^*_j$ を更新する。

$$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j \quad , \quad \boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j \boldsymbol{A}\,\boldsymbol{p}_j \quad , \quad \boldsymbol{r}^*_{j+1} = \boldsymbol{r}^*_j - \alpha_j \boldsymbol{A}^T \boldsymbol{p}^*_j \tag{2.17}$$

†1　直接法であっても疎行列に対応させたスカイライン法、マルチフロンタル法、ウェーブ・フロント法などがある [2.15)]。ただし、計算処理は共役勾配法に比較してかなり複雑である。

†2　共役勾配法は反復法でありながら（理論的には）有限の回数で収束することが保証されており、 1952 年に発表された当初は「科学史上に残る大発明」とされたらしい。この歴史については参考文献 2.16 が面白い。

6）誤差を評価して収束判定を満たすならば終了する。

7）式 2.18 で共役勾配方向 $\boldsymbol{p}_j$、$\boldsymbol{p}^*_j$ を更新する。

$$\beta_j=\frac{(\boldsymbol{r}_{j+1}\cdot\boldsymbol{r}^*_{j+1})}{(\boldsymbol{r}_j\cdot\boldsymbol{r}^*_j)} \quad , \quad \boldsymbol{p}_{j+1}=\boldsymbol{r}_{j+1}-\beta_j\boldsymbol{p}_j \quad , \quad \boldsymbol{p}^*_{j+1}=\boldsymbol{r}^*_{j+1}-\beta_j\boldsymbol{p}^*_j \tag{2.18}$$

8）ステップ 4）に戻る。

プログラム 2.10 に双共役勾配法による疎行列の解法を示す。SparseMatrix クラスに実装させる。入力は右辺ベクトル[*b*]と解ベクトルの初期値[*x*]である。13~22 行で初期値に対する残差ベクトルを計算する。16 行の Multiplicate はプログラム 2.4 で示した疎行列用の行列ベクトル積の計算処理である。24~58 行が反復計算内容である。28~32 行でステップ 4）の係数 α_j を計算する。29 行は転置行列に対するベクトル積の計算であり、62~74 行に示すとおりである。33~38 行でステップ 5）の計算を行い、40~48 行で収束を判定する。収束はベクトル[*b*]のノルムに対する残差の大きさで評価する。収束していない場合には 49~57 行で次の計算ステップの共役勾配方向を計算する。

プログラム 2.10　双共役勾配法による疎行列の解法

Popolo.Numerics.MatrixOperation.SparseMatrix class

```
 1 /// <summary>連立一次方程式 Ax=b の解 x を出力する</summary>
 2 /// <param name="vecB">右辺ベクトル b</param>
 3 /// <param name="vecX">出力:変数ベクトル x</param>
 4 public void SolveLinearEquation(IVector vecB, ref IVector vecX)
 5 {
 6   IVector ap = new Vector(Rows);
 7   IVector app = new Vector(Rows);
 8   IVector p = new Vector(Rows);
 9   IVector pp = new Vector(Rows);
10   IVector r = new Vector(Rows);
11   IVector rr = new Vector(Rows);
12
13   //初回の残差ベクトルを設定
14   double bnrm = 0;
15   double rnrm = 0;
16   Multiplicate(vecX, ref r);
17   for (int i = 0; i < Rows; i++)
18   {
19     r[i] = rr[i] = p[i] = pp[i] = vecB[i] - r[i];
20     bnrm += vecB[i] * vecB[i];
21     rnrm += r[i] * rr[i];
22   }
23
24   //収束計算開始
25   int maxIter = 10 * Rows;
26   for (int iter = 0; iter < maxIter; iter++)
27   {
28     Multiplicate(p, ref ap);
29     multiplicateT(pp, ref app);
30     double apnrm = 0;
31     for (int i = 0; i < Rows; i++) apnrm += ap[i] * pp[i];
32     double ak = rnrm / apnrm;
33     for (int i = 0; i < Rows; i++)
34     {
35       vecX[i] += ak * p[i];
36       r[i] -= ak * ap[i];
37       rr[i] -= ak * app[i];
38     }
39
40     //収束判定
41     Multiplicate(vecX, ref ap);
42     double err = 0;
43     for (int i = 0; i < Rows; i++)
44     {
45       ap[i] = vecB[i] - ap[i];
46       err += ap[i] * ap[i];
47     }
48     if (err / bnrm < 1e-10) return;
49     double rnrmbf = rnrm;
50     rnrm = 0;
```

```
51     for (int i = 0; i < Rows; i++) rnrm += r[i] * rr[i];
52     double bk = rnrm / rnrmbf;
53     for (int i = 0; i < Rows; i++)
54     {
55       p[i] = r[i] + bk * p[i];
56       pp[i] = rr[i] + bk * pp[i];
57     }
58   }
59   throw new Exception("iteration error in SparseMatrix");
60 }
61
62 /// <summary>転置行列と入力ベクトルとの積を計算して出力ベクトルに格納する</summary>
63 /// <param name="vec1">入力ベクトル</param>
64 /// <param name="vec2">出力：出力ベクトル</param>
65 public void multiplicateT(IVector vec1, ref IVector vec2)
66 {
67   vec2.Initialize(0);
68   for (int i = 0; i < Rows; i++)
69   {
70     Dictionary<int, double> row = elem[i];
71     foreach (int col in row.Keys)
72       vec2[col] += row[col] * vec1[i];
73   }
74 }
```

2.3　非線形方程式の解法

2.3.1　ニュートン・ラフソン法

ニュートン・ラフソン法は、非線形方程式の最も基本的な解法である。非線形方程式の解法とは言うものの、ある程度、関数形状がなめらかでないと収束しないことがある。経験上、建築熱環境分野の問題の多くは本手法を上手く適用すれば解けると推測するが、非線形性が非常に強い問題にあたった場合には、後述する Brent 法などの他の手法を試してみると良い。

図 2.2 はニュートン・ラフソン法による求解の原理を直感的に理解するための図である。$y = f(x) = 0$ を満たす x を求めるという問題が与えられたとする。グラフ上に描かれた $f(x)$ と x 軸が交わる点が解であり、この点にたどり着く方法が必要である。まず、適当な初期値 x_0 を設定する。x_0 における関数 $f(x)$の値および微分値を計算し、式 2.19 に従って x の値を更新する。これは図 2.2 において関数の接線を描いて x 軸と交差する点を求めることに相当する。設定した初期値から解に向かって関数がなめらかである場合には、図 2.2 に示されるように、更新の都度、x は解に近づく。$f(x)$が十分に 0 に近づいたと判断したら計算を打ち切る。

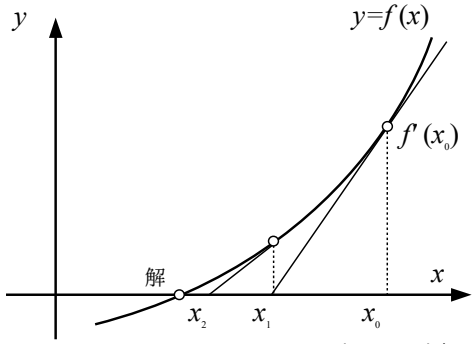


図 2.2　ニュートン・ラフソン法による求根

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.19}$$

前記の処理は数学的には次のように説明できる。x_nのまわりで関数 $f(x)$を 1 次までテイラー展開すると式 2.20 が得られる。この式が 0 となる点が解であるため、式 2.20 を 0 と等号で結んで x について

解くと式 2.21 となる。しかしこれはテイラー展開による近似式における解に過ぎないため、式 2.21 を繰り返し適用（式 2.19）する必要がある。

$$f(x)\approx f(x_n)+f'(x_n)(x-x_n) \tag{2.20}$$

$$x=x_n-\frac{f(x_n)}{f'(x_n)} \tag{2.21}$$

実際の問題では関数の微分値が得られない場合が多い。この場合には入力値をわずかに変化させて出力を求め、その差分を利用して式 2.22 で微分値を近似する。Δ は入力値の微小変化幅である。このような方法を数値微分と呼ぶ。

$$f'(x_n)=\frac{f(x_{n+\Delta})-f(x_n)}{\Delta} \tag{2.22}$$

【例題 2.2】

あるポンプの PQ 特性（16 章で解説）が式 2.23 で表現されるとする。また、配管抵抗により流量 m_w [m³/s]と圧力 P [kPa]の関係が式 2.24 で表現されるとする。この時、管路内に流れる水量を求めよ。

$$P=-1.32\text{E-}6\,m_w^4-4.90\text{E-}6\,m_w^3+2.08\text{E-}3\,m_w^2-0.314\,m_w+333 \tag{2.23}$$

$$P=4.66\text{E-}2\,m_w^2+0.567\,m_w+49.4 \tag{2.24}$$

【解】

式 2.23 を $f_1(m_w)$、式 2.24 を $f_2(m_w)$ とすると、$f_3(m_w) = f_1(m_w) - f_2(m_w) = 0$ を解けば良い。なお、$f_1(m_w)$、$f_2(m_w)$、$f_3(m_w)$をそれぞれグラフ化すると図 2.3 となる。66 m³/s あたりに解がありそうである。

上記を解くためのプログラムを 2.11 に示す。$f_3(m_w)$の微分値は直接求めることも可能であるが、ここでは練習のため、数値微分を用いる方法とした。実行すると G=66.6 m³/s が得られる。ところで、33~42 行では少し変わった方法で多項式の計算を行っている。多くのプログラム言語では実数のべき乗の計算をする関数が用意されているが、通常は単純な掛け算や足し算に比較して計算速度が遅いため、下記のような形式で記述することが多い。設備シミュレーションの分野においても多項式近似を行う場面は非常に多く、このような表現方法に慣れておくとプログラムの読解に有利だろう。

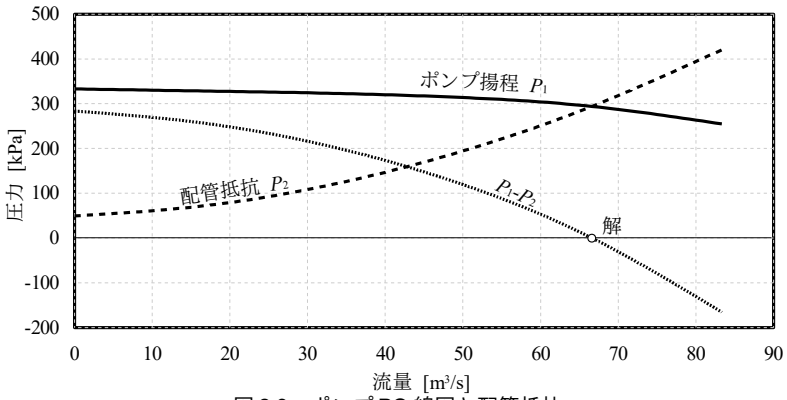


図 2.3　ポンプ PQ 線図と配管抵抗

プログラム **2.11**　ニュートン・ラフソン法による求根 **1**

```
 1 private static void NewtonRaphsonTest()
 2 {
 3   const int MAX_ITERATION = 100;  //最大反復回数
 4   const double DELTA = 0.0001;//微分値計算用微小値
 5   const double ERR_TOLERANCE = 0.001; //最大許容誤差
 6
 7   //mw の初期値は 0 とする
 8   double mw = 0;
 9
10   //誤差を計算
11   double fmw1 = fmw(mw);
12
13   //反復回数を記録
14   int iterNumber = 0;
```

```
15
16   //誤差が許容誤差未満になるまで反復計算
17   while (ERR_TOLERANCE < Math.Abs(fmw1) && iterNumber < MAX_ITERATION)
18   {
19     //数値微分値の計算
20     //mw を微小変動させて誤差を再計算
21     double fmw2 = fmw(mw + DELTA);
22     double fmwd = (fmw2 - fmw1) / DELTA;
23
24     //微分値を利用して mw を更新
25     mw -= fmw1 / fmwd;
26
27     //誤差を再計算
28     fmw1 = fmw(mw);
29   }
30   Console.WriteLine("mw= " + mw.ToString("F3"));
31 }
32
33 private static double fmw(double mw)
34 {
35   //ポンプの PQ 特性
36   double p1 = mw * (-0.314 + mw * (2.08e-3 + mw * (-4.9e-6 - 1.32e-6 * mw))) + 333;
37
38   //配管の抵抗
39   double p2 = mw * (0.567 + 4.66e-2 * mw) + 49.4;
40
41   return p1 - p2;
42 }
```

ニュートン・ラフソン法は様々なモデルの計算で頻出する手法のため、プログラム 2.12 に示すように汎用の関数としてまとめておくと便利である。1~4 行は誤差関数の入出力の定義であり、1 つの実数型の引数にもとづき、実数型の出力を返すメソッドを定義する。6~30 行のメソッドでは引数として誤差関数を受け取り、これにニュートン・ラフソン法を適用して求根する。32~55 行のメソッドは微分値を計算するための関数も与えられる場合であり、数値微分が不要な分、こちらの方が速い。

プログラム 2.12　ニュートン・ラフソン法の汎用メソッドの定義

Popolo.Numerics.Roots class

```
 1 /// <summary>誤差関数</summary>
 2 /// <param name="x">入力値</param>
 3 /// <returns>誤差</returns>
 4 public delegate double ErrorFunction(double x);
 5
 6 /// <summary>ニュートン法で求根する</summary>
 7 /// <param name="eFnc">誤差関数</param>
 8 /// <param name="x">初期値</param>
 9 /// <param name="delta">数値微分用デルタ</param>
10 /// <param name="errorTolerance">誤差量許容値</param>
11 /// <param name="collectionTolerance">修正量許容値</param>
12 /// <param name="maxIteration">最大反復回数</param>
13 /// <returns>解</returns>
14 public static double Newton(ErrorFunction eFnc, double x, double delta,
15   double errorTolerance, double collectionTolerance, int maxIteration)
16 {
17   int iNum = 0;
18   double err1 = eFnc(x);
19   while (errorTolerance < Math.Abs(err1))
20   {
21     if (maxIteration < iNum) throw new Exception("Newton's method iteration error");
22     double err2 = eFnc(x + delta);
23     double dX = (err1 * delta) / (err2 - err1);
24     x -= dX;
25     if (Math.Abs(dX) < collectionTolerance) break;
26     err1 = eFnc(x);
27     iNum++;
28   }
29   return x;
30 }
31
32 /// <summary>ニュートン法で求根する</summary>
33 /// <param name="eFnc">誤差関数</param>
34 /// <param name="eFncD">誤差関数の微分</param>
35 /// <param name="x">初期値</param>
36 /// <param name="errorTolerance">誤差量許容値</param>
37 /// <param name="collectionTolerance">修正量許容値</param>
38 /// <param name="maxIteration">最大反復回数</param>
```

```
39 /// <returns>解</returns>
40 public static double Newton(ErrorFunction eFnc, ErrorFunction eFncD,
41   double x, double errorTolerance, double collectionTolerance, int maxIteration)
42 {
43   int iNum = 0;
44   double err = eFnc(x);
45   while (errorTolerance < Math.Abs(err))
46   {
47     if (maxIteration < iNum) throw new Exception("Newton's method iteration error");
48     double dX = err / eFncD(x);
49     x -= dX;
50     if (Math.Abs(dX) < collectionTolerance) break;
51     err = eFnc(x);
52     iNum++;
53   }
54   return x;
55 }
```

プログラム 2.12 のようなメソッドを用意しておけば、例えば例題 2.2 の処理はプログラム 2.13 で記述することができる。3~11 行は匿名メソッドと呼ばれるメソッドの定義法である。プログラム 2.11 のように誤差関数を一々定義しなくても済むため、上手く使えばプログラムの可読性が向上する。

プログラム 2.13　ニュートン・ラフソン法による求根 2

```
 1 private static void NewtonRaphsonTest2()
 2 {
 3   //誤差関数を匿名メソッドで定義
 4   Roots.ErrorFunction eFnc = delegate (double mw)
 5   {
 6     //ポンプの PQ 特性
 7     double p1 = mw * (-0.314 + mw * (2.08e-3 + mw * (-4.9e-6 - 1.32e-6 * mw))) + 333;
 8     //配管の抵抗
 9     double p2 = mw * (0.567 + 4.66e-2 * mw) + 49.4;
10     return p1 - p2;
11   };
12   //ニュートンラフソン法を適用
13   double waterFlowRate = Roots.Newton(eFnc, 0, 0.0001, 0.001, 0.001, 100);
14
15   Console.WriteLine("mw= " + waterFlowRate.ToString("F3"));
16 }
```

ニュートン・ラフソン法に限らず、数値計算を行う際には数値微分を用いることが多い。このときにどの程度の値を微小変化幅Δとするかは計算の安定性に大きな影響を与える。

プログラム作成者が理解しておくべき点は大きく 2 つあり、1 つは機械精度（機械イプシロンとも呼ぶ）である。計算機は無限に精度の高い数値を取り扱うことはできないため、2 つの数値を足し合わせた際に両者の値が著しく異なると、小さい方の数値を表現できなくなる。機械精度は 1.0 に加算した時に 1.0 以外の数値として表現できる限界の数値である。逆に言えば、これ未満の数値を 1.0 に加算しても、コンピュータ上では何も起きない。

もう 1 つ、意識することが必要な点は、数値微分を階層的に重ねる可能性があるということである。例えば、第 4 章では湿り空気について解説を行い、いくつかの物性値計算において数値微分を用いる。湿り空気の物性は建築設備システムの基礎的情報であるため、以降の章で示すように各種の設備機器の計算においても繰り返し用いる。湿り空気の物性を含む形で数値微分が必要になった場合、その精度は湿り空気の物性計算の精度を下回る必要がある点に注意が必要である。例えば、数値微分を用いて湿り空気の湿球温度が 0.01 ℃ の精度で計算できるとする。この場合には、冷水コイルモデルで湿球温度を 0.0001 ℃ だけ変化させて数値微分を得ようとしても、そもそもの湿球温度の精度が 0.01 ℃ しか無いため、微分値は安定しない。

基本的な方針としては、まずは数値微分を用いない解析的な計算法を探すことが第一である。これが見つからない場合には、水や空気などの基本的な計算に関しては精度を相対的に高く設定し、個別機器、サブシステム、全体システムとなるにつれて精度を下げていくということが必要になる。

プログラム 2.14 に機械精度の計算処理を示す。

プログラム 2.14　機械精度の計算

```
 1 private static double GetMECHEPS()
 2 {
 3   double MECH_EPS = 1.0;
 4   while (true)
 5   {
 6     if (1.0 + MECH_EPS <= 1.0) return MECH_EPS * 2;
 7     else MECH_EPS = MECH_EPS * 0.5;
 8   }
 9 }
```

2.3.2　囲い込み法

設備のシミュレーションにおいて収束計算が必要となる際には、解が存在する範囲がわかっている場合も多い[†1]。このような場合には、二分法、割線法、挟み撃ち法、Brent 法などの方法を用いて、解を囲い込む様に収束計算を行うと確実性が高い。

二分法（Bisection method）は単純ではあるが確実な手法である。図 2.4 に二分法の概要を示す。解を含む範囲として初期値 x_1、x_2 を仮定する。中間点である x_3 において関数を評価し、同符号である初期値と入れ替える（この場合は x_2）。以降、この操作を繰り返し x_4、x_5、x_6...と次第に解の存在する領域を絞り込んでいき、所定の精度で絞り込みができた段階で終了する。

二分法による求根プログラムを 2.15 に示す。34 行が収束判定処理であり、絶対誤差が第 5 引数未満、修正量が第 6 引数未満となった場合に終了する。

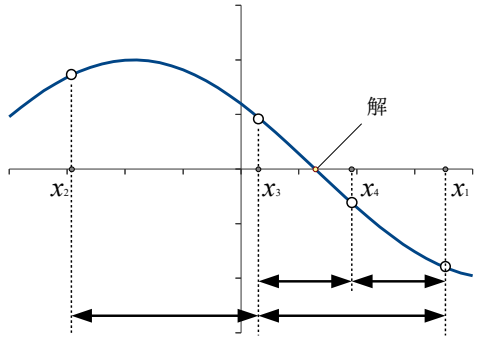


図 2.4　二分法による求根

プログラム 2.15　二分法による求根計算

Popolo.Numerics.Roots class

```
 1 /// <summary>二分法で求根する</summary>
 2 /// <param name="a">求根範囲 1</param>
 3 /// <param name="b">求根範囲 2</param>
 4 /// <param name="fa">求根範囲 1 での評価値</param>
 5 /// <param name="fb">求根範囲 2 での評価値</param>
 6 /// <param name="errTolerance">誤差量許容値</param>
 7 /// <param name="collecTolerance">修正量許容値</param>
 8 /// <param name="eFnc">誤差関数</param>
 9 /// <param name="maxIter">最大反復回数</param>
10 /// <returns>解</returns>
11 public static double Bisection(double a, double b, double fa, double fb,
12   double errTolerance, double collecTolerance, ErrorFunction eFnc, int maxIter)
13 {
14   //根が囲い込めていない場合はエラー
15   if (0 < fa * fb) throw new Exception("Bisection initial point error");
16
17   int iterNum = 0;
18   while (true)
19   {
20     //中間点を計算
21     double c = 0.5 * (a + b);
22     double fc = eFnc(c);
```

†1　例えば、冷温水であれば 0~50°C 程度の範囲に答えが存在するであろうと予想がつく。このような範囲を定めて収束計算をして解が見つからない場合には、むしろモデルの設計に問題があると判断した方が良い。

```
23     //入替え処理
24     if (Math.Sign(fc) == Math.Sign(fa))
25     {
26       fa = fc;
27       a = c;
28     }
29     else
30     {
31       fb = fc;
32       b = c;
33     }
34     if ((Math.Abs(fc) < errTolerance) || Math.Abs(a - b) < collecTolerance) return c;
35     iterNum++;
36     if (maxIter < iterNum) throw new Exception("Bisection Iteration Error");
37   }
38 }
39
40 /// <summary>二分法で求根する</summary>
41 /// <param name="eFnc">誤差関数</param>
42 /// <param name="a">求根範囲 1</param>
43 /// <param name="b">求根範囲 2</param>
44 /// <param name="errorTolerance">誤差量許容値</param>
45 /// <param name="collectionTolerance">修正量許容値</param>
46 /// <param name="maxIteration">最大反復回数</param>
47 /// <returns>解</returns>
48 public static double Bisection
49   (ErrorFunction eFnc, double a, double b, double errorTolerance, double collectionTolerance, int maxIteration)
50 { return Bisection(a, b, eFnc(a), eFnc(b), errorTolerance, collectionTolerance, eFnc, maxIteration); }
```

プログラム 2.16 は Brent 法による求根プログラムである[2.2)]。解を囲い込む範囲で逆 2 次補間処理を行うため、1 次収束である二分法よりも高速で収束する。また、誤差関数の形状により、反復計算中に求根範囲を飛び出そうとするような場合には適宜、二分法を実行するため、堅牢性も高い。

プログラム 2.16　Brent 法による求根計算

Popolo.Numerics.Roots class

```
 1 /// <summary>Brent 法で求根する</summary>
 2 /// <param name="a">求根範囲 1</param>
 3 /// <param name="b">求根範囲 2</param>
 4 /// <param name="errorTolerance">許容誤差</param>
 5 /// <param name="eFnc">誤差関数</param>
 6 /// <returns>解</returns>
 7 public static double Brent(double a, double b, double errorTolerance, ErrorFunction eFnc)
 8 {
 9   const int MAX_ITER = 100;
10
11   double e = 0;
12   double d = 0;
13   double fa = eFnc(a);
14   double fb = eFnc(b);
15   double c = a;
16   double fc = fa;
17
18   int iterNum = 0;
19   while (true)
20   {
21     //f(b)と f(c)が同一符号となった場合は入替え
22     if ((0 < fb && 0 < fc) || (fb < 0 && fc < 0))
23     {
24       c = a;
25       fc = fa;
26       e = d = b - a;
27     }
28
29     if (Math.Abs(fc) < Math.Abs(fa))
30     {
31       a = b;
32       b = c;
33       c = a;
34       fa = fb;
35       fb = fc;
36       fc = fa;
37     }
38
39     //収束判定
40     double tol = 2.0 * MECH_EPS * Math.Abs(b) + errorTolerance;
41     double mid = 0.5 * (c - b);
42     if (Math.Abs(mid) < tol || fb == 0.0) return b;
```

```
43
44      if ((Math.Abs(e) < tol) || Math.Abs(fa) <= Math.Abs(fb))
45      {
46        //区間幅の減少が小さい場合は二分法で処理
47        d = mid;
48        e = d;
49      }
50      else
51      {
52        //逆二次補間を試す
53        double p, q, r;
54        double s = fb / fa;
55        if (a == c)
56        {
57          p = 2.0 * mid * s;
58          q = 1.0 - s;
59        }
60        else
61        {
62          q = fa / fc;
63          r = fb / fc;
64          p = s * (2.0 * mid * q * (q - r) - (b - a) * (r - 1.0));
65          q = (q - 1.0) * (r - 1.0) * (s - 1.0);
66        }
67        if (0 < p) q = -q;
68        p = Math.Abs(p);
69        double min1 = 3.0 * mid * mid * q - Math.Abs(tol * q);
70        double min2 = Math.Abs(e * q);
71        if (2.0 * p < Math.Min(min1, min2))
72        {
73          e = d;
74          d = p / q;
75        }
76        else
77        {
78          d = mid;
79          e = d;
80        }
81      }
82      a = b;
83      fa = fb;
84      if (tol < Math.Abs(d)) b += d;
85      else b += Math.Sign(mid) * tol;
86      fb = eFnc(b);
87
88      iterNum++;
89      if (MAX_ITER < iterNum) throw new Exception("Brent Iteration Error");
90    }
91  }
```

2.3.3　3次方程式の根

2次方程式に関しては解の公式がよく知られているが、3次方程式と4次方程式に関しても代数的に解く方法が存在する。設備機器の特性は2次~3次程度の多項式で近似する場合が多く、低次の多項式について代数的解法を用意しておくと便利なことが多い。

3次方程式（式2.25）の解法はカルダーノの方法と呼ばれる[2.21)]。式2.26に示すように変数xをyで表すと、式2.25と式2.27に変換できる。ただし、pおよびqはそれぞれ式2.28ある。

$$a_0 x^3+a_1 x^2+a_2 x+a_3=0 \tag{2.25}$$

$$x=y-\frac{a_1}{3a_0} \tag{2.26}$$

$$y^3+3py+q=0 \tag{2.27}$$

$$p=-\frac{a_1^2}{9a_0^2}+\frac{a_2}{3a_0}, \qquad q=-\frac{2a_1^3}{27a_0^3}-\frac{a_1 a_2}{3a_0^2}+\frac{a_3}{a_0} \tag{2.28}$$

3次方程式の実数根は1つまたは3つであり、式2.29に示す判別式を用いて判定する。$D<0$であれば3つの実数根、$0 \le D$であれば1つの実数根である。3つの実数根を持つ場合には、式2.27の3つの解を式2.30~2.32で求め、これを式2.26に代入してxを計算する。ただしγとθは式2.33である。

$$D=q^2+4p^3 \tag{2.29}$$

$$y_0=2\sqrt{-p}\cos\frac{\theta}{3} \tag{2.30}$$

$$y_1=2\sqrt{-p}\cos\left(\frac{2\pi+\theta}{3}\right) \tag{2.31}$$

$$y_2=2\sqrt{-p}\cos\left(\frac{4\pi+\theta}{3}\right) \tag{2.32}$$

$$y=\sqrt{-p^3}\quad,\quad \theta=\tan^{-1}\sqrt{D}/q \tag{2.33}$$

1 つの実数根を持つ場合には式 2.34 で y を求め、これを式 2.26 に代入して x を計算する。

$$y_0=\left\{0.5\left(-q+\sqrt{D}\right)\right\}^{1/3}+\left\{0.5\left(-q-\sqrt{D}\right)\right\}^{1/3} \tag{2.34}$$

プログラム 2.17 に 3 次方程式の求根処理を示す。9~12 行で p と q を求めて 21 行で判別式を適用する。特殊な場合として、p=q=0 であれば式 2.26 の y に 0 を代入して処理を終える（14~18 行）。

プログラム 2.17　3 次方程式の求根処理（カルダーノの方法）

Popolo.Numerics.CubicEquation class

```
 1 /// <summary>3 次方程式の実数根を求める</summary>
 2 /// <param name="a">係数: a[0]*x^3+a[1]*x^2+a[2]*x+a[3]</param>
 3 /// <param name="x0">出力: 実数根 1</param>
 4 /// <param name="x1">出力: 実数根 2</param>
 5 /// <param name="x2">出力: 実数根 3</param>
 6 /// <param name="hasMultiSolution">出力: 複数解があるか否か</param>
 7 public static void Solve(double[] a, out double x0, out double x1, out double x2, out bool hasMultiSolution)
 8 {
 9   //p,q を計算
10   double bf = a[1] / (3 * a[0]);
11   double p = - bf * bf + a[2] / (3 * a[0]);
12   double q = bf * (2 * bf * bf - a[2] / a[0]) + a[3] / a[0];
13
14   if (Math.Abs(p) < MECH_EPS && Math.Abs(q) < MECH_EPS)
15   {
16     hasMultiSolution = false;
17     x0 = x1 = x2 = -bf;
18   }
19
20   //根の数の判定
21   double pq = q * q + 4 * p * p * p;
22   //実数根 3 つ
23   if (pq < 0)
24   {
25     hasMultiSolution = true;
26     double theta = Math.Atan2(Math.Sqrt(-pq) ,-q);
27     double sqp2 = 2 * Math.Sqrt(-p);
28     x0 = sqp2 * Math.Cos(theta / 3d) - bf;
29     x1 = sqp2 * Math.Cos((2 * Math.PI + theta) / 3d) - bf;
30     x2 = sqp2 * Math.Cos((4 * Math.PI + theta) / 3d) - bf;
31   }
32   //実数根 1 つ
33   else
34   {
35     hasMultiSolution = false;
36     double sqpq = Math.Sqrt(pq);
37     double r1 = 0.5 * (-q + sqpq);
38     double r2 = 0.5 * (-q - sqpq);
39     x0 =Math.Sign(r1) * Math.Pow(Math.Abs(r1), 1d / 3) + Math.Sign(r2) * Math.Pow(Math.Abs(r2), 1d / 3) - bf;
40     x1 = x2 = 0;
41   }
42 }
```

2.3.4　多変数のニュートン・ラフソン法

未知数が複数である非線形連立方程式に対してもニュートン・ラフソン法を適用することができる。1 変数の場合と同様に、解きたい非線形連立方程式 $f_i(\boldsymbol{x})$ を 1 次までテイラー展開すると式 2.35 が得られる。非線形連立方程式であるため $f_i(\boldsymbol{x})$ の $\boldsymbol{x}$ はベクトルであることに注意する。また、$\partial f_i / \partial x_j$ は、連立させるそれぞれの式を各変数で微分した行列でありヤコビアン（またはヤコブ行列）と呼

ぶ。δx_jはj番目の変数の修正量である。$f_i(\boldsymbol{x})$=0 が解であるから式 2.35 を 0 と結んで整理すると式 2.36 が得られる。この線形連立方程式を δx_jについて解き、得られた修正量で x の更新を繰り返すことで解にたどり着く。

$$f_i(\boldsymbol{x}) \approx f_i(\boldsymbol{x}) + \sum_{j=0}^{N} \frac{\partial f_i}{\partial x_j} \delta x_j \tag{2.35}$$

$$\sum_{j=0}^{N} \frac{\partial f_i}{\partial x_j} \delta x_j = -f_i(\boldsymbol{x}) \tag{2.36}$$

プログラム 2.18 に多変数のニュートン・ラフソン法の計算処理を示す。4 行は誤差関数であり、入出力ともにベクトルである。50~69 行は補助関数であり、誤差関数を用いて数値微分でヤコビアンを計算する。6~48 行がメインの処理である。反復回数が最大回数に達するか、x の変化率または $f(x)$の絶対値が収束条件に到達すれば計算を終了させるため、これらの条件を引数にする。25~47 行が反復計算処理である。29 行でヤコビアンを計算し、31 行で式 2.36 を解く。34~46 行で収束判定を行う。

プログラム 2.18　多変数のニュートン・ラフソン法

Popolo.Numerics.MultiRoots class

```
 1 /// <summary>誤差関数</summary>
 2 /// <param name="x">入力ベクトル</param>
 3 /// <param name="fx">出力ベクトル</param>
 4 public delegate void ErrorFunction(IVector x, ref IVector fx);
 5
 6 /// <summary>ニュートン法で求根する</summary>
 7 /// <param name="eFnc">誤差関数</param>
 8 /// <param name="x">入力:初期値,出力:収束値</param>
 9 /// <param name="errorTolerance">誤差量許容値</param>
10 /// <param name="collectionTolerance">入力変化率の収束条件</param>
11 /// <param name="maxIteration">最大反復回数</param>
12 /// <param name="iteration">反復回数</param>
13 /// <param name="error">最終誤差</param>
14 /// <returns>求根成功の真偽</returns>
15 public static bool Newton
16   (ErrorFunction eFnc, ref IVector x, double errorTolerance, double collectionTolerance,
17   int maxIteration, out int iteration, out double error)
18 {
19   iteration = 0;
20   int num = x.Length;
21   IVector fx = new Vector(num);
22   IMatrix jac = new Matrix(num, num);
23   eFnc(x, ref fx);
24   error = 0;
25   while (true)
26   {
27     if (maxIteration < iteration) return false;
28
29     computeJacobian(eFnc, ref x, fx, ref jac);
30     for (int i = 0; i < num; i++) fx[i] = -fx[i];
31     LinearAlgebra.SolveLinearEquations(ref jac, ref fx);
32     for (int i = 0; i < num; i++) x[i] += fx[i];
33
34     double maxCr = 0;
35     for (int i = 0; i < num; i++)
36     {
37       double bf = fx[i];
38       if (1e-5 < Math.Abs(x[i])) bf /= x[i];
39       else bf /= 1e-5;
40       maxCr = Math.Max(maxCr, Math.Abs(bf));
41     }
42     eFnc(x, ref fx);
43     error = 0;
44     for (int i = 0; i < num; i++) error += Math.Abs(fx[i]);
45     if (error < errorTolerance && maxCr < collectionTolerance) return true;
46     iteration++;
47   }
48 }
49
50 /// <summary>ヤコビアンを求める</summary>
51 /// <param name="eFnc">誤差関数</param>
52 /// <param name="x">入力ベクトル</param>
```

```
53 /// <param name="fx">入力ベクトルに対する出力 f(x)</param>
54 /// <param name="jac">出力:ヤコビアン</param>
55 private static void computeJacobian
56   (ErrorFunction eFnc, ref IVector x, IVector fx, ref IMatrix jac)
57 {
58   IVector vec = new Vector(x.Length);
59   for (int j = 0; j < x.Length; j++)
60   {
61     double tmp = x[j];
62     double dt = 1e-7 * Math.Abs(tmp);
63     if (dt < 1e-8) dt = 1e-8;
64     x[j] += dt;
65     eFnc(x, ref vec);
66     x[j] = tmp;
67     for (int i = 0; i < x.Length; i++) jac[i, j] = (vec[i] - fx[i]) / dt;
68   }
69 }
```

2.4　関数の極小値の探索

ポンプの変流量制御において冷温水の流量を最小化させる場合や、冷凍サイクルの計算において凝縮圧力を最小化させる場合など、関数の極小値を探索したい場合がある。これは数理最適化や数理計画法と呼ばれる問題であり、対象となる関数を最適化問題と呼ぶ[2,8]。本節では一変数関数の解法として黄金分割法を、多変数関数の解法として準ニュートン法を取り上げる。

2.4.1　一変数関数の最適化

一変数関数の極小値探索法としては、黄金分割法や Brent 法[2,2]がよく知られている。非線形方程式の解法の項で述べたように Brent 法は曲線近似を用いた方法である。後述の準ニュートン法で類似の方法を解説するため、本節では黄金分割法について述べる。黄金分割法は速度は遅いが確実に収束するというメリットがある[†1]。

黄金分割法は、ある範囲を限定した上で、その範囲内の極小値に漸近していく手法である。図 2.5 に黄金分割法による極小値の囲い込みの概念を示す。説明のため、左下方向に凸の関数と、右下方向に凸の関数の 2 つを用意した。ある関数 $f(x)$について a, b, c の 3 点で関数の値 $f(a), f(b), f(c)$が与えられているとする。$a < b < c$ で $f(b) < f(a)$かつ $f(b) < f(c)$の場合には、関数 $f(x)$は a から c の範囲内に極小値を持つ。極小値の範囲を絞り込むために、b から c の間（何故 a から b の間ではないのかについては後述）に x_1 を取り、関数値 $f(x_1)$を評価する。図 2.5 左に示すように $f(b) < f(x_1)$であれば、極小値は a~x_1 の範囲内にある。図 2.5 右に示すように $f(x_1) < f(b)$であれば、極小値は b~c の範囲内にある。前者であれば a, b, x_1 を新たな 3 点、後者であれば b, x_1, c を新たな 3 点として、それぞれ絞り込みを進める。

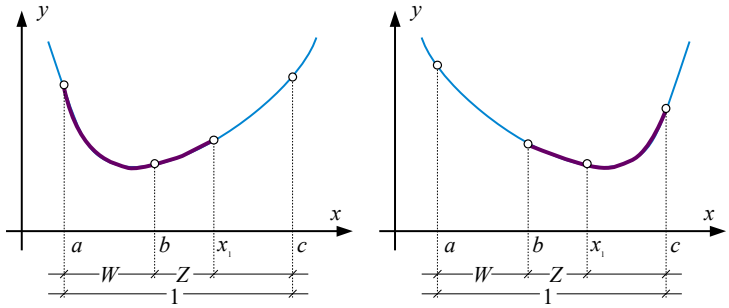


図 2.5　黄金分割法による極小値の囲い込み

ところで $f(x_1)$を実際に評価するまでは関数形状が図 2.5 の右か左かは不明であるが、いずれの場合であっても一定の絞り込み幅を確保できる戦略をとりたい。このためには x_1 の位置を以下の通りとす

†1　後述の多変数の極小値探索は、一変数の方法とは異なり収束が必ずしも保証されない。問題の性格上、やむを得ない場合には多変数の手法を適用する必要があるが、原則としてはモデルを修正して一変数関数の最適化問題に変更する方が良い。

る。*a*~*c* の幅（*c*-*a*）を 1 とおいた場合の *a* と *b* の幅（*b*-*a*）を *W*、*b* と x_1 の幅（x_1-*b*）を *Z* とする。即ち *W*=(*b*-*a*)/(*c*-*a*)、*Z*=(x_1-*b*)/(*c*-*a*)である。一回の絞込み計算の結果、図 2.5 左の場合には(*W*+*Z*)に範囲が狭まり、図 2.5 右の場合には(1-*W*)に範囲が狭まる。これらが等しくなるようにすれば関数形状の如何によらず一定の絞りこみ幅を保証できる。即ち式 2.37 が成立する必要がある。これを解くと *b*-*a*=*c*-x_1 となる。これは *ac* 間において x_1 を *b* と対称の位置に設定することが必要ということを意味する。

$$W+Z=1-W \tag{2.37}$$

ところで絞込み操作を連続的に行うという観点からは、そもそも前回の計算において *b* が、今回の計算での x_1 と同様の処理のもとに設定されたということが必要である。つまり、もともとの区間 *ac* に対する区間 *ab* の比率（=*W*）は、新たな区間である *bc* に対する区間 *b*x_1 の比率（=*Z*/(1-*W*)）と同じでなければならない。即ち式 2.38 が成立する必要がある。式 2.37 と 2.38 を連立させると式 2.39 が得られ、これを解くと *W*=0.38197 となる。つまり、最初に与えられた区間を 0.38197 : 0.61803 に分割するように第 3 の点を選択し、この操作を繰り返すことで極小値を絞り込んでいく[†1]。

$$W+Z=1-W \tag{2.38}$$

$$W^2-3W+1=0 \tag{2.39}$$

黄金分割法による極小値計算プログラムをプログラム 2.19 に示す。4 行目は最小化する関数の関数形状の定義である。引数として 1 の実数を取り、1 の実数値を返すプログラムとして定義している。11~71 行が極小値探索処理の本体である。17~25 行で探索を開始する 3 点の関数値を評価して初期化を行い、28~70 行で繰り返し計算を行う。

プログラム 2.19　黄金分割法による極小値計算プログラム

Popolo.Numerics.Minimization class

```
 1 /// <summary>最小化する関数</summary>
 2 /// <param name="x">入力値</param>
 3 /// <returns>出力値</returns>
 4 public delegate double MinimizeFunction(double x);
 5
 6 /// <summary>黄金分割法で極小値を探索する</summary>
 7 /// <param name="xMin">x の最小値</param>
 8 /// <param name="xMax">x の最大値</param>
 9 /// <param name="mFnc">最小化する関数</param>
10 /// <returns>極小値をとる x</returns>
11 public static double GoldenSection(double xMin, double xMax, MinimizeFunction mFnc)
12 {
13   const int MAX_ITER = 100;
14   const double ERR_TOL = 0.0001;
15   const double G_RATIO = 0.61803399;
16
17   //3つの点
18   double a = xMin;
19   double b = xMin + (xMax - xMin) * G_RATIO;
20   double c = xMax;
21
22   //3点での関数値を評価
23   double fa = mFnc(a);
24   double fb = mFnc(b);
25   double fc = mFnc(c);
26
27   int iterNum = 0;
28   while (true)
29   {
30     //4点目を設定・評価
31     if (b - a < c - b)
32     {
33       double x1 = a + (c - a) * G_RATIO;
34       double fx1 = mFnc(x1);
```

†1　この比率が黄金比と呼ばれる比率であり、建築計画分野においてはル・コルビュジエが自分の建築の構成を説明するために用いたことで有名である。

```
35        if (fx1 < fb)
36        {
37          a = b;
38          fa = fb;
39          b = x1;
40          fb = fx1;
41        }
42        else
43        {
44          c = x1;
45          fc = fx1;
46        }
47      }
48      else
49      {
50        double x1 = c - (c - a) * G_RATIO;
51        double fx1 = mFnc(x1);
52        if (fx1 < fb)
53        {
54          c = b;
55          fc = fb;
56          b = x1;
57          fb = fx1;
58        }
59        else
60        {
61          a = x1;
62          fa = fx1;
63        }
64      }
65
66      if (Math.Abs(c - a) < ERR_TOL) return b;
67
68      iterNum++;
69      if (MAX_ITER < iterNum) throw new Exception("Minimization Iteration Error");
70    }
71  }
```

【例題 2.3】

冷却塔消費電力 P_{ct} と冷凍機消費電力 P_{tb} が、冷却水温度 t_{cd} の関数として下記の式によって表現されているとする。この時、冷却塔と冷凍機の合計消費電力量が最小となる冷却水温度を求めよ。

$P_{ct} = 70\times(-2.4048\text{e-}5\times t_{cd}^5 + 3.2651\text{e-}3\times t_{cd}^4 - 0.1768\times t_{cd}^3 + 4.775\times t_{cd}^2 - 64.39\times t_{cd} + 347.73)$

$P_{tb} = 15\times(4.1126\text{e-}4\times t_{cd}^2 + 5.1299\text{e-}3\times t_{cd} + 0.41472)$

【解】

図 2.6 に上式にもとづいて計算した冷却水温度と消費電力量の関係を示す。低い冷却水温度は冷却塔にとってはファン動力の増大をもたらす。逆に、冷凍機にとっては高圧低圧の減少により、圧縮機動力の低下をもたらす。本問ではこれらのトレードオフの関係を捉えながら、合計の消費電力量の最小化を図ることを要求している。図 2.6 によれば 22°C 程度で電力消費量は最小化しそうである†1)。

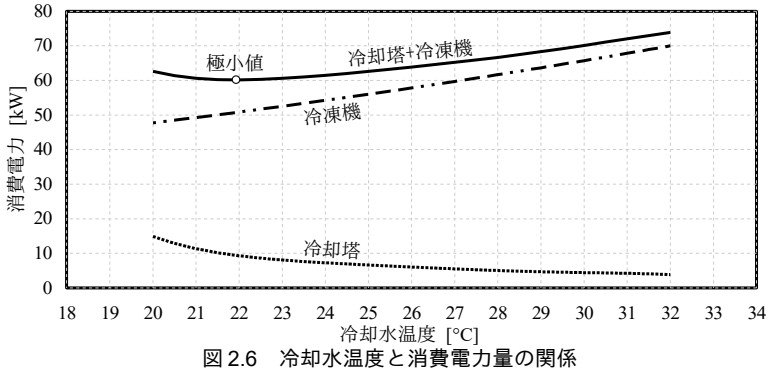


図 2.6　冷却水温度と消費電力量の関係

黄金分割法を用いて電力消費量の極小値を探索する。プログラムを 2.20 に示す。3~9 行は冷却塔と冷凍機の電力消費量の合算値を求める関数である。この関数を用いて 10 行目で 20 ~ 32°C の範囲で極小値を探

†1　極小値の探索という例題を成立させるために、本問では外気温度が 13°C 程度と低く、冷凍機効率もかなり低い機種を想定した。多くの場合、冷凍機の圧縮機動力は冷却塔よりもはるかに大きいため、限界まで冷却塔負荷を高めることで低い冷却水温度を追求した方が、全体としては消費電力量が小さくなる。即ち、通常は冷凍機と冷却塔の合算動力は例題のように極小値を持たず、左上がりの曲線となる。

索している。プログラムを実行すると解として 21.9°C が得られる。

プログラム 2.20　電力消費量の極小値探索プログラム

```
 1 private static void GoldenSectionTest()
 2 {
 3   Minimization.MinimizeFunction mFnc = delegate(double wTemp)
 4   {
 5     double pct = 3.4773e2 + wTemp * (-6.4390e1 + wTemp * (4.775 + wTemp *
 6       (-1.768e-1 + wTemp * (3.2651e-3 + wTemp * (-2.4048e-5)))));
 7     double ptb = 4.1472e-1 + wTemp * (5.1299e-3 + wTemp * (4.1126e-4));
 8     return pct * 15 + ptb * 70;
 9   };
10   Console.WriteLine(Minimization.GoldenSection(20, 32, mFnc).ToString("F1"));
11 }
```

2.4.2　多変数関数の最適化

多変数関数の極小値探索法としては、準ニュートン法、滑降シンプレックス法[2.4)]、共役勾配法[2.5)]、 Powell の方法[2.6)]、Levenberg-Marquardt 法[2.11)]などがある。近年、建築分野においても多くの応用研究が行われている遺伝的アルゴリズム[2.7)]も多変数関数の極小値探索手法の 1 つである。本書では準ニュートン法について解説する[2.20)]。

1）準ニュートン法

非線形かつ多変数な関数の場合には、確実に解にたどり着けることが保証された手法は存在しない。最適化すべき関数は波打ち、複数の極小値が点在しているため、反復計算で得られる解は最適解ではなく極小値に迷い込んだ準最適解である可能性が高い。1 変数の場合であれば、2.4.1 節に示したように解の範囲を囲い込むという作戦がとれるが、多次元空間ではこの方法は取れない（滑降シンプレックス法は多次元空間に多胞体を作り出して解を囲い込む方法であり、方法としては類似しているが解の一意性は保証されない）。従って、まずは、そもそも計算対象を多変数非線形関数の最適化という問題に持ち込まないということが第一に求められる工夫である。

2.3 節で解説したニュートン・ラフソン法と同様に、準ニュートン法も適当な初期値ベクトル $\boldsymbol{x}$（本節の太字変数はすべてベクトルであることに注意）から開始し、これを更新していくことで解にたどり着く方法である。k 番目の計算ステップにおいて、ベクトル $\boldsymbol{x}_k$ の近傍で 2 階のテイラー展開を行うと式 2.40 が得られる。ただし $\boldsymbol{F}(\boldsymbol{x}_k)$はヘッシアン（またはヘッセ行列）と呼ばれる行列であり、式 2.41 で定義される。

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)(\boldsymbol{x}-\boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_k)^T \boldsymbol{F}(\boldsymbol{x}_k)(\boldsymbol{x}-\boldsymbol{x}_k) \tag{2.40}$$

$$\boldsymbol{F}(\boldsymbol{x}_k) = \frac{\partial^2 f(\boldsymbol{x}_k)}{\partial x_i \partial x_j} \tag{2.41}$$

式 2.40 の右辺を最小化するためには式 2.42 でベクトル $\boldsymbol{x}$ を更新すれば良い。この方法はニュートン法と呼ばれる。

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k = -[\boldsymbol{F}(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)^T \tag{2.42}$$

しかし式 2.41 に示されるように数値微分によるヘッシアンの計算や式 2.42 を適用するための逆行列の計算は負荷が大きい。そこで、計算負荷を減らすためにヘッシアンの逆行列 $H = F^{-1}$ を直接に近似する行列の系列を作成する手法が提案されており、これらを準ニュートン法と呼ぶ。近似方法はいくつもがあるが BFGS 法（Broyden, Fletcher, Goldfarb and Shanno）は最もよく用いられる方法の 1 つである。BFGS 法のヘッシアンの逆行列 H の更新式は式 2.43~2.47で表される[2.22)]。

$$\boldsymbol{F}^{-1}\approx\boldsymbol{H}_{k+1}=\left(\boldsymbol{H}_k-\frac{\boldsymbol{H}_k\boldsymbol{q}_k\boldsymbol{q}_k^T\boldsymbol{H}_k}{\boldsymbol{q}_k^T\boldsymbol{H}_k\boldsymbol{q}_k}+\boldsymbol{v}_k\boldsymbol{v}_k^T\right)\gamma_k-\frac{\boldsymbol{p}_k\boldsymbol{p}_k^T}{\boldsymbol{p}_k^T\boldsymbol{q}_k} \tag{2.43}$$

$$\boldsymbol{p}_k=\boldsymbol{x}_{k+1}-\boldsymbol{x}_k \tag{2.44}$$

$$\boldsymbol{q}_k=\nabla f(\boldsymbol{x}_{k+1})^T-\nabla f(\boldsymbol{x}_k)^T \tag{2.45}$$

$$\boldsymbol{v}_k=\left(\boldsymbol{q}_k^T\boldsymbol{H}\,\boldsymbol{q}_k\right)^{1/2}\left(\frac{\boldsymbol{p}_k}{\boldsymbol{p}_k^T\boldsymbol{q}_k}-\frac{\boldsymbol{H}_k\boldsymbol{q}_k}{\boldsymbol{q}_k^T\boldsymbol{H}_k\boldsymbol{q}_k}\right) \tag{2.46}$$

$$\gamma_k=\frac{\boldsymbol{p}_k^T\boldsymbol{q}_k}{\boldsymbol{q}_k^T\boldsymbol{H}_k\boldsymbol{q}_k} \tag{2.47}$$

ヘッシアンの逆行列 H が得られれば式 2.42 を適用してベクトル $\boldsymbol{x}$ を更新できる。ただし、式 2.40 は 2 階のテーラー展開による近似に過ぎないため、式 2.48 に示すように更新式を定義し、直線探索を行うことでパラメータ α を最適化する。幸いなことに準ニュートン法によって得られる行列 H の系列はそれぞれ共役勾配方向にあるため、式 2.48 による修正は各回で独立に行って良い。

$$\phi(\alpha)=f(\boldsymbol{x}_k+\alpha(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k)) \tag{2.48}$$

極小値の直線探索には逆 2 次補間を用いる（2.4.1 節で解説した黄金探索でも良いが速度で劣る）。逆 2 次補間の概念を図 2.7 に示す。今、3 つの点 α_A、α_B、α_C とそれぞれの点での評価関数の評価値 $\phi(\alpha_A)$、$\phi(\alpha_B)$、$\phi(\alpha_C)$が与えられているとする。この時、3 点を貫く 2 次近似式の極小値は α*で生じ、その値は式 2.49 で計算できる。新たに推定された α*と α_A および α_B を用いて式 2.49 を再度適用する。この操作を繰り返すことで極小値を絞り込む。なお、プログラム全体での収束を保証するためには、直線探索の停止条件を式 2.50 に示す Goldstein のルールに従わせる必要がある。

$$\alpha^*=0.5\frac{(\alpha_C^2-\alpha_B^2)\phi(\alpha_A)+(\alpha_A^2-\alpha_C^2)\phi(\alpha_B)+(\alpha_B^2-\alpha_A^2)\phi(\alpha_C)}{(\alpha_C-\alpha_B)\phi(\alpha_A)+(\alpha_A-\alpha_C)\phi(\alpha_B)+(\alpha_B-\alpha_A)\phi(\alpha_C)} \tag{2.49}$$

$$\delta\le\frac{\phi(\alpha)}{\alpha\nabla f(\boldsymbol{x}_k)(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k)}\le 1-\delta \tag{2.50}$$

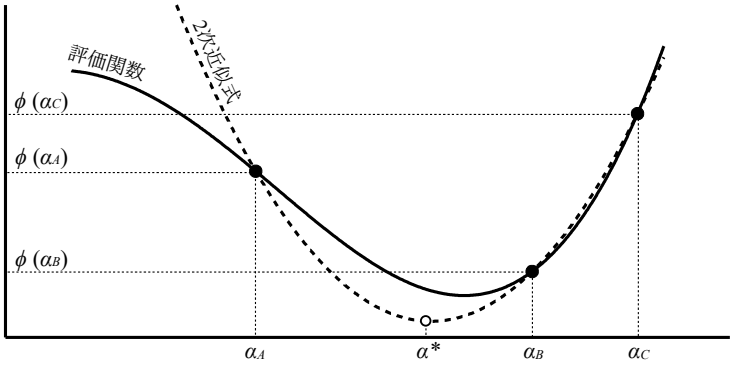


図 2.7　逆 2 次補間による極小値探索

準ニュートン法を実行するための MultiMinimization クラスを作成する。プログラム 2.21 に最小化する評価関数の定義と、これを用いた数値微分（$\nabla f(\boldsymbol{x})^T$の計算）の計算処理を示す。

プログラム 2.21　準ニュートン法の評価関数定義と数値微分処理

Popolo.Numerics.MultiMinimization class

```
1 /// <summary>最小化する関数</summary>
2 /// <param name="vecX">入力ベクトル</param>
3 /// <param name="iter">反復回数</param>
```

```
 4 /// <returns>評価値</returns>
 5 public delegate double MinimizeFunction(IVector vecX, int iter);
 6
 7 /// <summary>数値微分を計算する</summary>
 8 /// <param name="fx">計算を行う点での関数評価値</param>
 9 /// <param name="vecX">入力値ベクトル</param>
10 /// <param name="dif">出力：微分ベクトル</param>
11 /// <param name="mFnc">評価関数</param>
12 /// <param name="iter">反復回数</param>
13 private static void getDiff(double fx, ref IVector vecX, ref IVector dif, MinimizeFunction mFnc, int iter)
14 {
15   int num = vecX.Length;
16   for (int i = 0; i < num; i++)
17   {
18     double tmp = vecX[i];
19     double dx = Math.Abs(tmp) * 1e-7;
20     if (dx < 1e-8) dx = 1e-8;
21     vecX[i] += dx;
22     dif[i] = (mFnc(vecX, iter) - fx) / dx;
23     vecX[i] = tmp;
24   }
25 }
```

プログラム 2.22 に直線探索処理を示す。1~12 行は式 2.48 の実装である。第 1 引数は修正係数 α、第 2 引数は現在の $\boldsymbol{x}$ ベクトル、第 3 引数は方向ベクトル（$\boldsymbol{x}_{k+1}$-$\boldsymbol{x}_k$）である。15~157 行が直線探索処理である。式 2.49 に従って 101~110 行で α*を計算して逆 2 次補間を行う。

プログラム 2.22　直線探索処理（逆 2 次補間）

Popolo.Numerics. MultiMinimization class

```
 1 /// <summary>極小値探索用評価関数</summary>
 2 /// <param name="alpha">探索幅</param>
 3 /// <param name="vecX">入力ベクトル</param>
 4 /// <param name="dir">方向ベクトル</param>
 5 /// <param name="mFnc">評価関数</param>
 6 /// <param name="iter">反復回数</param>
 7 /// <returns>極小値</returns>
 8 private static double optFnc(double alpha, IVector vecX, IVector dir, MinimizeFunction mFnc, int iter)
 9 {
10   IVector vecX2 = new Vector(vecX.Length);
11   for (int i = 0; i < vecX.Length; i++) vecX2[i] = vecX[i] + dir[i] * alpha;
12   return mFnc(vecX2, iter);
13 }
14
15 /// <summary>直線探索を行う</summary>
16 /// <param name="vecX">入力値ベクトル</param>
17 /// <param name="dir">方向ベクトル</param>
18 /// <param name="dif">微分ベクトル</param>
19 /// <param name="mFnc">評価関数</param>
20 /// <param name="iteration">反復回数</param>
21 /// <param name="alpha">出力：直線探索幅</param>
22 /// <param name="fvecX">出力：極小値</param>
23 private static void lineSearch(IVector vecX, IVector dir, IVector dif, MinimizeFunction mFnc, int iteration,
24   out double alpha, out double fvecX)
25 {
26   const double RHO = 0.4;
27   int num = dir.Length;
28
29   //初期ステップ計算
30   alpha = 0;
31   for (int i = 0; i < num; i++) alpha += dir[i] * dir[i];
32   alpha = 1.0 / (3.0 * Math.Sqrt(alpha));
33
34   //Armijo-Goldstein 基準
35   double agL = 0;
36   for (int i = 0; i < num; i++) agL += dir[i] * dif[i];
37   double agR = agL * RHO;
38   agL *= (1 - RHO);
39
40   //alpha=0 を a とする
41   double fval0, fvalA;
42   double dstpA = 0;
43   fval0 = fvalA = optFnc(0, vecX, dir, mFnc, iteration);
44
45   //b,c 点を計算
46   double fvalC, dstpC;
47   double dstpB = alpha;
48   double fvalB = optFnc(dstpB, vecX, dir, mFnc, iteration);
```

```
 49   if (fvalA < fvalB)
 50   {
 51     dstpC = dstpB;
 52     fvalC = fvalB;
 53     int iter = 0;
 54     while (true)
 55     {
 56       dstpB /= 4.0;
 57       fvalB = optFnc(dstpB, vecX, dir, mFnc, iteration);
 58       if (fvalB < fvalA) break;
 59       iter++;
 60       if (100 < iter)
 71       {
 72         alpha = dstpA;
 73         fvecX = fvalA;
 74         return;
 75       }
 76     }
 77   }
 78   else
 79   {
 80     dstpC = dstpB * 2;
 81     fvalC = optFnc(dstpC, vecX, dir, mFnc, iteration);
 82     int iter = 0;
 83     while (fvalC < fvalB)
 84     {
 85       dstpA = dstpB;
 86       fvalA = fvalB;
 87       dstpB = dstpC;
 88       fvalB = fvalC;
 89       dstpC = dstpB * 2;
 90       fvalC = optFnc(dstpC, vecX, dir, mFnc, iteration);
 91       iter++;
 92       if (100 < iter) throw new Exception("QuasiNewton.lineSearch: iteration error");
 93     }
 94   }
 95
 96   //逆2次補間ループ
 97   int witer = 0;
 98   while (true)
 99   {
100     double bf1 = (dstpC - dstpB) * fvalA;
101     double bf2 = (dstpA - dstpC) * fvalB;
102     double bf3 = (dstpB - dstpA) * fvalC;
103     double denom = bf1 + bf2 + bf3;
104     if (denom < 1e-10) alpha = 0.5 * (dstpA + dstpC);
105     else
106     {
107       double numen = (dstpC + dstpB) * bf1 + (dstpA + dstpC) * bf2 + (dstpB + dstpA) * bf3;
108       alpha = 0.5 * numen / denom;
109     }
110     //範囲外に飛び出した場合は終了
111     if (alpha < dstpA || dstpC < alpha)
112     {
113       alpha = dstpB;
114       fvecX = fvalB;
115       return;
116     }
117
118     //収束判定
119     fvecX = optFnc(alpha, vecX, dir, mFnc, iteration);
120     if (fvecX <= fval0 + agR * alpha && fval0 + agL * alpha < fvecX) return;
121
122     //3点を更新
123     if (alpha < dstpB)
124     {
125       if (fvecX < fvalB)
126       {
127         dstpC = dstpB;
128         fvalC = fvalB;
129         dstpB = alpha;
130         fvalB = fvecX;
131       }
132       else
133       {
134         dstpA = alpha;
135         fvalA = fvecX;
136       }
137     }
138     else
```

```
139     {
140       if (fvecX < fvalB)
141       {
142         dstpA = dstpB;
143         fvalA = fvalB;
144         dstpB = alpha;
145         fvalB = fvecX;
146       }
147       else
148       {
149         dstpC = alpha;
150         fvalC = fvecX;
151       }
152     }
153     witer++;
154     if (100 < witer) return;
155   }
156 }
157
```

プログラム 2.23 に準ニュートン法による極小値探索処理を示す。38~46 行はヘッシアン逆行列 H の初期化処理であり、単位行列とする。1）計算開始時、2）直線探索失敗時、3）勾配ベクトルと方向集合の内積が負となる場合、4）反復計算が 2×変数の数の倍数に達した場合、に初期化を行う。48~54 行で方向集合を更新して 75 行で直線探索を行う。極小値に落ち込み、直線探索が失敗した場合には 77~99 行に示すように要素別に探索を行うが、それぞれの要素に分けても探索が失敗する場合には計算を打ち切る（86 行）。105~110 行で入力ベクトルを更新し、112~129 行で収束判定を行う。1）入力値の変化率、2）関数評価値の変化率、3）微分値の絶対値、それぞれが引数の値未満となれば収束したとする。反復計算を継続する場合には式 2.43~2.47を用いて 132~148 行でヘッシアンの逆行列を更新する。

プログラム 2.23　準ニュートン法による極小値探索処理

Popolo.Numerics. MultiMinimization class

```
  1 /// <summary>準ニュートン法で極小値を探索する</summary>
  2 /// <param name="vecX">初期値ベクトル</param>
  3 /// <param name="mFnc">探索する関数</param>
  4 /// <param name="maxIteration">最大反復回数</param>
  5 /// <param name="rErrXVal">入力変化率の収束条件</param>
  6 /// <param name="rErrFVal">関数評価値変化率の収束条件</param>
  7 /// <param name="rErrDel">微分値の収束条件</param>
  8 /// <param name="iteration">出力:反復回数</param>
  9 /// <returns>探索成功の真偽</returns>
 10 public static bool QuasiNewton
 11   (ref IVector vecX, MinimizeFunction mFnc, int maxIteration,
 12   double rErrXVal, double rErrFVal, double rErrDel, out int iteration)
 13 {
 14   iteration = 1;
 15   int num = vecX.Length;
 16   int nDir = 0;
 17   int locNum = 0;
 18   IVector dif = new Vector(num);
 19   double[] dif2 = new double[num];
 20   IVector dir = new Vector(num);
 21   IVector dir2 = new Vector(num);
 22   double[] qk = new double[num];
 23   double[] pk = new double[num];
 24   double[] qhk = new double[num];
 25   double[] hqk = new double[num];
 26   double[,] hINV = new double[num, num];
 27
 28   //初期化処理
 29   double fx = mFnc(vecX, iteration);
 30   getDiff(fx, ref vecX, ref dif, mFnc, iteration);
 31   for (int i = 0; i < num; i++) dir[i] = -dif[i];
 32   bool needInit = true;
 33
 34   while (true)
 35   {
 36     if (maxIteration < iteration) return false;
 37
```

```
38      if (needInit || (iteration % (2 * num) == 0))
39      {
40        for (int i = 0; i < num; i++)
41        {
42          for (int j = 0; j < num; j++) hINV[i, j] = 0;
43          hINV[i, i] = 1.0;
44        }
45        needInit = false;
46      }
47
48      //方向集合を更新
49      for (int i = 0; i < num; i++)
50      {
51        double bf = 0;
52        for (int j = 0; j < num; j++) bf += hINV[i, j] * dif[j];
53        dir[i] = -bf;
54      }
55
56      //勾配ベクトルと方向集合の内積が負の場合には初期化
57      double wk = 0;
58      for (int i = 0; i < num; i++) wk += dir[i] * dif[i];
59      if (0 <= wk) needInit = true;
60      else
71      {
72        //直線探索
73        double alpha = 0;
74        double fx2;
75        lineSearch(vecX, dir, dif, mFnc, iteration, out alpha, out fx2);
76
77        //探索失敗:要素別に探索
78        if (alpha == 0 && fx2 != 0)
79        {
80          if (dif[nDir] == 0) dir2[nDir] = (iteration % 2 == 1) ? 1 : -1;
81          else dir2[nDir] = -dif[nDir];
82          lineSearch(vecX, dir2, dif, mFnc, iteration, out alpha, out fx2);
83          if (alpha == 0)
84          {
85            locNum++;
86            if (locNum == num) return false;
87          }
88          else
89          {
90            locNum = 0;
91            vecX[nDir] += dir2[nDir] * alpha;
92            fx = fx2;
93            getDiff(fx, ref vecX, ref dif, mFnc, iteration);
94          }
95          dir2[nDir] = 0;
96          nDir++;
97          if (num == nDir) nDir = 0;
98          needInit = true;
99        }
100
101       //探索成功:入力値更新
102       else
103       {
104         locNum = 0;
105         for (int i = 0; i < num; i++)
106         {
107           pk[i] = dir[i] * alpha;
108           vecX[i] += pk[i];
109           dif2[i] = dif[i];
110         }
111
112         //収束判定
113         getDiff(fx2, ref vecX, ref dif, mFnc, iteration);
114         for (int i = 0; i < num; i++) qk[i] = dif[i] - dif2[i];
115         double delt = Math.Abs(fx2 - fx);
116         double mean = 0.5 * (Math.Abs(fx2) + Math.Abs(fx));
117         if (mean * rErrFVal < delt || delt < 1e-10)
118         {
119           double maxDX = 0;
120           double maxDF = 0;
121           for (int i = 0; i < num; i++)
122           {
123             double bf = Math.Abs(pk[i]);
124             if (rErrXVal < Math.Abs(vecX[i])) bf /= Math.Abs(vecX[i]);
125             maxDX = Math.Max(maxDX, bf);
126             maxDF = Math.Max(maxDF, Math.Abs(dif[i]));
127           }
```

```
128           if ((maxDF < rErrDel) && (maxDX < rErrXVal)) return true;
129         }
130         fx = fx2;
131
132         //ヘッシアンの逆行列近似を更新
133         double qhqC = 0;
134         double pqC = 0;
135         for (int i = 0; i < num; i++)
136         {
137           hqk[i] = 0;
138           for (int j = 0; j < num; j++) hqk[i] += hINV[i, j] * qk[j];
139           qhqC += hqk[i] * qk[i];
140           pqC += pk[i] * qk[i];
141         }
142         double pqInv = 1.0 / pqC;
143         double qhqInv = 1.0 / qhqC;
144         double gamma = pqC * qhqInv;
145         for (int i = 0; i < num; i++)
146           for (int j = 0; j < num; j++)
147             hINV[i, j] = (hINV[i, j] - hqk[i] * hqk[j] * qhqInv + qhqC * (pk[i] * pqInv - hqk[i] * qhqInv)
148               * (pk[j] * pqInv - hqk[j] * qhqInv)) * gamma + pk[i] * pk[j] * pqInv;
149       }
150     }
151     iteration++;
152   }
153 }
```

2）制約つき最適化問題

熱環境シミュレーションにおいて多変数関数の最適化を行う代表的な例は、温度や流量などの制御量を調整してエネルギー消費量を最小化する場面である。数値計算上は温度も流量も任意の実数をとることができるが、現実のシステムでは、通常はある一定の範囲内でしか値を制御できない。例えば熱源の冷水流量を制御する場合、計算モデル上はいくらでも流量を絞ることができるかもしれないが、現実の機器は下限流量が定められており、これを下回ると機器の故障などにつながる。従って、現実の問題を解く際には、単純に状態変数を最適化するだけではなく、いくつかの条件を満足することが前提となることが多い。このような問題を「制約つき最適化問題」と呼ぶ[2.8)]。

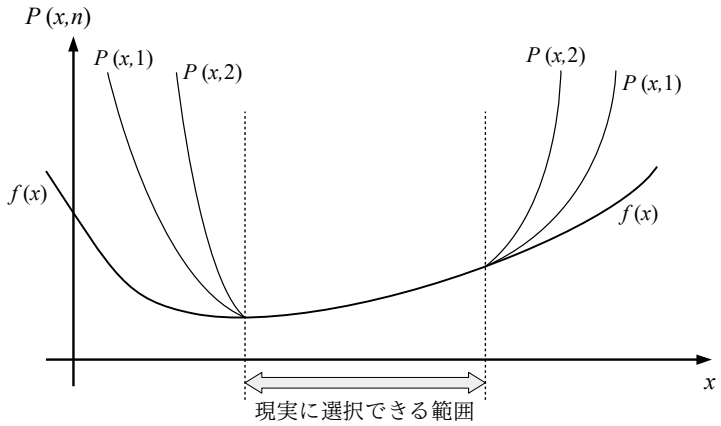


図 2.8　1 変数の場合のペナルティ関数

ペナルティ関数法は、制約条件のある最も簡単な制約つき最適化問題の解法である[†1)]。選択可能な状態値から外れるに従って無限大に発散する関数を作成し、これを最適化関数に加えた関数をペナルティ関数と定義する。このペナルティ関数に対して準ニュートン法などの一般の最適化手法を適用する。例えば $g_i(\boldsymbol{x}) < 0$ と $h_j(\boldsymbol{x}) = 0$ という制約条件がある場合に、ペナルティ関数 $P(x,n)$の例として式 2.51 が定義できる。α、β はそれぞれパラメータであり、制約条件の絶対値や性質に応じて設定する。また、ρ_n は n 回の反復計算の過程で徐々に大きくしていく。1 変数の場合のペナルティ関数の概念を図 2.8 に示す。基本となる関数 $f(x)$にペナルティを加え、反復計算が進むにつれて境界外のペナルティ関

†1　本例は外部ペナルティ関数法と呼ばれる。この他に、内部ペナルティ関数法、乗数法、逐次 2 次計画法、内点法などがある。

数の評価値を増大させることで、解が境界外で収束することを回避する。ペナルティ関数の定義の詳細は第 21 章の例題で具体的な制約つき最適化問題を挙げて解説する。

$$P(\boldsymbol{x},n)=f(\boldsymbol{x})+\rho_n\left\{\sum_{i=0}^{l}(max(0,g_i(\boldsymbol{x})))^{\alpha}+\sum_{j=0}^{m}\left|h_j(\boldsymbol{x})^{\beta}\right|\right\} \tag{2.51}$$

2.5 補間

例えば外気データが 1 時間間隔で計測されている一方で、計算は 30 分間隔で実行したい場合や、BEMS の計測エラーで異常値がある場合などには、前後のデータを頼りに、間に挟まれるデータの値を予測する必要が生じる。このような予測処理を「補間」と呼ぶ。これに対して、ある範囲のデータが与えられた上で、その外部の範囲に対して行う予測処理を「補外」と呼ぶ。補間は「内挿」、補外は「外挿」とも呼ばれる。以下、代表的な補間処理である三次スプライン補間について説明する。

図 2.9 に三次スプライン補間の概念を示す。離散的な N 個のデータの組、$(x_0, y_0), (x_1, y_1), (x_2, y_2)...(x_N, y_N)$が与えられた場合に、これらのデータの間の値を予測するため、N-1 個の補間関数 $S_n(x)$をモデル化する。ここで $S_n(x)$は式 2.52 で表される x の三次式である。

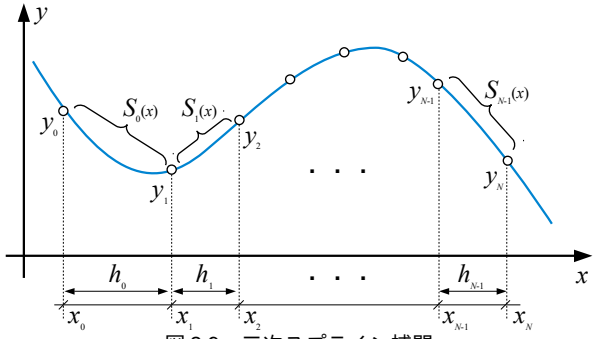


図 2.9　三次スプライン補間

$$S_n(x)=a_n+b_n(x-x_n)+c_n(x-x_n)^2+d_n(x-x_n)^3 \tag{2.52}$$

それぞれの補間関数 $S_n(x)$は端部で隣接する補間関数 $S_{n-1}(x)$と $S_{n+1}(x)$に接続する。この接続が滑らかに行われるように、下記の 1~4 の条件をつける。

1）$S_n(x_n)=y_n$　　左端でデータ点を通る

2）$S_n(x_{n+1})=S_{n+1}(x_{n+1})=y_{n+1}$　右端でデータ点を通る

3）$S'_n(x_{n+1})=S'_{n+1}(x_{n+1})$　　接続点で隣接補間関数と微分値が一致する

4）$S''_n(x_{n+1})=S''_{n+1}(x_{n+1})$　　接続点で隣接補間関数と二階微分値が一致する

式 2.52 のパラメータは a_n~d_n の 4 個であり、補間関数は全部で N-1 個あるため、パラメータの数は 4N-4 個である。一方で 1）と 2）でそれぞれ N-1 個、3）と 4）でそれぞれ N-2 個の条件式がたてられるため、全部で 4N-6 個の条件式が得られる。パラメータを確定するためには条件が 2 つ不足するため、両端での 2 階微分値が 0 になるという条件（$S''_0(x_0)=S''_{N-1}(x_{N-1})=0$）を加える。このような条件付けを行ったスプライン補間を特に「自然スプライン」と呼ぶ

上記条件にもとづいて a_n~d_n を求めると式 2.53~2.58が得られる。c_n は式 2.57 の解である。ただし c_0=0 である。係数行列である式 2.58 は帯行列であるため、Thomas Algorithm を用いて高速に解くこと

ができる。

$$a_n = y_n \tag{2.53}$$

$$b_n = \frac{a_{n+1} - a_n}{h_n} - \frac{h_n(c_{n+1} + 2c_n)}{3} \tag{2.54}$$

$$d_n = \frac{c_{n+1} - c_n}{3h_n} \tag{2.55}$$

$$h_n = x_{n+1} - x_n \tag{2.56}$$

$$\mathrm{H}\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} 3\left(\dfrac{a_2 - a_1}{h_1} - \dfrac{a_1 - a_0}{h_0}\right) \\ 3\left(\dfrac{a_3 - a_2}{h_2} - \dfrac{a_2 - a_1}{h_1}\right) \\ \vdots \\ 3\left(\dfrac{a_N - a_{N-1}}{h_{N-1}} - \dfrac{a_{N-1} - a_{N-2}}{h_{N-2}}\right) \end{bmatrix} \tag{2.57}$$

$$\mathrm{H} = \begin{bmatrix} 2(h_0 + h_1) & h_1 & & & & 0 \\ h_1 & 2(h_1 + h_2) & h_2 & & & \\ & h_2 & 2(h_2 + h_3) & h_3 & & \\ & & \ddots & \ddots & & \\ & & h_{N-3} & 2(h_{N-3} + h_{N-2}) & h_{N-2} & \\ 0 & & & h_{N-2} & 2(h_{N-2} + h_{N-1}) & \end{bmatrix} \tag{2.58}$$

プログラム 2.24 に係数 c_n を計算する処理を示す。7~17 行で式 2.57 を作成し、18 行で行列を解く。係数 c_n のみを計算しているが、c_n さえ得られれば 2.53~2.55 を用いて a_n, b_n, d_n は順次求められる。

プログラム 2.24　三次スプライン補間の係数計算処理

Popolo.Numerics.CubicSpline class

```
 1 /// <summary>三次スプライン補間のための係数を計算する</summary>
 2 /// <param name="x">X</param>
 3 /// <param name="y">Y</param>
 4 /// <returns>三次スプライン補間のための係数</returns>
 5 public static double[] GetParameters(double[] x, double[] y)
 6 {
 7   double[] h = new double[y.Length - 1];
 8   double[] a = new double[y.Length - 2];
 9   double[,] hm = new double[3, y.Length - 2];
10   for (int i = 0; i < y.Length - 1; i++) h[i] = x[i + 1] - x[i];
11   for (int i = 0; i < y.Length - 2; i++)
12   {
13     if (i != 0) hm[0, i] = h[i];
14     hm[1, i] = 2 * (h[i] + h[i + 1]);
15     if (i != y.Length - 3) hm[2, i] = h[i + 1];
16     a[i] = 3 * ((y[i + 2] - y[i + 1]) / h[i + 1] - (y[i + 1] - y[i]) / h[i]);
17   }
18   LinearAlgebra.SolveTridiagonalMatrix(hm, ref a);
19   double[] c = new double[y.Length];
20   a.CopyTo(c, 1);
21   c[0] = c[c.Length - 1] = 0;
22   return c;
23 }
```

c_n を用いて補間処理を行うプログラムを 2.25 に示す。44~71 行は a_n, b_n, d_n を計算して式 2.52 を適用する処理である。引数として補間式の番号（$S_n(x)$ の n の値）が与えられているが実際にはこの番号を求める処理も必要である。24~42 行のプログラムは二分法を用いて補間式の番号を求める処理である。実際には x の値は順序良く並ぶことが多いと予想できるが、このような場合には 1~22 行の処理

を用いる。12~20 行で頭から順番に補間の範囲を探索する。

プログラム **2.25**　三次スプライン補間処理

Popolo.Numerics.CubicSpline class

```
/// <summary>補間処理を行う</summary>
/// <param name="x">X</param>
/// <param name="y">Y</param>
/// <param name="c">係数</param>
/// <param name="x2">補完処理を行う位置</param>
/// <returns>補間値</returns>
/// <remarks>x2 が順序よく並ぶことを想定した処理</remarks>
public static double[] Interpolate(double[] x, double[] y, double[] c, double[] x2)
{
  double[] y2 = new double[x2.Length];
  int num = 0;
  for (int i = 0; i < x2.Length; i++)
  {
    //範囲確認
    if (x2[i] < x[0] || x[x.Length - 1] < x2[i])
      throw new Exception("CubicSpline: out of range");
    //補間範囲を特定して補間処理実行
    while (x[num + 1] < x2[i]) num++;
    y2[i] = interPolate(x, y, c, x2[i], num);
  }
  return y2;
}

/// <summary>補間処理を行う</summary>
/// <param name="x">X</param>
/// <param name="y">Y</param>
/// <param name="c">係数</param>
/// <param name="x2">補完処理を行う位置</param>
/// <returns>補間値</returns>
public static double Interpolate(double[] x, double[] y, double[] c, double x2)
{
  int low = 0;
  int high = x.Length - 1;
  //二分法で補間範囲を求める
  while (1 < high - low)
  {
    int mid = (low + high) >> 1;
    if (x2 < x[mid]) high = mid;
    else low = mid;
  }
  return interPolate(x, y, c, x2, low);
}

/// <summary>x2 の位置での補間値を計算する</summary>
/// <param name="x">X</param>
/// <param name="y">Y</param>
/// <param name="cf">係数</param>
/// <param name="x2">補間を行う位置</param>
/// <param name="num">x2 の補間式の番号</param>
/// <returns>補間値</returns>
private static double interPolate
  (double[] x, double[] y, double[] cf, double x2, int num)
{
  double dx = x2 - x[num];
  double h = x[num + 1] - x[num];
  double a = y[num];
  double b = (y[num + 1] - y[num]) / h - h * (cf[num + 1] + 2 * cf[num]) / 3d;
  double c = cf[num];
  double d = (cf[num + 1] - cf[num]) / (3d * h);
  return a + dx * (b + dx * (c + dx * d));
}
```

【例題 2.4】

BEMS から表 2.1 に示すような 1 時間間隔の二次側負荷データが入手できた。これをもとに 15 分間隔のシミュレーションの境界条件を得るために、三次スプライン補間を適用せよ。

表 2.1　二次側負荷データ

時刻	負荷	時刻	負荷	時刻	負荷	時刻	負荷
1:00	0	7:00	0	13:00	219	19:00	91
2:00	0	8:00	42	14:00	217	20:00	9
3:00	0	9:00	224	15:00	210	21:00	9
4:00	0	10:00	215	16:00	250	22:00	0
5:00	0	11:00	210	17:00	210	23:00	0
6:00	0	12:00	217	18:00	217	24:00	0

【解】

プログラム 2.26 に負荷データの補間処理を示す。3~71 行で表 2.1 の実測値を配列に入れ、データが循環するように 25 時間目に 1 時のデータを追加する。8 行で補間のための係数を計算し、12 行で 15 分間隔の補間値を算出する。負荷データであり 0 未満の値はとらないため、16 行で 0 以上に補正する。計算結果をグラフ化すると図 2.10 となる。実測データを通る滑らかな補間ができていることが確認できる。

プログラム 2.26　負荷データの補間処理

```
 1 private static void SplineTest()
 2 {
 3   double[] x = new double[25];
 4   for (int i = 0; i < x.Length; i++) x[i] = i;
 5   double[] y = new double[]
 6   { 0, 0, 0, 0, 0, 0, 0, 42, 224, 215, 210, 217,
 7     219, 217, 210, 250, 210, 217, 91, 9, 9, 0, 0, 0, 0 };
 8   double[] c = CubicSpline.GetParameters(x, y);
 9
10   double[] x2 = new double[24 * 4];
11   for (int i = 0; i < x2.Length; i++) x2[i] = 0.25 * i;
12   double[] y2 = CubicSpline.Interpolate(x, y, c, x2);
13
14   for (int i = 0; i < x2.Length; i++)
15   {
16     y2[i] = Math.Max(0, y2[i]); //負荷データのため 0 以上とする
17     Console.WriteLine(x2[i].ToString("F2") + "," + y2[i].ToString("F2"));
18   }
19 }
```

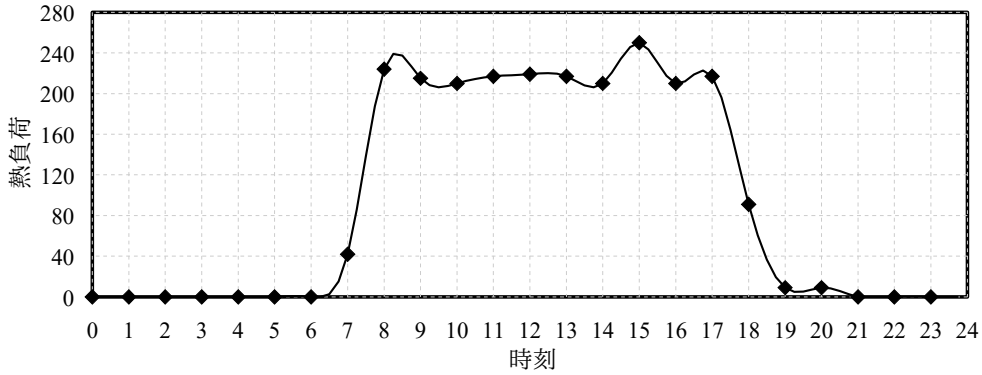


図 2.10　負荷データの補間結果

2.6　乱数列の作成

気象現象や人間行動などのモデル化を行う際には、その不確実性を表現したい場合がある。コンピュータは基本的には確定的な計算しかできないが、一定の計算処理を行うことで、あたかも不確実に見える数列を作ることができる。このような不確実な数列を乱数列と呼ぶ。本節では建築設備のシミュレーションにおいて重要性の高い一様乱数と正規乱数の計算法について述べる。

2.6.1　一様乱数

ある範囲内で全ての実数が同確率で出現するような乱数を一様乱数と呼ぶ。一様乱数は後述の正規乱数を含め、各種の確率分布に従う乱数を生成するために必要となる基礎的な乱数である。従って一様乱数を生成する関数は多くのプログラム言語において標準的に備えられている†1)が、言語によっては質の悪い（周期性が短い、出現に偏りがある）標準関数の場合があるため、モンテカルロ法の適用などの際には自分で実装することが望ましい。松本らによるメルセンヌ・ツイスター C 原語版[2.10)]を C#に移植したプログラムを 2.27 に示す。メルセンヌ・ツイスターは周期が 2^{19937} と極めて長いという長所がある。

†1　C#言語の場合には Donald E. Knuth のアルゴリズム[2.9)]を実装した Random クラスが定義されている。

プログラム 2.27　メルセンヌ・ツイスターによる一様乱数製造機

Popolo.Numerics.MersenneTwister class

```
/// <summary>メルセンヌ・ツイスターによる擬似乱数製造機</summary>
/// <remarks>
/// Makoto Matsumoto 氏の C 言語版を C#に移植
/// http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/CODES/mt19937ar.c
/// </remarks>
public class MersenneTwister
{
  private const int N = 624;
  private const int M = 397;
  private const uint MATRIX_A = 0x9908b0dfU;
  private const uint UPPER_MASK = 0x80000000U;
  private const uint LOWER_MASK = 0x7fffffffU;

  private UInt32[] mt = new UInt32[N];
  private int mti;
  private UInt32[] mag01 = new UInt32[] { 0x0U, MATRIX_A };

  /// <summary>コンストラクタ</summary>
  /// <param name="seed">乱数シード</param>
  public MersenneTwister(int seed)
  {
    mt[0] = (UInt32)(seed & 0xffffffffU);
    for (mti = 1; mti < N; mti++)
    {
      mt[mti] = (UInt32)(1812433253U * (mt[mti - 1] ^ (mt[mti - 1] >> 30)) + mti);
      mt[mti] &= 0xffffffffU;
    }
  }

  /// <summary>符号なし 32bit の擬似乱数を計算する</summary>
  /// <returns>符号なし 32bit の擬似乱数</returns>
  private UInt32 nextUInt32()
  {
    UInt32 y;
    if (mti >= N)
    {
      int kk;
      for (kk = 0; kk < N - M; kk++)
      {
        y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
        mt[kk] = mt[kk + M] ^ (y >> 1) ^ mag01[y & 0x1UL];
      }
      for (; kk < N - 1; kk++)
      {
        y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
        mt[kk] = mt[kk + (M - N)] ^ (y >> 1) ^ mag01[y & 0x1UL];
      }
      y = (mt[N - 1] & UPPER_MASK) | (mt[0] & LOWER_MASK);
      mt[N - 1] = mt[M - 1] ^ (y >> 1) ^ mag01[y & 0x1UL];

      mti = 0;
    }

    y = mt[mti++];

    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680U;
    y ^= (y << 15) & 0xefc60000U;
    y ^= (y >> 18);

    return y;
  }

  /// <summary>[0,1]の一様乱数列を計算する</summary>
  /// <returns>[0,1]の一様乱数列</returns>
  public double NextDouble()
  {
    return nextUInt32() * (1.0 / 4294967295.0);
  }
}
```

2.6.2　正規乱数

不確実な事象を扱う際には正規分布を仮定することが多く、正規分布に従う乱数である正規乱数の計算が必要となる場面は多い。正規乱数は一様乱数を用いて Box-Muller 法によって下記のステップで計算できる。

1. [-1,1]の範囲で 2 つの一様乱数 u, v を発生させる

2. $r^2 = u^2 + v^2$ を計算する

3. $1 < r$ の場合には 1.に戻る

4. $w=(-2 \log(r) / r)^{0.5}$ を計算し、$x = wu, y=wv$ とする

5. x および y は平均 0、分散 1 の正規分布となる

Box-Muller 法による正規乱数製造処理をプログラム 2.28 に示す。39~57 行が上記の処理である。一度の処理で 2 つの正規乱数が生成されるため、次回の呼び出しに備えて片方の乱数をストックする。

プログラム 2.28　Box-Muller 法による正規乱数製造処理

Popolo.Numerics.NormalRandom class

```
 1 /// <summary>正規分布に従う乱数系列を作成するクラス</summary>
 2 public class NormalRandom
 3 {
 4
 5   /// <summary>一様乱数製造</summary>
 6   private MersenneTwister rnd;
 7
 8   /// <summary>乱数ストック</summary>
 9   private double rndStock;
10
11   /// <summary>乱数ストックを持つか否か</summary>
12   private bool hasStock = false;
13
14   /// <summary>コンストラクタ</summary>
15   /// <param name="seed">乱数シード</param>
16   public NormalRandom(int seed)
17   {
18     rnd = new MersenneTwister(seed);
19   }
20
21   /// <summary>平均 0 分散 1 に従う正規乱数系列を取得する</summary>
22   /// <returns>平均 0 分散 1 に従う正規乱数</returns>
23   public double NextDouble()
24   {
25     if (hasStock)
26     {
27       hasStock = false;
28       return rndStock;
29     }
30     else
31     {
32       hasStock = true;
33       double rtn;
34       makeNormalRandomNumbers(rnd, out rtn, out rndStock);
35       return rtn;
36     }
37   }
38
39   /// <summary>BoxMuller 法で平均 0 分散 1 の正規乱数を計算する</summary>
40   /// <param name="rnd">一様乱数製造機</param>
41   /// <param name="nrnd1">正規乱数 1</param>
42   /// <param name="nrnd2">正規乱数 2</param>
43   private static void makeNormalRandomNumbers (MersenneTwister rnd, out double nrnd1, out double nrnd2)
44   {
45     double v = rnd.NextDouble() * 2 - 1;
46     double u = rnd.NextDouble() * 2 - 1;
47     double r2 = v * v + u * u;
48     while (1 < r2)
49     {
50       v = rnd.NextDouble() * 2 - 1;
51       u = rnd.NextDouble() * 2 - 1;
52       r2 = v * v + u * u;
53     }
54     double w = Math.Sqrt(-2.0 * Math.Log(r2) / r2);
55     nrnd1 = w * v;
56     nrnd2 = w * u;
57   }
58 }
```

【第２章 参考文献】

2.1） William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling : Numerical Recipes in C, 株式会社技術評論社, 1993

2.2） Richard P. Brent : Algorithms for Minimization without Derivatives, Dover Publications, Inc., 1973

2.3） 平瀬創也: C#で学ぶ偏微分方程式の数値解法 CAE プログラミング入門, 東京電機大学出版局, 2009

2.4） J. A. Nelder and R. Mead: A simplex method for function minimization, The Computer Journal, Vol.7 (4), pp.308-313, 1965

2.5） Hestenes, M.R., Stiefel, E.: Methods of Conjugate Gradients for Solving Linear Systems, Journal of Research of the National Bureau of Standards, Vol.49 No.6, pp.409-436, National Bureau of Standards, 1952

2.6） M. J. D. Powell: An efficient method for finding the minimum of a function of several variables without calculating derivatives, The Computer Journal, Vol.7, No.2, pp.155-162, 1964

2.7） John, H. Holland: Adaptation in Natural and Artificial Systems, MIT Press, 1992

2.8） 穴井宏和: 数理最適化の実践ガイド, 講談社, 2013

2.9） Donald E.Knuth: The Art of Computer Programming -Seminumerical algorithms-

2.10） M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 (1998)

2.11） Jorge J. More: The Levenberg-Marquardt Algorithm, Implementation and Theory, Conference on Numerical Analysis University of Dundee, Scotland, June28 – July 1, 1977

2.12） 小国力: 新数値計算法, 株式会社 サイエンス社, 初版, 1997

2.13） 平岡和幸, 堀玄: プログラミングのための線形代数, オーム社, 初版, 2004

2.14） 戸川隼人: マトリクスの数値計算, オーム社, 初版, 1971

2.15） 山本有作: 疎行列連立一次方程式の直接解法, 日本計算工学会 (JSCES), Vol.11, No.4, pp.1458-1462, 2006

2.16） 渡部善隆: 連立一次方程式の基礎知識 ~および Gauss の消去法の安定性について~, 数値解析チュートリアル 2004 資料, 2004

2.17） The Netlib repository, LAPACK, MINPACK, http://www.netlib.org, (accessed 2016-03-20)

2.18） GNU Scientific Library, http://www.gnu.org/software/gsl, (accessed 2016-03-20)

2.19） US. National Security Agency Photo Gallery, http://www.nsa.gov, (accessed 2016-03-20)

2.20） 大野豊, 磯野和男: 新版 数値計算ハンドブック, 第９章 最適化問題

2.21） 大野豊, 磯野和男: 新版 数値計算ハンドブック, 第６章 非線形方程式

2.22） David G. Luenberger, Yinyu Ye: Linear and Nonlinear Programming Third Edition, International series in operations research and management science, Springer, pp.285-317, 2009

第3章　水の物性
(Thermodynamic Properties of Water)

3.1　概要

水は空調分野の熱媒としては最も基本的な物質である。多くの熱源空調システムでは冷水や温水を用いて液体の状態で熱を搬送する。また、明治期の初期の暖房方式は、液体である温水よりも気体である蒸気で熱を搬送するシステムの方が主流であり、このような方式は、現在においても地域冷暖房施設や病院などで採用されることがある。この他には、配管内の圧力を上げることで 100 ºC を超える温水を搬送する高温水暖房[3.1)]や、固体として水を利用する氷蓄熱システムがある。

液体としての水を熱搬送に用いるシステムは、水の顕熱のみを利用するシステムであり、温度が重要な状態値となる。一般的には比熱は一定（=4.186 kJ/(kg·K)）とみなすため、温度変化に伴う熱移動の計算は容易である。一方、蒸気を熱搬送に用いるシステムでは水の潜熱も利用することになるため、熱量をエンタルピーで捉える必要がある。この場合には、蒸発潜熱の計算が必須となる。また、冷却コイルにおける冷却除湿、空調機における加湿、冷却塔における蒸発冷却などを評価する際にも、水の状態変化に伴うエネルギーを知る必要がある。

水および蒸気の物性値に関する国際的な基準としては、IAPWS 実用国際状態式 1997 が存在し、これにもとづく蒸気表が機械学会から刊行されている[3.2)]。IAPWS による方法は精度と規範性は高いが、数十の近似係数を用いる手法であるため、自用のプログラム作成を目的とする場合にはやや煩雑である。そこで本書では、飽和状態の計算には IAPWS の手法を示し、その他の状態値を計算する方法としては、Thomas らによる簡易な近似式[3.3)]を紹介する。

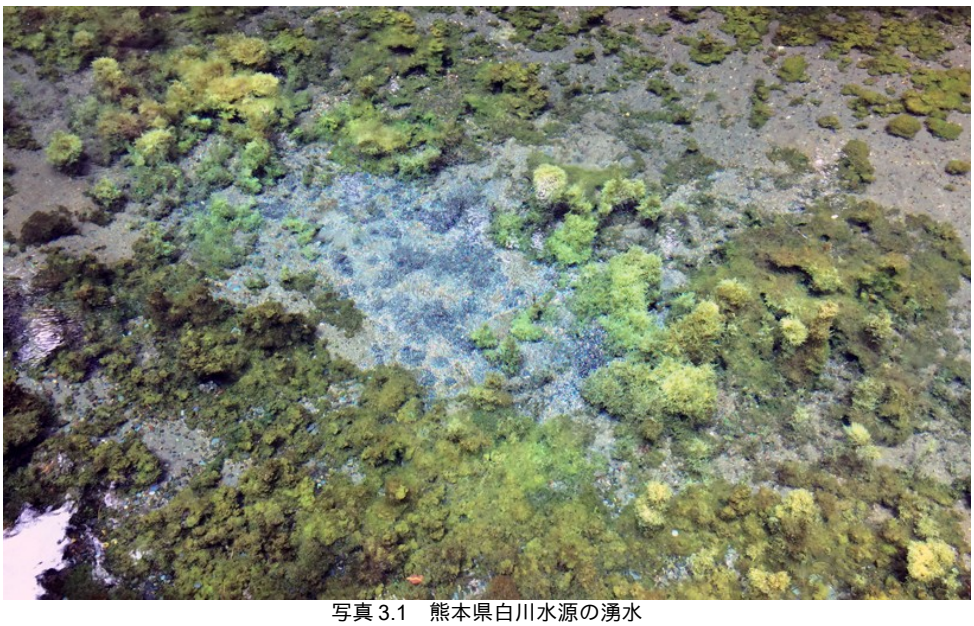


写真 3.1　熊本県白川水源の湧水

3.2 理論

3.2.1 飽和水蒸気圧と飽和温度の関係

IAPWS-IF97（IAPWS Indesutrial Formulation 1997 for the Thermodynamic Properties of Water and Steam）によれば、水および蒸気の物性把握の基礎となる飽和水蒸気圧 P_{Ws} [kPa]と飽和温度 T_{Ws} [K]の関係は式3.1~式3.3で表現できる。

$$\beta^2\alpha^2+n_1\beta^2\alpha+n_2\beta^2+n_3\beta\alpha^2+n_4\beta\alpha+n_5\beta+n_6\alpha^2+n_7\alpha+n_8=0 \tag{3.1}$$

$$\beta=P_{Ws}^{(1/4)} \tag{3.2}$$

$$\alpha=T_{Ws}+\frac{n_9}{T_{Ws}-n_{10}} \tag{3.3}$$

$n_1 = 0.11670521452767 \times 10^4$　　$n_2 = -0.72421316703206 \times 10^6$
$n_3 = -0.17073846940092 \times 10^2$　　$n_4 = 0.12020824702470 \times 10^5$
$n_5 = -0.32325550322333 \times 10^7$　　$n_6 = 0.14915108613530 \times 10^2$
$n_7 = -0.4823265731591 \times 10^4$　　$n_8 = 0.40511340542057 \times 10^6$
$n_9 = -0.23855557567849$　　$n_{10} = 0.65017534844798 \times 10^3$

式3.1は P_{Ws} と T_s について解くことができ、各々、式3.4と式3.5となる。

$$P_{Ws}=\left[\frac{2C}{-B+(B^2-4AC)^{1/2}}\right]^4 \tag{3.4}$$

$$T_{Ws}=\frac{n_{10}+D-\left[(n_{10}+D)^2-4(n_9+n_{10}D)\right]^{1/2}}{2} \tag{3.5}$$

ただし、A, B, C, D は式3.6~式3.12である。

$$A=\alpha^2+n_1\alpha+n_2 \tag{3.6}$$

$$B=n_3\alpha^2+n_4\alpha+n_5 \tag{3.7}$$

$$C=n_6\alpha^2+n_7\alpha+n_8 \tag{3.8}$$

$$D=\frac{2G}{-F-(F^2-4EG)^{1/2}} \tag{3.9}$$

$$E=\beta^2+n_3\beta+n_6 \tag{3.10}$$

$$F=n_1\beta^2+n_4\beta+n_7 \tag{3.11}$$

$$G=n_2\beta^2+n_5\beta+n_8 \tag{3.12}$$

式3.1の適用範囲は273.15 Kから647.096 K（臨界温度）までであり、氷点下に関しては別の近似式を使用する必要がある。飽和水蒸気圧 P_{Ws} と飽和温度 T_{Ws} の関係について、建築設備分野においては、式3.13で示されるWexler-Hylandの式がよく知られている[3.9) †1)]。

$$\ln(P_{Ws})=\begin{cases} C_1/T_{Ws}+C_2+C_3T_{Ws}+C_4T_{Ws}^2+C_5T_{Ws}^3+C_6T_{Ws}^4+C_7\ln T_{Ws} & (-100<T_{Ws}<0) \\ C_8/T_{Ws}+C_9+C_{10}T_{Ws}+C_{11}T_{Ws}^2+C_{12}T_{Ws}^3+C_{13}\ln T_{Ws} & (0<T_{Ws}<200) \end{cases} \tag{3.13}$$

$C_1 = -5.674359 \times 10^3$　　$C_2 = 6.3925247$　　$C_3 = -9.677843 \times 10^3$
$C_4 = 6.2215701 \times 10^{-7}$　　$C_5 = 2.0747825 \times 10^{-9}$　　$C_6 = -9.4840240 \times 10^{-13}$
$C_7 = 4.1635019$　　$C_8 = -5.8002206 \times 10^3$　　$C_9 = 1.3914993$
$C_{10} = -4.8640239 \times 10^{-2}$　　$C_{11} = 4.1764768 \times 10^{-5}$　　$C_{12} = -1.4452093 \times 10^{-8}$

†1 少し年配の先生方と話すと、古くはGoff-Gratchの式が用いられており、ある時期からWexler-Hylandの式を使い始めたと言う。本書ではIAPWS-IF97による計算法を取り上げたが、設備シミュレーションに求められる精度から言えば、どれを使っても大差無いように思う。どちらかと言えば設備シミュレーション好き同士が会った時の酒の肴としての価値の方が高い。

$C_{13} = 6.5459673$

飽和水蒸気圧 P_{Ws} から飽和温度 T_{Ws} を求める場合には、式 3.13 を T_{Ws} について解くことになるが、代数的には解けないため、ニュートン・ラフソン法などを使って数値計算的に求める必要がある。しかし、関数形状が単純であるため、近似式を用いても十分な精度を持つことができる。飽和水蒸気圧 P_{Ws} から飽和温度 T_{Ws} を求めるための近似式を式 3.14 と式 3.15 に示す[3.5)]。

$$T_{Ws} = \sum_{n=0}^{3} D_n Y^n \tag{3.14}$$

$$Y = \ln(1000 P_{Ws}) \tag{3.15}$$

ただし、

$0.0039 < P_{Ws} < 0.6112$（-50 °C ~ 0 °C）のとき、

$D_0 = -6.0662 \times 10$　　$D_1 = 7.4624$　　$D_2 = 2.0594 \times 10^{-1}$　　$D_3 = 1.6321 \times 10^{-2}$

$0.6112 < P_{Ws} < 12.350$（0 °C ~ 50 °C）のとき、

$D_0 = -7.7199 \times 10$　　$D_1 = 1.3198 \times 10$　$D_2 = 6.3772 \times 10^{-1}$　　$D_3 = 7.1098 \times 10^{-2}$

である。

3.2.2　飽和蒸気の物性

飽和水の比エンタルピー h_{Wsl} [kJ/kg]、比体積 υ_{Wsl} [m³/kg]、比エントロピー s_{Wsl} [kJ/kg]、飽和蒸気の比エンタルピー h_{Wsv} [kJ/kg]、比体積 υ_{Wsv} [m³/kg]、比エントロピー s_{Wsv} [kJ/kg]は式 3.16 で近似する[3.3)]。ただし、Y_{Ws} は各々の物性値を表している。また、T_{Wr} は式 3.17 で計算する。

$$Y_{Ws} = A + B T_{Wr}^{1/3} + C T_{Wr}^{5/6} + D T_{Wr}^{7/8} + \sum_{n=1}^{7} E_n T_{Wr}^{n} \tag{3.16}$$

$$T_{Wr} = \frac{T_{CP} - T_{Ws}}{T_{CP}} \tag{3.17}$$

式 3.16 の近似係数 A, B, C, D, E_n は各々、表 3.1 の通りである。物性値を直接に近似せず、臨界点や三重点での物性値との比率を近似対象としている。変数名称と物性値との対応を以下に記す。

T_{CP}　：臨界温度　：647.096 K

h_{CP}　：臨界点での比エンタルピー　：2099.3 kJ/kg

υ_{CP}　：臨界点での比体積　：0.003155 m³/kg

s_{CP}　：臨界点での比エントロピー　：4.4289 kJ/(kg·K)

P_{CP}　：臨界点での飽和水蒸気圧　：22089 kPa

γ_{TP}　：三重点での蒸発潜熱　：2500.9 kJ/kg

飽和蒸気の比体積に関しては、三重点から臨界点までの変化量が非常に大きく、近似性能を高めるために、飽和水蒸気圧と比体積の積である $P_{Ws}\,\upsilon_{Wsv}$ を近似の対象としている。即ち、一旦、$P_{Ws}\,\upsilon_{Wsv}$ を近似した後に、式 3.4 によって求めた P_{Ws} で除することで υ_{Wsv} を算出する。

表 3.1　飽和蒸気の物性値近似係数

	蒸発潜熱	飽和水の比エンタルピー		
Y_{Ws}	γ_{Ws} / γ_{TP}	h_{Wsl} / h_{CP}		
適用範囲	$273.16 \leq T_{Ws} \leq 647.3$	$273.16 \leq T_{Ws} \leq 300$	$300 \leq T_{Ws} \leq 600$	$600 \leq T_{Ws} \leq 647.3$
A	0	0	8.839230108e-1	1.0
B	7.79221E-1	0	0	-4.41057805E-1
C	4.62668	0	0	-5.52255517
D	-1.07931	0	0	6.43994847
E_1	-3.87446	6.24698837E+2	-2.67172935	-1.64578795
E_2	2.94553	-2.34385369E+3	6.22640035	-1.30574143
E_3	-8.06395	-9.50812101E+3	-1.31789573E+1	0
E_4	1.15633e1	7.16287928E+4	-1.91322436	0
E_5	-6.02884	-1.63535221E+5	6.87937653E+1	0
E_6	0	1.66531093E+5	-1.24819906E+2	0
E_7	0	-6.47854585E+4	7.21435404E+1	0

	飽和蒸気の 比エンタルピー	飽和水の比体積	飽和蒸気の比体積	飽和蒸気の 比エントロピー
Y_{Ws}	h_{Wsv} / h_{CP}	υ_{Wsl} / υ_{CP}	$P_{Ws}\,\upsilon_{Wsv}$ / $P_{CP}\,\upsilon_{CP}$	s_{Wsv} / s_{CP}
適用範囲	$273.16 \leq T_{Ws} \leq 647.3$	$273.16 \leq T_{Ws} \leq 647.3$	$273.16 \leq T_{Ws} \leq 647.3$	$273.16 \leq T_{Ws} \leq 647.3$
A	1.0	1.0	1.0	1.0
B	4.57874342E-1	-1.9153882	1.6351057	3.77391E-1
C	5.08441288	1.2015186E+1	5.2584599E+1	-2.78368
D	-1.48513244	-7.8464025	-4.4694653E+1	6.93135
E_1	-4.81351884	-3.888614	-8.9751114	-4.34839
E_2	2.69411792	2.0582238	-4.3845530E-1	1.34672
E_3	-7.39064542	-2.0829991	-1.9179576E+1	1.75261
E_4	1.04961689E+1	8.2180004E-1	3.6765319E+1	-6.22295
E_5	-5.46840036	4.7549742E-1	-1.9462437E+1	9.99004
E_6	0	0	0	0
E_7	0	0	0	0

	飽和水の比エントロピー		
Y_{Ws}	s_{Wsl} / s_{CP}		
適用範囲	$273.16 \leq T_{Ws} \leq 300$	$300 \leq T_{Ws} \leq 600$	$600 \leq T_{Ws} \leq 647.3$
A	0	9.12762917E-1	1.0
B	0	0	-3.24817650E-1
C	0	0	-2.990556709
D	0	0	3.2341900
E_1	-1.83692956E+3	-1.75702956	-6.78067859E-1
E_2	1.47066352E+4	1.68754095	-1.91910364
E_3	-4.31466046E+4	5.82215341	0
E_4	4.86066733E+4	-6.33354786E+1	0
E_5	7.9975096E+3	1.88076546E+2	0
E_6	-5.83339887E+4	-2.52344531E+2	0
E_7	3.31400718E+4	1.28058531E+2	0

3.2.3　液相の物性

建築設備を計算対象とする場合、定圧比熱や密度などの水の物性は一定値として扱うことが多い。しかし、例えば、冷温水コイルの詳細計算を行うために配管内表面における対流熱伝達率などを把握したい場合などには、水の物性値の正確な計算が求められることもある。水の密度 ρ_W [kg/m^3]、定圧比熱 c_{pW} [kJ/(kg·K)]、熱伝導率 λ_W [W/(m·K)]、粘性係数 μ_W [Pa·s]は温度の関数として近似が可能である。式 3.18 と式 3.19 に近似式を示す。

・密度 ρ_W、定圧比熱 c_{pW}、熱伝導率 λ_W

$$Y_W = \sum_{n=0}^{3} \alpha_n T_{Wr}^n \tag{3.18}$$

・粘性係数 μ_W

$$\mu_W = \exp\left(\sum_{n=0}^{4} \alpha_n T_{Wr}^n\right) \times 10^{-6} \tag{3.19}$$

式 3.18 および式 3.19 の近似係数 α_n は表 3.2 の通りである。いずれも大気圧（=101.325 kPa）での近似式であり、適用可能範囲は 0 ºC~100 ºC である。Wagner らによる Helmholtz 状態方程式による計算[3.10]と比較した結果、いずれの物性に関しても最大誤差率 0.3 %未満、平均誤差率 0.1 %未満の近似性能である†1)。式 3.18 および式 3.19 を用いて液相の各種物性を計算した結果を図 3.1 に示す。

表 3.2　水の物性値近似係数

	α_0	α_1	α_2	α_3	α_4
密度	9.8811040E+2	-1.3273604E+3	4.7162295E+3	-4.1245328E+3	-
定圧比熱	1.0570130	2.1952960E+1	-4.9895501E+1	3.6963413E+1	-
熱伝導率	-1.3734399E-1	4.2128755	-5.9412196	1.2794890	-
粘性係数	5.2136906E+1	-4.0910405E+2	1.3270844E+3	-1.9089622E+3	1.0489917E+3

動粘性係数 v_W [m²/s]および熱拡散率 a_W [m²/s]は上記の物性値から式 3.20 と式 3.21 で計算ができる。

・動粘性係数 v_W

$$v_W = \mu_W / \rho_W \tag{3.20}$$

・熱拡散率 a_W

$$a_W = \lambda_W / (1000\, c_{pW}\, \rho_W) \tag{3.21}$$

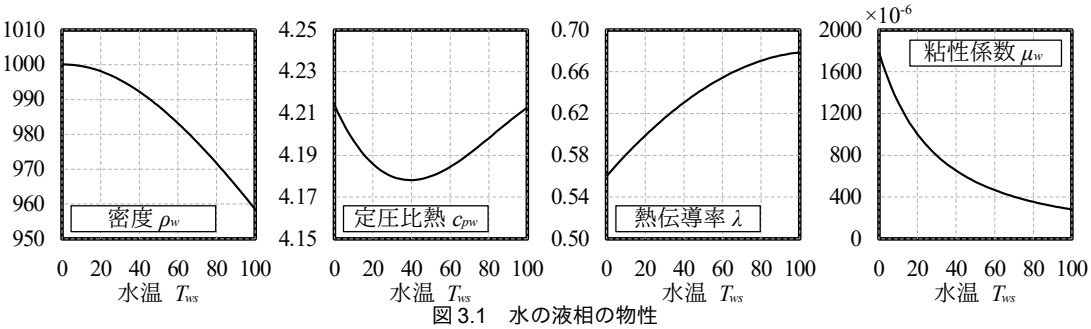


図 3.1　水の液相の物性

【例題 3.1】

下記条件で円管内に冷水が流れている。円管内表面の対流熱伝達率を計算せよ。

・円管の内径:　105.3 mm（配管用炭素鋼管 100A を想定）

・水量:　1,000 L/min　　・冷水の温度:　7 °C

【解】

円管内の乱流の対流熱伝達率は、Dittus-Boelter の実験式により、式 3.22 で表現できる[3.6) 3.7)]。ここで、N_u（ヌセルト数）、R_e（レイノルズ数）、P_r（プラントル数）は無次元数であり、それぞれ、式 3.23~式 3.25 で計算できる。α、λ、v、a は、流体の対流熱伝達率[W/(m²·K)]、熱伝導率[W/(m·K)]、動粘性係数[m²/s]、熱拡散率[m²/s]である。また、u は管内流速[m/s]、d は代表長さ[m]であり、本問の場合には管の直径である。従って、これらの式を α について解けば良い。

$$N_u = 0.023\, R_e^{0.8}\, P_r^{0.4} \tag{3.22}$$

$$N_u = \alpha\, d / \lambda \tag{3.23}$$

$$R_e = u\, d / v \tag{3.24}$$

$$P_r = v / a \tag{3.25}$$

†1　米国 NIST の物性値計算プログラム REFPROP version 7 により計算を行った。

まず T_r は、

$T_r = (647.096 - (7 + 273.15)) / 647.096 = 0.567$

となる。熱伝導率、定圧比熱、密度は式 3.18 に T_r を代入して、

$\lambda_W = -1.3734399E\text{-}1 + 4.2128755 \times 0.567 - 5.9412196 \times 0.567^2 + 1.2794890 \times 0.567^3$

$= 0.574$ W/(m·K)

$c_{pW} = 1.0570130 + 2.1952960E1 \times 0.567 - 4.9895501E1 \times 0.567^2 + 3.6963413E1 \times 0.567^3$

$= 4.201$ kJ/(kg·K)

$\rho_W = 9.8811040e2 - 1.3273604E3 \times 0.567 + 4.7162295E3 \times 0.567^2 - 4.1245328E3 \times 0.567^3$

$= 999.878$ kg/m^3

となる。粘性係数は式 3.19 に T_r を代入して、

$\mu_W = \exp(5.2136906E1 - 4.0910405E2 \times 0.567 + 1.3270844E3 \times 0.567^2$

$- 1.9089622E3 \times 0.567^3 + 1.0489917E3 \times 0.567^4)$

$= 1428.234 \times 10^{-6}$

となる。動粘性係数は式 3.20 に μ_W と ρ_W を代入して、

$\nu_W = 1428.234 \times 10^{-6} \div 999.878 = 1.4284 \times 10^{-6}$ m^2/s

となる。熱拡散率は式 3.21 に λ_W と c_{pW} を代入して、

$a_W = 0.574 \div (1000 \times 4.201 \times 999.878) = 0.1367 \times 10^{-6}$ m^2/s

となる。以上により、水の物性値が求まった。

管内流速は水量を管内面積で除することで求まり、

$u = 1000$ L/min $\div 60 \div 1000 \div ((0.1053 \div 2)^2 \times 3.14)$

$= 1.915$ m/s

となる。以上の結果から、レイノルズ数とプラントル数はそれぞれ、

$R_e = 1.915 \times 0.1053 \div (1.4284 \times 10^{-6})$

$= 0.1412 \times 10^6$

$P_r = (1.4284 \times 10^{-6}) \div (0.1367 \times 10^{-6})$

$= 10.45$

となる。従って、ヌセルト数は、

$N_u = 0.023 \times (0.1412 \times 10^6)^{0.8} \times 10.45^{0.4}$

$=782.17$

となる。式 3.23 にヌセルト数、熱伝導率、代表長さを代入すれば、

$a = 782.17 \times 0.574 \div 0.1053$

$= 4263.7$ W/(m^2·K)

となり、対流熱伝達率が得られる。

3.2.4　不凍液の物性

氷蓄熱設備を導入したシステムでは、0 °C 以下の温度で蓄熱を行うため、熱媒として冷水のかわりに不凍液（ブライン）を用いる。過去には塩化カルシウムブラインが用いられることもあったが、腐食性が高いため、現在ではグリコール系のブラインが主流である。グリコール系ブラインはエチレングリコール系とプロピレングリコール系に大別できるが、空調用途としてはエチレングリコール系を用いることが通常である。

ブラインは水とは異なった物性値を持ち、主にはブラインの温度とブラインの濃度に依存する。エチレングリコールに関しては、比熱 c_{peg} [kJ/(kg·K)]、密度 ρ_{eg} [kg/m^3]、熱伝導率 λ_{eg} [W/(m·K)]を濃度と温度の関数として式 3.26 で近似できる。また、粘性係数 μ_{eg} [Pa·s]は式 3.27 で近似できる。ただし、t_{egr} と w_{egr} は $t_{egr} = t_{eg} \div 60$、$w_{egr} = w_{eg} \div 40$ で正規化された温度と濃度であり、t_{eg} はブラインの温度[°C]、w_{eg} はブラインの体積濃度 [%]である。

$$Y_{eg} = \sum_{n=0}^{2} \sum_{m=0}^{2} \alpha_{eg,nm} w_{egr}^{n} \cdot t_{egr}^{m} \tag{3.26}$$

$$\mu_{eg}=\exp\left(\sum_{n=0}^{3}\sum_{m=0}^{3}\alpha_{eg\mu,nm}\,w_{egr}^{n}\cdot t_{egr}^{m}\right) \tag{3.27}$$

式 3.26 と式 3.27 の近似係数を表 3.3 に示す。近似式の適用範囲は t_{eg} <= 60 °C、10 % <= w_{eg} <= 40 % である。ASHRAE Handbook に記載の物性値表を基準に取ると[3.8]、粘性係数に関しては最大誤差率 1.1 %、平均誤差率 0.23 %、その他の物性値に関しては最大誤差率 0.13 %、平均誤差率 0.06 %で近似が可能である。

粘性係数は、エチレングリコールの濃度が上がるにつれて上昇するが、粘性係数が高いと配管抵抗が大きくなり搬送動力が増加する。また、例題 3.1 からわかるように、対流熱伝達率も低下するため、熱媒としての熱交換性能が低下する。従ってブラインの濃度は、凝結が起こらない範囲内で出来るだけ下げることが望ましい。氷蓄熱は通常-10 °C 前後で蓄熱を行うことが多いため、結果として、エチレングリコールの濃度は 30 %程度とすることが多い。なお、市販のブラインは純粋なエチレングリコールではなく、防錆や粘性係数の低減を目的とした添加剤を加えた水溶液であり、エチレングリコール濃度は 70~80 %程度である。従って、エチレングリコールの濃度を計算する際には、水に製品を混合した割合ではなく、この割合に更に製品自体のエチレングリコール濃度を乗じる必要があることに注意する。例えば 1 L の水に対して、あるブラインを 1 L の割合で混合した場合に、このブラインのエチレングリコール濃度が 75 %であれば、全体のエチレングリコール濃度は 50%×75%=38 %となる。式 3.26 と式 3.27 を適用する際には、この濃度を入力条件とする。

また、上記の通り、粘性係数の低減を目的とした添加剤の影響により、市販のブラインは純粋なエチレングリコールとはやや異なった物性を示す。従って、この点について精度の高い計算を行いたい場合にはメーカーから技術資料を入手して、改めて近似式を作成する方が良いだろう。

表 3.3　エチレングリコールの物性値近似係数

近似係数	α_{eg00}	α_{eg01}	α_{eg02}	α_{eg10}	α_{eg11}	α_{eg12}
	α_{eg20}	α_{eg21}	α_{eg22}	-	-	-
密度	1.000730E+3	1.142397E+1	-8.934849	7.384266E+1	-7.081561	6.265692E-1
	-7.758854	8.151257E-2	-4.658341E-1	-	-	-
定圧比熱	4.097664	6.490881E-2	4.601884E-3	-6.219971E-1	1.582082E-1	-1.508748E-2
	-7.478621E-2	-2.207541E-2	1.042091E-2	-	-	-
熱伝導率	5.582415E-1	1.285585E-1	-3.461736E-2	-1.977931E-1	-7.126944E-2	1.892756E-2
	3.423204E-2	8.921311E-3	-1.266517E-3	-	-	-
近似係数	$\alpha_{eg\mu00}$	$\alpha_{eg\mu01}$	$\alpha_{eg\mu02}$	$\alpha_{eg\mu03}$	$\alpha_{eg\mu10}$	$\alpha_{eg\mu11}$
	$\alpha_{eg\mu12}$	$\alpha_{eg\mu13}$	$\alpha_{eg\mu20}$	$\alpha_{eg\mu21}$	$\alpha_{eg\mu22}$	$\alpha_{eg\mu23}$
	$\alpha_{eg\mu30}$	$\alpha_{eg\mu31}$	$\alpha_{eg\mu32}$	$\alpha_{eg\mu33}$	-	-
粘性係数	2.250552E-1	-9.869133E-1	-8.577001E-1	6.022116E-1	2.400300	-5.167587
	1.040298E+1	-5.244681	-1.687822	7.744326	1.792170E+1	9.265506
	8.281442E-1	-4.035175	9.698964	-5.115606	-	-

3.3 計算法

プログラム 3.1 に水の物性値計算のための定数宣言を示す。

プログラム 3.1 水の物性計算の定数宣言

Popolo.ThermophysicalProperty.Water class

```
/// <summary>絶対温度と摂氏との変換定数</summary>
private const double CONVERT_C_TO_K = 273.15;

/// <summary>臨界点での比エンタルピー[kJ/kg]</summary>
private const double TEMPERATURE_AT_CRITICAL_POINT = 647.096;

/// <summary>臨界点での比エンタルピー[kJ/kg]</summary>
private const double ENTHALPY_AT_CRITICAL_POINT = 2099.3;

/// <summary>臨界点での比体積[m3/kg]</summary>
private const double SPECIFIC_VOLUME_AT_CRITICAL_POINT = 0.003155;

/// <summary>臨界点での比エントロピー[kJ/kgK]</summary>
private const double ENTROPY_AT_CRITICAL_POINT = 4.4289;

/// <summary>臨界点での飽和水蒸気圧[kPa]</summary>
private const double PRESSURE_AT_CRITICAL_POINT = 22089;

/// <summary>三重点での蒸発潜熱[kJ/kg]</summary>
private const double VAPORIZATION_HEAT_AT_TRIPLE_POINT = 2500.9;
```

3.3.1 飽和水蒸気圧と飽和温度の計算

プログラム 3.2 に飽和温度と飽和水蒸気圧の計算処理を示す。式 3.1~3.15にもとづく計算である。

プログラム 3.2 水の飽和温度および飽和水蒸気圧の計算

Popolo.ThermophysicalProperty.Water class

```
/// <summary>飽和温度[℃]から飽和水蒸気圧[kPa]を求める</summary>
/// <param name="saturationTemperature">飽和温度[℃]</param>
/// <returns>飽和水蒸気圧[kPa]</returns>
public static double GetSaturationPressure(double saturationTemperature)
{
  const double P_CONVERT = 0.001d;

  const double C1 = -5.6745359e3d;
  const double C2 = 6.3925247d;
  const double C3 = -9.6778430e-3d;
  const double C4 = 6.2215701e-7d;
  const double C5 = 2.0747825e-9d;
  const double C6 = -9.4840240e-13d;
  const double C7 = 4.1635019d;

  const double N1 = 0.11670521452767e4d;
  const double N2 = -0.72421316703206e6d;
  const double N3 = -0.17073846940092e2d;
  const double N4 = 0.12020824702470e5d;
  const double N5 = -0.32325550322333e7d;
  const double N6 = 0.14915108613530e2d;
  const double N7 = -0.4823265731591e4d;
  const double N8 = 0.40511340542057e6d;
  const double N9 = -0.23855557567849e0d;
  const double N10 = 0.65017534844798e3d;

  double ts = saturationTemperature + CONVERT_C_TO_K;

  //-100~0.01C//三重点以下は wexler-hyland の式
  if (saturationTemperature < 0.01)
    return Math.Exp(C1 / ts + C2 + C3 * ts + C4 * Math.Pow(ts, 2)
      + C5 * Math.Pow(ts, 3) + C6 * Math.Pow(ts, 4) + C7 * Math.Log(ts)) * P_CONVERT;
  //~647.096K//臨界温度までは IAPWS-IF97 実用国際状態式
  else
  {
    double alpha = ts + N9 / (ts - N10);
    double a2 = alpha * alpha;
    double A = a2 + N1 * alpha + N2;
    double B = N3 * a2 + N4 * alpha + N5;
    double C = N6 * a2 + N7 * alpha + N8;
    return Math.Pow(2 * C / (-B + Math.Pow(B * B - 4 * A * C, 0.5)), 4) / P_CONVERT;
  }
}
```

```
44
45 /// <summary>飽和水蒸気圧[kPa]から飽和温度[C]を求める</summary>
46 /// <param name="saturationPressure">飽和水蒸気圧[kPa]</param>
47 /// <returns>飽和温度[C]</returns>
48 public static double GetSaturationTemperature(double saturationPressure)
49 {
50   const double P_CONVERT = 0.001d;
51
52   const double D1 = -6.0662e1d;
53   const double D2 = 7.4624e0d;
54   const double D3 = 2.0594e-1d;
55   const double D4 = 1.6321e-2d;
56
57   const double N1 = 0.116705214527767e4d;
58   const double N2 = -0.72421316703206e6d;
59   const double N3 = -0.17073846940092e2d;
60   const double N4 = 0.12020824702470e5d;
61   const double N5 = -0.32325550322333e7d;
62   const double N6 = 0.14915108613530e2d;
63   const double N7 = -0.4823265731591e4d;
64   const double N8 = 0.40511340542057e6d;
65   const double N9 = -0.23855557567849e0d;
66   const double N10 = 0.65017534844798e3d;
67
68   //~0C//wexler-hyland の計算値を近似した式
69   if (saturationPressure < 0.611213)
70   {
71     double y = Math.Log(saturationPressure / P_CONVERT);
72     return D1 + y * (D2 + y * (D3 + y * D4));
73   }
74   //0C~//臨界圧力までは IAPWS-IF97 実用国際状態式
75   else
76   {
77     double ps = saturationPressure * P_CONVERT;
78     double beta = Math.Pow(ps, 0.25);
79     double b2 = beta * beta;
80     double E = b2 + N3 * beta + N6;
81     double F = N1 * b2 + N4 * beta + N7;
82     double G = N2 * b2 + N5 * beta + N8;
83     double D = 2 * G / (-F - Math.Pow(F * F - 4 * E * G, 0.5));
84     return (N10 + D - Math.Pow(Math.Pow(N10 + D, 2) - 4 * (N9 + N10 * D), 0.5)) / 2d - CONVERT_C_TO_K;
85   }
86 }
```

3.3.2　蒸発潜熱の計算

プログラム 3.3 に水の蒸発潜熱計算処理を示す。式 3.16 にもとづく計算である。1~8 行の *getTR* メソッドは、臨界点からの離れである T_r を計算するための関数（式 3.17）であり、以降のプログラムにおいても再利用する。

プログラム 3.3　水の蒸発潜熱の計算

Popolo.ThermophysicalProperty.Water class

```
 1 /// <summary>臨界温度からの離れを求める</summary>
 2 /// <param name="saturationTemperature">飽和温度[C]</param>
 3 /// <returns>臨界温度からの離れ</returns>
 4 private static double getTR(double saturationTemperature)
 5 {
 6   return (TEMPERATURE_AT_CRITICAL_POINT -
 7     (saturationTemperature + CONVERT_C_TO_K)) / TEMPERATURE_AT_CRITICAL_POINT;
 8 }
 9
10 /// <summary>飽和温度[℃]から蒸発潜熱[kJ/kg]を求める</summary>
11 /// <param name="saturationTemperature">飽和温度[℃]</param>
12 /// <returns>蒸発潜熱[kJ/kg]</returns>
13 public static double GetVaporizationLatentHeat (double saturationTemperature)
14 {
15   const double E1 = -3.87446;
16   const double E2 = 2.94553;
17   const double E3 = -8.06395;
18   const double E4 = 11.5633;
19   const double E5 = -6.02884;
20   const double B = 0.779221;
21   const double C = 4.62668;
22   const double D = -1.07931;
23
24   //0 度以上とする
25   saturationTemperature = Math.Max(0, saturationTemperature);
26
```

```
27   double tr = getTR(saturationTemperature);
28   if (tr < 0.0) return 0.0;
29   double y = B * Math.Pow(tr, 1.0 / 3.0) + C * Math.Pow(tr, 5.0 / 6.0) + D * Math.Pow(tr, 0.875);
30   y += tr * (E1 + tr * (E2 + tr * (E3 + tr * (E4 + tr * E5))));
31   return y * VAPORIZATION_HEAT_AT_TRIPLE_POINT;
32 }
```

3.3.3 飽和水と飽和蒸気の物性計算

プログラム 3.4 に飽和水の物性値計算処理を示す。式 3.16 にもとづく計算である。

プログラム 3.4 飽和水の物性値計算

Popolo.ThermophysicalProperty.Water class

```
 1 /// <summary>飽和温度[℃]から飽和水の比体積[m3/kg]を求める</summary>
 2 /// <param name="saturatedTemperature">飽和温度[℃]</param>
 3 /// <returns>飽和水の比体積[m3/kg]</returns>
 4 public static double GetSaturatedLiquidSpecificVolume(double saturatedTemperature)
 5 {
 6   const double A = 1.0;
 7   const double B = -1.9153882;
 8   const double C = 12.015186;
 9   const double D = -7.8464025;
10   const double E1 = -3.8886414;
11   const double E2 = 2.0582238;
12   const double E3 = -2.0829991;
13   const double E4 = 0.82180004;
14   const double E5 = 0.47549742;
15
16   double tr = getTR(saturatedTemperature);
17   double y = A + B * Math.Pow(tr, 1.0 / 3.0) + C * Math.Pow(tr, 5.0 / 6.0) + D * Math.Pow(tr, 0.875);
18   y += tr * (E1 + tr * (E2 + tr * (E3 + tr * (E4 + tr * E5))));
19
20   return y * SPECIFIC_VOLUME_AT_CRITICAL_POINT;
21 }
22
23 /// <summary>飽和温度[℃]から飽和水のエンタルピー[kJ/kg]を求める</summary>
24 /// <param name="saturatedTemperature">飽和温度[℃]</param>
25 /// <returns>飽和水のエンタルピー[kJ/kg]</returns>
26 public static double GetSaturatedLiquidEnthalpy(double saturatedTemperature)
27 {
28   //273.16<Ts<300 の係数
29   const double E11 = 624.698837;
30   const double E21 = -2343.85369;
31   const double E31 = -9508.12101;
32   const double E41 = 71628.7928;
33   const double E51 = -163535.221;
34   const double E61 = 166531.093;
35   const double E71 = -64785.4585;
36   //300<Ts<600 の係数
37   const double A2 = 0.8839230108;
38   const double E12 = -2.67172935;
39   const double E22 = 6.22640035;
40   const double E32 = -13.1789573;
41   const double E42 = -1.91322436;
42   const double E52 = 68.793763;
43   const double E62 = -124.819906;
44   const double E72 = 72.1435404;
45   //600<Ts の係数
46   const double A3 = 1.0;
47   const double B3 = -0.441057805;
48   const double C3 = -5.52255517;
49   const double D3 = 6.43994847;
50   const double E13 = -1.64578795;
51   const double E23 = -1.30574143;
52
53   double tk = saturatedTemperature + CONVERT_C_TO_K;
54   double tr = getTR(saturatedTemperature);
55   double y;
56   if (tk < 300.0)
57     y = tr * (E11 + tr * (E21 + tr * (E31 + tr * (E41 + tr * (E51 + tr * (E61 + tr * E71))))));
58   else if (tk < 600.0)
59     y = tr * (E12 + tr * (E22 + tr * (E32 + tr * (E42 + tr * (E52 + tr * (E62 + tr * E72)))))) + A2;
60   else
61     y = A3 + B3 * Math.Pow(tr, 1.0 / 3.0) + C3 * Math.Pow(tr, 5.0 / 6.0)
62   + D3 * Math.Pow(tr, 0.875) + tr * (E13 + tr * E23);
63
64   return y * ENTHALPY_AT_CRITICAL_POINT;
65 }
66
67 /// <summary>飽和温度[℃]から飽和水のエントロピー[kJ/kgK]を求める</summary>
```

```
/// <param name="saturatedTemperature">飽和温度[℃]</param>
/// <returns>飽和水のエントロピー[kJ/kgK]</returns>
public static double GetSaturatedLiquidEntropy(double saturatedTemperature)
{
  //273.16<Ts<300 の係数
  const double E11 = -1836.92956;
  const double E21 = 14706.6352;
  const double E31 = -43146.6046;
  const double E41 = 48606.6733;
  const double E51 = 7997.5096;
  const double E61 = -58333.9887;
  const double E71 = 33140.0718;
  //300<Ts<600 の係数
  const double A2 = 0.912762917;
  const double E12 = -1.75702956;
  const double E22 = 1.68754095;
  const double E32 = 5.82215341;
  const double E42 = -63.3354786;
  const double E52 = 188.076546;
  const double E62 = -252.344531;
  const double E72 = 128.058531;
  //600<Ts の係数
  const double A3 = 1.0;
  const double B3 = -0.324817650;
  const double C3 = -2.990556709;
  const double D3 = 3.2341900;
  const double E13 = -0.678067859;
  const double E23 = -1.91910364;

  double tk = saturatedTemperature + CONVERT_C_TO_K;
  double tr = getTR(saturatedTemperature);
  double y;
  if (tk < 300.0)
    y = tr * (E11 + tr * (E21 + tr * (E31 + tr * (E41 + tr * (E51 + tr * (E61 + tr * E71))))));
  else if (tk < 600.0)
    y = tr * (E12 + tr * (E22 + tr * (E32 + tr * (E42 + tr * (E52 + tr * (E62 + tr * E72)))))) + A2;
  else
    y = A3 + B3 * Math.Pow(tr, 1.0 / 3.0) + C3 * Math.Pow(tr, 5.0 / 6.0)
  + D3 * Math.Pow(tr, 0.875) + tr * (E13 + tr * E23);

  return y * ENTROPY_AT_CRITICAL_POINT;
}
```

プログラム 3.5 に飽和蒸気の物性値計算処理を示す。式 3.16 にもとづく計算である。

プログラム 3.5　飽和蒸気の物性値計算

Popolo.ThermophysicalProperty.Water class

```
/// <summary>飽和温度と飽和圧力から飽和蒸気の比体積を求める</summary>
/// <param name="saturatedTemperature">飽和温度[℃]</param>
/// <param name="saturatedPressure">飽和圧力[kPa]</param>
/// <returns>飽和蒸気の比体積[m3/kg]</returns>
public static double GetSaturatedVaporSpecificVolume(double saturatedTemperature, double saturatedPressure)
{
  const double A = 1.0d;
  const double B = 1.6351057d;
  const double C = 52.584599d;
  const double D = -44.694653;
  const double E1 = -8.9751114d;
  const double E2 = -0.43845530d;
  const double E3 = -19.179576d;
  const double E4 = 36.765319d;
  const double E5 = -19.462437d;

  double tr = getTR(saturatedTemperature);
  double y = A + B * Math.Pow(tr, 1.0 / 3.0) + C * Math.Pow(tr, 5.0 / 6.0) + D * Math.Pow(tr, 0.875);
  y += tr * (E1 + tr * (E2 + tr * (E3 + tr * (E4 + tr * E5))));

  return y * PRESSURE_AT_CRITICAL_POINT * SPECIFIC_VOLUME_AT_CRITICAL_POINT / saturatedPressure;
}

/// <summary>飽和温度[℃]から飽和蒸気のエンタルピー[kJ/kg]を求める</summary>
/// <param name="saturatedTemperature">飽和温度[℃]</param>
/// <returns>飽和蒸気のエンタルピー[kJ/kg]</returns>
public static double GetSaturatedVaporEnthalpy (double saturatedTemperature)
{
  const double E1 = -4.81351884;
  const double E2 = 2.69411792;
  const double E3 = -7.39064542;
  const double E4 = 10.4961689;
```

```
33   const double E5 = -5.46840036;
34   const double A = 1.0;
35   const double B = 0.457874342;
36   const double C = 5.08441288;
37   const double D = -1.48513244;
38
39   double tr = getTR(saturatedTemperature);
40   double y = A + B * Math.Pow(tr, 1.0 / 3.0) + C * Math.Pow(tr, 5.0 / 6.0) + D * Math.Pow(tr, 0.875);
41   y += tr * (E1 + tr * (E2 + tr * (E3 + tr * (E4 + tr * E5))));
42
43   return y * ENTHALPY_AT_CRITICAL_POINT;
44 }
45
46 /// <summary>飽和温度[℃]から飽和蒸気のエントロピー[kJ/kgK]を求める</summary>
47 /// <param name="saturatedTemperature">飽和温度[℃]</param>
48 /// <returns>飽和蒸気のエントロピー[kJ/kgK]</returns>
49 public static double GetSaturatedVaporEntropy (double saturatedTemperature)
50 {
51   const double E1 = -4.34839;
52   const double E2 = 1.34672;
53   const double E3 = 1.75261;
54   const double E4 = -6.22295;
55   const double E5 = 9.99004;
56   const double A = 1.0;
57   const double B = 0.377391;
58   const double C = -2.78368;
59   const double D = 6.93135;
60
61   double tr = getTR(saturatedTemperature);
62   double y = A + B * Math.Pow(tr, 1.0 / 3.0) + C * Math.Pow(tr, 5.0 / 6.0) + D * Math.Pow(tr, 0.875);
63   y += tr * (E1 + tr * (E2 + tr * (E3 + tr * (E4 + tr * E5))));
64
65   return y * ENTROPY_AT_CRITICAL_POINT;
66 }
```

以上の関数を用いて、蒸気表を計算するプログラムを 3.6 に示す。

プログラム 3.6　蒸気表の作成プログラム

Popolo.ThermophysicalProperty.Water class

```
 1 private static void SteamTableTest()
 2 {
 3     using (StreamWriter sWriter =
 4         new StreamWriter("SteamTable.csv", false, Encoding.GetEncoding("Shift_JIS")))
 5     {
 6     //タイトル行
 7     sWriter.WriteLine("飽和温度[℃], 飽和圧力[kPa], 飽和水比体積[m3/kg]," +
 8         " 飽和蒸気比体積[m3/kg], 飽和水エンタルピー[kJ/kg], " +
 9         "飽和蒸気エンタルピー[kJ/kg], 蒸発潜熱[kJ/kg], " +
10         "飽和水エントロピー[kJ/kgK], 飽和蒸気エントロピー[kJ/kgK]");
11
12     //蒸気物性の計算
13     for (int i = 0; i < 19; i++)
14     {
15         double ts;
16         if (i == 0) ts = 0.01;
17         else ts = i * 20;
18         double ps = Water.GetSaturationPressure(ts);
19         sWriter.WriteLine(ts + "," + ps + ","
20             + Water.GetSaturatedLiquidSpecificVolume(ts) + ","
21             + Water.GetSaturatedVaporSpecificVolume(ts, ps) + ","
22             + Water.GetSaturatedLiquidEnthalpy(ts) + ","
23             + Water.GetSaturatedVaporEnthalpy(ts) + ","
24             + Water.GetVaporizationLatentHeat(ts) + ","
25             + Water.GetSaturatedLiquidEntropy(ts) + ","
26             + Water.GetSaturatedVaporEntropy(ts));
27         }
28     }
29 }
```

計算結果を表 3.4 に示す。表 3.5 は IAPWS-IF97 による計算結果[3.2]である。平均の誤差率は 0.063%、最大誤差率は 0.796%であり、全領域に渡って精度の高い近似ができていることが確認できる。

表 3.4　蒸気表（本書の計算法）

T_{Ws} [°C]	P_{Ws} [kPa, MPa]	v_{Wsl} [m³/kg]	v_{Wsv} [m³/kg]	h_{Wsl} [kJ/kg]	h_{Wsv} [kJ/kg]	L_{Ws} [kJ/kg]	s_{Wsl} [kJ/(kg·K)]	s_{Wsv} [kJ/(kg·K)]
0.01	0.61166	0.00099991	206.075	0.14	2501.45	2501.18	0.00132	9.15155
20	2.3392	0.00100199	57.7820	83.27	2538.55	2454.14	0.29414	8.66551
40	7.3844	0.00100790	19.5241	167.73	2574.70	2406.26	0.57251	8.25492
60	19.946	0.00101707	7.6703	252.15	2609.89	2357.50	0.83364	7.90661
80	47.415	0.00102907	3.40626	336.19	2643.89	2307.53	1.07851	7.60896
100	101.42	0.00104364	1.67224	420.10	2676.33	2255.81	1.30940	7.35208
120	198.67	0.00106065	0.891482	504.37	2706.69	2201.59	1.52894	7.12759
140	361.50	0.00108014	0.508653	589.49	2734.34	2143.99	1.73946	6.92849
160	0.61814	0.00110231	0.306942	675.85	2758.56	2081.98	1.94275	6.74896
180	1.0026	0.00112753	0.193967	763.70	2778.55	2014.42	2.13999	6.58417
200	1.5547	0.00115641	0.127298	853.26	2793.44	1940.05	2.33201	6.43013
220	2.3193	0.00118982	0.0861441	944.77	2802.25	1857.46	2.51953	6.28341
240	3.3467	0.00122898	0.0597266	1038.67	2803.89	1764.98	2.70358	6.14088
260	4.6921	0.00127568	0.0421732	1135.79	2796.96	1660.55	2.88579	5.99938
280	6.4165	0.00133266	0.0301405	1237.46	2779.60	1541.28	3.06863	5.85518
300	8.5877	0.00140448	0.0216454	1345.67	2748.94	1402.72	3.25542	5.70308
320	11.284	0.00149979	0.0154601	1463.16	2699.85	1236.85	3.45013	5.53455
340	14.600	0.00163916	0.0107731	1595.05	2621.26	1026.17	3.66001	5.33245
360	18.666	0.00189861	0.00691947	1762.96	2479.00	715.45	3.91751	5.04698

表 3.5　蒸気表（IAPWS-IF97）

T_{Ws} [°C]	P_{Ws} [kPa, MPa]	v_{Wsl} [m³/kg]	v_{Wsv} [m³/kg]	h_{Wsl} [kJ/kg]	h_{Wsv} [kJ/kg]	L_{Ws} [kJ/kg]	s_{Wsl} [kJ/(kg·K)]	s_{Wsv} [kJ/(kg·K)]
0.01	0.61166	0.00100021	205.997	0.00	2500.91	2500.91	0.00000	9.15549
20	2.3392	0.00100184	57.7615	83.92	2537.47	2453.55	0.29650	8.66612
40	7.3844	0.00100788	19.5170	167.54	2573.54	2406.00	0.57243	8.25567
60	19.946	0.00101711	7.66766	251.15	2608.85	2357.69	0.83122	7.90817
80	47.415	0.00102904	3.40527	334.95	2643.01	2308.07	1.07539	7.61102
100	101.42	0.00104346	1.67186	419.10	2675.57	2256.47	1.30701	7.35408
120	198.67	0.00106033	0.891304	503.78	2705.93	2202.15	1.52782	7.12909
140	361.50	0.00107976	0.508519	589.20	2733.44	2144.24	1.73929	6.92927
160	0.61814	0.00110199	0.306818	675.57	2757.43	2081.86	1.94278	6.74910
180	1.0026	0.00112739	0.193862	763.19	2777.22	2014.03	2.13954	6.58407
200	1.5547	0.00115651	0.127222	852.39	2792.06	1939.67	2.33080	6.43030
220	2.3193	0.00119016	0.0861007	943.64	2801.05	1857.41	2.51782	6.28425
240	3.3467	0.00122946	0.0597101	1037.52	2803.06	1765.54	2.70194	6.14253
260	4.6921	0.00127613	0.0421755	1134.83	2796.64	1661.82	2.88472	6.00169
280	6.4165	0.00133285	0.0301540	1236.67	2779.82	1543.15	3.06807	5.85783
300	8.5877	0.00140422	0.0216631	1344.77	2749.57	1404.80	3.25474	5.70576
320	11.284	0.00149906	0.0154759	1462.05	2700.67	1238.62	3.44912	5.53732
340	14.600	0.00163751	0.0107838	1594.45	2622.07	1027.62	3.65995	5.33591
360	18.666	0.00189451	0.00694494	1761.49	2480.99	719.50	3.91636	5.05273

※$160 \leq T_{Ws}$の範囲に関してはP_sの単位は MPa

3.3.4　液相の物性の計算

プログラム 3.7 に液相のその他の物性値計算処理を示す。式 3.17 と式 3.18 にもとづく計算である。

プログラム **3.7**　液体の物性値計算

Popolo.ThermophysicalProperty.Water class

```
/// <summary>温度[C]から水（液体）の密度[kg/m3]を計算する</summary>
/// <param name="temperature">温度[C]</param>
/// <returns>水（液体）の密度[kg/m3]</returns>
public static double GetLiquidDensity(double temperature)
{
  double[] a = new double[] { 9.8811040e2, -1.3273604e3, 4.7162295e3, -4.1245328e3 };

  double tk = temperature + CONVERT_C_TO_K;
  double tr = (TEMPERATURE_AT_CRITICAL_POINT - tk) / TEMPERATURE_AT_CRITICAL_POINT;

  double rho = a[a.Length - 1];
  for (int i = a.Length - 2; 0 <= i; i--) rho = a[i] + rho * tr;
  return rho;
}
```

```
15
16 /// <summary>温度[C]から水（液体）の定圧比熱[kJ/(kg･K)]を計算する</summary>
17 /// <param name="temperature">温度[C]</param>
18 /// <returns>水（液体）の比熱[kJ/(kg･K)]</returns>
19 public static double GetLiquidIsobaricSpecificHeat(double temperature)
20 {
21   double[] a = new double[] { 1.0570130, 2.1952960e1, -4.9895501e1, 3.6963413e1 };
22
23   double tk = temperature + CONVERT_C_TO_K;
24   double tr = (TEMPERATURE_AT_CRITICAL_POINT - tk) / TEMPERATURE_AT_CRITICAL_POINT;
25
26   double cpw = a[a.Length - 1];
27   for (int i = a.Length - 2; 0 <= i; i--) cpw = a[i] + cpw * tr;
28   return cpw;
29 }
30
31 /// <summary>温度[C]から水（液体）の熱伝導率[W/(m･K)]を計算する</summary>
32 /// <param name="temperature">温度[C]</param>
33 /// <returns>水（液体）の熱伝導率[W/(m･K)]</returns>
34 public static double GetLiquidThermalConductivity(double temperature)
35 {
36   double[] a = new double[] { -1.3734399e-1, 4.2128755, -5.9412196, 1.2794890 };
37
38   double tk = temperature + CONVERT_C_TO_K;
39   double tr = (TEMPERATURE_AT_CRITICAL_POINT - tk) / TEMPERATURE_AT_CRITICAL_POINT;
40
41   double lambda = a[a.Length - 1];
42   for (int i = a.Length - 2; 0 <= i; i--) lambda = a[i] + lambda * tr;
43   return lambda;
44 }
45
46 /// <summary>温度[C]から水（液体）の粘性係数[Pa･s]を計算する</summary>
47 /// <param name="temperature">温度[C]</param>
48 /// <returns>水（液体）の粘性係数[Pa･s]</returns>
49 public static double GetLiquidViscosity(double temperature)
50 {
51   double[] a = new double[] { 5.2136906e1, -4.0910405e2, 1.3270844e3, -1.9089622e3, 1.0489917e3 };
52
53   double tk = temperature + CONVERT_C_TO_K;
54   double tr = (TEMPERATURE_AT_CRITICAL_POINT - tk) / TEMPERATURE_AT_CRITICAL_POINT;
55
56   double mu = a[a.Length - 1];
57   for (int i = a.Length - 2; 0 <= i; i--) mu = a[i] + mu * tr;
58   return Math.Exp(mu) * 0.000001;
59 }
60
61 /// <summary>温度[C]から水（液体）の動粘性係数[m2/s]を計算する</summary>
62 /// <param name="temperature">温度[C]</param>
63 /// <returns>水（液体）の動粘性係数[m2/s]</returns>
64 public static double GetLiquidDynamicViscosity(double temperature)
65 {
66   double mu = GetLiquidViscosity(temperature);
67   double rho = GetLiquidDensity(temperature);
68   return mu / rho;
69 }
70
71 /// <summary>温度[C]から水（液体）の熱拡散率[m2/s]を計算する</summary>
72 /// <param name="temperature">温度[C]</param>
73 /// <returns>水（液体）の熱拡散率[m2/s]</returns>
74 public static double GetLiquidThermalDiffusivity(double temperature)
75 {
76   double lambda = GetLiquidThermalConductivity(temperature);
77   double cp = GetLiquidIsobaricSpecificHeat(temperature);
78   double rho = GetLiquidDensity(temperature);
79   return lambda / (1000 * cp * rho);
80 }
```

【例題 3.2】

例題 3.1 を参考に、プログラム 3.7 を用いて、任意の配管径、水量、温度から管内表面の対流熱伝達率を計算するプログラムを作成せよ。

【解】

プログラム 3.8 に示す。

プログラム 3.8　配管内の対流熱伝達率の計算

Popolo.HVAC.Circuit.WaterPipe class

```
1 /// <summary>水温・直径・水量から配管内対流熱伝達率を計算する</summary>
2 /// <param name="waterTemperature">水温[C]</param>
3 /// <param name="diameter">直径[m]</param>
```

```
/// <param name="waterFlowRate">水量[L/min]</param>
/// <returns>対流熱伝達率[W/(m2･K)]</returns>
public static double GetConvectiveHeatTransferCoefficientOfTube
  (double waterTemperature, double diameter, double waterFlowRate)
{
    //動粘性係数[m2/s]・熱拡散率[m2/s]・熱伝導率[W/(m･K)]を計算
    double v = Water.GetLiquidDynamicViscosity(waterTemperature);
    double a = Water.GetLiquidThermalDiffusivity(waterTemperature);
    double lambda = Water.GetLiquidThermalConductivity(waterTemperature);

    //配管内流速[m/s]を計算
    double u = (waterFlowRate / 60 / 1000) / (Math.Pow(diameter / 2, 2) * Math.PI);

    //ヌセルト数を計算
    double reNumber = u * diameter / v;
    double prNumber = v / a;
    double nuNumber = 0.023 * Math.Pow(reNumber, 0.8) * Math.Pow(prNumber, 0.4);

    //ヌセルト数から対流熱伝達率[W/(m2･K)]を計算
    return nuNumber * lambda / diameter;
}
```

【第 3 章 記号表】

a	：熱拡散率 **[m²/s]**	v	：動粘性係数 **[m²/s]**
c_p	：定圧比熱 **[kJ/(kg･K)]**	W	：体積濃度 **[%]**
h	：比エンタルピー **[kJ/kg]**	α	：近似係数 **[-]**
N_u	：ヌセルト数 **[-]**	β	：近似係数 **[-]**
P_r	：プラントル数 **[-]**	λ	：熱伝導率 **[W/(m･K)]**
P_{Ws}	：飽和水蒸気圧 **[kPa]**	ρ	：密度 **[kg/m³]**
R_e	：レイノルズ数 **[-]**	υ	：比体積 **[m³/kg]**
s	：比エントロピー **[kJ/(kg･K)]**	γ	：蒸発潜熱 **[kJ/kg]**
T_{Ws}	：飽和温度 **[K]**		
subscripts			
CP	：臨界点	***Wsl***	：飽和水
eg	：エチレングリコール	***Wsv***	：飽和蒸気
W	：水		

【第 3 章 参考文献】

3.1） 井上宇市: 高温水暖房, 空気調和・衛生工学, Vol.40, No.3, pp.1~4, 1966

3.2） 日本機械学会 蒸気表, 日本機械学会, 丸善株式会社, 1999

3.3） Thomas F. Irvine, Jr. Peter E. Liley : Steam and Gas Tables with Computer Equations, Academic Press INC., 1984

3.4） 宇田川光弘 : パソコンによる空気調和計算法, p.313. オーム社, 1986

3.5） 宇田川光弘 : パソコンによる空気調和計算法, 第 3 章 湿り空気の状態値, pp26~43. オーム社, 1986

3.6） 西川兼康, 藤田恭伸 : 伝熱学, 第 7 章 p171, 理工学社, 1982

3.7） Dittus, F.W. And Boelter, L.M.K., Univ. Calif. Publs. Eng. 2(1930), 443

3.8） American Society of Heating, Refrigerating and Air-Conditioning Engineers. ASHRAE Fundamentals Handbook, Chapter 20 Physical Properties of Secondary Coolants (Brines), pp.20.5~20.11, 1997

3.9） Wexler, A., Hyland, R. W. : Final Report, ASHRAE Research Project RP 216 (1980), Part 1, 2, 3

3.10） Wagner, W. and Pruss, A.,"The IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use," J. Phys. Chem. Ref. Data, 31(2): 387-535, 2002

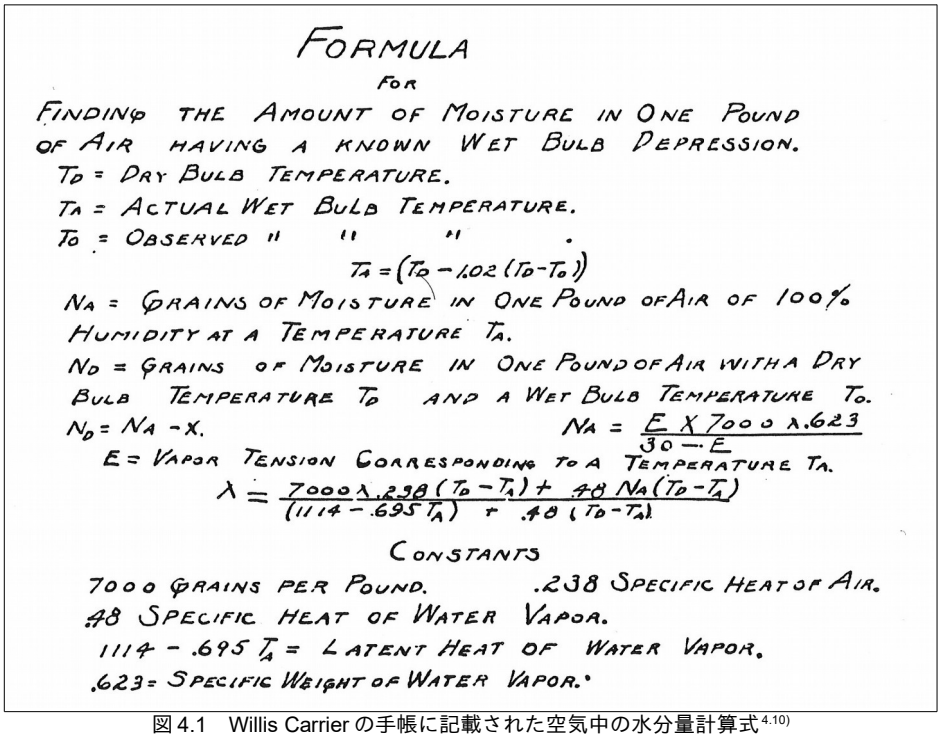
第4章　湿り空気の物性
(Thermodynamic Properties of Moist Air)

4.1　概要

我々は空気に包まれて生活をしており、空気の状態は空間の快適性に直接に影響を与える。また、いくつかの主要な空調機器・空調システムは熱を移動する媒体としても空気を用いており、空気の物性の理解は空調設備設計の基礎である。

空気は約 80 %が窒素、約 20 %が酸素であるが、これに加えて水蒸気を含んでいる。水分量の多寡は快適性や熱移動の性能に大きく影響を与えるため、空調設備分野では特に「湿り空気」と呼ぶことが多い。近代空調技術の父である Willis Carrier による空気調和機の発明は 1902 年のことであるが、この発明の最も画期的な点は、湿り空気の水分調整を可能にした点にある。湿り空気の特性を知り、加熱・冷却・加湿・除湿を操る技術は、空調設備設計者の最も基礎的な素養の一つである。

湿り空気の物性である乾球温度、湿球温度、絶対湿度、相対湿度、比体積、比エンタルピーなどの関係性をまとめた図として、図 4.2 に示す湿り空気線図がある。実務上はこのような図表から物性を読み取って設計を行うが、シミュレーションを行う際には具体的な数値を算出できる必要がある。2 種類の物性値を特定できれば、湿り空気の状態は唯一に定まる。本書では湿り空気の物性の内、乾球温度、湿球温度、絶対湿度、相対湿度、比エンタルピーを入力して状態値を予測するプログラムの作成方法について解説する。



FORMULA

FOR

FINDING THE AMOUNT OF MOISTURE IN ONE POUND OF AIR HAVING A KNOWN WET BULB DEPRESSION.

T_D = DRY BULB TEMPERATURE.

T_A = ACTUAL WET BULB TEMPERATURE.

T_O = OBSERVED " " " .

$$T_A = (T_D - 1.02(T_D - T_O))$$

N_A = GRAINS OF MOISTURE IN ONE POUND OF AIR OF 100% HUMIDITY AT A TEMPERATURE T_A.

N_D = GRAINS OF MOISTURE IN ONE POUND OF AIR WITH A DRY BULB TEMPERATURE T_D AND A WET BULB TEMPERATURE T_O.

$N_D = N_A - X.$

$$N_A = \frac{E \times 7000 \times .623}{30 - E}$$

E = VAPOR TENSION CORRESPONDING TO A TEMPERATURE T_A.

$$\lambda = \frac{7000 \times .238 (T_D - T_A) + .48 N_A (T_D - T_A)}{(1114 - .695 T_A) + .48 (T_D - T_A)}$$

CONSTANTS

7000 GRAINS PER POUND.　　.238 SPECIFIC HEAT OF AIR.

.48 SPECIFIC HEAT OF WATER VAPOR.

$1114 - .695 T_A$ = LATENT HEAT OF WATER VAPOR.

.623 = SPECIFIC WEIGHT OF WATER VAPOR.

図 4.1　Willis Carrier の手帳に記載された空気中の水分量計算式[4.10)]

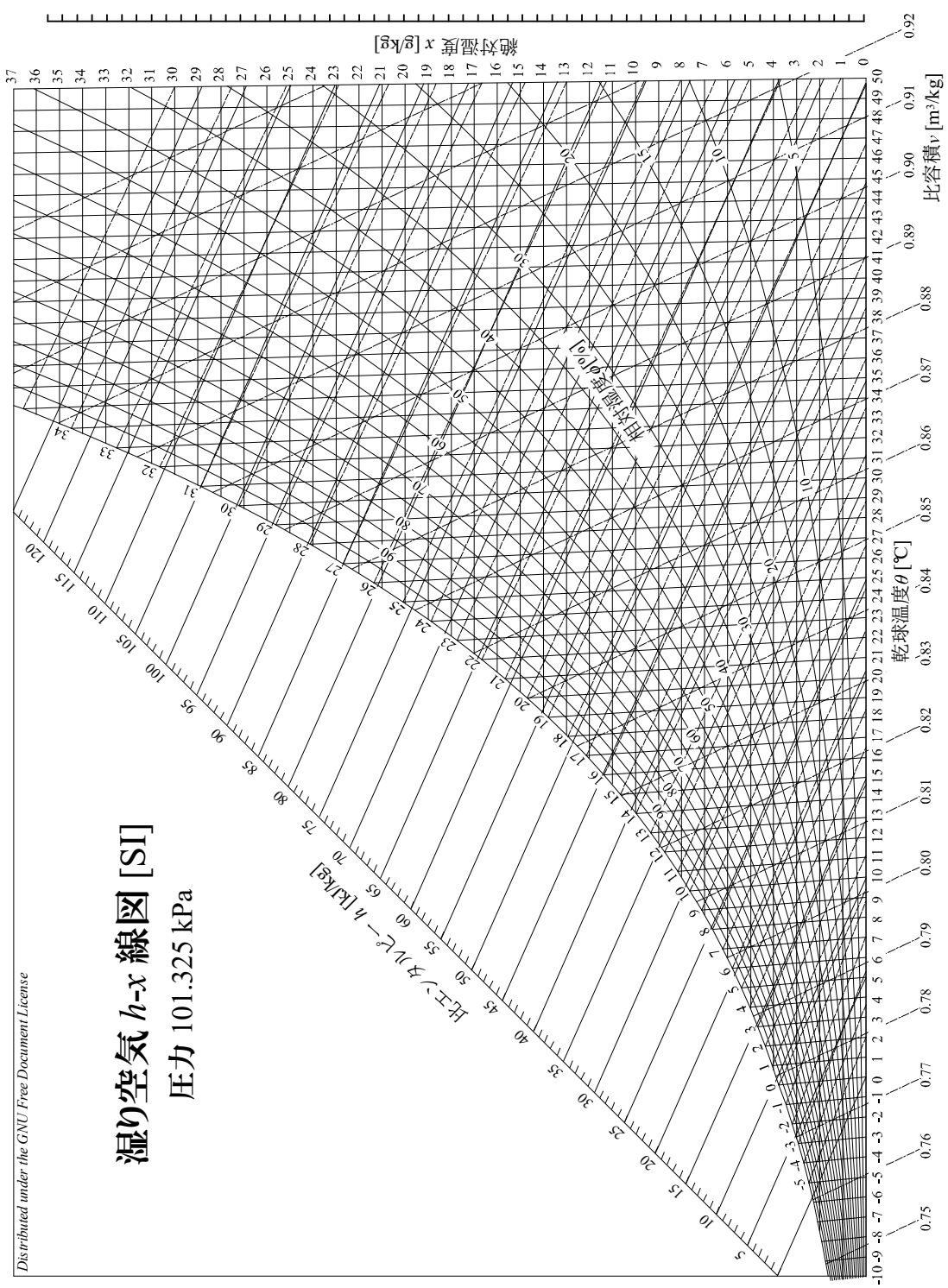


図 4.2　湿り空気 h-x 線図

4.2 理論

4.2.1 飽和水蒸気圧

湿り空気の物性値の把握にあたり、基礎となる計算は水の温度と飽和水蒸気圧との関係であるが、これに関しては既に第第 3 章で計算法を示した。

4.2.2 相対湿度

相対湿度 ϕ [%]は飽和水蒸気圧 P_{Ws} [kPa]と水蒸気分圧 P_W [kPa]の比率であり、式 4.1 で計算できる。

$$\phi = 100\frac{P_W}{P_{Ws}} \tag{4.1}$$

4.2.3 絶対湿度

絶対湿度 W [kg/kg]は、水蒸気分圧 P_W と大気圧 P [kPa]の関数として、式 4.2 で計算できる。ただし、0.62198 は水の分子量（=18.01528 g/mol）と乾き空気の分子量（=28.9645 g/mol）の比である。また、大気圧 P は、標高 Z [m]の関数として、式 4.3 で計算できる[4.5) 4.6) 4.7)]。通常、湿り空気線図は標高 0 m（101.325 kPa）として作成されている。

$$W = 0.62198\frac{P_W}{P - P_W} \tag{4.2}$$

$$P = 101.325\left(1 - 2.25577\times10^{-5}\,Z\right)^{5.2559} \tag{4.3}$$

【例題 4.1】

富士山頂での水の沸点を求めよ。

【解】

富士山の標高は 3,776m であるため、式 4.3 により、富士山頂での大気圧は、

$P = 101.325 \times (1 - 2.25577\times10^{-5} \times 3776)^{5.2559} = 63.5$ kPa

である。式 3.1~式 3.12 を使用して、飽和水蒸気圧 P_{Ws} に 63.5kPa を代入して飽和温度 T_{Ws} を求めると 87.3 °C となる[†1)]。

4.2.4 飽和度

飽和度 φ [%]は飽和絶対湿度 W_S[kg/kg]と絶対湿度 W [kg/kg]の比率であり、式 4.4 で計算できる。

$$\varphi = 100\frac{W}{W_s} \tag{4.4}$$

4.2.5 比体積

湿り空気の比体積 υ [m³/kg]は、乾球温度 t [°C]と絶対湿度 W の関数として式 4.5 で計算できる。ただし R_a は各気体に固有の気体定数であり、空気の場合には 0.287055 kJ/(kg·K)である。

$$\upsilon = \frac{(t+273.15)R_a}{P}(1+1.6078\,W) \tag{4.5}$$

4.2.6 乾球温度、湿球温度、絶対湿度、比エンタルピーの関係

乾球温度 t、絶対湿度 W、比エンタルピー h [kJ/kg]の関係は式 4.6 で表現できる。ここで、c_{pa} は乾き空気の定圧比熱（=1.006 kJ/(kg·K)）、c_{pv} は水蒸気の定圧比熱（=1.805 kJ/(kg·K)）、γ_0 は 0°C の水の蒸発潜熱（=2,501 kJ/kg）である。また、乾球温度 t の係数のみをまとめることで、湿り空気の比熱 c_{pma} [kJ/(kg·K)]は式 4.7 で計算できる。式 4.6 は乾球温度 t [°C]と絶対湿度 W についても解けるため、3 つの状態値のいずれか 2 つがわかれば残りの 1 つが解析的に求まることになる。

†1 このように圧力が低い場合には沸点も低くなるため、高温のお湯を得ることができなくなる。例えば、某航空会社の機内用インスタントラーメンは、低温のお湯でも麺が茹でられるように工夫がされているらしい。また、逆の作用の例としては圧力鍋が挙げられる。

$$h = c_{pa} t + (\gamma_0 + c_{pv} t) W \tag{4.6}$$

$$c_{pma} = c_{pa} + c_{pv} W \tag{4.7}$$

湿球温度 θ [°C]と絶対湿度 W、比エンタルピー h の関係は、式 4.8 で表現できる。ここで、W_s' [kg/kg]、h_s' [kJ/kg]はそれぞれ湿球温度 θ' [°C]での飽和空気の絶対湿度と比エンタルピーであり、h_w' [kJ/kg]は θ [°C]の水の比エンタルピーである。水の比エンタルピー h_w'を $c_{pw}\theta$ で表現し、式 4.6 を式 4.8 に代入して絶対湿度 W および乾球温度 t について整理すると式 4.9 と式 4.10 が得られる。ただし c_{pw} [kJ/(kg·K)]は水の定圧比熱である。式 4.9 と式 4.10 を用いることにより、湿球温度 θ が与えられた場合には、乾球温度 t もしくは絶対湿度 W と組み合わせて、残り 1 つの状態値が求められる。一方、乾球温度 t および絶対湿度 W が与えられた場合には、湿球温度 θ を代数的に解くことはできないため、ニュートン・ラプソン法などを利用して数値計算で解く必要がある。具体的な方法は後述する。

$$h + (W_s' - W) h_w' = h_s' \tag{4.8}$$

$$W = \frac{(\gamma_0 + (c_{pv} - c_{pw})\theta) W_s' + c_{pa}(\theta - t)}{\gamma_0 + c_{pv} t - c_{pw}\theta} \tag{4.9}$$

$$t = \frac{(\gamma_0 + (c_{pv} - c_{pw})\theta) W_s' + c_{pa}\theta + (c_{pw}\theta - \gamma_0) W}{c_{pv} W + c_{pa}} \tag{4.10}$$

4.2.7　その他の物性

1）熱伝導率

湿り空気の熱伝導率 λ [W/(m·K)]の近似式を式 4.11 に示す。近似性能は 0~100 °C の範囲で相対誤差 0.07 %未満である。

$$\lambda = 0.0241 + 0.000077 t \tag{4.11}$$

2）粘性係数

湿り空気の粘性係数 μ [Pa·s]の近似式を式 4.12 に示す。

$$\mu = \left(\frac{0.0074237}{t + 390.15} \right) \left(\frac{t + 273.15}{293.15} \right)^{1.5} \tag{4.12}$$

3）動粘性係数

湿り空気の動粘性係数 ν [m²/s]は粘性係数 μ に比体積 υ [m³/kg]を乗じて式 4.13 で計算する。

$$\nu = \mu \upsilon \tag{4.13}$$

4）膨張率

湿り空気の膨張率 β [1/K]の計算式を式 4.14 に示す。

$$\beta = \frac{1}{t + 273.15} \tag{4.14}$$

5）拡散係数

湿り空気の熱拡散係数 D_a [m²/s]の計算式を式 4.15 に示す。

$$D_a = \frac{\lambda \upsilon}{c_{pma}} \tag{4.15}$$

式 4.11~4.14によって各種物性を計算した結果を図 4.3 に示す。

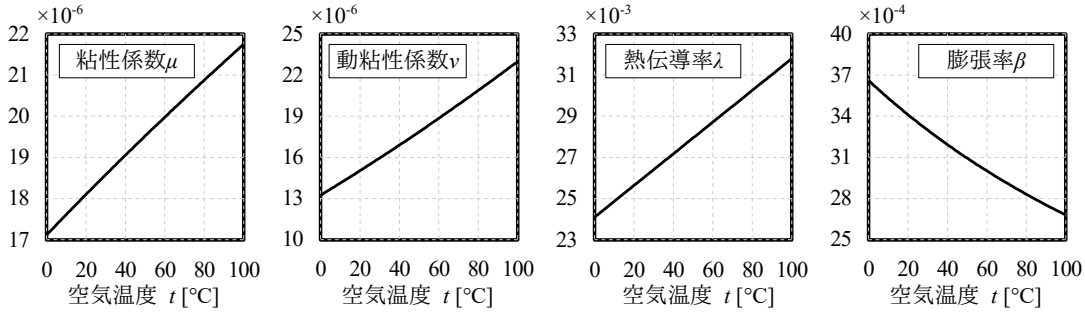


図 4.3　湿り空気の各種物性

4.3　計算法

乾球温度 t、湿球温度 θ、絶対湿度 W、相対湿度 ϕ、比エンタルピー h の内、2 つの状態値にもとづいて、残る 3 つの状態値を計算する方法を示す。

4.3.1　大気圧の計算

各種の定数宣言を 4.1 に示す。

プログラム 4.1　湿り空気の物性計算の定数宣言

Popolo.ThermophysicalProperty.MoistAir class

```
/// <summary>絶対温度と摂氏との変換定数</summary>
private const double CONVERT_C_TO_K = 273.15;

/// <summary>乾き空気の定圧比熱[kJ/kg-K]</summary>
private const double DRYAIR_ISOBARIC_SPECIFIC_HEAT = 1.005;

/// <summary>水蒸気の定圧比熱[kJ/kg-K]</summary>
private const double VAPOR_ISOBARIC_SPECIFIC_HEAT = 1.805;

/// <summary>0℃の水の定圧比熱[kJ/kg-K]</summary>
private const double WATER_ISOBARIC_SPECIFIC_HEAT = 4.186;

/// <summary>0℃の水の蒸発潜熱[kJ/kg]</summary>
private const double VAPORIZATION_LATENT_HEAT = 2501;

/// <summary>海抜 0m での大気圧=101.325[kPa]</summary>
private const double ATMOSPHERIC_PRESSURE_AT_SEALEVEL = 101.325;

/// <summary>乾き空気のガス定数[kJ/(kg K)]</summary>
private const double DRYAIR_GAS_CONSTANT = 0.287042;
```

式 4.3 を用いて標高に応じた大気圧を計算するためのプログラムを 4.2 に示す。

プログラム 4.2　大気圧の計算

Popolo.ThermophysicalProperty.MoistAir class

```
/// <summary>標高[m]に応じた大気圧[kPa]を取得する</summary>
/// <param name="altitude">標高[m]</param>
/// <returns>大気圧[kPa]</returns>
public static double GetAtmosphericPressure(double altitude)
{
  return ATMOSPHERIC_PRESSURE_AT_SEALEVEL * Math.Pow(1.0 - 2.25577e-5 * altitude, 5.2559);
}
```

4.3.2　水蒸気分圧と絶対湿度の計算

式 4.2 にもとづき、水蒸気分圧と絶対湿度を変換するためのプログラムを 4.3 に示す。

プログラム 4.3　絶対湿度と水蒸気分圧の計算

Popolo.ThermophysicalProperty.MoistAir class

```
/// <summary>絶対湿度[kg/kg]と大気圧[kPa]から水蒸気分圧[kPa]を求める</summary>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]：1 気圧は 101.325[kPa]</param>
/// <returns>水蒸気分圧[kPa]</returns>
public static double GetWaterVaporPartialPressureFromHumidityRatio
  (double humidityRatio, double atmosphericPressure)
{
  return atmosphericPressure * humidityRatio / (0.62198 + humidityRatio);
}
```

```
10
11 /// <summary>水蒸気分圧[kPa]と大気圧[kPa]から絶対湿度[kg/kg]を求める</summary>
12 /// <param name="waterVaporPartialPressure">水蒸気分圧[kPa]</param>
13 /// <param name="atmosphericPressure">大気圧[kPa]：1気圧は101.325[kPa]</param>
14 /// <returns>絶対湿度[kg/kg]</returns>
15 public static double GetHumidityRatioFromWaterVaporPartialPressure
16   (double waterVaporPartialPressure, double atmosphericPressure)
17 {
18   return 0.62198 * waterVaporPartialPressure / (atmosphericPressure - waterVaporPartialPressure);
19 }
```

4.3.3　乾球温度、絶対湿度、比エンタルピーの計算

乾球温度 t、絶対湿度 W、比エンタルピー h に関しては、式 4.6 によって相互に計算が可能である。各々の状態値を計算するためのプログラムを 4.4 に示す。

プログラム 4.4　乾球温度 t、絶対湿度 W、比エンタルピー h の計算

Popolo.ThermophysicalProperty.MoistAir class

```
 1 /// <summary>乾球温度[℃]と絶対湿度[kg/kg]から比エンタルピー[kJ/kg]を求める</summary>
 2 /// <param name="dryBulbTemperature">乾球温度[℃]</param>
 3 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
 4 /// <returns>比エンタルピー[kJ/kg]</returns>
 5 public static double GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
 6   (double dryBulbTemperature, double humidityRatio)
 7 {
 8   return DRYAIR_ISOBARIC_SPECIFIC_HEAT * dryBulbTemperature +
 9     humidityRatio * (VAPOR_ISOBARIC_SPECIFIC_HEAT * dryBulbTemperature + VAPORIZATION_LATENT_HEAT);
10 }
11
12 /// <summary>乾球温度[℃]と比エンタルピー[kJ/kg]から絶対湿度[kg/kg]を求める</summary>
13 /// <param name="dryBulbTemperature">乾球温度[℃]</param>
14 /// <param name="enthalpy">比エンタルピー[kJ/kg]</param>
15 /// <returns>絶対湿度[kg/kg]</returns>
16 public static double GetHumidityRatioFromDryBulbTemperatureAndEnthalpy
17   (double dryBulbTemperature, double enthalpy)
18 {
19   return (enthalpy - DRYAIR_ISOBARIC_SPECIFIC_HEAT * dryBulbTemperature) /
20     (VAPOR_ISOBARIC_SPECIFIC_HEAT * dryBulbTemperature + VAPORIZATION_LATENT_HEAT);
21 }
22
23 /// <summary>絶対湿度[kg/kg]と比エンタルピー[kJ/kg]から乾球温度[℃]を求める</summary>
24 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
25 /// <param name="enthalpy">比エンタルピー[kJ/kg]</param>
26 /// <returns>乾球温度[℃]</returns>
27 public static double GetDryBulbTemperatureFromHumidityRatioAndEnthalpy(double humidityRatio, double enthalpy)
28 {
29   return (enthalpy - VAPORIZATION_LATENT_HEAT * humidityRatio) /
30     (DRYAIR_ISOBARIC_SPECIFIC_HEAT + VAPOR_ISOBARIC_SPECIFIC_HEAT * humidityRatio);
31 }
32
33 /// <summary>湿り空気比熱[kJ/kgK]を計算する</summary>
34 /// <param name="humidityRatio">絶対湿度[kg/kg(DA)]</param>
35 /// <returns>湿り空気比熱[kJ/kgK]</returns>
36 public static double GetSpecificHeat(double humidityRatio)
37 {
38   return DRYAIR_ISOBARIC_SPECIFIC_HEAT + VAPOR_ISOBARIC_SPECIFIC_HEAT * humidityRatio;
39 }
```

4.3.4　乾球温度、湿球温度、絶対湿度の計算

乾球温度 t、湿球温度 θ、絶対湿度 W の計算には、式 4.9 と式 4.10 を用いる。ただし、前述のとおり、乾球温度 t と絶対湿度 W から湿球温度 θ を求める場合には反復計算が必要となる。まず湿球温度 θ に適当な値を仮定して、乾球温度と組み合わせて絶対湿度を計算する。算出された絶対湿度と、入力条件として与えられた絶対湿度との差分を誤差として評価し、誤差を 0 とするようにニュートン・ラフソン法で反復計算を行う。具体的には、乾球温度 T_1 と絶対湿度 W_1 が与えられた場合の誤差関数を式 4.16 で定義する。ここで、$f_1(T_1, \theta)$ は乾球温度と湿球温度から絶対湿度を計算する式 4.9 である。誤差関数 $err_1(\theta)$ が 0 となるように、θ の値を調整する。収束計算の目標精度が問題となるが、厳冬期の外気温湿度を 1 ºC、0.001 kg/kg とし、この条件において 0.01 %程度の誤差を許容するとすれ

ば、乾球温度で 1×0.0001=10^{-4}、絶対湿度で 0.001×0.0001=10^{-7}程度を誤差許容値とすれば良い。各々の状態値を計算するための計算処理をプログラム 4.5 に示す。

$$err_1(\theta)=\mathrm{W}_1-f_1(\mathrm{T}_1,\theta) \tag{4.16}$$

プログラム 4.5　乾球温度 *t*、絶対湿度 *W*、湿球温度 *θ* の計算

Popolo.ThermophysicalProperty.MoistAir class

```
/// <summary>乾球温度[℃]と湿球温度[℃]から絶対湿度[kg/kg]を求める</summary>
/// <param name="dryBulbTemperature">乾球温度[℃]</param>
/// <param name="wetBulbTemperature">湿球温度[℃]</param>
/// <param name="atmosphericPressure">大気圧[kPa]：1 気圧は 101.325[kPa]</param>
/// <returns>絶対湿度[kg/kg]</returns>
public static double GetHumidityRatioFromDryBulbTemperatureAndWetBulbTemperature
  (double dryBulbTemperature, double wetBulbTemperature, double atmosphericPressure)
{
  double ps = Water.GetSaturationPressure(wetBulbTemperature);
  double ws = GetHumidityRatioFromWaterVaporPartialPressure(ps, atmosphericPressure);
  double a = ws * (VAPORIZATION_LATENT_HEAT + (VAPOR_ISOBARIC_SPECIFIC_HEAT - WATER_ISOBARIC_SPECIFIC_HEAT)
    * wetBulbTemperature) + DRYAIR_ISOBARIC_SPECIFIC_HEAT * (wetBulbTemperature - dryBulbTemperature);
  double b = VAPORIZATION_LATENT_HEAT + VAPOR_ISOBARIC_SPECIFIC_HEAT * dryBulbTemperature
- WATER_ISOBARIC_SPECIFIC_HEAT * wetBulbTemperature;
  return a / b;
}

/// <summary>絶対湿度[kg/kg]と湿球温度[C]から乾球温度[C]を計算する</summary>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="wetBulbTemperature">湿球温度[C]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>乾球温度[C]</returns>
public static double GetDryBulbTemperatureFromHumidityRatioAndWetBulbTemperature
  (double humidityRatio, double wetBulbTemperature, double atmosphericPressure)
{
  double ps = Water.GetSaturationPressure(wetBulbTemperature);
  double ws = GetHumidityRatioFromWaterVaporPartialPressure(ps, atmosphericPressure);
  double a = ws * (VAPORIZATION_LATENT_HEAT
    + (VAPOR_ISOBARIC_SPECIFIC_HEAT - WATER_ISOBARIC_SPECIFIC_HEAT) * wetBulbTemperature);
  a += DRYAIR_ISOBARIC_SPECIFIC_HEAT * wetBulbTemperature;
  a += (WATER_ISOBARIC_SPECIFIC_HEAT * wetBulbTemperature - VAPORIZATION_LATENT_HEAT) * humidityRatio;
  double b = VAPOR_ISOBARIC_SPECIFIC_HEAT * humidityRatio + DRYAIR_ISOBARIC_SPECIFIC_HEAT;
  return a / b;
}

/// <summary>乾球温度[℃]と絶対湿度[kg/kg]から湿球温度[℃]を求める</summary>
/// <param name="dryBulbTemperature">乾球温度[℃]</param>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]：1 気圧は 101.325[kPa]</param>
/// <returns>湿球温度[℃]</returns>
public static double GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio
  (double dryBulbTemperature, double humidityRatio, double atmosphericPressure)
{
  Roots.ErrorFunction eFnc = delegate (double wbt)
  {
    return humidityRatio - GetHumidityRatioFromDryBulbTemperatureAndWetBulbTemperature
    (dryBulbTemperature, wbt, atmosphericPressure);
  };
  return Roots.Newton(eFnc, dryBulbTemperature, 1e-5, 1e-7, 1e-4, 20);
}
```

湿球温度 θ と比エンタルピー h から乾球温度 t を求める際にも反復計算が必要となる。入力条件である湿球温度を T_1'、比エンタルピーを H_1 とする。この時、誤差関数を式 4.17 で定義し、この誤差関数が 0 をとるようにニュートン・ラプソン法で反復計算を行う。ここで、$f_2(\mathrm{H}_1, t)$ は比エンタルピーと乾球温度から絶対湿度を計算する式 4.6 である。また、$f_3(\mathrm{W}_1, \mathrm{T}_1')$ は絶対湿度と湿球温度から乾球温度を計算する式 4.10 である。初期値 $t=\mathrm{T}_1$ とおき、誤差関数 $err_2(t)$が 0 となるように、t の値を調整する。許容誤差の考え方はプログラム 4.5 と同じである。計算処理をプログラム 4.6 に示す。

$$\begin{aligned}&\mathrm{W}_1=f_2(\mathrm{H}_1,t)\\&err_2(t)=t-f_3(\mathrm{W}_1,\mathrm{T}_1')\end{aligned} \tag{4.17}$$

プログラム **4.6**　湿球温度 θ と比エンタルピー h から乾球温度 t を求めるプログラム

Popolo.ThermophysicalProperty.MoistAir class

```
/// <summary>エンタルピー[kJ/kg]と湿球温度[C]から乾球温度[C]を計算する</summary>
/// <param name="enthalpy">エンタルピー[kJ/kg]</param>
/// <param name="wetBulbTemperature">湿球温度[C]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>乾球温度[C]</returns>
public static double GetDryBulbTemperatureFromWetBulbTemperatureAndEnthalpy
  (double wetBulbTemperature, double enthalpy, double atmosphericPressure)
{
  Roots.ErrorFunction eFnc = delegate (double dbt)
  {
    double hrt = GetHumidityRatioFromDryBulbTemperatureAndEnthalpy(dbt, enthalpy);
    return hrt - GetHumidityRatioFromDryBulbTemperatureAndWetBulbTemperature
    (dbt, wetBulbTemperature, atmosphericPressure);
  };
  return Roots.Newton(eFnc, wetBulbTemperature, 1e-5, 1e-7, 1e-4, 20);
}
```

4.3.5　その他の状態値の計算

プログラム 4.4、4.5、4.6 を組み合わせれば、乾球温度 t、湿球温度 θ、絶対湿度 W、比エンタルピー h の 4 種類の状態値の内、2 種類を与えることで他の 2 種類の状態値が計算できる。2 種類の状態値の組み合わせの数は ${}_4C_2$ で 6 通りあり、各々について残りの 2 種類の状態値を計算する関数が定義されるため、全体では 6×2=12 の関数がある。この内の 7 つの関数はプログラム 4.4、4.5、4.6 において提示した。残りの 5 つの状態値計算プログラムを 4.7 に示す。

プログラム **4.7**　t, θ, W, h の状態値相互変換プログラム

Popolo.ThermophysicalProperty.MoistAir class

```
/// <summary>エンタルピー[kJ/kg]と湿球温度[C]から絶対湿度[kg/kg]を計算する</summary>
/// <param name="enthalpy">エンタルピー[kJ/kg]</param>
/// <param name="wetBulbTemperature">湿球温度[C]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>絶対湿度[kg/kg]</returns>
public static double GetHumidityRatioFromWetBulbTemperatureAndEnthalpy
  (double wetBulbTemperature, double enthalpy, double atmosphericPressure)
{
  double dbt = GetDryBulbTemperatureFromWetBulbTemperatureAndEnthalpy
    (wetBulbTemperature, enthalpy, atmosphericPressure);
  return GetHumidityRatioFromDryBulbTemperatureAndEnthalpy(dbt, enthalpy);
}

/// <summary>乾球温度[C]とエンタルピー[kJ/kg]から湿球温度[C]を計算する</summary>
/// <param name="dryBulbTemperature">乾球温度[C]</param>
/// <param name="enthalpy">エンタルピー[kJ/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>湿球温度[C]</returns>
public static double GetWetBulbTemperatureFromDryBulbTemperatureAndEnthalpy
  (double dryBulbTemperature, double enthalpy, double atmosphericPressure)
{
  double hrt = GetHumidityRatioFromDryBulbTemperatureAndEnthalpy(dryBulbTemperature, enthalpy);
  return GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio
    (dryBulbTemperature, hrt, atmosphericPressure);
}

/// <summary>絶対湿度[kg/kg]とエンタルピー[kJ/kg]から湿球温度[C]を計算する</summary>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="enthalpy">エンタルピー[kJ/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>湿球温度[C]</returns>
public static double GetWetBulbTemperatureFromHumidityRatioAndEnthalpy
  (double humidityRatio, double enthalpy, double atmosphericPressure)
{
  double dbt = GetDryBulbTemperatureFromHumidityRatioAndEnthalpy(humidityRatio, enthalpy);
  return GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio
    (dbt, humidityRatio, atmosphericPressure);
}

/// <summary>乾球温度[C]と湿球温度[C]からエンタルピー[kJ/kg]を計算する</summary>
/// <param name="dryBulbTemperature">乾球温度[C]</param>
/// <param name="wetBulbTemperature">湿球温度[C]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>エンタルピー[kJ/kg]</returns>
public static double GetEnthalpyFromDryBulbTemperatureAndWetBulbTemperature
```

```
  (double dryBulbTemperature, double wetBulbTemperature, double atmosphericPressure)
{
  double hrt = GetHumidityRatioFromDryBulbTemperatureAndWetBulbTemperature
    (dryBulbTemperature, wetBulbTemperature, atmosphericPressure);
  return GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(dryBulbTemperature, hrt);
}

/// <summary>絶対湿度[kg/kg]と湿球温度[C]からエンタルピー[kJ/kg]を計算する</summary>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="wetBulbTemperature">湿球温度[C]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>エンタルピー[kJ/kg]</returns>
public static double GetEnthalpyFromHumidityRatioAndWetBulbTemperature
  (double humidityRatio, double wetBulbTemperature, double atmosphericPressure)
{
  double dbt = GetDryBulbTemperatureFromHumidityRatioAndWetBulbTemperature
    (humidityRatio, wetBulbTemperature, atmosphericPressure);
  return GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(dbt, humidityRatio);
}
```

相対湿度 ϕ に関しては、式 3.1~3.3の飽和水蒸気圧に関する式ならびに式 4.1 および式 4.2 を用いることで乾球温度 t と絶対湿度 W との間で相互に直接に変換が可能である。プログラムを 4.8 に示す。

プログラム 4.8　乾球温度 t、絶対湿度 W、相対湿度 ϕ の計算

Popolo.ThermophysicalProperty.MoistAir class

```
/// <summary>乾球温度[C]と相対湿度[%]から絶対湿度[kg/kg]を求める</summary>
/// <param name="dryBulbTemperature">乾球温度[C]</param>
/// <param name="relativeHumidity">相対湿度[%]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>絶対湿度[kg/kg]</returns>
public static double GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
  (double dryBulbTemperature, double relativeHumidity, double atmosphericPressure)
{
  //飽和水蒸気分圧[kPa]の計算
  double ps = Water.GetSaturationPressure(dryBulbTemperature);
  //水蒸気分圧[kPa]の計算
  double pw = 0.01 * relativeHumidity * ps;
  return GetHumidityRatioFromWaterVaporPartialPressure(pw, atmosphericPressure);
}

/// <summary>乾球温度[C]と絶対湿度[kg/kg]から相対湿度[%]を求める</summary>
/// <param name="dryBulbTemperature">乾球温度[C]</param>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>相対湿度[%]</returns>
public static double GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
  (double dryBulbTemperature, double humidityRatio, double atmosphericPressure)
{
  //水蒸気分圧[kPa]の計算
  double pw = GetWaterVaporPartialPressureFromHumidityRatio(humidityRatio, atmosphericPressure);
  //飽和水蒸気圧[kPa]の計算
  double ps = Water.GetSaturationPressure(dryBulbTemperature);
  if (ps <= 0.0) return 0.0;
  else return 100.0 * pw / ps;
}

/// <summary>絶対湿度[kg/kg]と相対湿度[%]から乾球温度[C]を求める</summary>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="relativeHumidity">相対湿度[%]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>乾球温度[C]</returns>
public static double GetDryBulbTemperatureFromHumidityRatioAndRelativeHumidity
  (double humidityRatio, double relativeHumidity, double atmosphericPressure)
{
  double ps = GetWaterVaporPartialPressureFromHumidityRatio(humidityRatio, atmosphericPressure);
  return Water.GetSaturationTemperature(ps / relativeHumidity * 100);
}
```

4.4~4.7に示したプログラムによって、乾球温度 t、絶対湿度 W、湿球温度 θ、比エンタルピー h の状態値は相互に計算可能である。また、4.8 に示したプログラムにより、乾球温度 t、絶対湿度 W、相対湿度 ϕ の計算も可能である。従って、これらの関数を組み合わせれば、入力条件が「湿球温度 θ と相対湿度 ϕ」あるいは「比エンタルピー h と相対湿度 ϕ」の組み合わせである場合を除き、全ての計算が可能である。プログラムを 4.9 に示す。

プログラム **4.9**　相対湿度 ϕ に関連する物性値の計算 **1**

Popolo.ThermophysicalProperty.MoistAir class

```
/// <summary>乾球温度[C]と湿球温度[C]から相対湿度[%]を求める</summary>
/// <param name="dryBulbTemperature">乾球温度[C]</param>
/// <param name="wetBulbTemperature">湿球温度[C]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>相対湿度[%]</returns>
public static double GetRelativeHumidityFromDryBulbTemperatureAndWetBulbTemperature
  (double dryBulbTemperature, double wetBulbTemperature, double atmosphericPressure)
{
  double hrt = GetHumidityRatioFromDryBulbTemperatureAndWetBulbTemperature
    (dryBulbTemperature, wetBulbTemperature, atmosphericPressure);
  return GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
    (dryBulbTemperature, hrt, atmosphericPressure);
}

/// <summary>湿球温度[C]と絶対湿度[kg/kg]から相対湿度[%]を求める</summary>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="wetBulbTemperature">湿球温度[C]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>相対湿度[%]</returns>
public static double GetRelativeHumidityFromWetBulbTemperatureAndHumidityRatio
  (double humidityRatio, double wetBulbTemperature, double atmosphericPressure)
{
  double dbt = GetDryBulbTemperatureFromHumidityRatioAndWetBulbTemperature
    (humidityRatio, wetBulbTemperature, atmosphericPressure);
  return GetRelativeHumidityFromDryBulbTemperatureAndWetBulbTemperature
    (dbt, wetBulbTemperature, atmosphericPressure);
}

/// <summary>乾球温度[C]とエンタルピー[kJ/kg]から相対湿度[%]を求める</summary>
/// <param name="dryBulbTemperature">乾球温度[C]</param>
/// <param name="enthalpy">エンタルピー[kJ/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>相対湿度[%]</returns>
public static double GetRelativeHumidityFromDryBulbTemperatureAndEnthalpy
  (double dryBulbTemperature, double enthalpy, double atmosphericPressure)
{
  double hrt = GetHumidityRatioFromDryBulbTemperatureAndEnthalpy(dryBulbTemperature, enthalpy);
  return GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
    (dryBulbTemperature, hrt, atmosphericPressure);
}

/// <summary>絶対湿度[kg/kg]とエンタルピー[kJ/kg]から相対湿度[%]を求める</summary>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="enthalpy">エンタルピー[kJ/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>相対湿度[%]</returns>
public static double GetRelativeHumidityFromHumidityRatioAndEnthalpy
  (double humidityRatio, double enthalpy, double atmosphericPressure)
{
  double dbt = GetDryBulbTemperatureFromHumidityRatioAndEnthalpy(humidityRatio, enthalpy);
  return GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
    (dbt, humidityRatio, atmosphericPressure);
}

/// <summary>乾球温度[C]と相対湿度[%]から湿球温度[C]を求める</summary>
/// <param name="dryBulbTemperature">乾球温度[C]</param>
/// <param name="relativeHumidity">相対湿度[%]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>湿球温度[C]</returns>
public static double GetWetBulbTemperatureFromDryBulbTemperatureAndRelativeHumidity
  (double dryBulbTemperature, double relativeHumidity, double atmosphericPressure)
{
  double hrt = GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
    (dryBulbTemperature, relativeHumidity, atmosphericPressure);
  return GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio
    (dryBulbTemperature, hrt, atmosphericPressure);
}

/// <summary>絶対湿度[kg/kg]と相対湿度[%]から湿球温度[C]を求める</summary>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="relativeHumidity">相対湿度[%]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>湿球温度[C]</returns>
public static double GetWetBulbTemperatureFromHumidityRatioAndRelativeHumidity
  (double humidityRatio, double relativeHumidity, double atmosphericPressure)
{
  double dbt = GetDryBulbTemperatureFromHumidityRatioAndRelativeHumidity
    (humidityRatio, relativeHumidity, atmosphericPressure);
```

```
 79   return GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio
 80     (dbt, humidityRatio, atmosphericPressure);
 81 }
 82
 83 /// <summary>乾球温度[C]と相対湿度[%]からエンタルピー[kJ/kg]を求める</summary>
 84 /// <param name="dryBulbTemperature">乾球温度[C]</param>
 85 /// <param name="relativeHumidity">相対湿度[%]</param>
 86 /// <param name="atmosphericPressure">大気圧[kPa]</param>
 87 /// <returns>エンタルピー[kJ/kg]</returns>
 88 public static double GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
 89   (double dryBulbTemperature, double relativeHumidity, double atmosphericPressure)
 90 {
 91   double hrt = GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
 92     (dryBulbTemperature, relativeHumidity, atmosphericPressure);
 93   return GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(dryBulbTemperature, hrt);
 94 }
 95
 96 /// <summary>絶対湿度[kg/kg]と相対湿度[%]からエンタルピー[kJ/kg]を求める</summary>
 97 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
 98 /// <param name="relativeHumidity">相対湿度[%]</param>
 99 /// <param name="atmosphericPressure">大気圧[kPa]</param>
100 /// <returns>エンタルピー[kJ/kg]</returns>
101 public static double GetEnthalpyFromHumidityRatioAndRelativeHumidity
102   (double humidityRatio, double relativeHumidity, double atmosphericPressure)
103 {
104   double dbt = GetDryBulbTemperatureFromHumidityRatioAndRelativeHumidity
105     (humidityRatio, relativeHumidity, atmosphericPressure);
106   return GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(dbt, humidityRatio);
107 }
```

「湿球温度 θ と相対湿度 ϕ」あるいは「比エンタルピー h と相対湿度 ϕ」が入力として与えられている場合には、ニュートン・ラフソン法などによる収束計算が必要となる。湿球温度 θ と相対湿度 ϕ から各種の物性値を計算するためのプログラムを 4.10 に示す。

プログラム 4.10　相対湿度 ϕ に関連する物性値の計算 2

Popolo.ThermophysicalProperty.MoistAir class

```
  1 /// <summary>湿球温度[C]と相対湿度[%]から乾球温度[C]を計算する</summary>
  2 /// <param name="wetBulbTemperature">湿球温度[C]</param>
  3 /// <param name="relativeHumidity">相対湿度[%]</param>
  4 /// <param name="atmosphericPressure">大気圧[kPa]</param>
  5 /// <returns>乾球温度[C]</returns>
  6 public static double GetDryBulbTemperatureFromWetBulbTemperatureAndRelativeHumidity
  7   (double wetBulbTemperature, double relativeHumidity, double atmosphericPressure)
  8 {
  9   Roots.ErrorFunction eFnc = delegate (double dbt)
 10   {
 11     double hrt = GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
 12     (dbt, relativeHumidity, atmosphericPressure);
 13     return dbt - GetDryBulbTemperatureFromHumidityRatioAndWetBulbTemperature
 14       (hrt, wetBulbTemperature, atmosphericPressure);
 15   };
 16   return Roots.Newton(eFnc, wetBulbTemperature, 1e-5, 1e-4, 1e-4, 20);
 17 }
 18
 19 /// <summary>湿球温度[C]と相対湿度[%]から絶対湿度[kg/kg]を計算する</summary>
 20 /// <param name="wetBulbTemperature">湿球温度[C]</param>
 21 /// <param name="relativeHumidity">相対湿度[%]</param>
 22 /// <param name="atmosphericPressure">大気圧[kPa]</param>
 23 /// <returns>絶対湿度[kg/kg]</returns>
 24 public static double GetHumidityRatioFromWetBulbTemperatureAndRelativeHumidity
 25   (double wetBulbTemperature, double relativeHumidity, double atmosphericPressure)
 26 {
 27   double dbt = GetDryBulbTemperatureFromWetBulbTemperatureAndRelativeHumidity
 28     (wetBulbTemperature, relativeHumidity, atmosphericPressure);
 29   return GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
 30     (dbt, relativeHumidity, atmosphericPressure);
 31 }
 32
 33 /// <summary>湿球温度[C]と相対湿度[%]から比エンタルピー[kJ/kg]を計算する</summary>
 34 /// <param name="wetBulbTemperature">湿球温度[C]</param>
 35 /// <param name="relativeHumidity">相対湿度[%]</param>
 36 /// <param name="atmosphericPressure">大気圧[kPa]</param>
 37 /// <returns>比エンタルピー[kJ/kg]</returns>
 38 public static double GetEnthalpyFromWetBulbTemperatureAndRelativeHumidity
 39   (double wetBulbTemperature, double relativeHumidity, double atmosphericPressure)
 40 {
 41   double dbt = GetDryBulbTemperatureFromWetBulbTemperatureAndRelativeHumidity
```

```
42     (wetBulbTemperature, relativeHumidity, atmosphericPressure);
43   return GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
44     (dbt, relativeHumidity, atmosphericPressure);
45 }
```

比エンタルピー h と相対湿度 ϕ から各種の物性値を計算するためのプログラムを 4.11 に示す。

プログラム 4.11　相対湿度 ϕ に関連する物性値の計算 3

Popolo.ThermophysicalProperty.MoistAir class

```
 1 /// <summary>比エンタルピー[kJ/kg]と相対湿度[%]から乾球温度[C]を計算する</summary>
 2 /// <param name="enthalpy">比エンタルピー[kJ/kg]</param>
 3 /// <param name="relativeHumidity">相対湿度[%]</param>
 4 /// <param name="atmosphericPressure">大気圧[kPa]</param>
 5 /// <returns>乾球温度[C]</returns>
 6 public static double GetDryBulbTemperatureFromEnthalpyAndRelativeHumidity
 7   (double enthalpy, double relativeHumidity, double atmosphericPressure)
 8 {
 9   Roots.ErrorFunction eFnc = delegate (double dbt)
10   {
11     return enthalpy - GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
12     (dbt, relativeHumidity, atmosphericPressure);
13   };
14   return Roots.Newton(eFnc, 25, 1e-5, 1e-4, 1e-4, 20);
15 }
16
17 /// <summary>比エンタルピー[kJ/kg]と相対湿度[%]から絶対湿度[kg/kg]を計算する</summary>
18 /// <param name="enthalpy">比エンタルピー[kJ/kg]</param>
19 /// <param name="relativeHumidity">相対湿度[%]</param>
20 /// <param name="atmosphericPressure">大気圧[kPa]</param>
21 /// <returns>絶対湿度[kg/kg]</returns>
22 public static double GetHumidityRatioFromEnthalpyAndRelativeHumidity
23   (double enthalpy, double relativeHumidity, double atmosphericPressure)
24 {
25   double dbt = GetDryBulbTemperatureFromEnthalpyAndRelativeHumidity
26     (enthalpy, relativeHumidity, atmosphericPressure);
27   return GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
28     (dbt, relativeHumidity, atmosphericPressure);
29 }
30
31 /// <summary>比エンタルピー[kJ/kg]と相対湿度[%]から湿球温度[C]を計算する</summary>
32 /// <param name="enthalpy">比エンタルピー[kJ/kg]</param>
33 /// <param name="relativeHumidity">相対湿度[%]</param>
34 /// <param name="atmosphericPressure">大気圧[kPa]</param>
35 /// <returns>湿球温度[C]</returns>
36 public static double GetWetBulbTemperatureFromEnthalpyAndRelativeHumidity
37   (double enthalpy, double relativeHumidity, double atmosphericPressure)
38 {
39   double dbt = GetDryBulbTemperatureFromEnthalpyAndRelativeHumidity
40     (enthalpy, relativeHumidity, atmosphericPressure);
41   return GetWetBulbTemperatureFromDryBulbTemperatureAndRelativeHumidity
42     (dbt, relativeHumidity, atmosphericPressure);
43 }
```

プログラム 4.4~4.11を使用することで、5 種類の物性値（乾球温度 t、湿球温度 θ、絶対湿度 W、比エンタルピー h、相対湿度 ϕ）の内、2 種類の物性値から残りの 3 種類の物性値の計算を行うことが可能である。比体積 v に関しても同様の方法で物性値相互の変換が可能であるが、関数の数が多くなるため、ここでは記載しない。式 4.5 にもとづいて、乾球温度、絶対湿度、比体積の相互の変換を行うプログラムを 4.12 に示す。

プログラム 4.12　乾球温度、相対湿度、比体積の計算

Popolo.ThermophysicalProperty.MoistAir class

```
 1 /// <summary>乾球温度[C]および絶対湿度[kg/kg]から比体積[m3/kg]を求める</summary>
 2 /// <param name="dryBulbTemperature">乾球温度[C]</param>
 3 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
 4 /// <param name="atmosphericPressure">大気圧[kPa]</param>
 5 /// <returns>比体積[m3/kg]</returns>
 6 public static double GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
 7   (double dryBulbTemperature, double humidityRatio, double atmosphericPressure)
 8 {
 9   return ((dryBulbTemperature + CONVERT_C_TO_K) * DRYAIR_GAS_CONSTANT) /
10     atmosphericPressure * (1.0 + 1.6078 * humidityRatio);
11 }
12
```

```
/// <summary>比体積[m3/kg]および絶対湿度[kg/kg]から乾球温度[C]を求める</summary>
/// <param name="specificVolume">比体積[m3/kg]</param>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>乾球温度[C]</returns>
public static double GetDryBulbTemperatureFromSpecificVolumeAndHumidityRatio
  (double specificVolume, double humidityRatio, double atmosphericPressure)
{
  return specificVolume / (1.0 + 1.6078 * humidityRatio) *
    atmosphericPressure / DRYAIR_GAS_CONSTANT - CONVERT_C_TO_K;
}

/// <summary>乾球温度[C]および比体積[m3/kg]から絶対湿度[kg/kg]を求める</summary>
/// <param name="dryBulbTemperature">乾球温度[C]</param>
/// <param name="specificVolume">比体積[m3/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>絶対湿度[kg/kg]</returns>
public static double GetHumidityRatioFromDryBulbTemperatureAndSpecificVolume
  (double dryBulbTemperature, double specificVolume, double atmosphericPressure)
{
  return (specificVolume / (dryBulbTemperature + CONVERT_C_TO_K)
    / DRYAIR_GAS_CONSTANT * atmosphericPressure - 1.0) / 1.6078;
}
```

4.2.7 節で解説を行ったその他の物性計算プログラムを 4.13 に示す。

プログラム 4.13　その他の物性の計算

Popolo.ThermophysicalProperty.MoistAir class

```
/// <summary>標高[m]に応じた大気圧[kPa]を取得する</summary>
/// <param name="altitude">標高[m]</param>
/// <returns>大気圧[kPa]</returns>
public static double GetAtmosphericPressure(double altitude)
{ return ATMOSPHERIC_PRESSURE_AT_SEALEVEL * Math.Pow(1.0 - 2.25577e-5 * altitude, 5.2559); }

/// <summary>湿り空気比熱[kJ/kg-K]を計算する</summary>
/// <param name="humidityRatio">絶対湿度[kg/kg(DA)]</param>
/// <returns>湿り空気比熱[kJ/kg-K]</returns>
public static double GetSpecificHeat(double humidityRatio)
{ return DRYAIR_ISOBARIC_SPECIFIC_HEAT + VAPOR_ISOBARIC_SPECIFIC_HEAT * humidityRatio; }

/// <summary>粘性係数[Pa s]を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[C]</param>
/// <returns>粘性係数[Pa s]</returns>
public static double GetViscosity(double drybulbTemperature)
{
  return (0.0074237 / (drybulbTemperature + 390.15))
    * Math.Pow((drybulbTemperature + CONVERT_C_TO_K) / 293.15, 1.5);
}

/// <summary>動粘性係数[m2/s]を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[C]</param>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>動粘性係数[m2/s]</returns>
public static double GetDynamicViscosity
  (double drybulbTemperature, double humidityRatio, double atmosphericPressure)
{
  return GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
    (drybulbTemperature, humidityRatio, atmosphericPressure) * GetViscosity(drybulbTemperature);
}

/// <summary>熱伝導率[W/(mK)]を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[C]</param>
/// <returns>熱伝導率[W/(mK)]</returns>
public static double GetThermalConductivity(double drybulbTemperature)
{ return 0.0241 + 0.000077 * drybulbTemperature; }

/// <summary>膨張率[1/K]を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[C]</param>
/// <returns>膨張率[1/K]</returns>
public static double GetExpansionCoefficient(double drybulbTemperature)
{ return 1 / (drybulbTemperature + CONVERT_C_TO_K); }

/// <summary>熱拡散率[m2/s]を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[C]</param>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>熱拡散率[m2/s]</returns>
public static double GetThermalDiffusivity
  (double drybulbTemperature, double humidityRatio, double atmosphericPressure)
```

```
53 {
54   double lambda = GetThermalConductivity(drybulbTemperature);
55   double cp = GetSpecificHeat(humidityRatio);
56   double rho = GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
57     (drybulbTemperature, humidityRatio, atmosphericPressure);
58   return lambda / (1000 * cp * rho);
59 }
```

乾球温度 t、絶対湿度 W、比エンタルピー h が与えられた場合の飽和状態計算処理をプログラム 4.14 に示す。

プログラム 4.14　飽和状態の計算

Popolo.ThermophysicalProperty.MoistAir class

```
 1 /// <summary>絶対湿度[kg/kg]から飽和乾球温度（露点温度）[℃]を求める</summary>
 2 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
 3 /// <param name="atmosphericPressure">大気圧[kPa]</param>
 4 /// <returns>飽和乾球温度（露点温度）[℃]</returns>
 5 public static double GetSaturationDryBulbTemperatureFromHumidityRatio
 6   (double humidityRatio, double atmosphericPressure)
 7 {
 8   double ps = GetWaterVaporPartialPressureFromHumidityRatio(humidityRatio, atmosphericPressure);
 9   return Water.GetSaturationTemperature(ps);
10 }
11
12 /// <summary>エンタルピー[kJ/kg]から飽和乾球温度（露点温度）[℃]を求める</summary>
13 /// <param name="enthalpy">エンタルピー[kJ/kg]</param>
14 /// <param name="atmosphericPressure">大気圧[kPa]</param>
15 /// <returns>飽和乾球温度（露点温度）[℃]</returns>
16 public static double GetSaturationDryBulbTemperatureFromEnthalpy
17   (double enthalpy, double atmosphericPressure)
18 { return GetDryBulbTemperatureFromEnthalpyAndRelativeHumidity(enthalpy, 100, atmosphericPressure); }
19
20 /// <summary>乾球温度[℃]から飽和絶対湿度[kg/kg]を求める</summary>
21 /// <param name="dryBulbTemperature">乾球温度[℃]</param>
22 /// <param name="atmosphericPressure">大気圧[kPa]</param>
23 /// <returns>飽和絶対湿度[kg/kg]</returns>
24 public static double GetSaturationHumidityRatioFromDryBulbTemperature
25   (double dryBulbTemperature, double atmosphericPressure)
26 {
27   double ps = Water.GetSaturationPressure(dryBulbTemperature);
28   return GetHumidityRatioFromWaterVaporPartialPressure(ps, atmosphericPressure);
29 }
30
31 /// <summary>エンタルピー[kJ/kg]から飽和絶対湿度[kg/kg]を求める</summary>
32 /// <param name="enthalpy">エンタルピー[kJ/kg]</param>
33 /// <param name="atmosphericPressure">大気圧[kPa]</param>
34 /// <returns>飽和絶対湿度[kg/kg]</returns>
35 public static double GetSaturationHumidityRatioFromEnthalpy(double enthalpy, double atmosphericPressure)
36 { return GetHumidityRatioFromEnthalpyAndRelativeHumidity(enthalpy, 100, atmosphericPressure); }
37
38 /// <summary>乾球温度[℃]から飽和エンタルピー[kJ/kg]を求める</summary>
39 /// <param name="drybulbTemperature">飽和温度[℃]</param>
40 /// <param name="atmosphericPressure">大気圧[kPa]</param>
41 /// <returns>飽和エンタルピー[kJ/kg]</returns>
42 public static double GetSaturationEnthalpyFromDrybulbTemperature
43   (double drybulbTemperature, double atmosphericPressure)
44 {
45   double ps = Water.GetSaturationPressure(drybulbTemperature);
46   double ws = GetHumidityRatioFromWaterVaporPartialPressure(ps, atmosphericPressure);
47   return GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(drybulbTemperature, ws);
48 }
49
50 /// <summary>絶対湿度[kg/kg]から飽和エンタルピー[kJ/kg]を求める</summary>
51 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
52 /// <param name="atmosphericPressure">大気圧[kPa]</param>
53 /// <returns>飽和エンタルピー[kJ/kg]</returns>
54 public static double GetSaturationEnthalpyFromHuidityRatio
55   (double humidityRatio, double atmosphericPressure)
56 {
57   double ts = GetSaturationDryBulbTemperatureFromHumidityRatio(humidityRatio, atmosphericPressure);
58   return GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(ts, humidityRatio);
59 }
```

【例題 4.2】

本章で解説を行ったプログラムを使用して湿り空気線図を作成せよ。ただし、横軸には乾球温度 t、縦軸には絶対湿度 W をとる t-W 線図とする[†1]。

†1　設計で通常使われる湿り空気線図は、比エンタルピー h と絶対湿度 W を斜めの軸に取った斜交座標による h-W 線図である。

【解】

プログラムを 4.15 に示す。乾球温度を変化させて、いくつかの相対湿度、エンタルピー、湿球温度の状態値との組み合わせから絶対湿度を算出し、CSV 形式のファイルへの書き出し処理を行なっている。計算結果をグラフ化した結果を図 4.4 に示す。飽和線上の状態値を計算したり、斜交座標への変換処理を行ったり、細部ではプログラムの微修正が必要であるが、湿り空気線図の基本的な形状は表現できていることが確認できる。

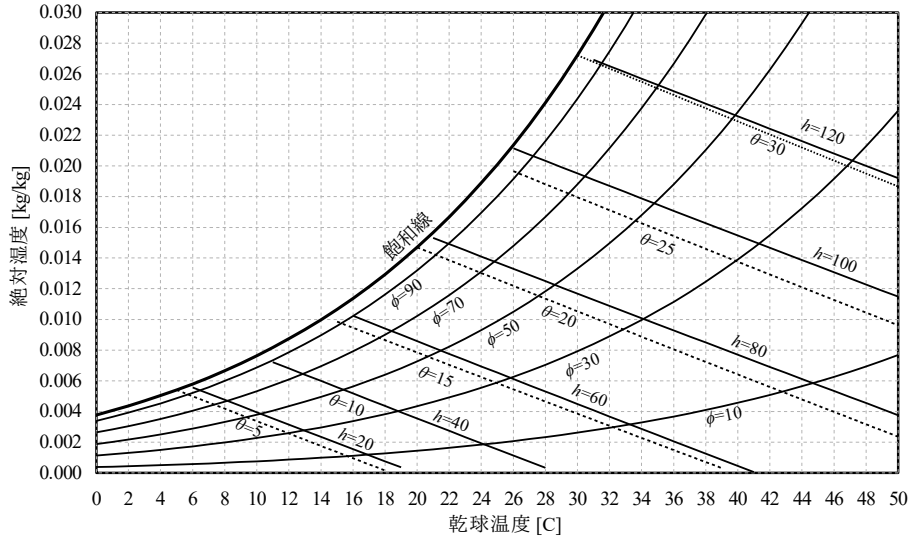


図 4.4　湿り空気線図作成プログラムの計算結果のグラフ化

プログラム **4.15**　湿り空気線図作成用 **CSV** データの作成プログラム

```
private static void moistAirTest()
{
  //標準気圧とする
  const double ATM = 101.325;

  //表計算ソフトでの描画に備え、計算結果を CSV 形式で書き出す
  using (StreamWriter sWriter = new StreamWriter("MoistAirTable.csv"))
  {
    //ヘッダ行書き出し
    sWriter.Write("T, SatT, RH10, RH30, RH50, RH70, RH90, ");
    sWriter.Write("H20, H40, H60, H80, H100, ");
    sWriter.WriteLine("WB5, WB10, WB15, WB20, WB25, WB30, WB35");

    //乾球温度は 0～50℃で 1℃刻みで描画する
    for (int t = 0; t <= 50; t++)
    {
      //飽和絶対湿度[kg/kg]の計算
      double ps = Water.GetSaturationPressure(t);
      double satW = MoistAir.GetHumidityRatioFromWaterVaporPartialPressure(ps, ATM);
      //結果書き出し
      sWriter.Write(t + "," + satW);

      //等相対湿度線を 10～90%の範囲で 20%刻みで描画する
      for (int i = 0; i < 5; i++)
      {
        //相対湿度[%]
        double rhd = 10 + i * 20;
        double rhdW = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity(t, rhd, ATM);
        if ((rhdW < satW) && (0 < rhdW)) sWriter.Write("," + rhdW);
        else sWriter.Write(",");
      }

      //等比エンタルピー線を 20～120kJ/kg の範囲で 20kJ/kg 刻みで描画する
      for (int i = 0; i < 5; i++)
      {
        //比エンタルピー[kJ/kg]
        double ent = 20 + i * 20;
        double entW = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndEnthalpy(t, ent);
        if ((entW < satW) && (0 < entW)) sWriter.Write("," + entW);
        else sWriter.Write(",");
      }
```

```
42
43        //等湿球温度線を 5～35℃の範囲で 5℃刻みで描画する
44        for (int i = 0; i < 7; i++)
45        {
46          //湿球温度[℃]
47          double wbt = 5 + i * 5;
48          double wbtW = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndWetBulbTemperature(t, wbt, ATM);
49          if ((wbtW < satW) && (0 < wbtW)) sWriter.Write("," + wbtW);
50          else sWriter.Write(",");
51        }
52        sWriter.WriteLine();
53      }
54    }
55 }
```

4.3.6　「湿り空気クラス」の作成

湿り空気の物性を一括して管理するために、湿り空気クラスを作成する。プロパティ定義およびコンストラクタをプログラム 4.16 に示す。22~38 行は標準のコンストラクタであり、乾球温度と絶対湿度を引数にして物性値を初期化する。引数を省略した場合には 41 行に示すように 24 ºC, 50 %で初期化する。43~54 行のコンストラクタは湿り空気のインスタンス（Immutable インターフェースを実装した読み取り専用）を受け取り、同じ内容の湿り空気を生成する。このようなコンストラクタを一般にコピーコンストラクタと呼ぶ。

プログラム 4.16　湿り空気クラスのプロパティおよびコンストラクタ

Popolo.ThermophysicalProperty.MoistAir class

```
 1 /// <summary>乾球温度[C]を設定・取得する</summary>
 2 public double DrybulbTemperature { get; set; }
 3
 4 /// <summary>湿球温度[C]を設定・取得する</summary>
 5 public double WetbulbTemperature { get; set; }
 6
 7 /// <summary>絶対湿度[kg/kg(DA)]を設定・取得する</summary>
 8 public double HumidityRatio { get; set; }
 9
10 /// <summary>相対湿度[%]を設定・取得する</summary>
11 public double RelativeHumidity { get; set; }
12
13 /// <summary>エンタルピー[kJ/kg]を設定・取得する</summary>
14 public double Enthalpy { get; set; }
15
16 /// <summary>比容積[m3/kg]を設定・取得する</summary>
17 public double SpecificVolume { get; set; }
18
19 /// <summary>大気圧[kPa]を設定・取得する</summary>
20 public double AtmosphericPressure { get; set; }
21
22 /// <summary>インスタンスを初期化する</summary>
23 /// <param name="drybulbTemperature">乾球温度[C]</param>
24 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
25 public MoistAir(double drybulbTemperature, double humidityRatio)
26 {
27   this.AtmosphericPressure = ATMOSPHERIC_PRESSURE_AT_SEALEVEL;
28   this.DrybulbTemperature = drybulbTemperature;
29   this.HumidityRatio = humidityRatio;
30   this.RelativeHumidity = GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
31     (drybulbTemperature, humidityRatio, ATMOSPHERIC_PRESSURE_AT_SEALEVEL);
32   this.Enthalpy = GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
33     (drybulbTemperature, humidityRatio);
34   this.WetbulbTemperature = GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio
35     (drybulbTemperature, humidityRatio, ATMOSPHERIC_PRESSURE_AT_SEALEVEL);
36   this.SpecificVolume = GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
37     (drybulbTemperature, humidityRatio, ATMOSPHERIC_PRESSURE_AT_SEALEVEL);
38 }
39
40 /// <summary>インスタンスを初期化する</summary>
41 public MoistAir() : this(24, 0.0093) { }
42
43 /// <summary>コピーコンストラクタ</summary>
44 /// <param name="moistAir">コピーする湿り空気状態</param>
45 public MoistAir(ImmutableMoistAir moistAir)
46 {
47   this.AtmosphericPressure = moistAir.AtmosphericPressure;
48   this.DrybulbTemperature = moistAir.DrybulbTemperature;
```

```
49   this.HumidityRatio = moistAir.HumidityRatio;
50   this.RelativeHumidity = moistAir.RelativeHumidity;
51   this.Enthalpy = moistAir.Enthalpy;
52   this.WetbulbTemperature = moistAir.WetbulbTemperature;
53   this.SpecificVolume = moistAir.SpecificVolume;
54 }
```

【第４章 記号表】

c_{pa}	：乾き空気の定圧比熱 **[kJ/(kg·K)]**	Z	：標高 **[m]**
c_{pma}	：湿り空気の定圧比熱 **[kJ/(kg·K)]**	β	：膨張率 **[1/K]**
c_{pv}	：水蒸気の定圧比熱 **[kJ/(kg·K)]**	λ	：熱伝導率 **[W/m·K]**
h	：比エンタルピー **[kJ/kg]**	ϕ	：相対湿度 **[%]**
P_{Ws}	：飽和水蒸気圧 **[kPa]**	φ	：飽和度 **[%]**
P_s	：水蒸気分圧 **[kPa]**	υ	：比体積 **[m^3/kg]**
P	：大気圧 **[kPa]**	μ	：粘性係数 **[Ps·s]**
R_a	：空気のガス定数 **(0.287055 kJ/(kg·K))**	γ_0	：**0 °C** の水の蒸発潜熱 **[kJ/kg]**
t	：乾球温度 **[°C]**	θ	：湿球温度 **[°C]**
W	：絶対湿度 **[kg/kg]**	ν	：動粘性係数 **[m^2/s]**
Ws	：飽和絶対湿度 **[kg/kg]**		

【第４章 参考文献】

4.1） American Society of Heating, Refrigerating and Air-Conditioning Engineers. ASHRAE Fundamentals Handbook, Chapter 6 Psychrometrics, pp.6.1~6.17, 1997

4.2） 宇田川光弘：パソコンによる空気調和計算法, 第３章 湿り空気の状態値, pp26~43. オーム社, 1986

4.3） 機械工学便覧, α 基礎編, α5 熱力学, p.α5-32, 33, 2007

4.4） Wexler, A., Hyland, R. W. : Final Report, ASHRAE Research Project RP 216 (1980), Part 1, 2, 3

4.5） Threlkeld, J.L. : Thermal Environmental Engineering (2nd Ed.), Prentice-Hall, 1970, pp.165~194

4.6） ASHRAE Brochure on Psychrometry, ASHRAE, 1977

4.7） Standard Atmosphere – Table and Data for Altitudes to 65000 Feet (NACA Report 1235, Washington, DC, 1955)

4.8） 手塚俊一, 藤田稔彦：湿り空気線図とその応用（1）~（6）, 空気調和・衛生工学, 57-12 号 ~ 58-5 号

4.9） Hirschfelder, J. O., Charles F. Curtiss, R. Byron Bird: Molecular Theory of Gases and Liquids, Rev. Ed., John Wiley & Sons, New York, 1964

4.10） Barry Donaldson, Bernard Nagengast, Gershon Meckler: Heat & Cold Mastering the Great Indoors, A selective History of Heating, Ventilation, Refrigeration & Air Conditioning, American Society of Heating, Refrigerating and Air Conditioning Engineers, p.280, Fig.11-40

第5章　冷媒の物性
(Thermodynamic Properties of Refrigerant)

5.1　概要

冷媒という言葉は、広義には冷凍サイクルの中で熱を伝達する物質を意味するが、狭義には特に圧縮式冷凍サイクルにおける熱媒を指す。いくつもの種類があり、ターボ冷凍機用の冷媒としては HFC-134a や HFC-245fa、パッケージ空調機用の冷媒としては R407a、R410a、R32 などが使われる。多くの冷媒はオゾン層破壊や地球温暖化に繋がる物質で構成されているため、環境負荷の低減を目的に段階的に冷媒の転換が進められており、計算対象となる熱源がどの冷媒を用いているのかは気に留める必要がある。

熱源の計算方法の詳細は後章で述べるが、特性式を使った単純なモデルであれば冷媒の物性値まで把握する必要はない。HASP/ACSS を始め、多くのエネルギーシミュレーションプログラムでは冷媒の物性値計算を行わない特性式による手法を採用している。特性式を用いたモデルと物理式によるモデルは、各々に長所と短所があり、検討の内容に応じて使い分けが必要であるが、物理式によるモデルを採用する場合には冷媒の物性値まで遡った計算が必要となる場合が多い。本章の冷媒物性値計算法は物理モデルを作成するための基礎である。冷媒も湿り空気と同様に 2 種類の状態値を特定すれば他の状態値が唯一つに定まる。本書では、空調分野の冷凍サイクルの計算において必要となることの多い、温度、圧力、比体積、比エントロピー、比エンタルピーの計算方法について記す。

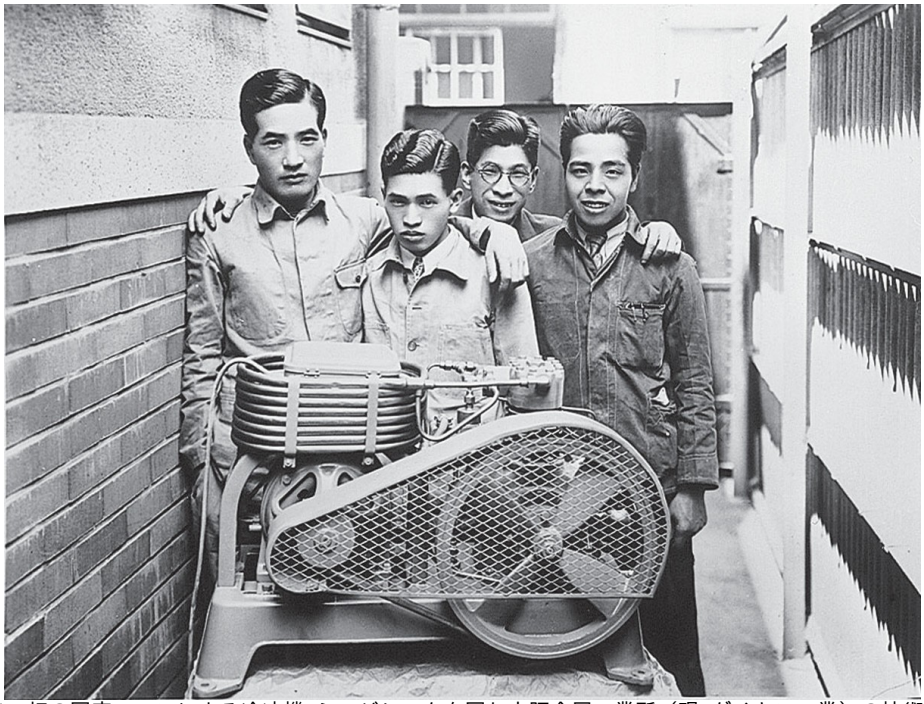


写真 5.1　初の国産フロンによる冷凍機 ミフジレータを囲む大阪金属工業所（現 ダイキン工業）の技術者達[5.18)]

5.2 理論

5.2.1 物性値相互の関係

1）理想気体の状態方程式

理想気体の圧力 P [kPa]、温度 T [K]、比体積 υ [m³/kg]の関係は式 5.1 で表現でき、ボイル-シャルルの法則、ゲイリュサックの法則、アボガドロの法則に従う。ここで n は物質量[mol/kg]、$\underline{R}$は一般気体定数 8.3144621 J/(K·mol)である。物質量 n と一般気体定数 $\underline{R}$ を乗じた数値を各気体に固有の気体定数 R として、式 5.2 で表現することもある。

$$P\upsilon = n\underline{R}T \tag{5.1}$$

$$P\upsilon = RT \tag{5.2}$$

理想気体の状態方程式は、分子の体積と分子間力が無いと仮定した場合の式であり、これらの仮定が成り立つとみなせる高温（分子の運動が大きく、相対的に分子間力が小さい）・低圧（空間内で分子の占める体積が小さい）の条件下では気体の状態を精度よく表現することができる。4.2 節で説明した湿り空気の各種の物性値の計算式も、理想気体であることを前提に導き出されたものである。一方で、湿り空気とは異なり、冷凍機内の冷媒は高圧や低温にもなりうるため、理想気体の状態方程式で表現することは精度の点で問題がある。

2）残存量（Residual Property）

高温低圧以外の条件下における気体（実在気体）の物性値は理想気体の状態方程式による計算結果とは異なった値をとり、この乖離量を残存量（Residual Property）と呼ぶ。例えば実在気体の圧力 P を温度 T と比体積 υ の関数とし、理想気体の物性値と残存量をそれぞれ上付き文字の「°」と「^{r}」で表現すると、式 5.3 に示すように、実在気体の状態量は理想気体の状態値と残存量の和として表すことができる。式変形をすると式 5.4 が得られる。

$$P(\upsilon, T) = P^{\circ} + P^{r} \tag{5.3}$$

$$P^{r} = P(\upsilon, T) - P^{\circ} = P(\upsilon, T) - \rho RT \tag{5.4}$$

圧力 P、温度 T、比体積 υ については式 5.4 を解くことで相互の変換が可能である。また、残存比エンタルピーと残存比エントロピーに関しては式 5.3 の微積分操作により求めることができる。まず、式 5.4 を比体積 υ について積分すると残存ヘルムホルツエネルギー A^r が得られる。ここで、ρ は密度[kg/m³]であり比体積 υ [m³/kg]の逆数である。

$$A^{r} = A - A^{\circ} = \int_{\upsilon}^{\infty} \left(P(\upsilon, T) - \rho RT\right) d\upsilon \tag{5.5}$$

残存比エントロピー s^r は、式 5.5 の温度 T に関しての微分値である。

$$s^{r} = s - s^{\circ} = -\left(\frac{\partial\,(A - A^{\circ})}{\partial\,T}\right)_{\upsilon} \tag{5.6}$$

残存比エンタルピー h^r は、残存比エントロピーと残存ヘルムホルツエネルギー A^r を用いて式 5.7 で表現できる。

$$h^{r} = h - h^{\circ} = (A - A^{\circ}) + T(s^{r} - s^{\circ}) + (P/\rho - RT) \tag{5.7}$$

残存ギブスエネルギー G^r は、残存ヘルムホルツエネルギー A^r を用いて式 5.8 で求めることができ

る。後述するが、残存ギブスエネルギー G^r は液相と気相の判定計算の際に使用する。

$$G^r = G - G^\circ = (A - A^\circ) + (P/\rho - RT) \tag{5.8}$$

3）比エントロピーと比エンタルピー

比エンタルピーは式 5.9 で計算できる。ここで、添字の *ref* は基準状態を示しており、T_{ref} と h_{ref} は基準状態における温度[K]と比エンタルピー[kJ/kg]である。基準状態をどこにとるかは任意であるが、例えば ASHRAE（American Society of Heating, Refrigerating and Air-Conditioning Engineers）や IIR (International Institute of Refrigeration)では 0 °C の飽和液のエントロピーを 1 kJ/(kg K)、比エンタルピーを 200 kJ/kg としている。

$$h(T,\rho) - h(T_{ref},\rho_{ref}) = [h(T,\rho) - h^\circ(T)] + [h^\circ(T) - h^\circ(T_{ref})] + [h^\circ(T_{ref}) - h(T_{ref},\rho_{ref})] \tag{5.9}$$

右辺第 1 項と第 3 項は残存比エンタルピーであり、それぞれ式 5.7 を用いて計算することが可能である。また、第 2 項は式 5.10 に示す通り、理想気体の定圧比熱 C_p° を温度 T について積分することで得られる。

$$h^\circ(T) - h^\circ(T_{ref}) = \int_{T_{ref}}^{T} C_p^\circ dT \tag{5.10}$$

比エントロピーは式 5.11 で計算できる。

$$\begin{aligned} s(T,\rho) - s(T_{ref},\rho_{ref}) = {} & [s(T,\rho) - s^\circ(T,\rho)] + [s^\circ(T,\rho) - s^\circ(T,\rho_{ref})] \\ & + [s^\circ(T,\rho_{ref}) - s^\circ(T_{ref},\rho_{ref})] + [s^\circ(T_{ref},\rho_{ref}) - s(T_{ref},\rho_{ref})] \end{aligned} \tag{5.11}$$

右辺第 1 項と第 4 項は残存比エントロピーであり、それぞれ式 5.6 を用いて計算することが可能である。また、第 2 項と第 3 項はそれぞれ式 5.12 と式 5.13 で計算できる。

$$s^\circ(T,\rho) - s^\circ(T,\rho_{ref}) = R \ln\left(\frac{\rho_{ref}}{\rho}\right) \tag{5.12}$$

$$s^\circ(T,\rho_{ref}) - s^\circ(T_{ref},\rho_{ref}) = \int_{T_{ref}}^{T} \frac{C_p^\circ}{T} dT - R \ln\frac{T}{T_{ref}} \tag{5.13}$$

4）比内部エネルギー

比内部エネルギー u は比エンタルピー h、圧力 P、密度 ρ の関数として式 5.14 で計算できる。

$$u = h - P/\rho \tag{5.14}$$

5）比熱

定積比熱 C_v [kJ/(kg·K)]と定圧比熱 C_p [kJ/(kg·K)]は、式 5.15 と 5.17 で計算できる。C_v^0 は理想気体の定積比熱であり、理想気体の定圧比熱 C_p^0 を用いて式 5.16 で計算できる。式 5.18 に示すように定圧比熱を定積比熱で除した値を比熱比 κ [-]と呼ぶ。理想気体を断熱圧縮する際の圧力と体積の関係は、比熱比を用いて式 5.19 で表現される（ポアソンの式）。比熱比が高い冷媒ほど断熱圧縮時の温度上昇が小さくなるため、効率が高くなる。

$$C_v = C_v^0 - T\left(\frac{\partial^2 A^r}{\partial T^2}\right)_V \tag{5.15}$$

$$C_v^0 = C_p^0 - R \tag{5.16}$$

$$C_p = C_v + \frac{T}{\rho^2}\left(\frac{\partial P}{\partial T}\right)_V^2 \Big/ \left(\frac{\partial P}{\partial \rho}\right)_T \tag{5.17}$$

$$\kappa = C_v / C_p \tag{5.18}$$

$$P\upsilon^{\kappa} = const \tag{5.19}$$

5.2.2 PVT 関係式

前節において記した通り、圧力 $P(\upsilon, T)$および理想気体の定圧比熱 $C_p^o(T)$について計算式が与えられれば、微積分操作を行うことで比エンタルピーと比エントロピーを計算することができる。圧力 P、比体積 υ、温度 Tの関係を表現した PVT 関係式に関しては多くの提案が行われている。

1）Van der Waals（ファン・デル・ワールス）状態方程式

Van der Waals の状態方程式を式 5.20 に示す[5.16]。理想気体の状態方程式で設けた 2 つの仮定（分子の体積と分子間力は 0）を補正するための係数 a と b が導入されている。

$$P = \frac{RT}{\upsilon - b} - \frac{a}{\upsilon^2} \tag{5.20}$$

式 5.20 の第一項では、分子の体積の影響により比体積 υ が b だけ減じられることが表現されている。また、第二項では、分子間力によって圧力測定面に対する分子の衝突回数が減じられ、その効果によって理想気体よりも圧力が減少することが表現されている。

2）ビリアル方程式

ビリアル方程式を式 5.21 に示す。B, C, D …は絶対温度 T [K]の関数として表現される近似係数である。B は 2 分子間の衝突による影響を、C は 3 分子間の衝突による影響を表現しており、D 以降も同様である。

$$\frac{P}{\rho RT} = 1 + B\rho + C\rho^2 + D\rho^3 + \cdots \tag{5.21}$$

3）Martin-Hou（マーティン・ハウ）状態方程式

式 5.22 に Martin-Hou の状態方程式を示す[5.7]。実在気体の実測結果にもとづき、実験結果の近似性能が高くなるように式形状に工夫がなされている。液相の近似はできない点は、後述の MBWR 状態方程式等と比較すると劣るが、近似係数は少ないため、気相のみの計算であれば有利である。

$$P = \frac{RT}{\upsilon - b} + \sum_{i=2}^{5} \frac{a_{i,1} + a_{i,2}T + a_{i,3}\exp\left(-k\,T/T_c\right)}{(\upsilon - b)^i} \tag{5.22}$$

4）MBWR 状態方程式

Modified Benedict-Webb-Rubin（MBWR）状態方程式[5.3) 5.4)]を式 5.23 に示す。液相から気相にかけて広い範囲で近似性能が高く、R123 の国際状態方程式としても用いられている。ここで α は絶対温度 T の多項式であり、合計で 32 の係数により表現されている。なお、δ は密度を臨界密度 ρ_c との比率（$\delta = \rho/\rho_c$）で表現したものであり、対臨界密度（reduced density）と呼ぶ。

$$P = \sum_{n=1}^{9} a_n \rho^n + \exp\left(-\delta\right)^2 \sum_{n=10}^{15} a_n \rho^{2n-17} \tag{5.23}$$

5）ヘルムホルツ状態方程式

式 5.24 は Tillner-Roth らによるヘルムホルツ関数形の状態方程式である[5.8]。R32 の国際状態方程式としてはこの形式が用いられている。τ は温度 T を臨界温度 T_c との比率（$\tau = T/T_c$）で表現したものであり、対臨界温度（reduced temperature）と呼ぶ。MBWR 式とは異なり、ヘルムホルツ自由エネルギー A が温度および密度の関数として表現されているため、定圧比熱の近似式を用いずに式 5.24 の

微分操作のみで圧力や比エンタルピーなどの他の物性値を算出することができる点が特徴である。一方で、係数の指数が実数値をとり、式の形状は MBWR に比較してさらに複雑であるため、プログラムの作成にあたっては煩雑さがある。

$$\frac{A}{RT}=\Phi^{o}(\tau,\delta)+\Phi^{r}(\tau,\delta) \tag{5.24}$$

5.2.3　本書で採用する PVT 関係式と定圧比熱の近似式

1）残存量の計算

本書では式 5.25 に示す PVT 関係式を用いて冷媒の物性計算を行う[5.17]。ビリアル方程式において係数 A, B, C...を温度 T の多項式で表現したものである。連続する自然数のべき数表現であるため、プログラムが実装し易いという利点がある。

$$P=\rho RT+\sum_{m=0}^{M}\sum_{n=2}^{N}\alpha_{m,n-2}\tau^{m}\delta^{n} \tag{5.25}$$

式 5.25 を用いて、5.2.1 節に記した方法により残存量（Residual Property）を計算すると以下の通りとなる。

・残存圧力

$$P^{r}=P-P^{o}=\sum_{m=0}^{M}\sum_{n=2}^{N}\alpha_{m,n-2}\tau^{m}\delta^{n} \tag{5.26}$$

・残存ヘルムホルツエネルギー

$$A^{r}=\frac{1}{\rho_c}\sum_{m=0}^{M}\sum_{n=2}^{N}a_{m,n-2}\frac{1}{n-1}\tau^{m}\delta^{(n-1)} \tag{5.27}$$

・残存比エントロピー

$$s^{r}=-\left(\frac{\partial A^{r}}{\partial T}\right)_{V}=\frac{1}{\rho_c T}\sum_{m=1}^{M}\sum_{n=2}^{N}a_{m,n-2}\frac{m}{n-1}\tau^{m}\delta^{(n-1)} \tag{5.28}$$

・残存比エンタルピー

$$h^{r}=A^{r}+Ts^{r}+\left(\frac{P}{\rho}-RT\right)=\frac{1}{\rho_c}\sum_{m=0}^{M}\sum_{n=2}^{N}a_{m,n-2}\frac{n+m}{n-1}\tau^{m}\delta^{(n-1)} \tag{5.29}$$

・残存ギブスエネルギー

$$G^{r}=A^{r}+\left(P/\rho-RT\right)=\frac{1}{\rho_c}\sum_{m=0}^{M}\sum_{n=2}^{N}a_{m,n-2}\frac{n}{n-1}\tau^{m}\delta^{(n-1)} \tag{5.30}$$

式 5.26~式 5.30 が全て係数 $\alpha_{m,n}$ に関しての線形式となっていることが重要であり、最小二乗法を用いることで全ての状態値を考慮しながら係数 $\alpha_{m,n}$ の推定が可能となる。このような処理を多重性質同時回帰（Multi-property regression）と呼ぶ[5.13) 5.14]。なお、実際に状態値の計算を行う際には式 5.26~式 5.30 に加え、式 5.9~式 5.13 を用いる。

理想気体の定圧比熱の近似式としては式 5.31 を用いる[5.9]。これは R123 の国際状態方程式において用いられている近似式と同一の式形状である。

$$C_{p}^{o}=R\cdot\sum_{k=0}^{K}c_{cp,k}\left(\frac{T}{T_c}\right)^{k} \tag{5.31}$$

式 5.31 を温度で積分すると式 5.32 が得られる。式 5.9 と式 5.11 に示したとおり、式 5.32 は比エンタルピーの計算で用いる。

$$\int_{T_{ref}}^{T} C_p^{o}\,dT = R\sum_{k=0}^{K}\frac{c_{cp,k}}{T_c^{k}(k+1)}\left(T^{k+1}-T_{ref}^{k+1}\right) \tag{5.32}$$

式 5.31 を温度で除した後に温度で積分すると式 5.33 が得られる。式 5.11 と式 5.13 に示したとおり、式 5.33 は比エントロピーの計算で用いる。

$$\int_{T_{ref}}^{T}\frac{C_p^{o}}{T}dT = R\left\{c_{cp,0}\ln\left(\frac{T}{T_{ref}}\right)+\sum_{k=1}^{K}\frac{c_{cp,k}}{T_c^{k}}\left(T^{k}-T_{ref}^{k}\right)\right\} \tag{5.33}$$

定積比熱と定圧比熱は、式 5.25 に式 5.15 と式 5.17 を適用し、式 5.34 と 5.35 で計算できる。

$$C_v = C_v^0 - T\left(\frac{\partial^2 A^r}{\partial T^2}\right)_V = \frac{1}{\rho_c T}\sum_{m=2}^{M}\sum_{n=2}^{N} a_{m,n-2}\frac{m(m+1)}{1-n}\tau^m\delta^{(n-1)} \tag{5.34}$$

$$\begin{aligned} C_p - C_v &= \frac{T}{\rho^2}\left(\frac{\partial P}{\partial T}\right)_V^2 \Big/ \left(\frac{\partial P}{\partial \rho}\right)_T \\ &= \frac{T}{\rho^2}\left(\rho R - \frac{1}{T}\sum_{m=1}^{M}\sum_{n=2}^{N} m\cdot a_{m,n-2}\tau^m\delta^n\right)^2 \Big/ \left(RT + \frac{1}{\rho_c}\sum_{m=0}^{M}\sum_{n=2}^{N} n\cdot a_{m,n-2}\tau^m\delta^{(n-1)}\right) \end{aligned} \tag{5.35}$$

2）近似係数

パッケージ空調機の代表的な冷媒である R32 と R410A、大型冷凍機の代表的な冷媒である R134a について近似係数を計算した結果を表 5.1~表 5.12 に示す。飽和温度および飽和圧力の初期値計算用の係数 $c_{ts,k}$、$c_{ps,k}$ の使用方法については後述する。

表 5.1　R32 の PVT 式の係数 $a_{m,n}$

		n		
		2	3	4
m	0	9.76659057E+03	3.46090534E+04	-4.10246731E+04
	1	1.09321877E+03	-8.08424152E+03	1.44666922E+04
	2	-1.30851505E+05	-1.84472550E+05	2.95380164E+05
	3	1.25919687E+05	3.32809980E+05	-5.45340044E+05
	4	-4.36905538E+04	-1.47241829E+05	2.70623107E+05
		n		
		5	6	7
m	0	1.19662253E+04	-1.69944599E+02	1.90910763E+02
	1	-1.07181678E+04	3.56517164E+03	-4.41108945E+02
	2	-7.95365835E+04	-1.02795826E+04	3.36291347E+03
	3	1.97719548E+05	-7.54149219E+03	-3.26330690E+03
	4	-1.23411075E+05	1.66223110E+04	0

表 5.2　R32 の理想気体の定圧比熱近似係数 $c_{cp,k}$

$c_{cp,0}$	$c_{cp,1}$	$c_{cp,2}$	$c_{cp,3}$	$c_{cp,4}$
4.0186023E+00	-3.1370881E-01	6.8796834E-01	2.6831619E+00	-1.3934091E+00

表 5.3　R32 の臨界状態値およびガス定数

臨界温度 [K]	臨界密度 [kg/m^3]	臨界圧力 [kPa]	ガス定数 [-]
351.255	424	5782	0.15982

表 5.4　R32 の飽和温度・飽和圧力初期値計算用係数

	n=0	n=1	n=2	n=3
$c_{ts,n}$	241	269	-298	148
$c_{ps,n}$	-33645	141476	-204886	102795

状態式の適用可能範囲（圧力：0.95~4.0 Mpa、過冷却度：10 °C、過熱度：80 °C）

表 5.5　R410A の PVT 式の係数 $\alpha_{m,n}$

		n		
		2	3	4
m	0	6.71960062E+03	3.19356936E+04	-3.24183793E+04
	1	8.63923773E+02	-5.71715972E+03	6.86534581E+03
	2	-8.58414719E+04	-1.92186729E+05	3.20048484E+05
	3	8.21904683E+04	3.15725662E+05	-5.50790659E+05
	4	-2.86299865E+04	-1.36826045E+05	2.59241307E+05
		n		
		5	6	7
m	0	8.54825317E-01	8.12952272E+03	-1.73274221E+03
	1	-3.31906471E+03	7.13086961E+02	-5.60282667E+01
	2	-1.26786726E+05	3.65772262E+03	3.54791238E+03
	3	2.63679502E+05	-3.77400688E+04	-4.92447924E+02
	4	-1.39979135E+05	2.75454235E+04	-1.41852889E+03

表 5.6　R410A の理想気体の定圧比熱近似係数 $c_{cp,k}$

$c_{cp,0}$	$c_{cp,1}$	$c_{cp,2}$	$c_{cp,3}$	$c_{cp,4}$
3.8200427E+00	2.5486066E+00	1.0799339E+00	9.8471295E-01	-7.6910990E-01

表 5.7　R410A の臨界状態値およびガス定数

臨界温度 [K]	臨界密度 [kg/m^3]	臨界圧力 [kPa]	ガス定数 [-]
344.508	459.53	4902.6	0.114548

表 5.8　R410A の飽和温度・飽和圧力初期値計算用係数

	n=0	n=1	n=2	n=3
$c_{ts,n}$	238	260	-292	151
$c_{ps,n}$	-27955	118095	-171854	86584

状態式の適用可能範囲（圧力：0.95~4.0 Mpa、過冷却度：10 °C、過熱度：80 °C）

表 5.9　R134a の PVT 式の係数 $\alpha_{m,n}$

		n		
		2	3	4
m	0	-1.79588354E+06	4.19965683E+06	-3.01978229E+06
	1	2.34822126E+05	-6.11593615E+05	5.03422032E+05
	2	5.95593217E+06	-1.41258495E+07	1.06739901E+07
	3	-6.23600250E+06	1.33368075E+07	-8.68522771E+06
	4	1.85868920E+06	-3.38523621E+06	1.48937436E+06
		n		
		5	6	-
m	0	8.24368800E+05	-5.20373039E+04	-
	1	-1.66971575E+05	1.93377103E+04	-
	2	-3.60638582E+06	4.46949815E+05	-
	3	2.59982114E+06	-3.13344200E+05	-
	4	-1.89758898E+05	0.00000000E+00	-

表 5.10　R134a の理想気体の定圧比熱近似係数 $c_{cp,k}$

$c_{cp,0}$	$c_{cp,1}$	$c_{cp,2}$	$c_{cp,3}$	$c_{cp,4}$
1.9747230E-01	2.6221924E+01	-3.8424120E+01	3.8198553E+01	-1.4361012E+01

表 5.11　R134a の臨界状態値およびガス定数

臨界温度 [K]	臨界密度 [kg/m^3]	臨界圧力 [kPa]	ガス定数 [-]
374.21	511.9	4059.3	0.081490

表 5.12　R134a の飽和温度・飽和圧力初期値計算用係数

	n=0	n=1	n=2	n=3
$c_{ts,n}$	242	513	-1252	1355
$c_{ps,n}$	-13992	65405	-104220	56756

状態式の適用可能範囲（圧力：0.2~1.5 Mpa、過冷却度：20 °C、過熱度：30 °C）

5.3 計算法

定数宣言と列挙型の定義をプログラム 5.1 に示す。

プログラム 5.1　定数宣言と列挙型の定義

Popolo.ThermophysicalProperty.Refrigerant class

```
/// <summary>絶対温度と摂氏との変換定数</summary>
private const double CONVERT_C_TO_K = 273.15;

/// <summary>冷媒の種類</summary>
public enum Fluid
{
    /// <summary>R410A</summary>
    R410A,
    /// <summary>R32</summary>
    R32,
    /// <summary>R134a</summary>
    R134a
}

/// <summary>冷媒の状態</summary>
public enum Phase
{
    /// <summary>気相</summary>
    Vapor,
    /// <summary>液相</summary>
    Liquid,
    /// <summary>気液平衡</summary>
    Equilibrium
}
```

5.3.1 「冷媒物性計算クラス」の作成

表 5.1~表 5.12 に示したように冷媒計算のためのパラメータ数は多いため、これらのパラメータを保持するインスタンスを生成して物性計算を行う仕様とする。プログラム 5.2 にパラメータを保持するインスタンス変数の定義を示す。

プログラム 5.2　インスタンス変数

Popolo.ThermophysicalProperty.Refrigerant class

```
/// <summary>近似係数 m の数</summary>
private readonly int m_number;

/// <summary>近似係数 n の数</summary>
private readonly int n_number;

/// <summary>臨界温度[K]</summary>
private readonly double criticalTemperature;

/// <summary>臨界密度[kg/m3]</summary>
private readonly double criticalDensity;

/// <summary>臨界圧力[kPa]</summary>
private readonly double criticalPressure;

/// <summary>ガス定数</summary>
private readonly double gasConstant;

/// <summary>基準状態の温度[K]</summary>
private readonly double refTemperature;

/// <summary>基準状態の密度[kg/m3]</summary>
private readonly double refDensity;

/// <summary>基準状態のエンタルピー[kJ/kg]</summary>
private readonly double refEnthalpy;

/// <summary>基準状態のエントロピー[kJ/kgK]</summary>
private readonly double refEntropy;

/// <summary>PVT 近似係数</summary>
private readonly double[,] alpha;

/// <summary>定圧比熱[kJ/kg]近似係数</summary>
private readonly double[] ccp;
```

```
36
37 /// <summary>飽和圧力[kPa]の初期値推測用近似係数</summary>
38 private readonly double[] cps;
39
40 /// <summary>飽和温度[K]の初期値推測用近似係数</summary>
41 private readonly double[] cts;
```

プログラム 5.3 にコンストラクタを示す。8~32 行は R410A の場合のパラメータ初期化処理である。39~56 行に示すクラス変数を用いて初期化する。記載は省略したが、R32 および R134a についても同様の処理を実装する。

プログラム 5.3　近似係数の初期化処理

Popolo.ThermophysicalProperty.Refrigerant class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="fluid">冷媒種類</param>
 3 public Refrigerant(Fluid fluid)
 4 {
 5   int index;
 6   switch (fluid)
 7   {
 8     case Fluid.R410A:
 9       m_number = 5;
10       n_number = 8;
11       criticalTemperature = 71.358 + CONVERT_C_TO_K;
12       criticalDensity = 459.53;
13       criticalPressure = 4902.6;
14       gasConstant = 8.3144621 / 72.585;
15       refTemperature = 0 + CONVERT_C_TO_K;
16       refDensity = 1170;
17       refEnthalpy = 200;
18       refEntropy = 1.0;
19       ccp = ccp_R410A;
20       cps = cps_R410A;
21       cts = cts_R410A;
22       alpha = new double[m_number, n_number - 2];
23       index = 0;
24       for (int m = 0; m < alpha.GetLength(0); m++)
25       {
26         for (int n = 0; n < alpha.GetLength(1); n++)
27         {
28           alpha[m, n] = alpha_R410A[index];
29           index++;
30         }
31       }
32       break;
33     case Fluid.R32:
34       //以下省略：R32 の初期化処理
35     case Fluid.R134a:
36       //以下省略：R134a の初期化処理
37 }
38
39 /// <summary>PVT 近似係数</summary>
40 private static double[] alpha_R410A = new double[] {
41  6.7196006E+03, 3.1935694E+04, -3.2418379E+04, 8.5482532E-01, 8.1295227E+03,
42  -1.7327422E+03, 8.6392377E+02, -5.7171597E+03, 6.8653458E+03, -3.3190647E+03,
43  7.1308696E+02, -5.6028267E+01, -8.5841472E+04, -1.9218673E+05, 3.2004848E+05,
44  -1.2678673E+05, 3.6577226E+03, 3.5479124E+03, 8.2190468E+04, 3.1572566E+05,
45  -5.5079066E+05, 2.6367950E+05, -3.7740069E+04, -4.9244792E+02, -2.8629986E+04,
46  -1.3682605E+05, 2.5924131E+05, -1.3997914E+05, 2.7545424E+04, -1.4185289E+03 };
47
48 /// <summary>定圧比熱[kJ/kg]近似係数</summary>
49 private static double[] ccp_R410A = new double[] {
50  3.8200427E+00, 2.5486066E+00, 1.0799339E+00, 9.8471295E-01, -7.6910990E-01 };
51
52 /// <summary>飽和圧力[kPa]の初期値推測用近似係数</summary>
53 private static double[] cps_R410A = new double[] { 86584, -171854, 118095, -27955 };
54
55 /// <summary>飽和温度[K]の初期値推測用近似係数</summary>
56 private static double[] cts_R410A = new double[] { 151, -292, 260, 238 };
```

5.3.2　PVT 関係式の計算

プログラム 5.4 に PVT 関係式の計算処理を示す。1~18 行は温度と密度から圧力を求める関数であり、式 5.25 の実装である。20~51 行は式 5.25 を密度について解くための関数である。ニュートン・ラフソン法による反復計算を用いるため、第 3 引数である density に密度の初期値を適切に設定する必

要がある。式 5.25 は温度 T について微分可能であるため、数値微分を用いず 31~48 行で微分値の計算関数を定義する。0.2 MPa、0 ºC の条件において 0.001 %程度の誤差に抑えることを目標に、$200\times0.00001\approx10^{-3}$、$273.15\times0.00001\approx10^{-3}$ を許容誤差とした。

プログラム 5.4　PVT 関係式の計算処理

Popolo.ThermophysicalProperty.Refrigerant class

```
 1 /// <summary>温度[K]および密度[kg/m3]から圧力[kPa]を計算する</summary>
 2 /// <param name="temperature">温度[K]</param>
 3 /// <param name="density">密度[kg/m3]</param>
 4 /// <returns>圧力[kPa]</returns>
 5 public double GetPressureFromTemperatureAndDensity(double temperature, double density)
 6 {
 7   double tau = criticalTemperature / temperature;
 8   double rho = density / criticalDensity;
 9
10   double pressure = 0;
11   for (int m = alpha.GetLength(0) - 1; 0 <= m; m--)
12   {
13     double buff = 0;
14     for (int n = alpha.GetLength(1) - 1; 0 <= n; n--) buff = buff * rho + alpha[m, n];
15     pressure = pressure * tau + buff;
16   }
17   return pressure * rho * rho + density * gasConstant * temperature;
18 }
19
20 /// <summary>ニュートン法で圧力[kPa]および温度[K]から密度[kg/m3]を計算する</summary>
21 /// <param name="pressure">圧力[kPa]</param>
22 /// <param name="temperature">温度[K]</param>
23 /// <param name="density">密度[kg/m3]</param>
24 private void getDensityFromPressureAndTemperature
25   (double pressure, double temperature, ref double density)
26 {
27   //誤差関数
28   Roots.ErrorFunction eFnc = delegate (double dns)
29   { return pressure - GetPressureFromTemperatureAndDensity(temperature, dns); };
30
31   //誤差関数の微分値
32   Roots.ErrorFunction eFncD = delegate (double dns)
33   {
34     double tau = criticalTemperature / temperature;
35     double rho = dns / criticalDensity;
36
37     double dPdRho = 0;
38     for (int m = alpha.GetLength(0) - 1; 0 <= m; m--)
39     {
40       double buff = 0;
41       int len = alpha.GetLength(1) + 1;
42       for (int n = len; 2 <= n; n--) buff = buff * rho + n * alpha[m, n - 2];
43       dPdRho = dPdRho * tau + buff;
44     }
45     dPdRho *= rho;
46
47     return -(dPdRho / criticalDensity + gasConstant * temperature);
48   };
49
50   density = Roots.Newton(eFnc, eFncD, density, 1e-3, 1e-3, 10);
51 }
```

5.3.3　液相と気相の判定

式 5.25~式 5.29 に示される各状態値は、他の状態値から陽に求められず、収束計算が必要となるものが多い。収束計算にあたっては適切な初期値を仮定する必要があり、このためにはまず、液相と気相の判定を行うことが必要である。液相と気相の判定のフローを図 5.1 に示す。

入力として圧力 P と絶対温度 T が与えられたとする。与えられた圧力を飽和圧力とし、飽和温度を計算する。収束計算となるため飽和温度の初期値を仮定する必要があるが、収束の発散が生じない程度の精度があればよく、例えば式 5.36 で表現される多項式を用いる。n の次数としては 3 程度で十分である。

飽和温度と飽和圧力が与えられたため、式 5.25 を用いて飽和密度を計算することができる。ただ

し、式 5.25 もまた密度 ρ について解析的に解くことができないため、初期値を定めた上での収束計算が必要となる。初期値の推定のためには、飽和液に関しては式 5.37、式 5.38 で表現される修正 Rackett 式を用い[5.10]、飽和蒸気に関しては理想気体を仮定して式 5.39 を用いる。

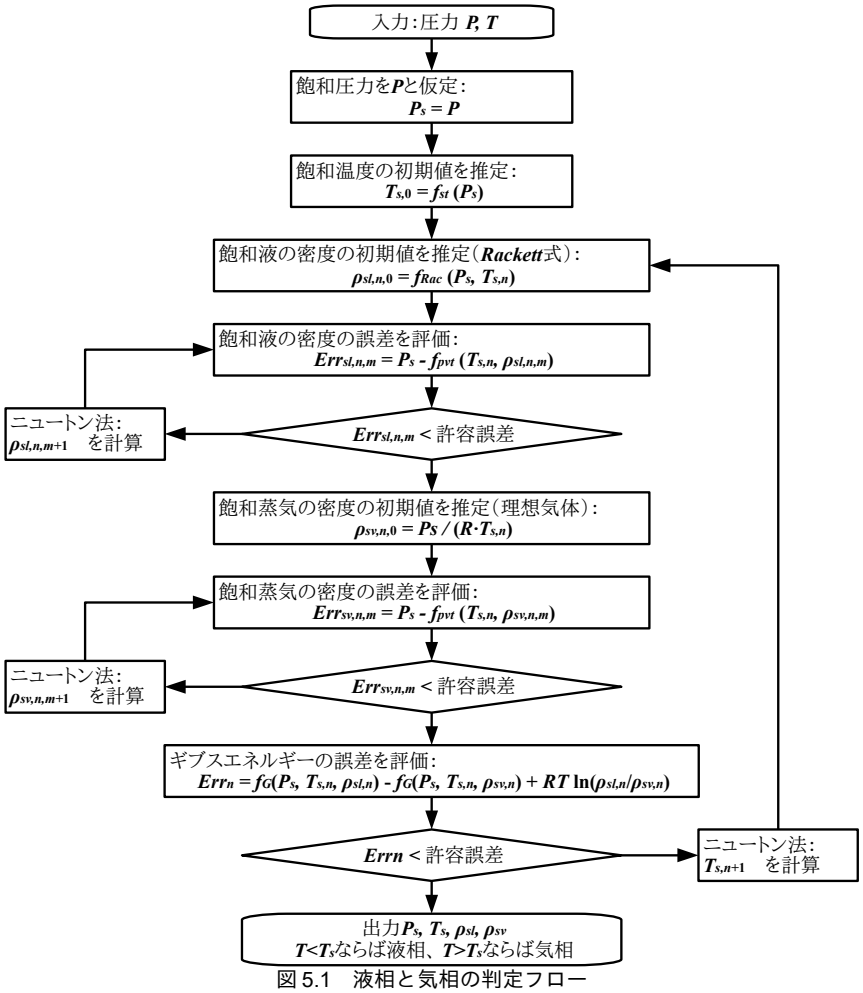


図 5.1　液相と気相の判定フロー

$$T_{s,0}=\sum_{k=0}^{K} c_{ps,k}\left(\frac{P_s}{P_c}\right)^k \tag{5.36}$$

$$\frac{1}{\rho_{sl}}=\frac{R\cdot T_c}{P_c} Z_c^{\left(1+(1-T/T_c)^{2/7}\right)} \tag{5.37}$$

$$Z_c=\frac{P_c\cdot \upsilon_c}{R\cdot T_c} \tag{5.38}$$

$$\rho_{sv}=\frac{P}{R\cdot T} \tag{5.39}$$

飽和温度と飽和密度が求まれば、式 5.30 により残存ギブスエネルギー G^r を計算することができ、飽和蒸気と飽和液の両者について G^r を求める。等温等圧条件下においてギブスエネルギーは一定値であるため、飽和液と飽和蒸気のギブスエネルギーは等しく、式 5.40 が成立する[5.9]。なお、添字の *sl* と *sv* はそれぞれ飽和液と飽和蒸気を表している。反復計算によって最初に仮定した飽和温度 T_s を

修正し、最終的に得られた飽和温度 T_s が入力条件である T よりも大きければ液相、小さければ気相である。

$$G^r_{sl} - G^r_{sv} + R\,T \ln\left(\frac{\rho_{sl}}{\rho_{sv}}\right) = 0 \tag{5.40}$$

なお、図 5.1 は飽和温度 T_s を未知数にする場合の飽和状態の計算法であるが、飽和圧力 P_s を未知変数としても同様の処理により飽和状態を求めることができる。この場合には飽和圧力の初期値推定に式 5.41 を用いる。

$$P_{s,0} = \sum_{k=0}^{K} c_{ts,k} \left(\frac{T_s}{T_c}\right)^k \tag{5.41}$$

飽和状態の物性値計算処理をプログラム 5.5 に示す。80~100 行は残存ギブスエネルギーの計算処理であり、式 5.30 の実装である。47~78 行は、飽和蒸気および飽和液の残存ギブスエネルギーの差分を計算する処理であり、式 5.40 の左辺である。飽和状態においてはこの値が 0 となる。1~22 行は飽和圧力にもとづく飽和温度および飽和密度の計算処理である。15~18 行において、式 5.36 に従って飽和温度の初期値（仮定）を計算する。19 行はでニュートン・ラフソン法による収束計算を実行する。同様に、24~45 行は飽和温度にもとづく飽和圧力および飽和密度の計算処理である。

プログラム 5.5　飽和状態の物性値計算処理

Popolo.ThermophysicalProperty.Refrigerant class

```
 1 /// <summary>飽和圧力[kPa]から飽和物性値を計算する</summary>
 2 /// <param name="pressure">飽和圧力[kPa]</param>
 3 /// <param name="saturatedLiquidDensity">飽和液の密度[kg/m3]</param>
 4 /// <param name="saturatedVaporDensity">飽和蒸気の密度[kg/m3]</param>
 5 /// <param name="saturatedTemperature">飽和温度[K]</param>
 6 public void GetSaturatedPropertyFromPressure(double pressure,
 7   out double saturatedLiquidDensity, out double saturatedVaporDensity, out double saturatedTemperature)
 8 {
 9   //誤差関数
10   double sld = 0;
11   double svd = 0;
12   Roots.ErrorFunction eFnc = delegate (double tmp)
13   { return getGibbsEnergyDifference(tmp, pressure, out sld, out svd); };
14
15   //飽和温度の初期値を推定
16   double pr = pressure / criticalPressure;
17   double ts = cts[0];
18   for (int i = 1; i < cts.Length; i++) ts = ts * pr + cts[i];
19   saturatedTemperature = Roots.Newton(eFnc, ts, 1e-5, 1e-5, 1e-3, 10);
20   saturatedLiquidDensity = sld;
21   saturatedVaporDensity = svd;
22 }
23
24 /// <summary>飽和温度[K]から飽和物性値を計算する</summary>
25 /// <param name="temperature">飽和温度[K]</param>
26 /// <param name="saturatedLiquidDensity">飽和液の密度[kg/m3]</param>
27 /// <param name="saturatedVaporDensity">飽和蒸気の密度[kg/m3]</param>
28 /// <param name="saturatedPressure">飽和圧力[kPa]</param>
29 public void GetSaturatedPropertyFromTemperature(double temperature,
30   out double saturatedLiquidDensity, out double saturatedVaporDensity, out double saturatedPressure)
31 {
32   //誤差関数
33   double sld = 0;
34   double svd = 0;
35   Roots.ErrorFunction eFnc = delegate (double pres)
36   { return getGibbsEnergyDifference(temperature, pres, out sld, out svd); };
37
38   //飽和圧力の初期値を推定
39   double tr = temperature / criticalTemperature;
40   double ps = cps[0];
41   for (int i = 1; i < cps.Length; i++) ps = ps * tr + cps[i];
42   saturatedPressure = Roots.Newton(eFnc, ps, 1e-5, 1e-5, 1e-3, 10);
43   saturatedLiquidDensity = sld;
44   saturatedVaporDensity = svd;
45 }
```

```
46
47 /// <summary>温度と圧力から液体および気体のギブスエネルギーの差分を計算する</summary>
48 /// <param name="temperature">温度[K]</param>
49 /// <param name="pressure">圧力[kPa]</param>
50 /// <param name="liquidDensity">液体の密度[kg/m3]</param>
51 /// <param name="vaporDensity">気体の密度[kg/m3]</param>
52 /// <returns>ギブスエネルギーの差分[kJ]</returns>
53 private double getGibbsEnergyDifference
54   (double temperature, double pressure, out double liquidDensity, out double vaporDensity)
55 {
56   double tr = temperature / criticalTemperature;
57
58   //液体の密度
59   //初期値を Rackett 式で設定
60   double rc = criticalPressure / (criticalDensity * criticalTemperature * gasConstant);
61   double rhol = 1d / (Math.Pow(rc, 1d + Math.Pow(1d - tr, 2d / 7d))
62     / criticalPressure * gasConstant * criticalTemperature);
63   getDensityFromPressureAndTemperature(pressure, temperature, ref rhol);
64   liquidDensity = rhol;
65
66   //気体の密度を推定
67   //初期値を理想気体の状態式で設定
68   double rhov = pressure / (temperature * gasConstant);
69   getDensityFromPressureAndTemperature(pressure, temperature, ref rhov);
70   vaporDensity = rhov;
71
72   //Gibbs エネルギーの差分を計算
73   double gL = getResidualGibbsFreeEnergy(temperature, liquidDensity);
74   double gV = getResidualGibbsFreeEnergy(temperature, vaporDensity);
75
76   return gL - gV + gasConstant * temperature * Math.Log(liquidDensity / vaporDensity);
78 }
79
80 /// <summary>残存ギブスエネルギー[kJ]を計算する</summary>
81 /// <param name="temperature">絶対温度[K]</param>
82 /// <param name="density">密度[kg/m3]</param>
83 /// <returns>残存ギブスエネルギー[kJ]</returns>
84 private double getResidualGibbsFreeEnergy(double temperature, double density)
85 {
86   double tau = criticalTemperature / temperature;
87   double rho = density / criticalDensity;
88
89   double gr = 0;
90   for (int m = alpha.GetLength(0) - 1; 0 <= m; m--)
91   {
92     double buff = 0;
93     int len = alpha.GetLength(1) + 1;
94     for (int n = len; 2 <= n; n--) buff = buff * rho + alpha[m, n - 2] * n / (n - 1d);
95     gr = gr * tau + buff;
96   }
97   gr *= rho;
98
99   return gr / criticalDensity;
100 }
```

5.3.4　温度と密度にもとづく物性値の計算

温度と密度にもとづく比エンタルピーの計算処理をプログラム 5.6 に示す。1~9 行は比エンタルピーの計算処理であり、式 5.9 の実装である。11~31 行は残存比エンタルピーの計算処理（式 5.29）、33~48 行は理想気体の定圧比熱の温度積分値の計算処理（式 5.32）であり、比エンタルピー計算の補助関数である。

プログラム 5.6　温度と密度にもとづく比エンタルピーの計算処理

Popolo.ThermophysicalProperty.Refrigerant class

```
 1 /// <summary>温度[K]および密度[kg/m3]から比エンタルピー[kJ/kg]を計算する</summary>
 2 /// <param name="temperature">温度[K]</param>
 3 /// <param name="density">密度[kg/m3]</param>
 4 /// <returns>比エンタルピー[kJ/kg]</returns>
 5 public double GetEnthalpyFromTemperatureAndDensity(double temperature, double density)
 6 {
 7   double h = getResidualEnthalpy(temperature, density) - getResidualEnthalpy(refTemperature, refDensity);
 8   return h + getIntegralCp0(temperature) + refEnthalpy;
 9 }
10
11 /// <summary>絶対温度[K]と密度[kg/m3]から残存比エンタルピー[kJ/kg]を計算する</summary>
12 /// <param name="temperature">絶対温度[K]</param>
```

```
/// <param name="density">密度[kg/m3]</param>
/// <returns>残存比エンタルピー[kJ/kg]</returns>
private double getResidualEnthalpy(double temperature, double density)
{
  double tau = criticalTemperature / temperature;
  double rho = density / criticalDensity;

  double hr = 0;
  for (int m = alpha.GetLength(0) - 1; 0 <= m; m--)
  {
    double buff = 0;
    int len = alpha.GetLength(1) + 1;
    for (int n = len; 2 <= n; n--) buff = buff * rho + alpha[m, n - 2] * (n + m) / (n - 1d);
    hr = hr * tau + buff;
  }
  hr *= rho;

  return hr / criticalDensity;
}

/// <summary>温度[K]から理想気体の定圧比熱[kJ/kgK]の積分値を計算する</summary>
/// <param name="temperature">温度[K]</param>
/// <returns>理想気体の定圧比熱[kJ/kgK]の積分値</returns>
private double getIntegralCp0(double temperature)
{
  double tr = temperature / criticalTemperature;
  double tr0 = refTemperature / criticalTemperature;
  double cp0 = 0;
  double cp0r = 0;
  for (int i = ccp.Length - 1; 0 <= i; i--)
  {
    cp0 = (cp0 + ccp[i] / (i + 1) * criticalTemperature) * tr;
    cp0r = (cp0r + ccp[i] / (i + 1) * criticalTemperature) * tr0;
  }
  return (cp0 - cp0r) * gasConstant;
}
```

温度と密度にもとづく比エントロピーの計算処理をプログラム 5.7 に示す。1~11 行は比エントロピーの計算処理であり、式 5.11 の実装である。13~33 行は残存比エントロピーの計算処理（式 5.28）、35~55 行は理想気体の定圧比熱を温度で除した値の温度積分値の計算処理（式 5.33）であり、比エントロピー計算の補助関数である。

プログラム 5.7　温度と密度にもとづく比エントロピーの計算処理

Popolo.ThermophysicalProperty.Refrigerant class

```
/// <summary>温度[K]および密度[kg/m3]から比エントロピー[kJ/kgK]を計算する</summary>
/// <param name="temperature">温度[K]</param>
/// <param name="density">密度[kg/m3]</param>
/// <returns>比エントロピー[kJ/kgK]</returns>
public double GetEntropyFromTemperatureAndDensity(double temperature, double density)
{
  double s = getResidualEntropy(temperature, density) - getResidualEntropy(refTemperature, refDensity);
  s += gasConstant * Math.Log(refDensity / density);
  s += getIntegralCp0T(temperature) - gasConstant * Math.Log(temperature / refTemperature);
  return s + refEntropy;
}

/// <summary>絶対温度[K]と密度[kg/m3]から残存比エントロピー[kJ/kgK]を計算する</summary>
/// <param name="temperature">絶対温度[K]</param>
/// <param name="density">密度[kg/m3]</param>
/// <returns>残存比エントロピー[kJ/kgK]</returns>
private double getResidualEntropy(double temperature, double density)
{
  double tau = criticalTemperature / temperature;
  double rho = density / criticalDensity;

  double sr = 0;
  for (int m = alpha.GetLength(0) - 1; 1 <= m; m--)
  {
    double buff = 0;
    int len = alpha.GetLength(1) + 1;
    for (int n = len; 2 <= n; n--) buff = buff * rho + alpha[m, n - 2] * m / (n - 1d);
    sr = sr * tau + buff;
  }
  sr *= rho * tau;

  return sr / (criticalDensity * temperature);
```

```
33 }
34
35 /// <summary>温度[K]から理想気体の定圧比熱[kJ/kgK]/T の積分値を計算する</summary>
36 /// <param name="temperature">温度[K]</param>
37 /// <returns>理想気体の定圧比熱[kJ/kgK]/T の積分値</returns>
38 private double getIntegralCp0T(double temperature)
39 {
40   double tr = temperature / criticalTemperature;
41   double tr0 = refTemperature / criticalTemperature;
42   double cp0 = 0;
43   double cp0r = 0;
44
45   for (int i = ccp.Length - 1; 0 < i; i--)
46   {
47     cp0 = cp0 * tr + ccp[i] / i;
48     cp0r = cp0r * tr0 + ccp[i] / i;
49   }
50   cp0 *= tr;
51   cp0r *= tr0;
52
53   return ((cp0 + ccp[0] * Math.Log(temperature))
54     - (cp0r + ccp[0] * Math.Log(refTemperature))) * gasConstant;
55 }
```

温度と密度にもとづく比内部エネルギーの計算処理をプログラム 5.8 に示す。式 5.14 を使用して比エンタルピー、圧力、密度から計算する。

プログラム 5.8　温度と密度にもとづく比内部エネルギーの計算処理

Popolo.ThermophysicalProperty.Refrigerant class

```
1 /// <summary>温度[K]および密度[kg/m3]から内部エネルギー[kJ/kg]を計算する</summary>
2 /// <param name="temperature">温度[K]</param>
3 /// <param name="density">密度[kg/m3]</param>
4 /// <returns>内部エネルギー[kJ/kg]</returns>
5 public double GetInternalEnergyFromTemperatureAndDensity(double temperature, double density)
6 {
7   double p = GetPressureFromTemperatureAndDensity(temperature, density);
8   return GetEnthalpyFromTemperatureAndDensity(temperature, density) - p / density;
9 }
```

温度と密度にもとづく比熱および比熱比の計算処理をプログラム 5.9 に示す。1~52 行が実在気体の定圧比熱および定積比熱の計算であり、理想気体の状態値を用いて式 5.34 および式 5.35 で計算する。65~80 行は理想気体の定圧比熱および定積比熱の計算であり、式 5.16 と式 5.31 の実装である。54~63 行は比熱比の計算であり、式 5.18 の実装である。

プログラム 5.9　温度と密度にもとづく比熱および比熱比の計算処理

Popolo.ThermophysicalProperty.Refrigerant class

```
 1 /// <summary>絶対温度[K]と密度[kg/m3]から定積比熱[kJ/kgK]を計算する</summary>
 2 /// <param name="temperature">絶対温度[K]</param>
 3 /// <param name="density">密度[kg/m3]</param>
 4 /// <returns>定積比熱[kJ/kgK]</returns>
 5 public double GetIsovolumetricSpecificHeatFromTemperatureAndDensity(double temperature, double density)
 6 {
 7   double tau = criticalTemperature / temperature;
 8   double rho = density / criticalDensity;
 9
10   double cv = 0;
11   for (int m = alpha.GetLength(0) - 1; 2 <= m; m--)
12   {
13     double buff = 0;
14     int len = alpha.GetLength(1) + 1;
15     for (int n = len; 2 <= n; n--) buff = buff * rho + alpha[m, n - 2] * m * (m + 1) / (1d - n);
16     cv = cv * tau + buff;
17   }
18   cv *= rho * tau * tau;
19
20   return cv / (criticalDensity * temperature) + getIsovolumetricHeatCapacityOfIdealGas(temperature);
21 }
22
23 /// <summary>絶対温度[K]と密度[kg/m3]から定圧比熱[kJ/kgK]を計算する</summary>
24 /// <param name="temperature">絶対温度[K]</param>
25 /// <param name="density">密度[kg/m3]</param>
26 /// <returns>定圧比熱[kJ/kgK]</returns>
27 public double GetIsobaricSpecificHeatFromTemperatureAndDensity(double temperature, double density)
28 {
```

```
29   double tau = criticalTemperature / temperature;
30   double rho = density / criticalDensity;
31
32   double cp1 = 0;
33   double cp2 = 0;
34   for (int m = alpha.GetLength(0) - 1; 0 <= m; m--)
35   {
36     double buff1 = 0;
37     double buff2 = 0;
38     for (int n = alpha.GetLength(1) + 1; 2 <= n; n--)
39     {
40       if (m != 0) buff1 = buff1 * rho + alpha[m, n - 2] * m;
41       buff2 = buff2 * rho + alpha[m, n - 2] * n;
42     }
43     if (m != 0) cp1 = cp1 * tau + buff1;
44     cp2 = cp2 * tau + buff2;
45   }
46   cp1 = density * gasConstant - cp1 * rho * rho * tau / temperature;
47   cp2 = gasConstant * temperature + cp2 * rho / criticalDensity;
48
49   double bf = cp1 / density;
50   return GetIsovolumetricSpecificHeatFromTemperatureAndDensity(temperature, density)
51     + temperature * bf * bf / cp2;
52 }
53
54 /// <summary>絶対温度[K]と密度[kg/m3]から比熱比[-]を計算する</summary>
55 /// <param name="temperature">絶対温度[K]</param>
56 /// <param name="density">密度[kg/m3]</param>
57 /// <returns>比熱比[-]</returns>
58 public double GetSpecificHeatRatioFromTemperatureAndDensity(double temperature, double density)
59 {
60   double cv = GetIsovolumetricSpecificHeatFromTemperatureAndDensity(temperature, density);
61   double cp = GetIsobaricSpecificHeatFromTemperatureAndDensity(temperature, density);
62   return cp / cv;
63 }
64
65 /// <summary>絶対温度[K]から理想気体の定圧比熱[kJ/kgK]を計算する</summary>
66 /// <param name="temperature">温度[K]</param>
67 /// <returns>理想気体の定圧比熱[kJ/kgK]</returns>
68 private double getIsobaricHeatCapacityOfIdealGas(double temperature)
69 {
70   double tr = temperature / criticalTemperature;
71   double cp0 = ccp[ccp.Length - 1];
72   for (int i = ccp.Length - 2; 0 <= i; i--) cp0 = cp0 * tr + ccp[i];
73   return cp0 * gasConstant;
74 }
75
76 /// <summary>温度[K]から理想気体の定積比熱[kJ/kgK]を計算する</summary>
77 /// <param name="temperature">絶対温度[K]</param>
78 /// <returns>理想気体の定積比熱[kJ/kgK]</returns>
79 private double getIsovolumetricHeatCapacityOfIdealGas(double temperature)
80 { return getIsobaricHeatCapacityOfIdealGas(temperature) - gasConstant; }
```

5.3.5　比エンタルピーと圧力にもとづく物性値の計算

比エンタルピーと圧力にもとづく物性値の計算処理をプログラム 5.10 に示す。与えられた圧力にもとづき、13 行において飽和状態の物性値を計算する。15 行と 16 行において飽和状態の比エンタルピーを計算し、液相・二相・気相の判定を行う。

液相および気相の場合には、飽和温度近辺に温度の初期値をとり（42~51 行）、ニュートン・ラフソン法による収束計算を行うことで、与えられた比エンタルピーに合致する温度を求める。

二相域の場合には式 5.42 で気液混合割合 *rate* を計算し、比体積、比エントロピー、比内部エネルギーを計算する（24~41 行）。例えば比体積の場合には式 5.43 で計算することができる。

$$rate=(H-H_{sl})/(H_{sv}-H_{sl}) \tag{5.42}$$

$$\upsilon_{tp}=rate\times\upsilon_{sv}+(1-rate)\times\upsilon_{sl} \tag{5.43}$$

プログラム 5.10　比エンタルピーと圧力にもとづく物性値計算処理

Popolo.ThermophysicalProperty.Refrigerant class

```
1 /// <summary>圧力[kPa]と比エンタルピー[kJ/kg]から冷媒状態を計算する</summary>
2 /// <param name="pressure">圧力[kPa]</param>
3 /// <param name="enthalpy">比エンタルピー[kJ/kg]</param>
```

```
 4 /// <param name="temperature">絶対温度[K]</param>
 5 /// <param name="density">密度[kg/m3]</param>
 6 /// <param name="entropy">比エントロピー[kJ/kgK]</param>
 7 /// <param name="internalEnergy">比内部エネルギー[kJ/kg]</param>
 8 public void GetStateFromPressureAndEnthalpy(double pressure, double enthalpy,
 9   out double temperature, out double density, out double entropy, out double internalEnergy)
10 {
11   //冷媒の状態を判定
12   double rhoL, rhoV, tSat;
13   GetSaturatedPropertyFromPressure(pressure, out rhoL, out rhoV, out tSat);
14   Phase phase;
15   double hl = GetEnthalpyFromTemperatureAndDensity(tSat, rhoL);
16   double hv = GetEnthalpyFromTemperatureAndDensity(tSat, rhoV);
17   if (enthalpy < hl) phase = Phase.Liquid;
18   else
19   {
20     if (hv < enthalpy) phase = Phase.Vapor;
21     else phase = Phase.Equilibrium;
22   }
23
24   //冷媒の状態に応じて密度初期値を設定
25   if (phase == Phase.Equilibrium)
26   {
27     temperature = tSat;
28     double vRate = (enthalpy - hl) / (hv - hl);
29     double lRate = 1 - vRate;
30     //密度
31     density = 1 / (vRate / rhoV + lRate / rhoL);
32     //比エントロピー
33     double sL = GetEntropyFromTemperatureAndDensity(tSat, rhoL);
34     double sV = GetEntropyFromTemperatureAndDensity(tSat, rhoV);
35     entropy = sL * lRate + sV * vRate;
36     //比内部エネルギー
37     double uL = GetInternalEnergyFromTemperatureAndDensity(tSat, rhoL);
38     double uV = GetInternalEnergyFromTemperatureAndDensity(tSat, rhoV);
39     internalEnergy = uL * lRate + uV * vRate;
40     return; //二相域の場合には温度確定のため収束計算不要
41   }
42   else if (phase == Phase.Liquid)
43   {
44     temperature = tSat - 3;
45     density = rhoL;
46   }
47   else
48   {
49     temperature = tSat + 3;
50     density = rhoV;
51   }
52
53   //ニュートン法で温度を収束計算
54   double dns = density;
55   Roots.ErrorFunction eFnc = delegate (double tmp)
56   {
57     getDensityFromPressureAndTemperature(pressure, tmp, ref dns);
58     return enthalpy - GetEnthalpyFromTemperatureAndDensity(tmp, dns);
59   };
60   temperature = Roots.Newton(eFnc, temperature, 1e-5, 1e-3, 1e-3, 10);
61
62   //比エントロピー、比内部エネルギーを計算
63   entropy = GetEntropyFromTemperatureAndDensity(temperature, density);
64   internalEnergy = GetInternalEnergyFromTemperatureAndDensity(temperature, density);
65 }
```

5.3.6　比エントロピーと圧力にもとづく物性値の計算

比エントロピーと圧力にもとづく物性値の計算処理をプログラム 5.11 に示す。計算方法は比エンタルピーを入力とする場合と同様である。

プログラム 5.11　比エントロピーと圧力にもとづく物性値計算処理

Popolo.ThermophysicalProperty.Refrigerant class

```
 1 /// <summary>圧力[kPa]と比エントロピー[kJ/kg]から他の物性値を計算する</summary>
 2 /// <param name="pressure">圧力[kPa]</param>
 3 /// <param name="entropy">比エントロピー[kJ/kgK]</param>
 4 /// <param name="temperature">絶対温度[K]</param>
 5 /// <param name="density">密度[kg/m3]</param>
 6 /// <param name="enthalpy">比エンタルピー[kJ/kg]</param>
 7 /// <param name="internalEnergy">比内部エネルギー[kJ/kg]</param>
 8 public void GetStateFromPressureAndEntropy(double pressure, double entropy,
 9   out double temperature, out double density, out double enthalpy, out double internalEnergy)
```

```
10 {
11   //冷媒の状態を判定
12   double rhoL, rhoV, tSat;
13   GetSaturatedPropertyFromPressure(pressure, out rhoL, out rhoV, out tSat);
14   Phase phase;
15   double sl = GetEntropyFromTemperatureAndDensity(tSat, rhoL);
16   double sv = GetEntropyFromTemperatureAndDensity(tSat, rhoV);
17   if (entropy < sl) phase = Phase.Liquid;
18   else
19   {
20     if (sv < entropy) phase = Phase.Vapor;
21     else phase = Phase.Equilibrium;
22   }
23
24   //冷媒の状態に応じて密度初期値を設定
25   if (phase == Phase.Equilibrium)
26   {
27     temperature = tSat;
28     double vRate = (entropy - sl) / (sv - sl);
29     double lRate = 1 - vRate;
30     //密度
31     density = 1 / (vRate / rhoV + lRate / rhoL);
32     //比エンタルピー
33     double hL = GetEnthalpyFromTemperatureAndDensity(tSat, rhoL);
34     double hV = GetEnthalpyFromTemperatureAndDensity(tSat, rhoV);
35     enthalpy = hL * lRate + hV * vRate;
36     //比内部エネルギー
37     double uL = GetInternalEnergyFromTemperatureAndDensity(tSat, rhoL);
38     double uV = GetInternalEnergyFromTemperatureAndDensity(tSat, rhoV);
39     internalEnergy = uL * lRate + uV * vRate;
40     return; //二相域の場合には温度確定のため収束計算不要
41   }
42   else if (phase == Phase.Liquid)
43   {
44     temperature = tSat - 3;
45     density = rhoL;
46   }
47   else
48   {
49     temperature = tSat + 3;
50     density = rhoV;
51   }
52
53   //ニュートン法で温度を収束計算
54   double dns = density;
55   Roots.ErrorFunction eFnc = delegate (double tmp)
56   {
57     getDensityFromPressureAndTemperature(pressure, tmp, ref dns);
58     return entropy - GetEntropyFromTemperatureAndDensity(tmp, dns);
59   };
60   temperature = Roots.Newton(eFnc, temperature, 1e-5, 1e-3, 1e-3, 10);
61
62   //比エンタルピー、比内部エネルギーを計算
63   enthalpy = GetEnthalpyFromTemperatureAndDensity(temperature, density);
64   internalEnergy = GetInternalEnergyFromTemperatureAndDensity(temperature, density);
65 }
```

5.3.7　温度と圧力にもとづく物性値の計算

温度と圧力にもとづく物性値の計算処理をプログラム 5.12 に示す。1~46 行は他の物性を一括計算する処理であり、計算方法は比エンタルピーを入力とする場合と同様である。ただし、二相域の場合には温度情報から物性が定まらないため、25~31 行に示すように飽和状態の物性値を出力する。48~61 行は温度と圧力から密度のみを計算する処理である。

プログラム 5.12　温度と圧力にもとづく物性値計算処理

Popolo.ThermophysicalProperty.Refrigerant class

```
 1 /// <summary>圧力[kPa]と絶対温度[K]から他の物性値を計算する</summary>
 2 /// <param name="pressure">圧力[kPa]</param>
 3 /// <param name="temperature">絶対温度[K]</param>
 4 /// <param name="entropy">比エントロピー[kJ/kgK]</param>
 5 /// <param name="density">密度[kg/m3]</param>
 6 /// <param name="enthalpy">比エンタルピー[kJ/kg]</param>
 7 /// <param name="internalEnergy">比内部エネルギー[kJ/kg]</param>
 8 public void GetStateFromPressureAndTemperature(double pressure, double temperature,
 9   out double entropy, out double density, out double enthalpy, out double internalEnergy)
10 {
```

```
11   //冷媒の状態を判定
12   double rhoL, rhoV, tSat;
13   GetSaturatedPropertyFromPressure(pressure, out rhoL, out rhoV, out tSat);
14   Phase phase;
15   if (temperature < tSat) phase = Phase.Liquid;
16   else
17   {
18     if (tSat < temperature) phase = Phase.Vapor;
19     else phase = Phase.Equilibrium;
20   }
21
22   //冷媒の状態に応じて密度初期値を設定
23   if (phase == Phase.Equilibrium)
24   {
25     //二相域の場合には温度情報から物性は定まらない
26     //飽和液の物性を出力する
27     density = 1 / rhoL;
28     enthalpy = GetEnthalpyFromTemperatureAndDensity(tSat, rhoL);
29     internalEnergy = GetInternalEnergyFromTemperatureAndDensity(tSat, rhoL);
30     entropy = GetEntropyFromTemperatureAndDensity(tSat, rhoL);
31     return;
32   }
33   else if (phase == Phase.Liquid) density = rhoL + 0.1;
34   else density = rhoV - 0.1;
35
36   //ニュートン法で温度を収束計算
37   double tmp = temperature;
38   Roots.ErrorFunction eFnc = delegate (double dns)
39   { return pressure - GetPressureFromTemperatureAndDensity(tmp, dns); };
40   density = Roots.Newton(eFnc, density, 1e-5, 1e-3, 1e-3, 10);
41
42   //比エンタルピー、比内部エネルギー、エントロピーを計算
43   enthalpy = GetEnthalpyFromTemperatureAndDensity(temperature, density);
44   internalEnergy = GetInternalEnergyFromTemperatureAndDensity(temperature, density);
45   entropy = GetEntropyFromTemperatureAndDensity(temperature, density);
46 }
47
48 /// <summary>圧力[kPa]および温度[K]から密度[kg/m3]を計算する</summary>
49 /// <param name="pressure">圧力[kPa]</param>
50 /// <param name="temperature">温度[K]</param>
51 public double GetDensityFromPressureAndTemperature(double pressure, double temperature)
52 {
53   //密度の初期値を仮定
54   double rho, rhoL, rhoV, satT;
55   GetSaturatedPropertyFromPressure(pressure, out rhoL, out rhoV, out satT);
56   if (satT < temperature) rho = rhoV;
57   else rho = rhoL;
58
59   getDensityFromPressureAndTemperature(pressure, temperature, ref rho);
60   return rho;
61 }
```

【例題 5.1】

R410A について、飽和蒸気線および飽和液線ならびに下記条件の冷凍サイクルを計算し、 PQ 線図上に図示せよ。

・蒸発温度は 5 ℃ とする

・過熱度は 5 ℃、過冷却度は 10 ℃ とする

・凝縮圧力は 3500 kPa とする

・圧縮機では断熱圧縮が行われるとする

【解】プログラム 5.13 と図 5.2 に計算プログラムおよび計算結果の図を示す。なお、冷凍サイクルの詳細については第 14 章 圧縮式冷凍機で解説する。

プログラム 5.13　R410A の飽和蒸気線および飽和液線ならびに冷凍サイクルの計算

```
 1 public static void RefrigerantTest1()
 2 {
 3   using (StreamWriter sWriter = new StreamWriter
 4     ("RefrigerantTest1.csv", false, Encoding.GetEncoding("Shift_JIS")))
 5   {
 6     //冷媒種類は R410A
 7     Refrigerant r410a = new Refrigerant(Refrigerant.Fluid.R410A);
 8
 9     //飽和液と飽和蒸気の密度
10     double rhoSl, rhoSv;
11
12     //飽和液線と飽和蒸気線の描画
13     sWriter.WriteLine ("飽和 P[kPa], 飽和液 h[kJ/kg], 飽和蒸気 h[kJ/kg]");
```

```
14      //圧力範囲[kPa]
15      const double minPressure = 700;
16      const double maxPressure = 3900;
17      //飽和線の描画点数
18      const int plotNumber = 50;
19      double delta = (maxPressure - minPressure) / plotNumber;
20      for (int i = 0; i <= plotNumber; i++)
21      {
22        double satTemp;
23        double pressure = minPressure + delta * i;
24        r410a.GetSaturatedPropertyFromPressure
25          (pressure, out rhoSl, out rhoSv, out satTemp);
26        double rhoHl = r410a.GetEnthalpyFromTemperatureAndDensity(satTemp, rhoSl);
27        double rhoHv = r410a.GetEnthalpyFromTemperatureAndDensity(satTemp, rhoSv);
28        sWriter.WriteLine(pressure + "," + rhoHl + "," + rhoHv);
29      }
30
31      sWriter.WriteLine();
32      sWriter.WriteLine("冷凍サイクルの計算");
33
34      //冷凍サイクルの計算
35      double evapP;//蒸発圧力[kPa]
36      double evapT = 5; //蒸発温度[C]
37      r410a.GetSaturatedPropertyFromTemperature
38        (evapT + 273.15, out rhoSl, out rhoSv, out evapP);
39      double tCmpIn = evapT + 273.15 + 5;  //圧縮機入口温度[K] (過熱度 5C)
40      double rhoCmpIn = r410a.GetDensityFromPressureAndTemperature(evapP, tCmpIn);
41      double hCmpIn = r410a.GetEnthalpyFromTemperatureAndDensity(tCmpIn, rhoCmpIn);
42      double sCmpIn = r410a.GetEntropyFromTemperatureAndDensity(tCmpIn, rhoCmpIn);
43      sWriter.WriteLine("圧縮機入口比エンタルピー[kJ/kg], 圧縮機入口圧力[kPa]");
44      sWriter.WriteLine(hCmpIn + "," + evapP);
45
46      double condP = 3500; //凝縮圧力[kPa]
47      double tCmpOut, rhoCmpOut, hCmpOut, uCmpOut;
48      r410a.GetStateFromPressureAndEntropy
49        (condP, sCmpIn, out tCmpOut, out rhoCmpOut, out hCmpOut, out uCmpOut);
50      sWriter.WriteLine("圧縮機出口比エンタルピー[kJ/kg], 圧縮機出口圧力[kPa]");
51      sWriter.WriteLine(hCmpOut + "," + condP);
52
53      double condT; //凝縮温度[K]
54      r410a.GetSaturatedPropertyFromPressure(condP, out rhoSl, out rhoSv, out condT);
55      double tEvpIn = condT - 10;  //圧縮機入口温度[K] (過冷却度 10C)
56      double rhoEvpIn = r410a.GetDensityFromPressureAndTemperature (condP, tEvpIn);
57      double hEvpIn = r410a.GetEnthalpyFromTemperatureAndDensity(tEvpIn, rhoEvpIn);
58      double sEvpIn = r410a.GetEntropyFromTemperatureAndDensity(tEvpIn, rhoEvpIn);
59      sWriter.WriteLine("膨張弁入口比エンタルピー[kJ/kg], 膨張弁入口圧力[kPa]");
60      sWriter.WriteLine(hEvpIn + "," + condP);
61
62      sWriter.WriteLine("膨張弁出口比エンタルピー[kJ/kg], 膨張弁出口圧力[kPa]");
63      sWriter.WriteLine(hEvpIn + "," + evapP);
64    }
65  }
```

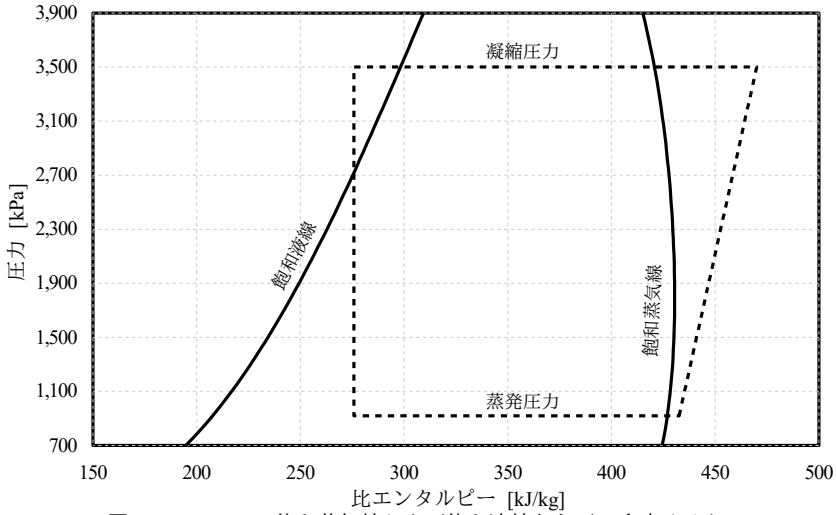


図 5.2　R410A の飽和蒸気線および飽和液線ならびに冷凍サイクル

【第 5 章 記号表】

A	：ヘルムホルツ自由エネルギー [kJ]	T	：絶対温度 [K]
C_p	：定圧比熱 [kJ/(kg·K)]	u	：比内部エネルギー [kJ/kg]
G	：ギブス自由エネルギー [kJ]	V	：比体積 [m^3/kg]
h	：比エンタルピー [kJ/kg]	Z	：圧縮係数 [-]
P	：圧力 [kPa]	δ	：対臨界密度 [-]
R	：ガス定数 [kPa·m^3/(kg·K)]	ρ	：密度 [kg/m^3]
s	：比エントロピー [kJ/(kg·K)]	τ	：対臨界温度 [-]
sub scripts			
c	：臨界点	sl	：飽和液
ref	：基準状態	sv	：飽和蒸気
s	：飽和状態	tp	：二相域
super scripts			
r	：理想気体からの偏差で表現した状態値	0	：理想気体の状態値

【第 5 章 参考文献】

5.1） Billington & Roberts; Building Service Engg Pergamon. Press 1982

5.2） Mark O. McLinden, Eric W. Lemmon and Richard T. Jacobsen : Thermodynamic properties ofr the alternative refrigerants, Int J. Refrige., Vol. 21, No.4, pp.322-338, 1998

5.3） Benedict, M., Webb, G.B. And Rubin, L.C., An empirical equation for thermodynamic properties of light hydrocarbons and their mixtures. I. Methane, ethane, propane and n-butane. J. Chem. Phys., 1940, 8, 334-345.

5.4） Jacobsen, R.T. and Stewart, R.B. : Thermodynamic properties of nitrogen including liquid and vapor phases from 63 K to 2000 K with pressures to 10,000 bar. J. Phys. Chem. Ref. Data, 2, (1973), pp.757-922.

5.5） C. Y. Chan and G. G. Haselden : Computer-based refrigerant thermodynamic properties. Part 1 Basic equations, International Journal of Refrigeration, Volume 4, Issue 1, (Jan., 1981), pp. 7–12

5.6） 機械工学便覧, α 基礎編, α5 熱力学, p.α5-22, 2007

5.7） Joseph J. Martin and Yu-Chun Hou : Development of an equation of state for gases, A.I.Ch.E. Journal, Vol.1, No.2, pp.142-151, June, 1955

5.8） R. Tillner-Roth and A. Yokozeki, An International Standard Equation of State for Difluoromethane (R-32) for Temperatures from the Triple Point at 136.34 K to 435 K and Pressures upto 70 MPa, J. Phys. Chem. Eng. Data, 26-6, (1997), pp.1273-1328

5.9） Younglove B.A. and McLinden, M.O. : An international standard equation-of-state formulation of the thermodynamic properties of refrigerant 123 (2,2-dichloro-1,1,1-trifluoroethane). J. Phys. Chem. Ref. Data, 23-5, (Oct., 1994), pp.731-779.,

5.10） Robert C. Reid, John M. Prausnitz and Bruce E. Poling : The properties of gases & liquids, Fourth edition, McGraw-Hill inc., pp.67

5.11） 森村雅人, 岡部智人, 佐藤春樹 ：HFC 系冷媒の液体状態式, The 24th Japan Symposium on Thermophysical Properties, 2003

5.12） Joseph J. Martin and Yu-Chun Hou : Development of an equation of state for gases, American Institute of Chemical Engineers. Journal, Vol.1, No.2, (June 1955), pp.142-151

5.13） K. E. Starling and J. F. Wolfe : Equation of state from multiproperty analysis - general development, proc. Okla. Acad. Sci. 52, (1972), pp.73-81

5.14） W. Wagner : A method to establish equations of state exactly representing all saturated state variables applied to nitrogen, Cryogenics, Volume 12, Issue 3, (June 1972), pp.214-221

5.15） Joseph J. Marting and Yu-chan hou : Development of an equation of state for gases, A.I.Ch.E. Journal, Vol.1, No.2, (June 1955) pp.142-151

5.16） Johannes Diderik van der Waals : Over de Continuiteit van den Gasen Vloeistoftoestand. Leiden, 1873.

5.17） 富樫英介 ：空調設備の年間エネルギーシミュレーションのための HFC32(R32)の熱物性値計算法の開発, 空気調和・衛生工学会学術論文集, No.204, pp.66-74 2014

5.18） 拓く ダイキン工業 90 年史, 第 1 章 ダイキンの礎 -町工場から大企業への躍進, p.16, ダイキン工業株式会社, 2015

第6章　日射
(Solar Irradiance)

6.1　概要

化石燃料は太古の太陽エネルギーであり、また、風力や地熱も太陽エネルギー拡散の不均一性にもとづくエネルギーである。従って、日射は最も根源的なエネルギーであるとも言える。

日射は、気象に関しての情報という意味では乾球温度や相対湿度などと同じであるが、これらと比較すると「方向」の概念があるために取り扱いがやや難しい。例えば、我々が通常入手できる日射データは水平面天空日射量や法線面直達日射量であるが、実際に、ある特定の面に入射する日射量を把握する際にはこれらの情報を基礎とした計算処理が必要となるからである。このためには、ある日時における太陽の位置を特定し、検討対象となる平面と太陽の幾何学的な位置関係を計算する必要がある。これらの計算は、建築のガラス面や躯体へ入射する日射量を把握し建築熱負荷計算を行うための基礎となる。また、太陽熱集熱器、太陽光発電パネルなどの太陽を利用した設備機器の評価にあたっても必須の計算処理である。

本章では、太陽位置の計算法、傾斜面に対する日射入射角の計算法、水平面全天日射量から水平面天空日射量および法線面直達日射量を計算する方法について解説する。

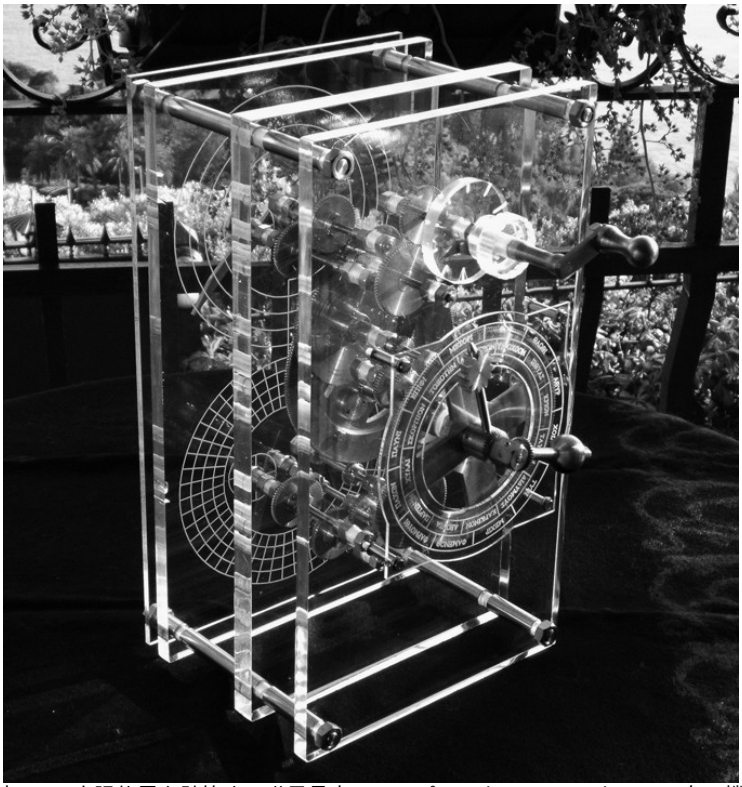


写真6.1　太陽位置を計算する世界最古のコンピュータ アンティキティラ島の機械

(*Creative commons Attribution-ShareAlike, Attribution: I, Mogi*)

6.2　理論

6.2.1　太陽位置

1）太陽高度と太陽方位角

太陽位置は地上から見た太陽の高さと方位によって定められる。図 6.1 に示す通り、太陽の高さは太陽高度 h、太陽の方位は太陽方位角 α で表現する。角度の取り方は自由であるが、太陽方位角については南を 0°、東を-90°、西を 90°、北を 180°と取ることが多い。また、太陽高度に関しては地平面を 0°、天頂を 90°と取ることとする。

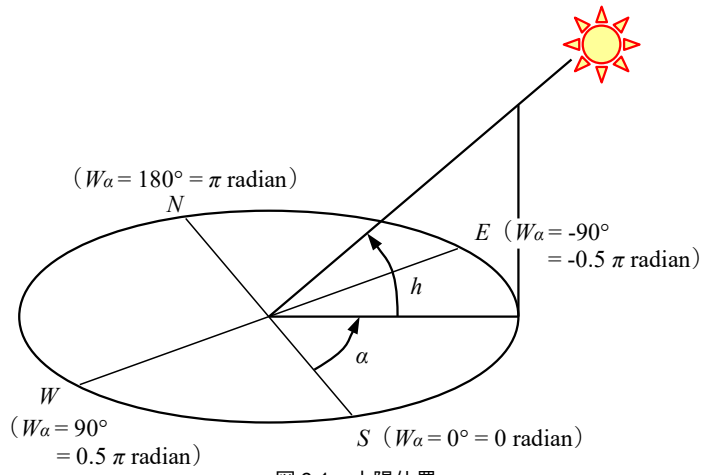


図 6.1　太陽位置

太陽高度 h と太陽方位角 α は式 6.1 と式 6.2 で計算する。

$$\sin h = \sin\varphi \sin\delta + \cos\varphi \cos\delta \cos t \tag{6.1}$$

$$\cos\alpha = \frac{\sin h \sin\varphi - \sin\delta}{\cos h \cos\varphi} \tag{6.2}$$

φ は緯度であり、北半球では正の値、南半球では負の値を設定する。

δ は太陽赤緯（赤道からのずれ）であり、元旦起算の通し日 n を用いて式 6.3 と式 6.4 で近似する。

$$\sin\delta = 0.397949 \sin B \tag{6.3}$$

$$B = 360\frac{n-81}{365} \tag{6.4}$$

t は時角であり、式 6.5 で計算する。

$$t = 15(\tau_t - 12) \tag{6.5}$$

τ_t は真太陽時であり、式 6.6 で計算する。

$$\tau_t = \tau_m + e \tag{6.6}$$

τ_m は平均太陽時[h]、e は均時差[h]であり、それぞれ式 6.7 と式 6.8 で計算する[6.5]。ただし τ_{st} と L は計算を行う地点の標準時と経度、L_s は標準時子午線の経度（日本の場合は兵庫県明石市であり 135°）である。

$$\tau_m = \tau_{st} + \frac{L - L_s}{15} \tag{6.7}$$

$$e = 0.1645 \sin 2B - 0.1255 \cos B - 0.025 \sin B \tag{6.8}$$

なお、式 6.2 を α について解くと式 6.9 が得られるが、α は $t > 0$ の時に正の値、$t < 0$ の場合に負の

値となる。

$$\alpha = \pm \arccos\left(\frac{\sin h \sin\varphi - \sin\delta}{\cos h \cos\varphi}\right) \tag{6.9}$$

【例題 6.1】

東京（北緯 35°41'、東経 139°46'）の 11 月 16 日 10:00 における太陽位置を計算せよ。

【解】

まず、緯度は 35 + 41 / 60 = 35.7°、経度は 139 + 46 / 60 = 139.8°である。

太陽赤緯 δ を計算する。11 月 16 日は元旦起算で 319 日目であるため、

B = 360 (319-80) / 365 = 236°

sin δ = 0.397949 sin(236°)

δ = -19.2°

均時差 e は、

e = 0.1645 sin(472°) - 0.1255 cos(236°) - 0.025 sin(236°) = 0.24 h = 14.7 min

となる。平均太陽時 τ_m は、

τ_m = 10 + (139.8 – 135) / 15 = 10.32

となる。平均太陽時 τ_m に均時差 e を加算し、真太陽時 τ_t は、

τ_t = 10.32 + 0.24 = 10.56

となる。従って時角 t は、

t = 15 × (10.56-12) = 21.57°

太陽高度 h は、

sin h = sin(35.7°) sin(-19.2°) + cos(35.7°) cos(-19.2°) cos(21.57°) = 0.522

h = 31.4°

となる。また cosα は、

cos α = (sin(31.4°) sin(35.7°) - sin(-19.2°)) / (cos(31.4°) cos(35.7°)) = 0.913

となり、時角 $t > 0$ のため、

α = +arccos(0.913) = -24°

となる。

2）日の出と日没

日の出の時角 t_{sr} [°]は式 6.10 で計算する。従って、標準時における日の出時刻 τ_{stsr} [hour]は式 6.5~6.7 を用いて式 6.11 で計算できる。また、1 日の可照時間 τ_d [hour]は式 6.12 で計算できる。日の出時刻に可照時間を加算することで日没時刻 τ_{stss} は式 6.13 で計算できる。あるいは式 6.10 の正負を逆転させれば式 6.11 により計算することもできる。ところで-tanφ tanδ が-1~1 の範囲外では式 6.10 の計算ができないが、北極圏および南極圏では-1 未満や 1 を超える季節がある。-1 を下回る場合には終日、日は沈まず、逆に 1 を超える場合には日は昇らない。これらはそれぞれ白夜、極夜と呼ばれる。

$$t_{sr} = -\cos^{-1}\left(-\tan\varphi \tan\delta\right) \tag{6.10}$$

$$\tau_{stsr} = \tau_{msr} - \frac{\left(L - L_s\right)}{15} = \tau_{tsr} - e - \frac{\left(L - L_s\right)}{15} = \frac{t_{sr} - \left(L - L_s\right)}{15} + 12 - e \tag{6.11}$$

$$\tau_d = 2\left|t_{sr}\right| / 15 \tag{6.12}$$

$$\tau_{stss} = \tau_{stsr} + \tau_d \tag{6.13}$$

6.2.2 傾斜面に入射する日射

1）日射の直達成分と拡散成分

傾斜面への入射を計算するためには、直達日射と拡散日射に分けて日射を捉える必要がある。

直達日射 I_D（Direct Raditaion）とは、雲や大気に吸収・拡散されること無く太陽から直接的に傾斜面に入射する日射である。直達日射は太陽との位置関係に依存する。従って、時々刻々移動する太

陽からの直達日射を定量的に捉えるために、各時刻の日射方向と垂直な傾斜面を想定し、この傾斜面に入射する日射を法線面直達日射 I_{DN}（Dirct Normal Radiation）と呼ぶ。

拡散日射 I_d（diffuse Radiation）とは、観測された全日射から直達日射を除いた部分である。雲や大気で散乱した後に傾斜面に到達する天空日射 I_{SKY}（Sky Radiation）や地表面に到達した後に反射光として傾斜面に到達する反射日射 I_R（Reflected Radiation）である。

地上の水平面における全日射を水平面全天日射 I_{HOL} と呼ぶ。この場合には反射日射は無い（I_R = 0）ため、$I_d = I_{SKY}$ であり、式 6.14 で計算できる。

$$I_{HOL} = I_{DN} \sin h + I_{SKY} \tag{6.14}$$

2）傾斜面に入射する直達日射

直達日射 I_D [W/m²]は法線面直達日射 I_{DN} [W/m²]を用いて式 6.15 で計算できる。

$$I_D = I_{DN} \cos\theta \tag{6.15}$$

θ は傾斜面の法線と太陽光線が成す角度であり、6.2.1 節で求めた太陽位置情報ならびに図 6.2 に示す傾斜面の傾き W_β および回転角度 W_α を用いて式 6.16 で計算できる。太陽方位と同じように、回転角度 W_α は南を 0°、東を-90°、西を 90°、北を 180°とする。傾きは水平面を 0°、垂直面を 90°とする。

$$\cos\theta = \sin h \cos W_\beta + \cos h \sin W_\beta \cos(\alpha - W_\alpha) \tag{6.16}$$

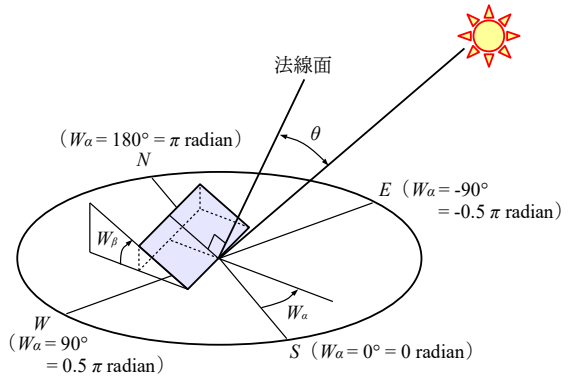


図 6.2　傾斜面の傾きと回転角度

3）傾斜面に入射する拡散日射

傾斜面に入射する拡散日射 I_{SR} [W/m²]は水平面天空日射 I_{SKY} [W/m²]と水平面全天日射 I_{HOL} [W/m²]を用いて式 6.17 で計算できる。ここで F_S [-]と F_G [-]はそれぞれ傾斜面の天空および地表面に対する形態係数であり、式 6.18 と式 6.19 で計算する。ρ_G [-]は地表面反射率でありアルベードとも呼ばれる。郊外の緑地などでは 0.2 程度[6.15]、都市部においては 0.3~0.5 程度[6.14]の値を取る。

$$I_{SR} = F_S I_{SKY} + \rho_G F_G I_{HOL} \tag{6.17}$$

$$F_S = \frac{1 + \cos W_\beta}{2} \tag{6.18}$$

$$F_G = 1 - F_S \tag{6.19}$$

【例題 6.2】

南南東（-22.5°）の向きに傾斜角度 55°と 35°で設置された太陽熱集熱器へ入射する日射の直達日射量をそれぞれ求めよ。ただし、地点および日時は例題 6.1 と同一の条件とする。また、法線面直達日射は 1,000 W/m² とする。

【解】

例題 6.1 により太陽高度 h は 35.7°であり、$\sin h = 0.522$、$\cos h = 0.853$ である。

55°の集熱器に関しては、

$\cos\theta = 0.522 \times \cos(55°) + \cos(35.7°) \times \sin(55°) \times \cos(-24° - (-22.5)) = 0.998$

となり、入射量は 1,000×0.998 = 998 W/m^2 となる。なお、$\theta = 3.8°$である。

同様に 35°の集熱器に関しては、

$\cos\theta = 0.522 \times \cos(35°) + \cos(35.7°) \times \sin(35°) \times \cos(-24° - (-22.5)) = 0.916$

となり、入射量は 1,000×0.916 = 916 W/m^2 となる。なお、$\theta = 23.6°$である。

太陽高度の低い冬季においては、傾斜角度の大きい太陽熱集熱器が有利であることがわかる。

4）見かけの太陽高度

日除けの計算などを行う際には見かけの太陽高度 ϕ [radian]という概念が必要になる。見かけの太陽高度 ϕ は、ある傾斜面の法線方向に対する太陽高度であり、プロファイル角（Profile Angle）と呼ばれることも多い。図 6.3 に見かけの太陽高度を示す。γ [radian]は傾斜面の法線の方位を基準にした太陽方位角であり、式 6.20 である。図 6.2 に示すような傾斜面の場合には、見かけの太陽高度 ϕ は式 6.21 で計算する。また、傾斜面から d_p だけ離れた位置にある物体は、垂直方向に d_{pH}、水平方向に d_{pW} ずれた位置に影を落とすが、図 6.2 からわかるようにこれらは式 6.22 と 6.23 で計算できる。

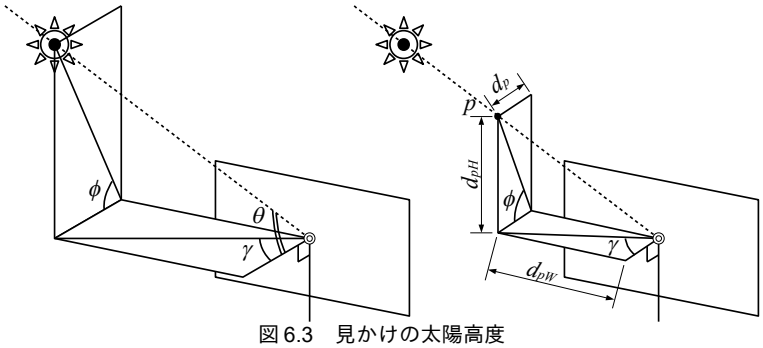


図 6.3　見かけの太陽高度

$$\gamma = \alpha - W_\alpha \tag{6.20}$$

$$\tan\gamma = \frac{\sin h \sin W_\beta - \cos h \cos W_\beta (\sin\alpha \sin W_\alpha - \cos\alpha \cos W_\alpha)}{\cos\theta} \tag{6.21}$$

$$d_{pH} = d_p \tan\phi \tag{6.22}$$

$$d_{pW} = d_p \tan\gamma \tag{6.23}$$

6.2.3　直散分離

通常、建物の屋上等に設置される日射量計は全天日射計であるため、BEMS（Building Energy and Environment Management System）などで収集できる日射は水平面全天日射 I_{HOL} [W/m^2]であることが多い†1)。一方で、前節までに記した通り、任意の傾斜面に入射する日射を計算するためには法線面直達日射 I_{DN} と水平面天空日射 I_{SKY} を別々に把握する必要がある。そこで、水平面全天日射 I_{HOL} を法線面直達日射 I_{DN} と水平面天空日射 I_{SKY} に分割する処理が必要となる。これは直散分離問題と呼ばれ、既に多くの研究成果が報告されている[6.7) 6.8) 6.10) 6.11) 6.12) 6.13)]。以下にいくつかの代表的な手法を示す。

1）宇田川・木村の手法[6.6)]

式 6.24~式 6.26 は宇田川、木村によって提案された推定法であり、反復計算が不要であるという利

†1　BEMS で日射関連のデータを読む際には単位に気をつける必要がある。「W/m^2」であれば瞬時値であり、概念としては「日射」となる。「Wh/m^2」であれば積算値であり、概念としては「日射量」となる。日射は太陽位置と関連が深いため、いずれの概念のもとで記録されたデータであるのか把握し、適切な時刻と結びつけて計算にかける必要がある。なお、英語では日射（瞬時値）を「Irradiance」、日射量（積算値）を「Insolation」と称する。

点がある。I_0は大気圏外日射 [W/m²]であり式 6.27 で近似する。I_{SC}は太陽定数であり 1,367 W/m²である[†1]。

$$I_{DN}=\begin{cases}(-0.43+1.43\,K_{Tt})\,I_0 & (I_{HOL}\geq K_{TtC})\\ (2.277-1.258\sin h+0.2396\sin^2 h)\,K_{Tt}^3\,I_0 & (I_{HOL}<K_{TtC})\end{cases} \tag{6.24}$$

$$K_{Tt}=\frac{I_{HOL}}{I_0\sin h} \tag{6.25}$$

$$K_{TtC}=(0.5163+0.333\sin h+0.00803\sin^2 h)\,I_0\sin h \tag{6.26}$$

$$I_0=I_{SC}\left\{1+0.033\cos\left(360\frac{n}{365}\right)\right\} \tag{6.27}$$

2）Erbs の手法[6.17)]

計算式は式 6.28 のとおりであり、反復計算不要な手法である。

$$I_{SKY}=\begin{cases}(1.0-0.09\,K_{Tt})\,I_0 & (K_{TtC}\leq 0.22)\\ \left(0.9511-0.1604\,K_{Tt}+4.388\,K_{Tt}^2-16.638\,K_{Tt}^3+12.336\,K_{Tt}^4\right)I_0 & (0.22<K_{TtC}\leq 0.8)\\ 0.1651\,I_0 & (0.8<K_{TtC})\end{cases} \tag{6.28}$$

3）永田の手法[6.9)]

式 6.29 に示すように、水平面天空日射 I_{SKY} [W/m²]を、水平面全天日射 I_{HOL} [W/m²]、法線面直達日射 I_{DN} [W/m²]、大気透過率 P [-]の関数としてモデル化する。大気透過率 P は式 6.30 に示す Bouguer の式により計算する。反復計算が必要であり、まず適当な大気透過率 P を設定して式 6.30 によって法線面直達日射 I_{DN}を計算する。次に法線面直達日射 I_{DN}を用いて式 6.29 で水平面天空日射 I_{SKY}を計算する。これらを式 6.14 に代入すれば水平面全天日射 I_{HOL}が求まるため、この値と観測値との差を誤差と評価して大気透過率 P を更新する。本手法は HASP 用標準年気象データの作成にも用いられた歴史的な手法であるが、晴天日を前提としたモデルであるため、曇天日に関してはやや天空日射が小さく計算される傾向がある。

$$I_{SKY}=\sin h\,(I_0-I_{DN})(0.66-0.32\sin h)\left\{0.5+(0.4-0.3\,P)\sin h\right\} \tag{6.29}$$

$$I_{DN}=I_0\,P^{1/\sin h} \tag{6.30}$$

【例題 6.3】

例題 6.1 と同一の日時において、水平面全天日射 I_{HOL}が 500W/m²と記録された。直散分離を行い、法線面直達日射 I_{DN}と水平面天空日射 I_{SKY}に分離せよ。

【解】

例題 6.1 により太陽高度 h は 35.7°であるため、sin h = 0.522 である。式 6.27 により、大気圏外日射は、

I_0 = 1,370 × {1+0.033 cos(360 × 319 / 365)} = 1,402 W/m²

である。

また、K_{Tt}およびK_{TtC}は各々、

K_{Tt} = 500 / (1,402 × 0.522) = 0.684

K_{TtC} = (0.5163+0.333 × 0.522 + 0.00803 × 0.522²) ×0.522 × 1,402 = 506

I_{HOL} < K_{TtC} であるため式 6.24 下段を用い、

I_{DN} = 2.277 – 1.258 × 0.522 + 0.2396 × 0.522² = 756 W/m²

となる。式 6.15 により、水平面天空日射 I_{SKY}は、

I_{SKY} = 500 – 756 × 0.522 = 106 W/m²

となる。

†1　第 8 回 測器・観測法委員会（CIMO-VIII, 1981）による勧告値

6.2.4 夜間放射

本節の内容は日射とは直接には関わらないが、日射と同様に建築外表面における熱収支式に表れることが多いため、便宜的に本章で取り扱う。

大気は地表面に向けて熱を放射し、地表面は宇宙に向けて熱を放射する。この両者の差分を実効放射 R_{eff} [W/m^2]と呼ぶ。大気から地表面に向けた放射を大気放射 R_{atm} [W/m^2]と呼び、式 6.31 で計算する。天空が大気温度 T_{atm} [K]に等しい固体（黒色）に覆われているならば、放射熱移動の理論式に従い、黒体の放射定数 σ [W/(m^2·K^4)]に絶対温度の 4 乗を乗じた熱流となる。しかし現実には大気中の水滴（雲）や塵以外はこの条件を満たさないため、放射量を低減する必要がある。式 6.31 の ε_{atm} [-]はこの低減係数であり、雲量 C_R [-]を用いて式 6.32 で計算できる。雲量とは全天に占める雲の割合であり、日本では 0~10 までの 11 段階で表現される。B_r は射出率[-]と呼ばれ、水蒸気分圧 P_W [kPa]を用いて式 6.33（Brunt の式）で計算する。Brunt の式は晴天時の実測値にもとづくものであるため、式 6.32 では Philipps の研究結果を用いて雲量の影響を表現している[6.18) 6.16)]。

$$R_{atm}=\varepsilon_{atm}\sigma T_{atm}^4 \tag{6.31}$$

$$\varepsilon_{atm}=\left(1-0.62\frac{C_R}{10}\right)B_r+0.62\frac{C_R}{10} \tag{6.32}$$

$$B_r=0.526+0.209\sqrt{P_w} \tag{6.33}$$

地表面から大気へ向けた放射から式 6.31 を差し引き、実効放射 R_{eff} は式 6.34 で計算できる。ただし T_g [K]は地表面の温度である。

$$R_{eff}=\sigma T_g^4-R_{atm}=\sigma\left(T_g^4-\varepsilon_{atm}T_{atm}^4\right) \tag{6.34}$$

地表面温度 T_g と大気温度 T_{atm} が等しいとすれば、式 6.34 は式 6.35 で表すことができ、この場合の実効放射を特に夜間放射 R_N [W/m^2]と呼ぶ。熱負荷計算において屋外の物体の表面温度が与えられることは少なく、大気温度と等しいとみなさざるを得ないことが殆どであるため、建築外表面の熱収支を計算する際には式 6.35 の夜間放射を用いることが多い。

$$R_N=\left(1-0.62\frac{C_R}{10}\right)\left(1-B_r\right)\sigma T_{atm}^4 \tag{6.35}$$

6.3 計算法

6.3.1 太陽位置の計算

太陽位置の計算プログラムを 6.1 に示す。18~24 行は時角の計算処理であり、式 6.4~式 6.8 を用いる。29 行で式 6.1 を用いて sin$h$ を求めるが、この値が負の場合には日の出前であるため 34~35 行で高度・方位ともに 0 を設定して計算を打ち切る。39~43 行は日中の場合（0 < sin$h$）の処理であり、式 6.2 を用いて太陽方位角 α を計算する。47~59 行と 61~73 行はそれぞれ太陽高度と太陽方位角のみを出力する関数である。

プログラム 6.1　太陽位置の計算

Popolo.Weather.Sun class

```
1 /// <summary>Degree を Radian に変換する係数</summary>
2 private const double DEG_TO_RAD = 2d * Math.PI / 360d;
3
4 /// <summary>太陽位置を計算する</summary>
5 /// <param name="latitude">計算地点の緯度（北緯が正）[degree]</param>
6 /// <param name="longitude">計算地点の経度（北緯が正）[degree]</param>
```

```
 7 /// <param name="standardLongitude">標準時地点の経度（東経が正）[degree]</param>
 8 /// <param name="dTime">日時</param>
 9 /// <param name="altitude">太陽高度[rad]</param>
10 /// <param name="orientation">太陽方位角[rad]</param>
11 public static void GetSunPosition
12   (double latitude, double longitude, double standardLongitude, DateTime dTime,
13   out double altitude, out double orientation)
14 {
15   //緯度をRadianに変換
16   double phi = DEG_TO_RAD * latitude;
17
18   double b = (360d * (dTime.DayOfYear - 81) / 365d) * DEG_TO_RAD;
19   double sd = 0.397949 * Math.Sin(b);
20   double cd = Math.Sqrt(1 - sd * sd);
21   double e = 0.1645 * Math.Sin(2 * b) - 0.1255 * Math.Cos(b) - 0.025 * Math.Sin(b);
22   double taum = dTime.Hour + dTime.Minute / 60d + dTime.Second / 3600d
23   + e + (longitude - standardLongitude) / 15d;
24   double t = ((taum - 12) * 15) * DEG_TO_RAD;
25
26   //Sin太陽高度を計算
27   double sp = Math.Sin(phi);
28   double cp = Math.Cos(phi);
29   double sh = sp * sd + cp * cd * Math.Cos(t);
30
31   //日の出前、日没後の場合
32   if (sh <= 0)
33   {
34     altitude = 0;
35     orientation = 0;
36   }
37   else
38   {
39     altitude = Math.Asin(sh);
40     double ch = Math.Sqrt(1.0 - sh * sh);
41     double ca = (sh * sp - sd) / (ch * cp);
42     orientation = Math.Acos(ca);
43     orientation *= Math.Sign(t);
44   }
45 }
46
47 /// <summary>太陽高度[radian]を計算する</summary>
48 /// <param name="latitude">緯度[degree]</param>
49 /// <param name="longitude">計算地点の経度[degree]</param>
50 /// <param name="standardLongitude">標準時地点の経度（東経が正）[degree]</param>
51 /// <param name="dTime">日時</param>
52 /// <returns>太陽高度[radian]</returns>
53 public static double GetSunAltitude
54   (double latitude, double longitude, double standardLongitude, DateTime dTime)
55 {
56   double altitude, azimuth;
57   GetSunPosition(latitude, longitude, standardLongitude, dTime, out altitude, out azimuth);
58   return altitude;
59 }
60
61 /// <summary>太陽方位角[radian]を計算する</summary>
62 /// <param name="latitude">緯度[degree]</param>
63 /// <param name="longitude">計算地点の経度[degree]</param>
64 /// <param name="standardLongitude">標準時地点の経度（東経が正）[degree]</param>
65 /// <param name="dTime">日時</param>
66 /// <returns>太陽方位角[radian]</returns>
67 public static double GetSunAzimuth
68   (double latitude, double longitude, double standardLongitude, DateTime dTime)
69 {
70   double altitude, azimuth;
71   GetSunPosition(latitude, longitude, standardLongitude, dTime, out altitude, out azimuth);
72   return azimuth;
73 }
```

プログラム 6.2 に日の出および日没時刻の計算処理を示す。1~9 行および 11~19 行は、日の出と日没のいずれかの処理かを切り替えるためのメソッドであり、本体は 21~52 行である。38 行までに-$\tan\varphi$ $\tan\delta$ を求め、-1~1 の範囲内に無い場合には白夜または極夜として処理を終える。式 6.10 および式 6.11 を適用し、44 行で標準時における日の出または日没時刻 τ_{stsr} を計算する。日の出と日没に関しては 43 行で正負を調整して切り替える。τ_{stsr} の単位は[hour]であるため、46~50 行で端数を[min]および[sec]に換算する。

プログラム **6.2**　日の出および日没時刻の計算

Popolo.Weather.Sun class

```
/// <summary>日没の時刻を求める</summary>
/// <param name="latitude">計算地点の緯度[degree]</param>
/// <param name="longitude">計算地点の経度[degree]</param>
/// <param name="standardLongitude">標準時地点の経度（東経が正）[degree]</param>
/// <param name="dTime">日時</param>
/// <returns>日没の時刻</returns>
public static DateTime GetSunSetTime
  (double latitude, double longitude, double standardLongitude, DateTime dTime)
{ return getSunRiseAndSunSetTime(latitude, longitude, standardLongitude, dTime, true); }

/// <summary>日の出の時刻を求める</summary>
/// <param name="latitude">計算地点の緯度[degree]</param>
/// <param name="longitude">計算地点の経度[degree]</param>
/// <param name="standardLongitude">標準時地点の経度（東経が正）[degree]</param>
/// <param name="dTime">日時</param>
/// <returns>日の出の時刻</returns>
public static DateTime GetSunRiseTime
  (double latitude, double longitude, double standardLongitude, DateTime dTime)
{ return getSunRiseAndSunSetTime(latitude, longitude, standardLongitude,dTime, false); }

/// <summary>日の出または日没時刻を求める</summary>
/// <param name="latitude">計算地点の緯度[degree]</param>
/// <param name="longitude">計算地点の経度[degree]</param>
/// <param name="standardLongitude">標準時地点の経度（東経が正）[degree]</param>
/// <param name="dTime">日時</param>
/// <param name="isSunSet">計算対象は日没か否か</param>
/// <returns>日の出の時刻</returns>
private static DateTime getSunRiseAndSunSetTime
  (double latitude, double longitude, double standardLongitude, DateTime dTime, bool isSunSet)
{
  //緯度をRadianに変換
  double phi = DEG_TO_RAD * latitude;

  double b = (360d * (dTime.DayOfYear - 81) / 365d) * DEG_TO_RAD;
  double sd = 0.397949 * Math.Sin(b);
  double tptd = -Math.Tan(phi) * Math.Tan(Math.Asin(sd));
  //白夜・極夜の場合
  if (tptd < -1 || 1 < tptd) return new DateTime(dTime.Year, dTime.Month, dTime.Day, 0, 0, 0);

  double cd = Math.Sqrt(1 - sd * sd);
  double e = 0.1645 * Math.Sin(2 * b) - 0.1255 * Math.Cos(b) - 0.025 * Math.Sin(b);
  double tsr = -Math.Acos(tptd) / DEG_TO_RAD;
  if (isSunSet) tsr *= -1;
  double tstsr = (tsr - longitude + standardLongitude) / 15 + 12 - e;

  int hour = (int)Math.Truncate(tstsr);
  tstsr = 60 * (tstsr - hour);
  int minute = (int)Math.Truncate(tstsr);
  tstsr = 60 * (tstsr - minute);
  int sec = (int)Math.Truncate(tstsr);
  return new DateTime(dTime.Year, dTime.Month, dTime.Day, hour, minute, sec);
}
```

6.3.2　傾斜面に入射する日射の計算

傾斜面への日射を計算するにあたっては傾斜面の幾何学的情報を保持するクラスを定義しておくと便利である。このようなクラスを用意しておけば、窓ガラス・壁面・太陽熱集熱器等の計算の際に再利用できる。傾斜面（Incline）クラスの列挙型・インスタンス変数・プロパティの定義をプログラム 6.3 に示す。

プログラム **6.3**　傾斜面クラスの列挙型・インスタンス変数・プロパティの定義

Popolo.Weather.Incline class

```
/// <summary>16方位</summary>
public enum Orientation
{
  /// <summary>北北東</summary>
  NNE = -7,
  /// <summary>北東</summary>
  NE = -6,
  /// <summary>東北東</summary>
  ENE = -5,
  /// <summary>東</summary>
  E = -4,
  /// <summary>東南東</summary>
```

```
 13   ESE = -3,
 14   /// <summary>南東</summary>
 15   SE = -2,
 16   /// <summary>南南東</summary>
 17   SSE = -1,
 18   /// <summary>南</summary>
 19   S = 0,
 20   /// <summary>南南西</summary>
 21   SSW = 1,
 22   /// <summary>南西</summary>
 23   SW = 2,
 24   /// <summary>西南西</summary>
 25   WSW = 3,
 26   /// <summary>西</summary>
 27   W = 4,
 28   /// <summary>西北西</summary>
 29   WNW = 5,
 30   /// <summary>北西</summary>
 31   NW = 6,
 32   /// <summary>北北西</summary>
 33   NNW = 7,
 34   /// <summary>北</summary>
 35   N = 8
 36 }
 37
 38 /// <summary>正弦、余弦</summary>
 39 private double sinBeta, cosBeta;
 40
 41 /// <summary>方位角[radian]（南を0、東を負、西を正とする）を取得する</summary>
 42 public double HorizontalAngle { private set; get; }
 43
 44 /// <summary>傾斜角[radian]（水平面を0、垂直面を1/2π とする）を取得する</summary>
 45 public double VerticalAngle { private set; get; }
 46
 47 /// <summary>天空への形態係数[-]を取得する</summary>
 48 public double ConfigurationFactorToSky { private set; get; }
 49
 50 /// <summary>地表面への形態係数[-]を取得する</summary>
 51 public double ConfigurationFactorToGround
 52 {
 53   get { return 1 - ConfigurationFactorToSky; }
 54 }
```

プログラム 6.4 に傾斜面クラスのコンストラクタを示す。7~15 行は前処理であり、方位と傾きの値を範囲内に調整する。18 行は天空への形態係数の計算であり、式 6.18 にもとづき 31~35 行で関数を定義している。式 6.16 の中で $\sin W_\beta$ と $\cos W_\beta$ は太陽位置に依存しない数値であるため、21~22 行で初期化する。25~29 行はコンストラクタのオーバーロードである。方位を実数で指定するのではなく、プログラム 6.3 で定義した 16 方位を用いて指定する。このような定義を行っておくと、クラスを使用する際に西回りが正、東回りが負などのルールを覚える必要がなくなり便利である。

プログラム 6.4　傾斜面クラスのコンストラクタ

Popolo.Weather.Incline class

```
  1 /// <summary>インスタンスを初期化する</summary>
  2 /// <param name="horizontalAngle">方位角[radian]（南：0、東：負、西：正）</param>
  3 /// <param name="verticalAngle">傾斜角[radian]（水平面：0、垂直面：1/2π）</param>
  4 public Incline(double horizontalAngle, double verticalAngle)
  5 {
  6   //方位角は-180~180
  7   double pi2 = Math.PI * 2;
  8   horizontalAngle = horizontalAngle % pi2;
  9   if (Math.PI < horizontalAngle) HorizontalAngle = horizontalAngle - pi2;
 10   else if (horizontalAngle < -Math.PI) HorizontalAngle = horizontalAngle + pi2;
 11   else HorizontalAngle = horizontalAngle;
 12
 13   //傾斜角は0~180
 14   VerticalAngle = verticalAngle % Math.PI;
 15   if (Math.PI < VerticalAngle) VerticalAngle = Math.PI - VerticalAngle;
 16
 17   //天空への形態係数を計算
 18   ConfigurationFactorToSky = GetConfigurationFactorToSky(verticalAngle);
 19
 20   //傾斜面固有の正接・余弦を計算しておく
 21   sinBeta = Math.Sin(verticalAngle);
```

```
22   cosBeta = Math.Cos(verticalAngle);
23 }
24
25 /// <summary>コンストラクタ</summary>
26 /// <param name="orientation">16 方位</param>
27 /// <param name="verticalAngle">傾斜角[radian]（水平面：0、垂直面：1/2π）</param>
28 public Incline(Orientation orientation, double verticalAngle)
29   : this(Math.PI / 8 * (int)orientation, verticalAngle) { }
30
31 /// <summary>天空に対する傾斜面の形態係数[-]を計算する</summary>
32 /// <param name="verticalAngle">傾斜面の傾斜角[radian]（水平面を 0 とする）</param>
33 /// <returns>天空に対する傾斜面の形態係数[-]</returns>
34 public static double GetConfigurationFactorToSky(double verticalAngle)
35 { return (1d + Math.Cos(verticalAngle)) / 2d; }
```

プログラム 6.5 に、傾斜面に入射する日射を計算する処理を示す。コンストラクタで初期化しておいた $\sin W_\beta$ と $\cos W_\beta$ を用いながら、式 6.15~6.19で計算を行う。太陽高度と太陽方位角を実数で直接入力するメソッドの他に、太陽クラス（後述）のインスタンスを引数で取るメソッドも用意する。

プログラム 6.5　傾斜面に入射する日射の計算

Popolo.Weather.Incline class

```
 1 /// <summary>傾斜面の法線に対する太陽光線入射角の余弦 cosθ[-]を計算する</summary>
 2 /// <param name="altitude">太陽高度[radian]</param>
 3 /// <param name="orientation">太陽方位角[radian]</param>
 4 /// <returns>傾斜面の法線に対する太陽光線入射角の余弦 cosθ[-]</returns>
 5 public double GetDirectSolarRadiationRate(double altitude, double orientation)
 6 {
 7   double sh = Math.Sin(altitude);
 8   double ch = Math.Cos(altitude);
 9   double caa = Math.Cos(orientation - HorizontalAngle);
10   return Math.Max(0, sh * cosBeta + ch * sinBeta * caa);
11 }
12
13 /// <summary>傾斜面の法線に対する太陽光線入射角の余弦 cosθ[-]を計算する</summary>
14 /// <param name="sun">太陽</param>
15 /// <returns>傾斜面の法線に対する太陽光線入射角の余弦 cosθ[-]</returns>
16 public double GetDirectSolarRadiationRate(ImmutableSun sun)
17 { return GetDirectSolarRadiationRate(sun.Altitude, sun.Orientation); }
18
19 /// <summary>傾斜面に入射する直達日射[W/m2]を計算する</summary>
20 /// <param name="sun">太陽</param>
21 /// <returns>傾斜面に入射する直達日射[W/m2]</returns>
22 public double GetDirectSolarIrradiance(ImmutableSun sun)
23 { return GetDirectSolarRadiationRate(sun) * sun.DirectNormalRadiation; }
24
25 /// <summary>傾斜面に入射する拡散日射[W/m2]を計算する</summary>
26 /// <param name="sun">太陽</param>
27 /// <param name="albedo">地表面反射率[-]</param>
28 /// <returns>傾斜面に入射する拡散日射[W/m2]</returns>
29 public double GetDiffuseSolarIrradiance(ImmutableSun sun, double albedo)
30 {
31   return ConfigurationFactorToSky * sun.DiffuseHorizontalRadiation
32     + albedo * ConfigurationFactorToGround * sun.GlobalHorizontalRadiation;
33 }
34
35 /// <summary>傾斜面に入射する日射[W/m2]を計算する</summary>
36 /// <param name="sun">太陽</param>
37 /// <param name="albedo">地表面反射率[-]</param>
38 /// <returns>傾斜面に入射する日射[W/m2]</returns>
39 public double GetSolarIrradiance(ImmutableSun sun, double albedo)
40 { return GetDirectSolarIrradiance(sun) + GetDiffuseSolarIrradiance(sun, albedo); }
```

6.3.3　直散分離の計算

プログラム 6.6 に直散分離手法の列挙型定義を示す。6.2.3 節で示していない手法も幾つか実装するが、いずれも 6.2.3 節の計算方法で対応できる。

プログラム 6.6　直散分離手法の列挙型定義

Popolo.Weather.Sun class

```
1 /// <summary>直散分離の手法</summary>
2 public enum SeparationMethod
3 {
4   /// <summary>Berlage</summary>
5   Berlage,
6   /// <summary>松尾</summary>
7   Matsuo,
```

```
 8   /// <summary>永田</summary>
 9   Nagata,
10   /// <summary>Liu-Jordan</summary>
11   LiuAndJordan,
12   /// <summary>宇田川</summary>
13   Udagawa,
14   /// <summary>渡辺</summary>
15   Watanabe,
16   /// <summary>赤坂</summary>
17   Akasaka,
18   /// <summary>三木</summary>
19   Miki,
20   /// <summary>Erbs</summary>
21   Erbs
22 }
```

プログラム 6.7 に直散分離の計算を示す。4~97 行がメインプログラムである。18~24 行は前処理であり、太陽高度が 0 以下の場合あるいは全天日射が 0 以下の場合には直達日射および天空日射も 0 となる。25 行は著しく太陽高度が低い場合の処理である。sin h で除する計算処理が発生するため、h に下限値を設けている。29 行は大気圏外日射の計算処理であり、式 6.27 に従い、99~105 行に定義した関数を用いて計算する。

31 行以降は、手法別の計算処理である。反復計算が不要な宇田川、三木、Erbs の手法に関してはそれぞれ、31~39 行、40~49 行、50~62 行で計算する。大気透過率の反復計算が必要な手法に関しては 64~93 行で計算を行う。大気透過率の範囲は 0~1 までであるから、まず 0 と 1 で計算を行い、解が存在するかを確認する。大気透過率にもとづき天空日射と直達日射を計算する関数は 107~151 行に定義している。手法別に条件分岐をかけており、例えば 6.2.3 節で解説した永田の手法であれば 141~144 行で計算を行う。66~73 行は大気透過率 0 で計算値が全天日射を上回る場合の処理であり、天空日射 I_{SKY}=観測値とし、直達日射 I_{DN} には 0 を設定する。逆に大気透過率 1 でも計算値が全天日射を下回る場合には、75~83 行に示すように、計算によって得られた直達日射と天空日射の比率で観測値を按分する。これ以外の場合には大気透過率が 0~1 の範囲で解を持つということであるから、2.3.2 節で示した二分法を用いて 0~1 の間で解を求める。誤差関数は 87~91 行に示す通りである。

プログラム 6.7　直散分離の計算

Popolo.Weather.Sun class

```
 1 /// <summary>太陽定数[W/m2]</summary>
 2 public const double SOLAR_CONSTANT = 1367d;
 3
 4 /// <summary>水平面全天日射[W/m2]の直散分離を行う</summary>
 5 /// <param name="globalHorizontalRadiation">水平面全天日射[W/m2]</param>
 6 /// <param name="latitude">緯度[degree]</param>
 7 /// <param name="longitude">計算地点の経度[degree]</param>
 8 /// <param name="standardLongitude">標準時地点の経度（東経で正）[degree]</param>
 9 /// <param name="dTime">日時</param>
10 /// <param name="directSolarRadiation">法線面直達日射[W/m2]</param>
11 /// <param name="method">直散分離の手法</param>
12 /// <param name="diffuseHorizontalRadiation">天空日射[W/m2]</param>
13 public static void SeparateGlobalHorizontalRadiation
14   (double globalHorizontalRadiation, double latitude, double longitude,
15   double standardLongitude, DateTime dTime, SeparationMethod method,
16   out double directSolarRadiation, out double diffuseHorizontalRadiation)
17 {
18   //太陽高度の正弦の計算
19   double h = GetSunAltitude(latitude, longitude, standardLongitude, dTime);
20   if (h <= 0 || globalHorizontalRadiation <= 0)
21   {
22     directSolarRadiation = diffuseHorizontalRadiation = 0;
23     return;
24   }
25   h = Math.Max(0.05, h);  //誤差が拡大するため３度以上とする
26   double sinH = Math.Sin(h);
27
```

```
  //大気圏外日射[w/m2]の計算
  double io = GetExtraterrestrialRadiation(dTime.DayOfYear);
  double dn, dff;
  //宇田川の手法
  if (method == SeparationMethod.Udagawa)
  {
    double ktt = globalHorizontalRadiation / (io * sinH);
    double ktc = io * sinH * (0.5163 + sinH * (0.333 + 0.00803 * sinH));
    if (ktc <= globalHorizontalRadiation) dn = (-0.43 + 1.43 * ktt) * io;
    else dn = (2.277 + sinH * (-1.258 + 0.2396 * sinH)) * Math.Pow(ktt, 3) * io;
    dff = globalHorizontalRadiation - dn * sinH;
  }
  //Erbsの手法
  else if (method == SeparationMethod.Erbs)
  {
    double ktt = globalHorizontalRadiation / (io * sinH);
    if (ktt < 0.22) dff = globalHorizontalRadiation * (1.0 - 0.09 * ktt);
    else if (ktt < 0.8) dff = globalHorizontalRadiation * (0.9511 + ktt * (-0.1604
      + ktt * (4.388 + ktt * (-16.638 + 12.336 * ktt))));
    else dff = 0.1651 * globalHorizontalRadiation;
    dn = (globalHorizontalRadiation - dff) / sinH;
  }
  //三木の手法
  else if (method == SeparationMethod.Miki)
  {
    double lkt = Math.Min(1, globalHorizontalRadiation / (io * sinH));
    double skt = (lkt - 0.15 - 0.2 * sinH) / 0.6;
    double skd;
    if (skt <= 0) skd = 0;
    else skd = skt * skt * (3 - 2 * skt);
    double lkd = Math.Min(skd * lkt, Math.Pow(0.8, (7 + sinH) / (1 + 7 * sinH)));
    double lks = Math.Max(lkt - lkd, 0.005);
    dn = lkd * io;
    dff = globalHorizontalRadiation - dn * sinH;
  }
  //数値計算による方法
  else
  {
    //大気透過率=0で観測値を上回る場合
    getDirectAndDiffuseRadiation(0, sinH, io, method, out dn, out dff);
    if (globalHorizontalRadiation < dn * sinH + dff)
    {
      directSolarRadiation = 0;
      diffuseHorizontalRadiation = globalHorizontalRadiation;
      return;
    }

    //大気透過率=1で観測値を下回る場合
    getDirectAndDiffuseRadiation(1, sinH, io, method, out dn, out dff);
    if (dn * sinH + dff < globalHorizontalRadiation)
    {
      double rate = globalHorizontalRadiation / (dn * sinH + dff);
      directSolarRadiation = dn * sinH * rate;
      diffuseHorizontalRadiation = dff * rate;
      return;
    }

    //大気透過率を二分法で収束計算
    //誤差関数を定義
    Roots.ErrorFunction eFnc = delegate(double atmTrans)
    {
      getDirectAndDiffuseRadiation(atmTrans, sinH, io, method, out dn, out dff);
      return globalHorizontalRadiation - (dn * sinH + dff);
    };
    Roots.Bisection(0, 1, 0.001, eFnc);
  }
  //0以上とする
  directSolarRadiation = Math.Max(0, dn);
  diffuseHorizontalRadiation = Math.Max(0, dff);
}

/// <summary>大気圏外日射[W/m2]を計算する</summary>
/// <param name="daysOfYear">通日(1月1日=1, 12月31日=365)</param>
/// <returns>大気圏外日射[W/m2]</returns>
public static double GetExtraterrestrialRadiation(int daysOfYear)
{
  return SOLAR_CONSTANT * (1d + 0.033 * Math.Cos(2d * Math.PI * daysOfYear / 365d));
}

/// <summary>法線面直達日射[W/m2]と水平面天空日射[W/m2]を計算する</summary>
```

```
/// <param name="aTransmissivity">大気透過率[-]</param>
/// <param name="sinAltitude">太陽高度の正弦</param>
/// <param name="exRadiation">大気圏外日射[W/m2]</param>
/// <param name="method">直散分離手法</param>
/// <param name="directNormalRadiation">法線面直達日射[W/m2]</param>
/// <param name="diffuseHorizontalRadiation">水平面天空日射[W/m2]</param>
private static void getDirectAndDiffuseRadiation
  (double aTransmissivity, double sinAltitude, double exRadiation,
  SeparationMethod method,
  out double directNormalRadiation, out double diffuseHorizontalRadiation)
{
  double ps = Math.Pow(aTransmissivity, 1 / sinAltitude);
  directNormalRadiation = exRadiation * ps;
  double shi = sinAltitude * (exRadiation - directNormalRadiation);
  diffuseHorizontalRadiation = 0;

  switch (method)
  {
    case SeparationMethod.Akasaka:
      diffuseHorizontalRadiation =
    shi * 0.95 * Math.Pow(aTransmissivity, 1 / (0.5 + 2.5 * sinAltitude)) *
    Math.Pow(1 - aTransmissivity, 2d / 3d);
      break;
    case SeparationMethod.Berlage:
      diffuseHorizontalRadiation = shi * 0.5 / (1 - 1.4 * Math.Log(aTransmissivity));
      break;
    case SeparationMethod.LiuAndJordan:
      diffuseHorizontalRadiation = sinAltitude * exRadiation * (0.271 - 0.2939 * ps);
      break;
    case SeparationMethod.Matsuo:
      diffuseHorizontalRadiation =
    shi * (1 - aTransmissivity) * 1.2 / (1 - 1.4 * Math.Log(aTransmissivity));
      break;
    case SeparationMethod.Nagata:
      diffuseHorizontalRadiation = shi * (0.66 - 0.32 * sinAltitude) *
        (0.5 + (0.4 - 0.3 * aTransmissivity) * sinAltitude);
      break;
    case SeparationMethod.Watanabe:
      double qq = (0.9013 + 1.123 * sinAltitude) * Math.Pow
        (aTransmissivity, 0.489 / sinAltitude) * Math.Pow(1 - ps, 2.525);
      diffuseHorizontalRadiation = sinAltitude * exRadiation * qq / (1 + qq);
      break;
  }
}
```

6.3.4　夜間放射の計算

プログラム 6.8 に夜間放射の計算処理を示す。式 6.35 の実装である。雲量は 0~10 の 11 段階表示であり、プログラム 6.8 では式 6.35 の係数 0.62 を予め 10 で除している。夜間放射は太陽に依存する値ではないため、天空に関する情報を扱う Sky クラスのメソッドとする。

プログラム 6.8　夜間放射の計算処理

Popolo.Weather.Sky class

```
/// <summary>夜間放射[W/m2]を計算する</summary>
/// <param name="temperature">外気乾球温度[C]</param>
/// <param name="cloudCover">雲量[-]</param>
/// <param name="waterVaporPartialPressure">水蒸気分圧[kPa]</param>
/// <returns>夜間放射[W/m2]</returns>
public static double GetNocturnalRadiation
  (double temperature, int cloudCover, double waterVaporPartialPressure)
{
  double br = 0.526 + 0.209 * Math.Sqrt(waterVaporPartialPressure);
  return (1d - 0.062 * cloudCover) * (1d - br) * BLACK_CONSTANT * Math.Pow(temperature + 273.15, 4);
}
```

6.3.5　「太陽クラス」の作成

太陽の計算には、高度、方位、各種日射量、計算基準地点（経度・緯度）など、様々な情報を組み合わせる必要があるため、これらの情報をパッケージ化したクラスを定義するとプログラムが簡明になる。本節では太陽クラス（Sun class）の作成について説明する。

プログラム 6.9 にインスタンス変数およびプロパティの定義を示す。太陽位置（高度・方位角）は、インスタンスメソッドを通して値を変更するため、外部から操作できないように private アク

セッサとする。また、計算地点（緯度・経度・標準時経度）についても初期化後に変更ができないようにprivateアクセッサとする。

プログラム 6.9　Sun クラスのインスタンス変数・プロパティの定義

Popolo.Weather.Sun class

```
 1 /// <summary>法線面直達日射量[W/m2]</summary>
 2 private double directNormalRadiation;
 3
 4 /// <summary>水平面天空日射量[W/m2]</summary>
 5 private double diffuseHorizontalRadiation;
 6
 7 /// <summary>水平面全天日射量[W/m2]</summary>
 8 private double globalHorizontalRadiation;
 9
10 /// <summary>太陽高度[radian]を取得する</summary>
11 public double Altitude { private set; get; }
12
13 /// <summary>太陽方位角[radian]を取得する</summary>
14 public double Orientation { private set; get; }
15
16 /// <summary>計算地点の緯度（北が正）[degree]を取得する</summary>
17 public double Latitude { private set; get; }
18
19 /// <summary>計算地点の経度（東が正）[degree]を取得する</summary>
20 public double Longitude { private set; get; }
21
22 /// <summary>標準時を規定する地点の経度（東が正）[degree]を取得する</summary>
23 public double StandardLongitude { private set; get; }
24
25 /// <summary>法線面直達日射量[W/m2]を設定・取得する</summary>
26 public double DirectNormalRadiation
27 {
28   get { return directNormalRadiation; }
29   set { directNormalRadiation = Math.Max(0, value); }
30 }
31
32 /// <summary>水平面天空（散乱）日射量[W/m2]を設定・取得する</summary>
33 public double DiffuseHorizontalRadiation
34 {
35   get { return diffuseHorizontalRadiation; }
36   set { diffuseHorizontalRadiation = Math.Max(0, value); }
37 }
38
39 /// <summary>水平面全天日射量[W/m2]を設定・取得する</summary>
40 public double GlobalHorizontalRadiation
41 {
42   get { return globalHorizontalRadiation; }
43   set { globalHorizontalRadiation = Math.Max(0, value); }
44 }
45
46 /// <summary>現在の日時を取得する</summary>
47 public DateTime CurrentDateTime { private set; get; }
```

プログラム 6.10 にコンストラクタおよびインスタンスメソッドを示す。コンストラクタでは計算を行う地点の位置情報を保存する。12~21 行に定義する Update メソッドにより、特定の時刻の太陽位置を計算し、太陽高度と太陽方位角の値を更新する。23~43 行はプログラム 6.7 で定義した関数とほぼ同様の機能の関数であるが、適宜インスタンス変数を参照して計算を行うため、static メソッドに比較して引数が省略されている。

プログラム 6.10　Sun クラスのコンストラクタとインスタンスメソッド

Popolo.Weather.Sun class

```
 1 /// <summary>コンストラクタ</summary>
 2 /// <param name="latitude">計算地点の緯度（北が正）[degree]</param>
 3 /// <param name="longitude">計算地点の経度（東が正）[degree]</param>
 4 /// <param name="standardLongitude">標準時を規定する地点の経度[degree]</param>
 5 public Sun(double latitude, double longitude, double standardLongitude)
 6 {
 7   Latitude = latitude;
 8   Longitude = longitude;
 9   StandardLongitude = standardLongitude;
10 }
11
```

```
/// <summary>太陽位置等を更新する</summary>
/// <param name="dateTime">日時</param>
public void Update(DateTime dateTime)
{
  CurrentDateTime = dateTime;
  double al, or;
  GetSunPosition(Latitude, Longitude, StandardLongitude, dateTime, out al, out or);
  Altitude = al;
  Orientation = or;
}

/// <summary>大気圏外日射量[W/m2]を計算する</summary>
/// <returns>大気圏外日射量[W/m2]</returns>
public double GetExtraterrestrialRadiation()
{ return GetExtraterrestrialRadiation(CurrentDateTime.DayOfYear); }

/// <summary>法線面直達日射[W/m2]を計算する</summary>
/// <param name="atmosphericTransmissivity">大気透過率[-]</param>
/// <returns>法線面直達日射[W/m2]</returns>
public double GetDirectNormalRadiation(double atmosphericTransmissivity)
{ return GetDirectNormalRadiation(Altitude, atmosphericTransmissivity, CurrentDateTime.DayOfYear); }

/// <summary>直散分離を行う</summary>
/// <param name="method">直散分離の手法</param>
/// <param name="globalHorizontalRadiation">水平面全天日射[W/m2]</param>
public void SeparateGlobalHorizontalRadiation(double globalHorizontalRadiation, SeparationMethod method)
{
  GlobalHorizontalRadiation = globalHorizontalRadiation;
  SeparateGlobalHorizontalRadiation(globalHorizontalRadiation,
    Latitude, Longitude, StandardLongitude, CurrentDateTime, method,
    out directNormalRadiation, out diffuseHorizontalRadiation);
}
```

【例題 6.4】

夏至（6 月 22 日）および冬至（12 月 22 日）において、四方位鉛直面および水平面に入射する直達日射 I_D を計算せよ。ただし、計算地点は東京（北緯 35.7°、東経 139.8°）、大気拡散率 P は 0.65 とする。

【解】

大気透過率 P が与えられているため、式 6.27 および式 6.30 により法線面直達日射量 I_{DN} が計算可能である。さらに式 6.16 を用いて各時刻の $\cos\theta$ を計算し、式 6.15 で傾斜面に入射する直達日射 I_D を計算する。

プログラム 6.11 に計算処理を示す。8~13 行で四方位鉛直面および水平面を定義する。16 行で計算地点を東京として Sun クラスのインスタンスを生成する。30 分毎に出力値を得ることとし、26 行で時刻を更新する。27 行で大気透過率から法線面直達日射を計算し、30 行でそれぞれの傾斜面の $\cos\theta$ と乗じて直達日射を得る。計算結果をグラフ化すると図 6.4 が得られる。

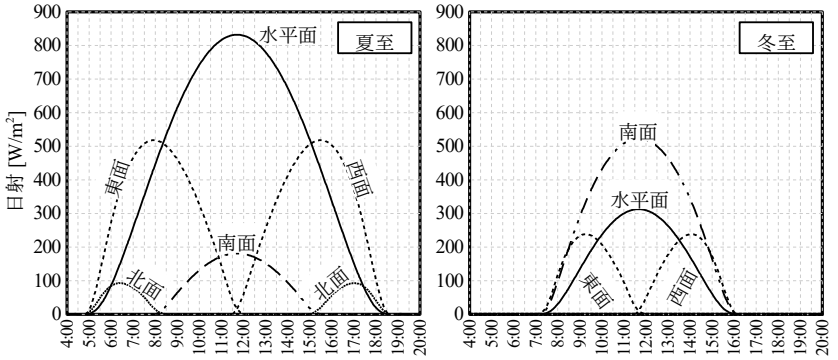


図 6.4　夏至および冬至に傾斜面が受ける日射 [W/m²]

プログラム **6.11**　夏至および冬至の鉛直面・水平面入射日射の計算

```
1 private static void InclineTest(bool isSummer)
2 {
3   DateTime dTime;
4   if (isSummer) dTime = new DateTime(2014, 6, 22, 4, 0, 0);
5   else dTime = new DateTime(2014, 12, 22, 4, 0, 0);
6
7   //傾斜面を作成
8   Incline[] inc = new Incline[5];
9   inc[0] = new Incline(Incline.Orientation.W, 0.5 * Math.PI);
10  inc[1] = new Incline(Incline.Orientation.E, 0.5 * Math.PI);
11  inc[2] = new Incline(Incline.Orientation.N, 0.5 * Math.PI);
```

```
12   inc[3] = new Incline(Incline.Orientation.S, 0.5 * Math.PI);
13   inc[4] = new Incline(Incline.Orientation.S, 0);
14
15   //東京地点で初期化
16   Sun sun = new Sun(35.7, 139.8, 135);
17
18   using (StreamWriter sWriter =
19   new StreamWriter("Incline.csv", false, Encoding.GetEncoding("Shift_JIS")))
20   {
21     //タイトル用
22     sWriter.WriteLine("時刻, 東面, 西面, 北面, 南面, 水平面");
23
24     for (int i = 0; i <= 32; i++)
25     {
26       sun.Update(dTime.AddMinutes(30 * i));
27       double idn = sun.GetDirectNormalRadiation(0.65);
28       sWriter.Write(sun.CurrentDateTime.ToShortTimeString());
29       for (int j = 0; j < inc.Length; j++)
30         sWriter.Write(", " + inc[j].GetDirectSolarRadiationRate(sun) * idn);
31       sWriter.WriteLine();
32     }
33   }
34 }
```

【例題 6.5】

晴天日と曇天日について直散分離を行い、手法間で結果を比較せよ。観測された水平面全天日射を表 6.1 に示す。地点は東京（北緯 35.7°、東経 139.8°）であり、晴天日は 2014 年 8 月 6 日、曇天日は 2014 年 7 月 20 日に観測されたデータである。

表 6.1　水平面全天日射の時系列データ [W/m^2]

時刻	5:00	6:00	7:00	8:00	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00
晴天	0	53	244	444	631	778	881	928	917	844	719	550	358	161	17	0
曇天	0	28	81	275	339	569	594	486	575	536	378	356	142	8	3	0

【解】

プログラム 6.12 に手法別の直散分離計算を示す。3~18 行は入力データの初期化処理である。列挙型で定義した直散分離のリストを 26 行で取得する。観測データは過去 1 時間に計測された平均値であるため、代表的な太陽位置は計測時点の 30 分前とする。このため時刻は 4:30 で初期化した後、36 行で 1 時間ずつ進める。40 行で直散分離を行うメソッドを呼び出し、41 行で外部ファイルに書き出す。計算結果をグラフ化すると、晴天日においては各手法とほぼ同様の傾向を示す一方で、曇天日においては Berlage、松尾、永田、Liu らの手法の水平面天空日射がやや小さめに計上されることが確認できるだろう。

プログラム 6.12　手法別の直散分離計算

```
 1 private static void RadiationSeparateTest(bool isCloudy)
 2 {
 3   double[] ghRadiation;
 4   DateTime sTime;
 5   if (isCloudy)
 6   {
 7     //東京都 2014/7/20 5-20 時の水平面全天日射[W/m2] (曇天日)
 8     ghRadiation = new double[] { 0, 28, 81, 275, 339, 569, 594, 486, 575, 536, 378, 356, 142, 8, 3, 0 };
 9     sTime = new DateTime(2014, 7, 20, 4, 30, 0);
10   }
11   else
12   {
13     //東京都 2014/8/6 5-20 時の水平面全天日射[W/m2] (晴天日)
14     ghRadiation = new double[] { 0, 53, 244, 444, 631, 778, 881, 928, 917, 844, 719, 550, 358, 161, 17, 0 };
15     sTime = new DateTime(2014, 8, 6, 4, 30, 0);
16   }
17
18   //東京地点で初期化
19   Sun sun = new Sun(35.7, 139.8, 135);
20   using (StreamWriter sWriter =
21   new StreamWriter("sun.csv", false, Encoding.GetEncoding("Shift_JIS")))
22   {
23     //直散分離手法一覧を取得
24     Array mtds = Enum.GetValues(typeof(Sun.SeparationMethod));
25
26     //タイトル行
27     sWriter.Write("時刻, 全天日射[W/m2]");
28     foreach (Sun.SeparationMethod mtd in mtds) sWriter.Write(", " + mtd.ToString() + ", ");
29     sWriter.WriteLine();
30     //直散分離計算実行
31     for (int i = 0; i < ghRadiation.Length; i++)
32     {
```

```
33        sun.Update(sTime.AddHours(i));
34        sWriter.Write(sun.CurrentDateTime.Hour + ", " + ghRadiation[i]);
35        foreach (Sun.SeparationMethod mtd in mtds)
36        {
37          sun.SeparateGlobalHorizontalRadiation(ghRadiation[i], mtd);
38          sWriter.Write(", " + sun.DirectNormalRadiation + ", " + sun.DiffuseHorizontalRadiation);
39        }
40        sWriter.WriteLine();
41      }
42    }
43 }
```

【第 6 章 記号表】

B_r	：射出率 **[-]**	R_{atm}	：大気放射 **[W/m²]**
C_R	：雲量 **[-]**	R_{eff}	：実効放射 **[W/m²]**
e	：均時差 **[hour]**	t	：時角 **[radian]**
F_S	：傾斜面の天空への形態係数 **[-]**	T_{atm}	：大気温度 **[K]**
F_G	：傾斜面の地表面への形態係数 **[-]**	W_α	：傾斜面の方位角 **[radian]**
h	：太陽高度 **[radian]**	W_β	：傾斜面の傾斜角度 **[radian]**
I_0	：大気圏外日射 **[W/m²]**	α	：太陽方位角 **[radian]**
I_d	：水平面拡散日射 **[W/m²]**	ε_{atm}	：放射率低減係数 **[-]**
I_D	：傾斜面へ入射する直達日射 **[W/m²]**	φ	：緯度 **[radian]**
I_{DN}	：法線面直達日射 **[W/m²]**	ϕ	：見かけの太陽高度 **[radian]**
I_{HOL}	：水平面全天日射 **[W/m²]**	δ	：太陽赤緯 **[radian]**
I_{SC}	：太陽定数 **(1,367 W/m²)**	σ	：黒体の放射定数 **[W/(m²·K⁴)]**
I_{SKY}	：水平面天空日射 **[W/m²]**	τ_m	：平均太陽時 **[hour]**
I_{SR}	：傾斜面へ入射する拡散日射 **[W/m²]**	τ_{st}	：標準時 **[hour]**
L	：経度 **[°]**	τ_t	：真太陽時 **[hour]**
L_s	：標準時子午線の経度 **[°]**	θ	：太陽光と傾斜面の法線が成す角度 **[radian]**
P_W	：水蒸気分圧 **[kPa]**		
subscripts			
sr	：日の出	***ss***	：日没

【第 6 章 参考文献】

6.1） 環境工学教科書, 環境工学教科書研究会編著, 彰国社, pp.78-81, 1996

6.2） ASHRAE Transaction Fundamentals, pp.29.16-29.51

6.3） 宇田川光弘: パソコンによる空気調和計算法, pp.45-68, オーム社, 1986

6.4） 松尾陽, 横山浩一, 石野久彌, 川元昭吾: 空調設備の動的熱負荷計算入門, 日本建築設備士協会, pp.24-26, 1980

6.5） Duffie, J.A., Beckman, W.A. : Solar Engineering of Thermal Processes, Wiley-Interscience, 1980 pp.63~121

6.6） 宇田川光弘, 木村建一: 水平面全天日射量観測値よりの直達日射量の推定, 日本建築学会論文報告集, 267(1978), pp.83~90

6.7） Berlage,Von H.P.:Zur Theorie der Beleuchtung einer horizontalen Flache durch Tageslicht,Meteorologische Zeitschrift, May 1928,pp.174-180

6.8） 松尾陽:日本建築学会論文報告集,快晴時の日射について 日射量に関する研究 2,pp.21-24,1960

6.9） 永田忠彦:晴天空による水平面散乱の日射の式の試案,日本建築学会学術講演梗概集,1978

6.10） Liu,B.Y.H & Jordan,R.C:The interrelationship and characteristic distribution of direct, diffuse and total solar radiation, solar energy, Vol.4, No.3, 1960

6.11） 渡辺俊行:水平面全天日射量の直散分離と傾斜面日射量の推定,日本建築学会論文報告集,No.330,pp.96-108,19830830

6.12） H.Akasaka:Model of circumsolar radiation and diffuse sky radiation including cloudy sky, ISES, Solar World Congress, 1991

6.13） 三木信博:標準気象データの日射直散分離に関する研究 その 6 日射直散分離法の提案,日本建築学会学術講演梗概集,pp.857-858,1991

6.14） 伊藤大輔, 足永靖信: 都市形状の分光アルベドに関するスケールモデル実験と数値計算, 日本建築学会環境系論文集, 第 74 巻, 第 641 号, pp.863-868, 2009

6.15） 木村建一, 建築設備基礎, 第 22 章 日射

6.16） 木村建一, 建築設備基礎, 第 24 章 放射+対流の熱伝達

6.17） D.G. Erbs, S.A. Klein and J.A. Duffie: Estimation of the Diffuse Radiation Fraction for Hourly, Daily and Monthly-Average Global Radiation, Solar Energy, Vol.28, No.4, pp.293-302, 1982

6.18） Philipps, H.: Zur Theorie der Warmestrahlung in Bodennahe, Grerl. Beitr. Z. Geophys. 56, p.229, 1940

第7章　気象データ
(Weather Data)

7.1　概要

シミュレーションを行う際には、まず、検討対象の範囲を定める必要がある。検討対象範囲内の各種の情報はシミュレーションを行う過程で明らかにしていくが、検討対象範囲外の情報は外部から固定値として与える。このような外部から一方的に与えられる固定的な条件を境界条件と呼ぶ。

建物 1 棟のシミュレーションを実行する場合には、気象条件を境界条件とすることが多い。気象データと呼ばれるものとしては、気温、湿度、日射、風速、雲量、雨量、積雪量などが挙げられるが、特に熱負荷・設備シミュレーションにおいて影響度が大きい要素は、気温、湿度、日射である。

気象条件を与える方法は大きく 2 つあり、1 つは予め用意しておいた確定的なデータを用いる方法であり、もう 1 つは計算の都度、確率的にデータを生成する方法である。多くの熱負荷計算プログラムでは標準年気象データ（HASP, 拡張 AMEDAS など）を与えて計算を行うが、これは確定的なデータを用いる方法である。確定的な方法は計算結果の解釈が容易であるという利点がある一方で、無償で公開されたデータが少ない、モンテカルロ法などで確率的な検討をすることには向いていない、という弱点もある。また、規範性の高い確定的データは有償・高額であることが通常であり、学生らが自らが作成したシミュレーションプログラムを簡易に試験するという場面では敷居が高い。そこで本章では、まず代表的な確定的データについて概説し、確率的な気象データの生成方法を解説する。

写真 7.1　明治 16 年 3 月 1 日の日本最古の天気図

（気象庁 気象科学館にて撮影）

7.2　理論

7.2.1　確定的気象データ

1）拡張アメダス気象データ

拡張アメダス気象データは日本において最もよく知られた確定的気象データであり、全国の気象官署とアメダス（AMeDAS :Automated Meteorological Data Acquisition System）における計測値にもとづいて開発された。気象官署のデータは信頼性が高いが観測地点数が少ない。一方でアメダスデータは全国で 840 を超える観測地点があるが、無人観測であるために気象官署ほどの信頼性は無い。拡張アメダス気象データでは両者の短所を補い合い、アメダスで計測されていないデータ（日射、水蒸気圧、大気放射量）の補充、欠測データの補間を慎重に行うことで気象官署と遜色のない精度で全国 842 地点に対応している。具体的には 1981~2000 年の気温、絶対湿度、水平面全天日射量、大気放射量、風向、風速、降水量、日照時間の時刻別データ（実在年気象データ）が提供されている。また、実在年気象データにもとづいて設計用気象データおよび標準年気象データも提供されている。

設計にあたっては特定の年の気象データに対応するだけでは不十分であり、特別に外気条件が厳しい日を定義し、その外気条件に対応可能な容量を計画する必要がある。このような目的のもとに作成された外気条件を設計用気象データと呼ぶ。一方で、年間のシミュレーションを行う場合には、その地点を代表する標準的な気象データを入力とすることが多い。このような目的のもとに作成された外気条件を標準年気象データと呼ぶ。

2）BEST 用気象データ

総合エネルギーシミュレーションツール BEST で用いることを想定した気象データであり、1 分間隔の計算に対応していることが特徴である。また、拡張アメダスデータで提供されている要素に加えて「照度」と「降水量」が用意されている。

7.2.2　確率的気象データ

1）時系列解析

時間とともに変動するデータを時系列データと呼び、気象データは典型的な時系列データである。時系列解析は、時系列データの特徴の記述、モデル化、予測などを行うことを言う。

気象データは、短期間には不規則に変動（不規則変動）しているように見えるが、長期のデータを俯瞰すると、地球の公転と自転に起因する 1 日周期、1 年周期の変動（循環変動）を読み取ることができる。また、さらに長期的には一定の傾向を持つ変動（トレンド）も確認できる。時系列データを読み解きモデル化するためには、まず、これらの変動（不規則変動・循環変動・トレンド）を分解する必要がある[†1)]。時系列データを時点 t の関数として Y_t とおくと、循環変動 C_t、トレンド T_t、不規則変動 I_t を用いて式 7.1 ないしは式 7.2 で表現することができる。式 7.1 を加法モデル、式 7.2 を乗法モデルと呼び、本書では加法モデルを採用する。なお、気象データからパラメータを推定する作業は非常に複雑であるため参考文献[7.1)]に譲る。

$$y_t = C_t + T_t + I_t \tag{7.1}$$

$$y_t = C_t \times T_t \times I_t \tag{7.2}$$

†1　時系列分析の分野では周期が確定している 1 年周期の変動を特に季節変動と呼び、その他の循環変動と切り分けることが多い。即ち、時系列データを季節（Seasonal）変動・不規則（Irregular）変動・トレンド（Trend）・循環（Cyclical）変動の 4 成分に分解する。

2）気象要素

モデル化を行う気象要素は、乾球温度、絶対湿度、水平面全天日射量の 3 要素とする。ただし、水平面全天日射量 I_{HOL} [W/m^2]は大気透過率 P [-]を求めた後に式 7.3~7.6を用いて変換を行う[7.2) 7.3)]。

$$I_d = I_0 \sin h \frac{Q}{1+Q} \tag{7.3}$$

$$Q = (0.8672 + 0.7505 \sin h) P^{0.421/\sin h} (1 - P^{1/\sin h})^{2.277} \tag{7.4}$$

$$I_{DN} = I_0 P^{1/\sin h} \tag{7.5}$$

$$I_{HOL} = I_{DN} \sin h + I_d \tag{7.6}$$

3）トレンドのモデル化

各年で独立に平均 0、分散 σ_T^2 の正規分布に従うとする。

4）周期変動のモデル化

周期変動は地球の公転に影響を受ける年周期成分 CA_t と自転に影響を受ける日周期成分 CC_t に分離できる。周期的変動はフーリエ級数を用いて式 7.7 で計算する。ここで A_i および B_i は係数、T は周期であり、日周期の場合には T=24、年周期の場合には T=365 となる。次数 N は 3 ないしは 4 程度で十分な近似が可能なことが確かめられている。なお、大気透過率の日周期成分に関しては曇天（雲量=10）か晴天（雲量<10）かによって 2 種類のモデルを用意する。雲量の状態は晴天と曇天の 2 つの状態を持つマルコフ過程として式 7.8 で計算する。P_{FF} は晴天から晴天への推移確率、P_{CC} は曇天から曇天への推移確率であり、これらも年周期変動を持つものとしてモデル化する。また、$t \to \infty$における不変分布 $P_{C,\infty}$は $P_{F,t+1} = P_{F,t}$の条件により、式 7.9 で計算できる。なお、本モデルのパラメータを推定するために用いた気象庁のデータは 3 時間毎に観測された雲量であるため、Δh=3 である。

以上により、周期変動としては気象 3 要素の年周期成分、乾球温度と絶対湿度の日周期成分、大気透過率の日周期成分（曇天+晴天）、雲量状態推移確率の年周期成分（曇天+晴天）で、合計 9 つのモデルを用意する。

$$C_n = \frac{A_0}{2} + \sum_{i=1}^{N} \left[A_i \cos\left(\frac{2\pi i n}{T}\right) + B_i \sin\left(\frac{2\pi i n}{T}\right) \right] \tag{7.7}$$

$$\begin{bmatrix} P_{F,t+\Delta h} \\ P_{C,t+\Delta h} \end{bmatrix} = \begin{bmatrix} P_{FF,t} & 1-P_{CC,t} \\ 1-P_{FF,t} & P_{CC,t} \end{bmatrix} = \begin{bmatrix} P_{F,t} \\ P_{C,t} \end{bmatrix} \tag{7.8}$$

$$P_{C,\infty} = (1 - P_{FF,t}) / (2 - P_{CC,t} - P_{FF,t}) \tag{7.9}$$

5）不規則変動のモデル化

大気透過率に関しては式 7.10 に示す AR モデルで計算する。ラグ次数 p=1 とする。α は係数、$N(0,\sigma^2)$に従う正規乱数である。は平均 0 分散乾球温度と絶対湿度に関しては不規則変動の相関を表現するために式 7.11 と 7.12 に示す多変量 AR モデルで計算する。α と β は係数であり、ラグ次数 p=3 とする。

$$I_{ATn} = \sum_{i=1}^{p} \alpha_{ATi} I_{ATn-i} + N(0, \sigma_{AT}) \tag{7.10}$$

$$I_{DTn} = SD_{DTn} \left\{ \sum_{i=1}^{p} (\alpha_{DTi} I_{DTSn-i} + b_{DTi} I_{HRSn-i}) + N(0, \sigma_{DT}) \right\} \tag{7.11}$$

$$I_{HRn}=SD_{HRn}\left\{\sum_{i=1}^{p}\left(\alpha_{HRi}I_{DTSn-i}+b_{HRi}I_{HRSn-i}\right)+N\left(0,\sigma_{HR}\right)\right\} \tag{7.12}$$

大気透過率と乾球温度ならびに大気透過率と絶対湿度の不規則変動成分の相関を表現するため、雲量状態を加味した大気透過率の不規則変動成分の大きさ I_{ATcr} を式 7.13 で定義し、これを用いて乾球温度と絶対湿度の年周期成分および日周期成分を補正する[†1]。式 7.13 の第一項は式 7.10 で計算した *AR* モデルに基づく不規則変動成分である。第二項は雲量の状態に応じた大気透過率日周期成分の実現値である。第三項の CC_{ATF} と CC_{ATC} はそれぞれ晴天と曇天の日周期成分であり、これらを各時点の晴天率と曇天率（不変分布）で重み付け平均することで期待値としての大気透過率日周期成分を求め、第二項から減じることで雲量状態を加味した不規則変動成分を計算する。

$$I_{ATcr}=I_{AT}+CC_{AT}-\left(P_{F,\infty}CC_{ATF}+P_{C,\infty}CC_{ATC}\right) \tag{7.13}$$

式 7.14~7.16を用いて乾球温度の年周期成分と日周期成分、絶対湿度の年周期成分を補正する。絶対湿度の日周期成分に関しては大気透過率不規則変動成分との相関が小さいため、補正は行わない。

$$CA_{DTcr}=CA_{DT}\times\left(cf_{shDT}\cdot I_{ATcr}\right) \tag{7.14}$$

$$CA_{HRcr}=CA_{HR}\times\left(cf_{shHR}\cdot I_{ATcr}\right) \tag{7.15}$$

$$CC_{DTcr}=CC_{DT}\times\left(cf_{swADT}\cdot I_{ATcr}+cf_{swBDT}\right) \tag{7.16}$$

6）気象時系列の再構成

以上の関係式を用いて気象時系列を再構成する手順を図 7.1 に示す。

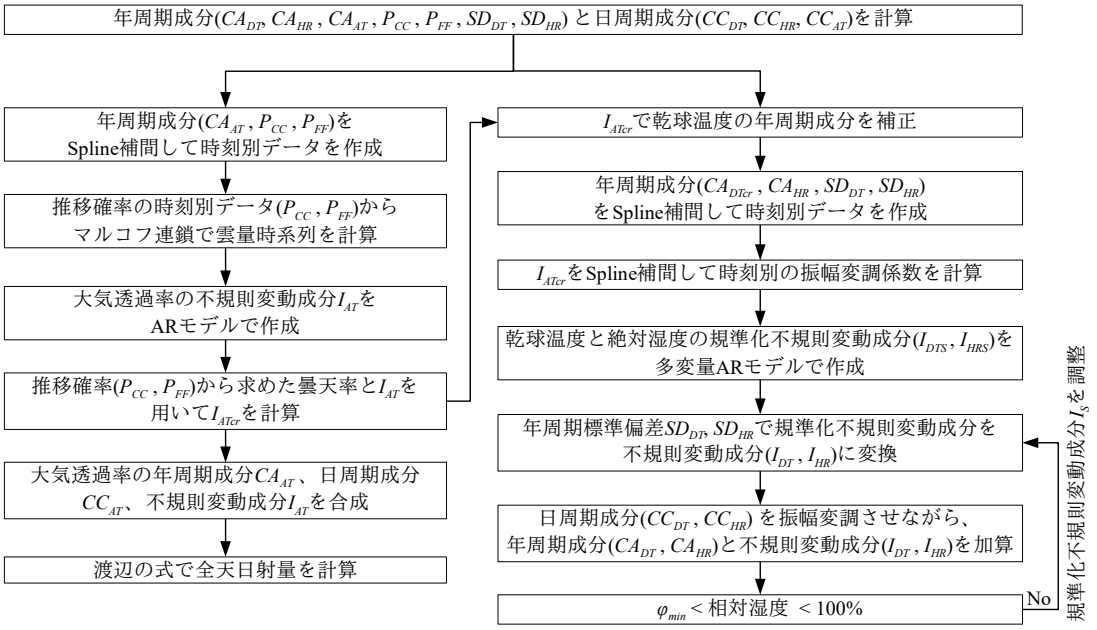


図 7.1　気象時系列の再構成フロー

まず確定的成分として年周期成分と日周期成分を計算する。日周期成分および不規則変動成分への合成のため、年周期成分は第 2 章で解説した Spline 補間をかけて、時刻別データに変換する。推移確率 P_{CC} と P_{FF} を用いて時刻別の雲量を計算する。ただし、時点 0 の状態は式 7.9 で示される不変分布にもとづいて確率的に発生させる。大気透過率の不規則変動成分は単純な AR モデルで作成する。ただし 50 時間の助走期間を取る。以上により、雲量状態と不規則変動成分が得られるため、乾球温度と

†1　このような迂遠な方法を取らず、最初から 3 要素の多変量 *AR* モデルを作成すれば良いように思われるかもしれない。しかし大気透過率は夜間に欠測となるため、実際には直接的に多変量 *AR* モデルを適用することが難しい。

絶対湿度の計算に備えて式 7.13 を用いて I_{ATcr} を求めておく。年周期成分、日周期成分、不規則変動成分を合算して大気透過率を計算し、式 7.3~7.6を用いて全天日射量に変換する。ただし、現実には日中に大気透過率が完全に 0 となることは無いため、最小値を 0.001 とする。

乾球温度と絶対湿度に関しては、大気透過率の計算の際に算出した I_{ATcr} を用いて年周期成分を式 7.16 で補正する。さらに補正した年周期成分に Spline 補間を行い、時刻別データを作成する。また、振幅変調の際には時刻別に式 7.14 と式 7.15 を適用する必要があるため、I_{ATcr} も時刻別データに変換する。多変量 AR モデルを適用し、不規則変動成分を計算する。ただし 50 時間の助走期間を取る。以上により求めた年周期成分と不規則変動成分に、振幅変調をした日周期成分を加えることで乾球温度と絶対湿度を計算する。ただし、相対湿度が 100%を超えないように不規則変動成分を調整する。また、実績値にもとづき設けた各地域の相対湿度下限値 φ_{min} を下回らないようにする。

7）モデルパラメータ

日本の代表 6 都市（東京・大阪・札幌・仙台・福岡・那覇）のパラメータを表 7.1~7.6に示す。

表 7.1　東京のパラメータ

周期成分フーリエ係数									
	A_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
CA_DT	16.1553	-9.1951	-0.5818	0.6386	-0.4619	4.5427	-0.3686	0.5093	-0.2657
CA_HR	8.4347	-5.8197	0.2694	0.3759	-0.1934	2.9745	-1.2801	0.4618	-0.0777
CA_AT	0.3797	0.1610	0.0332	0.0231	-0.0285	-0.0061	0.0052	0.0085	-0.0225
CA_FF	0.8132	0.0770	0.0077	0.0152	-0.0207	0.0002	-0.0101	0.0192	-0.0217
CA_CC	0.7975	-0.0427	-0.0228	-0.0255	0.0199	-0.0055	0.0156	-0.0112	0.0037
CA_DTSD	1.2483	0.0497	-0.0513	-	-	-0.1111	0.0172	-	-
CA_HRSD	1.3130	-0.4424	-0.1839	-	-	0.2039	0.0424	-	-
CC_DT	0.0000	-1.5906	0.5391	-0.0523	-	1.7772	-0.2794	-0.0602	-
CC_HR	0.0000	0.1410	-0.0029	-0.0107	-	0.1681	0.0217	-0.0091	-
CC_ATC	0.0450	0.3672	0.0000	-0.0390	-	0.0230	0.0000	-0.0037	-
CC_ATF	0.3807	0.2269	0.0000	-0.0008	-	0.0062	0.0000	-0.0019	-
AR モデル係数									
	a_1	a_2	a_3	b_1	b_2	b_3	σ	-	-
AT	0.8306	-	-	-	-	-	0.0289	-	-
DT	1.1366	-0.0915	-0.1000	0.0836	0.0017	-0.0253	0.4572	-	-
HR	0.0212	0.0186	-0.0262	1.0330	-0.0618	-0.0197	0.3600	-	-
その他									
cf_{shDT}	cf_{shHR}	cf_{swA}	cf_{swB}	φ_{min}	$\sigma_{T\ DT}$	$\sigma_{T\ HR}$	$\sigma_{T\ AT}$	-	-
2.584	-2.806	1.118	1.249	10	0.5686	0.3078	0.0241	-	-

表 7.2　大阪のパラメータ

周期成分フーリエ係数									
	A_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
CA_DT	16.8268	-9.3956	-0.3928	0.3107	-0.2057	4.4420	-0.5665	0.5255	-0.3072
CA_HR	9.4201	-6.0004	0.5700	0.1399	-0.0428	2.8796	-1.3209	0.5317	-0.0812
CA_AT	0.3703	0.0655	-0.0209	0.0046	-0.0212	0.0317	0.0183	0.0068	-0.0205
CA_FF	0.8090	0.0109	-0.0331	0.0088	-0.0146	0.0223	0.0094	0.0152	-0.0122
CA_CC	0.7729	-0.0411	-0.0171	-0.0087	0.0242	-0.0324	0.0147	-0.0103	0.0210
CA_DTSD	1.3801	0.1494	-0.0147	-	-	-0.1642	0.0041	-	-
CA_HRSD	1.2499	-0.3970	-0.1582	-	-	0.1771	0.1389	-	-
CC_DT	0.0000	-1.7793	0.6353	-0.0995	-	1.6188	-0.2374	-0.1190	-
CC_HR	0.0000	-0.1133	0.0500	0.0032	-	0.2190	-0.0122	0.0042	-
CC_ATC	0.0040	0.3535	0.0000	-0.0401	-	-0.0277	0.0000	0.0049	-
CC_ATF	0.3579	0.1904	0.0000	0.0053	-	-0.0113	0.0000	0.0056	-
AR モデル係数									
	a_1	a_2	a_3	b_1	b_2	b_3	σ	-	-
AT	0.7394	-	-	-	-	-	0.0906	-	-
DT	1.1584	-0.0837	-0.1388	0.0931	-0.0088	-0.0284	0.4360	-	-
HR	-0.0086	0.0413	-0.0146	1.0330	1.0589	-0.0643	-0.0403	-	-
その他									
cf_{shDT}	cf_{shHR}	cf_{swA}	cf_{swB}	φ_{min}	$\sigma_{T\ DT}$	$\sigma_{T\ HR}$	$\sigma_{T\ AT}$	-	-
0	-4.173	1.486	1.337	10	0.5154	0.2984	0.0323	-	-

表 7.3　札幌のパラメータ

周期成分フーリエ係数									
	A_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
CA_DT	8.8674	-11.5445	-0.7236	0.5122	-0.2416	5.3594	-0.4775	0.5302	-0.2817
CA_HR	5.8837	-4.3882	0.5599	0.2440	-0.2687	2.3810	-1.3414	0.4957	-0.1201
CA_AT	0.3778	0.0967	-0.0072	-0.0008	0.0109	0.0038	0.0028	-0.0116	0.0003
CA_FF	0.7343	-0.0526	-0.0393	0.0037	0.0138	0.0256	0.0114	0.0118	-0.0041
CA_CC	0.7694	-0.0497	0.0083	0.0141	0.0022	-0.0067	-0.0044	0.0022	0.0001
CA_DTSD	1.5794	0.4146	0.0894	-	-	-0.0883	0.1445	-	-
CA_HRSD	0.9420	-0.3845	-0.0669	-	-	0.3613	-0.0884	-	-
CC_DT	0.0000	-2.0102	0.6600	-0.0203	-	1.2000	-0.0492	-0.1270	-
CC_HR	0.0000	-0.1166	-0.0264	0.0067	-	0.1300	0.0315	-0.0058	-
CC_ATC	0.0772	0.4520	0.0662	-0.0249	-	0.0431	0.0163	-0.0144	-
CC_ATF	0.3787	0.2029	0.0115	0.0138	-	0.0036	0.0034	-0.0043	-
AR モデル係数									
	a_1	a_2	a_3	b_1	b_2	b_3	σ	-	-
AT	0.7219	-	-	-	-	-	0.1025	-	-
DT	1.1456	-0.1402	-0.0701	0.1611	-0.0552	-0.0481	0.5696	-	-
HR	0.0625	-0.0017	-0.0330	0.9749	-0.0038	-0.0330	0.4885	-	-
その他									
cf_{shDT}	cf_{shHR}	cf_{swA}	cf_{swB}	φ_{min}	σ_{T_DT}	σ_{T_HR}	σ_{T_AT}	-	-
0.723	-1.613	1.346	1.343	10	0.5686	0.3078	0.0241	-	-

表 7.4　仙台のパラメータ

周期成分フーリエ係数									
	A_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
CA_DT	12.3043	-9.6166	-0.6660	0.6683	-0.4191	5.0537	-0.4351	0.4763	-0.1837
CA_HR	7.4230	-5.1868	0.4997	0.4007	-0.3268	2.9537	-1.4521	0.4731	-0.0609
CA_AT	0.3801	0.1609	-0.0026	0.0128	-0.0174	-0.0168	0.0156	0.0043	-0.0167
CA_FF	0.7986	0.0491	-0.0205	0.0078	-0.0144	-0.0169	0.0182	-0.0022	-0.0056
CA_CC	0.7758	-0.1076	-0.0254	-0.0058	0.0141	0.0066	0.0007	-0.0130	0.0040
CA_DTSD	1.4371	0.0498	0.0356	-	-	-0.1038	0.1049	-	-
CA_HRSD	1.1640	-0.5124	-0.1287	-	-	0.3059	-0.0090	-	-
CC_DT	0.0000	-1.9449	0.7034	-0.0493	-	1.3603	-0.1132	-0.1271	-
CC_HR	0.0000	-0.0205	0.0410	-0.0030	-	0.1717	0.0334	0.0134	-
CC_ATC	0.0180	0.3333	0.0000	-0.0400	-	0.0282	0.0000	-0.0099	-
CC_ATF	0.3852	0.2452	0.0000	0.0117	-	0.0005	0.0000	0.0002	-
AR モデル係数									
	a_1	a_2	a_3	b_1	b_2	b_3	σ	-	-
AT	0.7646	-	-	-	-	-	0.0964	-	-
DT	1.0833	-0.0766	-0.0789	0.1604	-0.0292	-0.0603	0.4915	-	-
HR	0.0353	0.0100	-0.0327	1.0105	-0.0091	-0.0448	0.3376	-	-
その他									
cf_{shDT}	cf_{shHR}	cf_{swA}	cf_{swB}	φ_{min}	σ_{T_DT}	σ_{T_HR}	σ_{T_AT}	-	-
1.794	-2.182	1.284	1.304	10	0.6328	0.3540	0.0200	-	-

表 7.5　福岡のパラメータ

周期成分フーリエ係数									
	A_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
CA_DT	16.8268	-9.3956	-0.3928	0.3107	-0.2057	4.4420	-0.5665	0.5255	-0.3072
CA_HR	9.4201	-6.0004	0.5700	0.1399	-0.0428	2.8796	-1.3209	0.5317	-0.0812
CA_AT	0.3703	0.0655	-0.0209	0.0046	-0.0212	0.0317	0.0183	0.0068	-0.0205
CA_FF	0.8090	0.0109	-0.0331	0.0088	-0.0146	0.0223	0.0094	0.0152	-0.0122
CA_CC	0.7929	-0.0411	-0.0171	-0.0087	0.0242	-0.0324	0.0147	-0.0103	0.0210
CA_DTSD	1.3338	0.1684	0.1002	-	-	-0.2047	0.0312	-	-
CA_HRSD	1.2744	-0.1366	-0.1652	-	-	0.1132	0.1945	-	-
CC_DT	0.0000	-1.7793	0.6353	-0.0995	-	1.6188	-0.2374	-0.1190	-
CC_HR	0.0000	-0.1133	0.0500	0.0032	-	0.2190	-0.0122	0.0042	-
CC_ATC	0.0040	0.3535	0.0000	-0.0401	-	-0.0277	0.0000	0.0049	-
CC_ATF	0.3579	0.1904	0.0000	0.0053	-	-0.0113	0.0000	0.0056	-
AR モデル係数									
	a_1	a_2	a_3	b_1	b_2	b_3	σ	-	-
AT	0.7344	-	-	-	-	-	0.0871	-	-
DT	1.1385	-0.1107	-0.0907	0.0996	-0.0029	-0.0418	0.4749	-	-
HR	0.0264	0.0160	-0.0145	0.9199	0.0400	-0.0172	0.3547	-	-
その他									
cf_{shDT}	cf_{shHR}	cf_{swA}	cf_{swB}	φ_{min}	σ_{T_DT}	σ_{T_HR}	σ_{T_AT}	-	-
-0.162	-3.323	1.324	1.290	10	0.4559	0.2169	0.0221	-	-

表 7.6　那覇のパラメータ

周期成分フーリエ係数									
	A_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
CA_DT	22.9623	-5.3214	-0.2766	-0.1536	0.1923	2.9532	-0.0588	0.1686	-0.0156
CA_HR	13.8874	-5.6110	0.0933	-0.0870	0.1813	2.1287	-0.3409	0.0453	0.1980
CA_AT	0.3407	0.0017	0.0121	-0.0082	0.0056	0.0724	0.0061	0.0043	-0.0122
CA_FF	0.7991	-0.0288	-0.0036	0.0057	0.0024	0.0727	-0.0097	0.0093	-0.0065
CA_CC	0.7882	0.0221	-0.0010	0.0026	-0.0179	-0.0490	0.0134	-0.0110	0.0071
CA_DTSD	0.8363	0.2361	0.0041	-	-	-0.1779	-0.0364	-	-
CA_HRSD	1.9009	0.3594	-0.4965	-	-	-0.3534	0.4481	-	-
CC_DT	0.0000	-1.1943	0.4402	-0.0188	-	0.7588	-0.1384	-0.0821	-
CC_HR	0.0000	-0.1003	0.0246	0.0117	-	-0.0056	0.0006	0.0013	-
CC_ATC	0.0626	0.3517	0.0000	-0.0314	-	-0.0491	0.0000	0.0113	-
CC_ATF	0.3428	0.2665	0.0000	0.0165	-	-0.0426	0.0000	-0.0061	-
AR モデル係数									
	a_1	a_2	a_3	b_1	b_2	b_3	σ	-	-
AT	0.7364	-	-	-	-	-	0.1009	-	-
DT	0.9343	-0.0137	-0.0049	0.0676	0.0114	-0.0145	0.6282	-	-
HR	-0.0082	0.0229	0.1372	1.0330	0.7498	0.1372	0.3404	-	-
その他									
cf_{shDT}	cf_{shHR}	cf_{swA}	cf_{swB}	φ_{min}	σ_{T_DT}	σ_{T_HR}	σ_{T_AT}	-	-
1.899	-3.168	0.582	0.811	30	0.4332	0.3588	0.0210	-	-

7.3　計算法

プログラム 7.1 に列挙型およびインスタンス変数宣言を示す。1~16 行で代表 6 都市の列挙型を定義する。18~42 行がインスタンス変数宣言である。

プログラム 7.1　列挙型およびインスタンス変数宣言

Popolo.Weather.RandomWeather class

```
 1 /// <summary>地点</summary>
 2 public enum Location
 3 {
 4   /// <summary>東京</summary>
 5   Tokyo,
 6   /// <summary>大阪</summary>
 7   Osaka,
 8   /// <summary>福岡</summary>
 9   Fukuoka,
10   /// <summary>札幌</summary>
11   Sapporo,
12   /// <summary>仙台</summary>
13   Sendai,
14   /// <summary>那覇</summary>
15   Naha
16 }
17
18 /// <summary>計算地点の太陽</summary>
19 private static Sun sun;
20
21 /// <summary>トレンド成分標準偏差</summary>
22 private static double sdevTDT, sdevTHR, sdevTAT;
23
24 /// <summary>AR モデル係数</summary>
25 private static double[] iDTCof, iHRCof;
26
27 /// <summary>AR モデル係数</summary>
28 private static double iATCof, iDTSD, iHRSD, iATSD;
29
30 /// <summary>振幅変調・シフト係数</summary>
31 private static double shiftDT, swingDTa, swingDTb, shiftHR;
32
33 /// <summary>年周期フーリエ係数</summary>
34 private static readonly double[] acaDT, bcaDT, acaHR, bcaHR, acaAT, bcaAT,
35   acFF, bcFF, acCC, bcCC, acDTSD, bcDTSD, acHRSD, bcHRSD;
36
37 /// <summary>日周期フーリエ係数</summary>
38 private static readonly double[] accDT, bccDT, accHR, bccHR, acCLD, bcCLD, acFAR, bcFAR;
39
40 /// <summary>相対湿度下限値</summary>
41 private static double minimumRelativeHumidity = 10;
```

式 7.3~7.6に従い、大気透過率[-]から水平面全天日射[W/m²]を推定する処理をプログラム 7.2 に示す。第 2 引数は第 6 章で定義した Sun クラスのインスタンスであり、太陽位置は設定済みとする。

プログラム 7.2　大気透過率による水平面全天日射の推定処理

Popolo.Weather.RandomWeather class

```
/// <summary>渡辺の手法で水平面全天日射[W/m2]を計算する</summary>
/// <param name="aTransmissivity">大気透過率[-]</param>
/// <param name="sun">計算地点の太陽</param>
private static double getGlobalHorizontalRadiation(double aTransmissivity, ImmutableSun sun)
{
  double sinAltitude = Math.Sin(sun.Altitude);
  double exRadiation = sun.GetExtraterrestrialRadiation();
  double ps = Math.Pow(aTransmissivity, 1 / sinAltitude);
  double directNormalRadiation = exRadiation * ps;
  double q = (0.8672 + 0.7505 * sinAltitude) * Math.Pow(aTransmissivity, 0.421 / sinAltitude)
    * Math.Pow(1d - Math.Pow(aTransmissivity, 1 / sinAltitude), 2.277);
  double diffuseHorizontalRadiation = exRadiation * sinAltitude * (q / (1 + q));
  return directNormalRadiation * sinAltitude + diffuseHorizontalRadiation;
}
```

周期的変動成分の計算処理をプログラム 7.3 に示す。1~37 行が年周期変動成分、39~66 行が日周期変動成分の計算処理である。いずれも式 7.7 に従い、各時点の状態値を計算する。

プログラム 7.3　周期的変動成分の計算

Popolo.Weather.RandomWeather class

```
/// <summary>年周期成分を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[C]</param>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="atmTransmissivity">大気透過率[-]</param>
/// <param name="fairToFair">晴れから晴れへの推移確率</param>
/// <param name="cloudToCloud">曇りから曇りへの推移確率</param>
/// <param name="dbtSigma">乾球温度不規則変動成分の標準偏差</param>
/// <param name="hrtSigma">絶対湿度不規則変動成分の標準偏差</param>
private static void makeAnnualData
  (out double[] drybulbTemperature, out double[] humidityRatio, out double[] atmTransmissivity,
   out double[] fairToFair, out double[] cloudToCloud, out double[] dbtSigma, out double[] hrtSigma)
{
  //日別年周期データを作成
  drybulbTemperature = new double[365];
  humidityRatio = new double[365];
  atmTransmissivity = new double[365];
  fairToFair = new double[365];
  cloudToCloud = new double[365];
  dbtSigma = new double[365];
  hrtSigma = new double[365];
  for (int i = 0; i < acaDT.Length; i++)
  {
    double wk = 2d * Math.PI * i / 365d;
    for (int j = 0; j < 365; j++)
    {
      double cwk = Math.Cos(wk * j);
      double swk = Math.Sin(wk * j);
      drybulbTemperature[j] += acaDT[i] * cwk - bcaDT[i] * swk;
      humidityRatio[j] += acaHR[i] * cwk - bcaHR[i] * swk;
      atmTransmissivity[j] += acaAT[i] * cwk - bcaAT[i] * swk;
      fairToFair[j] += acFF[i] * cwk - bcFF[i] * swk;
      cloudToCloud[j] += acCC[i] * cwk - bcCC[i] * swk;
      dbtSigma[j] += acDTSD[i] * cwk - bcDTSD[i] * swk;
      hrtSigma[j] += acHRSD[i] * cwk - bcHRSD[i] * swk;
    }
  }
}

/// <summary>日周期成分を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[C]</param>
/// <param name="humidityRatio">絶対湿度[kg/kg]</param>
/// <param name="atmTransFair">晴れの日の大気透過率[-]</param>
/// <param name="atmTransCloudy">曇の日の大気透過率[-]</param>
private static void makeCircadianData
  (out double[] drybulbTemperature, out double[] humidityRatio,
   out double[] atmTransFair, out double[] atmTransCloudy)
{
  //時刻別データを作成
  drybulbTemperature = new double[24];
  humidityRatio = new double[24];
  atmTransCloudy = new double[24];
```

```
52   atmTransFair = new double[24];
53   for (int i = 0; i < accDT.Length; i++)
54   {
55     double wk = 2d * Math.PI * i / 24d;
56     for (int j = 0; j < 24; j++)
57     {
58       double cwk = Math.Cos(wk * j);
59       double swk = Math.Sin(wk * j);
60       drybulbTemperature[j] += accDT[i] * cwk - bccDT[i] * swk;
61       humidityRatio[j] += accHR[i] * cwk - bccHR[i] * swk;
62       atmTransCloudy[j] += acCLD[i] * cwk - bcCLD[i] * swk;
63       atmTransFair[j] += acFAR[i] * cwk - bcFAR[i] * swk;
64     }
65   }
66 }
```

日別データを Spline 補間して時刻別データに変換するプログラムを 7.4 に示す。プログラム 7.3 で得られる年周期成分は日別のデータであるため、必要に応じてこのプログラムを実行することで時刻別データを得る。14 行で Spline 補間のための係数を取得し、15 行で補間を実行する。

プログラム 7.4　日別データから時刻別データへの変換処理

Popolo.Weather.RandomWeather class

```
 1 /// <summary>日別データを Spline 補間して時刻別データを作成する</summary>
 2 /// <param name="dailyData">日別データ</param>
 3 /// <returns>時刻別データ</returns>
 4 private static double[] interpolateDailyData(double[] dailyData)
 5 {
 6   double[] dd = new double[dailyData.Length + 2];
 7   dailyData.CopyTo(dd, 1);
 8   dd[0] = dd[dd.Length - 2];
 9   dd[dd.Length - 1] = dd[1];
10   double[] sx = new double[dailyData.Length + 2];
11   double[] hx = new double[dailyData.Length * 24];
12   for (int i = 0; i < sx.Length; i++) sx[i] = -12 + 24 * i;
13   for (int i = 0; i < hx.Length; i++) hx[i] = i;
14   double[] cDD = CubicSpline.GetParameters(sx, dd);
15   return CubicSpline.Interpolate(sx, dd, cDD, hx);
16 }
```

乾球温度と絶対湿度の不規則変動成分の計算処理をプログラム 7.5 に示す。第一引数は第 2 章で解説した正規乱数製造機である。第二引数と第三引数は現在時点を含む直近 3 時点の乾球温度と絶対湿度の不規則変動成分である。10~13 行と 14~20 行でそれぞれ式 7.11 と式 7.12 を適用して次の時点の不規則変動成分を求めている。18~23 行で配列の値を 1 時点分、シフトさせている。

プログラム 7.5　不規則変動成分（乾球温度と絶対湿度）の計算

Popolo.Weather.RandomWeather class

```
 1 /// <summary>不規則変動成分を更新する</summary>
 2 /// <param name="nrnd">正規乱数製造機</param>
 3 /// <param name="drybulbTemperature">乾球温度[C]</param>
 4 /// <param name="humidityRatio">絶対湿度[g/kg]</param>
 5 private static void updateRandomComponent
 6   (NormalRandom nrnd, ref double[] drybulbTemperature, ref double[] humidityRatio)
 7 {
 8   double nrnd1 = nrnd.NextDouble();
 9   double nrnd2 = nrnd.NextDouble();
10   double dbt = nrnd1 * iDTSD +
11     drybulbTemperature[2] * iDTCof[0] + humidityRatio[2] * iDTCof[1] +
12     drybulbTemperature[1] * iDTCof[2] + humidityRatio[1] * iDTCof[3] +
13     drybulbTemperature[0] * iDTCof[4] + humidityRatio[0] * iDTCof[5];
14   double hrt = nrnd2 * iHRSD +
15     drybulbTemperature[2] * iHRCof[0] + humidityRatio[2] * iHRCof[1] +
16     drybulbTemperature[1] * iHRCof[2] + humidityRatio[1] * iHRCof[3] +
17     drybulbTemperature[0] * iHRCof[4] + humidityRatio[0] * iHRCof[5];
18   drybulbTemperature[0] = drybulbTemperature[1];
19   drybulbTemperature[1] = drybulbTemperature[2];
20   drybulbTemperature[2] = dbt;
21   humidityRatio[0] = humidityRatio[1];
22   humidityRatio[1] = humidityRatio[2];
23   humidityRatio[2] = hrt;
24 }
```

確率的気象データの計算処理を 7.6 に示す。

14 行は第 2 引数の地点に応じてプログラム 7.1 で示した各種のインスタンス変数を表 7.1~7.6の値で初期化する処理である。紙面の都合により、具体的な実装は省略する。 16, 17 行は第 2 章で解説した乱数列製造機である。

出力は時刻別の乾球温度、絶対湿度、水平面全天日射、晴天・曇天の状態であり、第 3 引数で与えた年数×365 日×24 時間（=8760）の配列である。計算を簡略化するため、うるう年の存在は無視する。21 行で総日数と総時間数を求め、22~28 行で配列を初期化する。

30~39 行はトレンド成分の計算であり、単純な正規乱数に従うとする。

43 行でプログラム 7.3 を用いて年周期成分を計算し、44~48 行でプログラム 7.4 を用いて時刻別データに変換する。同様に 52 行でプログラム 7.3 を用いて日周期成分を計算する。

54~135 行は全天日射量の計算処理である。62~77 行で式 7.8 を適用し、マルコフ連鎖で雲量の状態を計算する。66~68 行は 0 時点の計算処理であり、式 7.9 で示した不変分布から確率的に雲量状態を発生させる。また本モデルは 3 時間ピッチのモデルであるため、76 行に示すように t+1, t+2 時点も同じ値を取るとする。

79~96 行は大気透過率の不規則変動成分の計算である。0 時点においては 82~89 行で助走期間として 100 時間の計算を行う。90~96 行で式 7.10 を用いて AR モデルで不規則変動成分を発生させる。

99~126 行で大気透過率の周期成分と不規則変動成分を重ねあわせ、水平面全天日射を計算する。ただし 105 行に示す通り、太陽高度が 0 以下の場合には太陽が地平面以下にあるため、大気透過率は確定的に 0 となる。乾球温度と絶対湿度の補正のため、108、109 行で式 7.13 の I_{ATcr} を計算しておく。最後に 123 行でプログラム 7.2 を呼び出し、大気透過率から水平面全天日射へ変換する。

129~131 行は乾球温度と絶対湿度の補正処理であり、式 7.14~7.16を適用している。補正値は日別データであるため、138~140 行でプログラム 7.4 を用いて時刻別データに変換する。

150~173 行は乾球温度と絶対湿度の計算である。 158 行で多変量 AR モデルを用いて不規則変動成分を更新する。159~161 行で周期成分と不規則変動成分を重ねあわせる。 162~167 行で相対湿度が上下限値（φ_{min}~100 %）におさまる絶対湿度を計算し、範囲外となる場合には 168~170 行で不規則変動成分を修正する。

プログラム 7.6　確率的気象データの計算処理

Popolo.Weather.RandomWeather class

```
 1 /// <summary>確率的に気象データを発生させる</summary>
 2 /// <param name="seed">乱数シード</param>
 3 /// <param name="location">地点</param>
 4 /// <param name="year">発生させる年数</param>
 5 /// <param name="drybulbTemperature">出力:乾球温度[C]</param>
 6 /// <param name="humidityRatio">出力:絶対湿度[g/kg]</param>
 7 /// <param name="radiation">出力:水平面全天日射[W/m2]</param>
 8 /// <param name="isFair">出力:晴れか否か</param>
 9 public static void MakeWeather
10   (int seed, Location location, int year, out double[] drybulbTemperature,
11   out double[] humidityRatio, out double[] radiation, out bool[] isFair)
12 {
13   //係数初期化
14   initParametor(location);
15
16   MersenneTwister rnd = new MersenneTwister(seed);
17   NormalRandom nRnd = new NormalRandom(seed);
18   DateTime dt = new DateTime(2001, 1, 1, 0, 0, 0);
19
20   int totalDay = year * 365;
```

```
 21   int totalHour = totalDay * 24;
 22   drybulbTemperature = new double[totalHour];
 23   humidityRatio = new double[totalHour];
 24   radiation = new double[totalHour];
 25   double[] swing = new double[totalDay];
 26   double[] dbtRnd = new double[totalDay];
 27   double[] hrtRnd = new double[totalDay];
 28   isFair = new bool[totalHour];
 29
 30   //トレンド成分を計算
 31   double[] trendDT = new double[year];
 32   double[] trendHR = new double[year];
 33   double[] trendAT = new double[year];
 34   for (int i = 0; i < year; i++)
 35   {
 36     trendDT[i] = nRnd.NextDouble() * sdevTDT;
 37     trendHR[i] = nRnd.NextDouble() * sdevTHR;
 38     trendAT[i] = nRnd.NextDouble() * sdevTAT;
 39   }
 40
 41   //確定的年周期成分を計算
 42   double[] caDBT, caHRT, caATM, caFTF, caCTC, caDTSIG, caHRSIG;
 43   makeAnnualData(out caDBT, out caHRT, out caATM, out caFTF, out caCTC, out caDTSIG, out caHRSIG);
 44   caATM = interpolateDailyData(caATM);
 45   caFTF = interpolateDailyData(caFTF);
 46   caCTC = interpolateDailyData(caCTC);
 47   caDTSIG = interpolateDailyData(caDTSIG);
 48   caHRSIG = interpolateDailyData(caHRSIG);
 49
 50   //確定的日周期成分を計算
 51   double[] ccDBT, ccHRT, ccATMF, ccATMC;
 52   makeCircadianData(out ccDBT, out ccHRT, out ccATMF, out ccATMC);
 53
 54   //水平面全天日射の計算
 55   int tHour = 0;
 56   for (int i = 0; i < totalDay; i++)
 57   {
 58     int yHour = tHour % 8760;
 59     int cYear = i / 365;
 60
 61     //晴れ曇りの状態をマルコフ連鎖で計算
 62     for (int j = 0; j < 8; j++)
 63     {
 64       int ch = tHour + 3 * j;
 65       //0時点の状態は不変分布から計算
 66       if (ch == 0)
 67         isFair[0] = isFair[1] = isFair[2] =
 68          ((1 - caCTC[yHour]) / (2 - caFTF[yHour] - caCTC[yHour])) < rnd.NextDouble();
 69       //その他の時点の状態は推移確率から計算
 70       else
 71       {
 72         bool curF = isFair[ch - 1];
 73         if (curF) isFair[ch] = rnd.NextDouble() < caFTF[yHour];
 74         else isFair[ch] = caCTC[yHour] < rnd.NextDouble();
 75       }
 76       isFair[ch + 2] = isFair[ch + 1] = isFair[ch];
 77     }
 78
 79     //不規則変動成分の計算
 80     double[] iATM = new double[24];
 81     //時点0の場合には助走計算
 82     if (i == 0)
 83     {
 84       for (int j = 0; j < 50; j++)
 85       {
 86         iATM[23] = iATM[23] * iATCof + nRnd.NextDouble() * iATSD;
 87         iATM[23] = iATM[23] * iATCof + nRnd.NextDouble() * iATSD;
 88       }
 89     }
 90     iATM[0] = iATM[23] * iATCof + nRnd.NextDouble() * iATSD;
 91     iATM[1] = iATM[0] * iATCof + nRnd.NextDouble() * iATSD;
 92     for (int j = 1; j < 12; j++)
 93     {
 94       iATM[j * 2] = iATM[j * 2 - 1] * iATCof + nRnd.NextDouble() * iATSD;
 95       iATM[j * 2 + 1] = iATM[j * 2] * iATCof + nRnd.NextDouble() * iATSD;
 96     }
 97
 98     //確定成分と不規則変動成分を集計
 99     int nSum = 0;
100     double rSum = 0;
```

```
101     for (int j = 0; j < 24; j++)
102     {
103       int ch = tHour + j;
104       sun.Update(dt.AddHours(j));
105       if (0 < sun.Altitude)
106       {
107         nSum++;
108         double idisF = (1 - caCTC[yHour]) / (2 - caFTF[yHour] - caCTC[yHour]);
109         rSum -= idisF * ccATMF[j] + (1 - idisF) * ccATMC[j];
110         if (isFair[ch])
111         {
112           radiation[ch] = trendAT[cYear] + caATM[yHour] + iATM[j] + ccATMF[j];
113           rSum += iATM[j] + ccATMF[j];
114         }
115         else
116         {
117           radiation[ch] = trendAT[cYear] + caATM[yHour] + iATM[j] + ccATMC[j];
118           rSum += iATM[j] + ccATMC[j];
119         }
120
121         //大気透過率から水平面全天日射を計算（渡辺の式）
122         radiation[ch] = Math.Min(1.0, Math.Max(0.001, radiation[ch]));
123         radiation[ch] = getGlobalHorizontalRadiation(radiation[ch], sun);
124       }
125       else radiation[ch] = 0;
126     }
127     //不規則成分で振幅変調および絶対値シフト
128     rSum /= nSum;
129     dbtRnd[i] = caDBT[i % 365] +rSum * shiftDT;
130     hrtRnd[i] = caHRT[i % 365] +rSum * shiftHR;
131     swing[i] = rSum * swingDTa + swingDTb;
132
133     dt = dt.AddDays(1);
134     tHour += 24;
135   }
136
137   //振幅変調・絶対値シフトデータを時刻別データにSpline補間
138   dbtRnd = interpolateDailyData(dbtRnd);
139   hrtRnd = interpolateDailyData(hrtRnd);
140   swing = interpolateDailyData(swing);
141
142   //VARモデル助走計算
143   double[] idbt = new double[3];
144   double[] ihrt = new double[3];
145   for (int i = 0; i < 100; i++)
146     updateRandomComponent(nRnd, ref idbt, ref ihrt);
147
148   //確定成分と不規則変動成分を合成
149   tHour = 0;
150   for (int i = 0; i < totalDay; i++)
151   {
152     int yHour = tHour % 8760;
153     int cYear = i / 365;
154
155     for (int j = 0; j < 24; j++)
156     {
157       int ch = tHour + j;
158       updateRandomComponent(nRnd, ref idbt, ref ihrt);
159       drybulbTemperature[ch] = trendDT[cYear] +
160         idbt[idbt.Length - 1] * caDTSIG[yHour] + dbtRnd[ch] + ccDBT[j] * swing[ch];
161       humidityRatio[ch] = trendHR[cYear] + hrtRnd[ch] + ccHRT[j];
162       double hrtMax = MoistAir.GetSaturationHumidityRatioFromDryBulbTemperature
163         (drybulbTemperature[ch], 101.325) * 1000 - humidityRatio[ch];
164       double hrtMin =
165         MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
166             (drybulbTemperature[ch], minimumRelativeHumidity, 101.325) * 1000;
167       hrtMin -= humidityRatio[ch];
168       ihrt[2] = (Math.Max(Math.Min(ihrt[2] * caHRSIG[yHour], hrtMax), hrtMin));
169       ihrt[2] /= caHRSIG[yHour];
170       humidityRatio[ch] += ihrt[2] * caHRSIG[yHour];
171     }
172     tHour += 24;
173   }
174 }
```

【例題7.1】

各都市の月平均乾球温度と相対湿度を求め、クリモグラフを作成せよ。

【解】

計算処理をプログラム 7.7 に示す。12~39 行でループをかけ、各都市について計算を行う。17, 18 行で 1 年間の気象データを作成し、22~30 行で月別に積算する。31~38 行で各月の時間数で除して平均値を計算した後、外部出力を行う。結果をグラフ化すると図 7.2 が得られる。

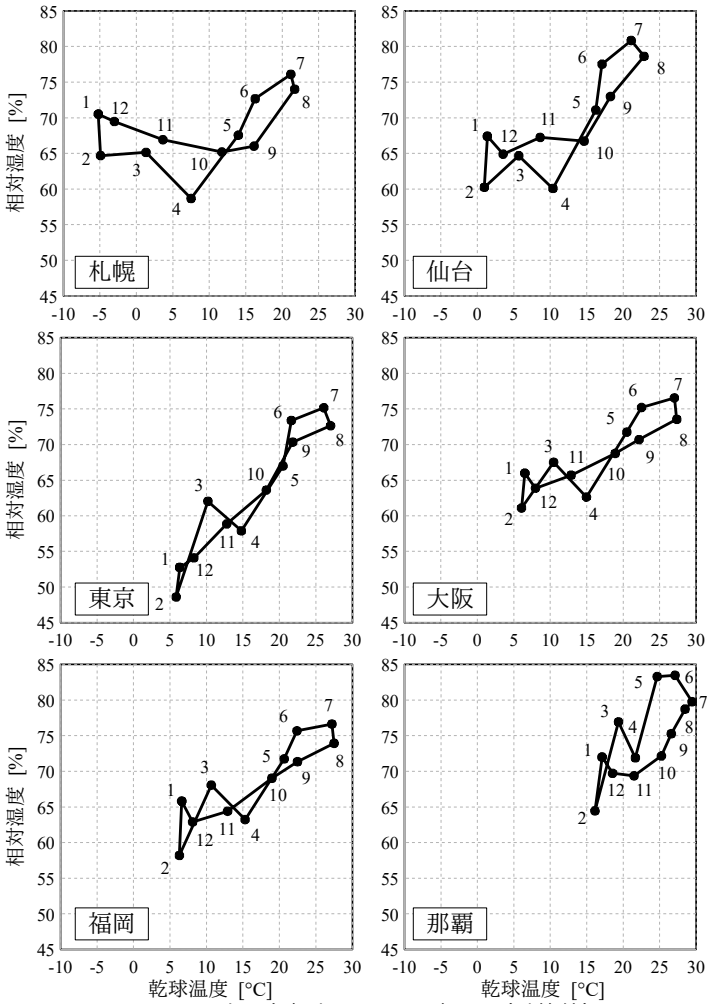


図 7.2　日本の各都市のクリモグラフ（計算値）

プログラム 7.7　クリモグラフ作成処理

Popolo.Weather.RandomWeather class

```
 1 private static void RandomWeatherTest(int seed)
 2 {
 3   int[] days =
 4     new int[] { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
 5   using (StreamWriter sWriter = new StreamWriter
 6   ("weather.csv", false, Encoding.GetEncoding("Shift_JIS")))
 7   {
 8     for (int i = 0; i < days.Length; i++)
 9       sWriter.Write("," + (i + 1).ToString("F0") + "月");
10     sWriter.WriteLine();
11
12     foreach (RandomWeather.Location loc
13       in Enum.GetValues(typeof(RandomWeather.Location)))
14     {
15       double[] dbt, hrt, rad;
16       bool[] fcf;
17       RandomWeather.MakeWeather
18         (seed, loc, 1, out dbt, out hrt, out rad, out fcf);
19       double[] dbtAve = new double[12];
```

```
20        double[] rhdAve = new double[12];
21        DateTime dt = new DateTime(1999, 1, 1, 0, 0, 0);
22        for (int i = 0; i < dbt.Length; i++)
23        {
24          dbtAve[dt.Month - 1] += dbt[i];
25          double rhd =
26            MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
27            (dbt[i], hrt[i] / 1000d, 101.325);
28          rhdAve[dt.Month - 1] += rhd;
29          dt = dt.AddHours(1);
30        }
31        sWriter.Write(loc.ToString() + "乾球温度");
32        for (int i = 0; i < dbtAve.Length; i++)
33          sWriter.Write("," + (dbtAve[i] / (days[i] * 24)).ToString("F2"));
34        sWriter.WriteLine();
35        sWriter.Write(loc.ToString() + "相対湿度");
36        for (int i = 0; i < rhdAve.Length; i++)
37          sWriter.Write("," + (rhdAve[i] / (days[i] * 24)).ToString("F2"));
38        sWriter.WriteLine();
39      }
40    }
41 }
```

【第７章 記号表】

A_n	：フーリエ級数 **[-]**	P	：大気透過率 **[-]**
B_n	：フーリエ級数 **[-]**	P_C	：曇天の確率 **[-]**
CA	：年周期成分	P_{CC}	：曇天→曇天の確率 **[-]**
CC	：日周期成分	P_F	：晴天の確率 **[-]**
cf_{sh}	：日平均値補正係数	P_{FF}	：晴天→晴天の確率 **[-]**
cf_{sw}	：振幅補正係数	T	：フーリエ級数の周期
h	：太陽高度	T	：トレンド成分
I	：不規則変動成分	Y	：原系列
I_{DN}	：法線面直達日射 **[W/m²]**	α	：**VAR** 係数
I_d	：水平面天空日射 **[W/m²]**	β	：**VAR** 係数
I_{HOL}	：水平面全天日射 **[W/m²]**	φ_{min}	：相対湿度下限値 **[%]**
I_0	：交換熱量 **[kW]**	σ	：標準偏差
N	：時点		
添字：			
AT	：大気透過率	F	：晴天
C	：曇天	HR	：絶対湿度
cr	：補正済	S	：規準化
DT	：乾球温度	SD	：標準偏差

【第７章 参考文献】

7.1） 富樫英介: 設備システムの省エネルギー化が不動産価値に与える影響の定量的評価方法に関する研究 第2報 省エネ投資リスク評価のための確率的気象モデルの開発, 空気調和・衛生工学会論文集 No.221, pp.21-29, 2015

7.2） 浦野良美, 渡辺俊行, 林徹夫, 高尾直樹, 村高秀人：水平面全天日射量観測値から方位別日射量を推定する方法について, 日本建築学会九州支部研究報告, No.27, pp.97-100 , 1983

7.3） 渡辺俊行, 浦野良美, 林徹夫：水平面全天日射量の直散分離と傾斜面日射量の推定, 日本建築学会論文報告集, No.330, pp.96-108, 1983

7.4） 二宮秀與,赤坂裕: BEST 用気象データ, 空気調和・衛生工学 82(11), pp.933-938, 2008

7.5） 北川源四郎: 時系列解析入門, 岩波書店, 2005

7.6） 田中孝文: R による時系列分析入門, シーエーピー出版, 2008

7.7） 吉田 治典 , 寺井 俊夫: 気象データの時系列モデルと確率的熱負荷に関する研究, 日本建築学会計画系論文集 (463), pp.11-19, 1994

7.8） 吉田 治典 , 寺井 俊夫: 熱負荷計算用気象データのモデル化 ：気温の日周期成分についての検討, 日本建築学会計画系論文報告集 (391), pp.39-49, 1988

7.9） 早川一也, 清水浩明: モンテカルロ法による熱負荷計算のための気象データの研究, 空気調和・衛生工学会論文集, No.0(見本号), 1976.03, pp.1~15

7.10） 赤坂裕: 年間毎時刻雲量, 日照率, 日射データの作成法, 日本建築学会計画系論文報告集, No.370, pp.1-12, 1986

7.11） 赤坂裕: 年間毎時刻雲量データの作成法, 日本建築学会計画系論文報告集, No.326, pp.103-110, 1983

7.12） 日本建築学会: 拡張アメダス気象データ 1981-2000, 鹿児島 TLO, 2005

7.13） 松本真一, 赤坂裕, 永村一雄, 二宮秀與, 村上周三, 井川憲男,武田和大, 窪田真樹: 外皮・躯体と設備・機器の総合エネルギーシミュレーションツール「BEST」の開発, （その 114）BEST で使用される拡張アメダス気象データの開発状況, 空気調和・衛生工学会大会学術講演論文集, pp.9-12, 2013

第8章　熱交換
(Heat Exchange)

8.1　概要

建築熱環境設計の目的の一つは、対象となる空間の熱の出入りを制御して目標となる温熱環境を提供することにある。第 3 章~第 5 章で記したとおり、熱を運搬する際には、水・空気・冷媒などの熱媒を使用することになるが、ある熱媒から別の熱媒への熱の移動を熱交換と呼び、その装置を熱交換器と総称する。従って、熱交換に関する計算技術は、建築熱環境システムを構成する多くの機器に応用可能な基礎的な技術である。本章では種々の熱交換器で共通して成立する基礎的な理論と計算法について解説する。具体的な機器への応用方法については、後章において個別に行うこととする。

8.2　理論

8.2.1　対数平均温度差

高温流体と低温流体の間の熱交換を考えることにする。高温側流体の入口温度を T_{hi} [K]、出口温度を T_{ho} [K]、比熱を c_h [J/(kg·K)]、質量流量を m_h [kg/s]とすると、高温側流体が失う熱量 Q [W]は式 8.1 で表現できる。

$$Q=m_h c_h (T_{hi}-T_{ho}) \tag{8.1}$$

同様に、低温側流体の入口温度を T_{ci} [K]、出口温度を T_{co} [K]、比熱を c_c [J/(kg·K)]、質量流量を m_c [kg/s]とすると、低温側流体が得る熱量 Q [W]は式 8.2 で表現できる。

$$Q=m_c c_c (T_{co}-T_{ci}) \tag{8.2}$$

熱交換器内において、両流体間の熱交換は熱伝導率の高い隔壁を介して行われる。熱交換器内の温度分布の例を図 8.1 に示す。

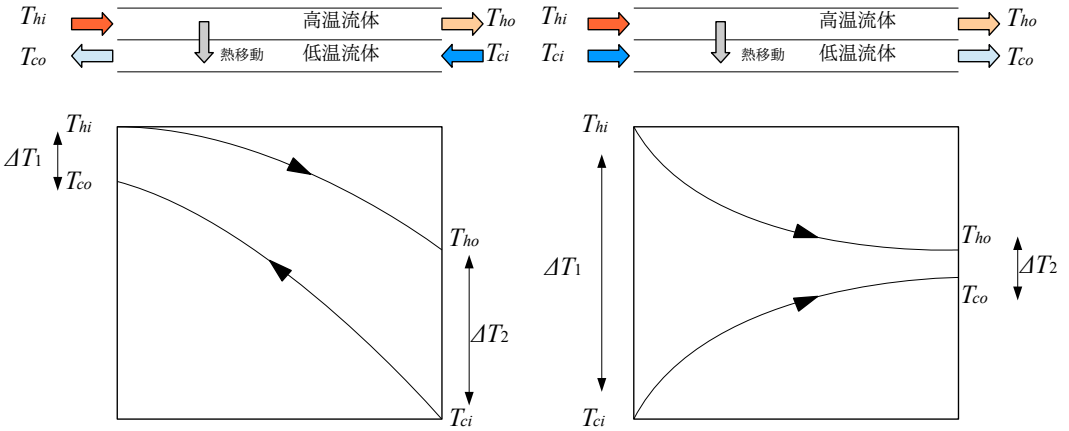


図 8.1　熱交換器内の温度分布（左：向流型、右：並流型）

逆向きに流体が流れる場合には、熱交換器内の温度分布は図 8.1 左の様になる。このような逆向きに流体が流れる種類の熱交換器を、向流の熱交換器と呼ぶ。隔壁を介した、ある点での熱流は温度差

（図中の高さ方向の差分）に依存し、式 8.3 で表現できる。ここで K [W/(m²·K)]は熱通過率、A [m²]は熱交換器の伝熱面積である†1)。式 8.3 を積分すれば式 8.4 が得られ、熱交換器全体での平均的な熱流を求めることができる。

$$dQ = K\left(T_h - T_c\right)dA \tag{8.3}$$

$$Q = KA\Delta T_m \tag{8.4}$$

ΔT_m [K]は熱交換器内の平均的な温度差を示しており、この値をどのように求めるかが問題となる。向流型熱交換器の場合には解析解が求められており、式 8.5 で計算することができる。ここで、ΔT_1 [K]と ΔT_2 [K]はそれぞれ式 8.6 と式 8.7 で表されるように、出入口における両流体の温度差である。特に式 8.5 で表現される温度差を対数平均温度差 ΔT_{LM} [K]と呼ぶ。ただし、適用にあたっては、以下に挙げる 5 つの仮定が成立している必要がある。特に仮定 4 に関しては、冷却除湿コイルや冷媒の直膨コイルなどについては成立しておらず、対数平均温度差の代わりに対数平均エンタルピー差という概念を用いて計算することがある。詳細は後章において述べることとする。

1）熱通過率 K [W/(m²·K)]は一定である。

2）両流体の質量流量は一定である。

3）両流体の比熱は一定である。

4）液化や気化など、流体の相変化は生じない

5）熱交換以外の熱移動（損失）は生じない

$$\Delta T_m = \Delta T_{LM} = \frac{\Delta T_1 - \Delta T_2}{\ln\left(\Delta T_1 / \Delta T_2\right)} \quad \text{（向流型の場合）} \tag{8.5}$$

$$\Delta T_1 = T_{hi} - T_{co} \quad \text{（向流型の場合）} \tag{8.6}$$

$$\Delta T_2 = T_{ho} - T_{ci} \quad \text{（向流型の場合）} \tag{8.7}$$

熱交換器内で高温側と低温側の流体が同一方向に流れる熱交換器を並流型の熱交換器と呼び、並流型についても ΔT_m の解析解がある。ΔT_1 と ΔT_2 をそれぞれ式 8.8 と式 8.9 で求め、式 8.5 に代入すれば良い。図 8.1 右に並流型熱交換器の場合の温度分布を示す。

$$\Delta T_1 = T_{hi} - T_{ci} \quad \text{（並流型の場合）} \tag{8.8}$$

$$\Delta T_2 = T_{ho} - T_{co} \quad \text{（並流型の場合）} \tag{8.9}$$

向流と並流以外の熱交換器一般の平均温度差 ΔT_m に関しては、必ずしも解析解が用意されていない。例えば冷却塔においては空気が水平方向に流れ、水が垂直方向に流れるため、直交流型の熱交換器と呼ばれるが、解析的に平均温度差を求めることはできない。これらの熱交換器について簡便に計算を行うために、補正係数 F_G [-]を用いる方法がある。補正係数 F_G は向流型の熱交換器の平均温度差である対数平均温度差 ΔT_{LM} に比較して、検討の対象となる熱交換器の温度差がいくらになるのかという割合を示す係数である。補正係数 F_G を用いると熱交換器の交熱流 Q は式 8.10 で表現できる。

$$Q = KA F_G \Delta T_{LM} \tag{8.10}$$

典型的な熱交換器に関しては、図 8.2 に示すように出入口温度および補正係数 F_G の関係が図表として与えられている[8.1)]。手計算の場合にはこのような図表から読み取る[8.5) 8.6) 8.7)]。

†1　厳密には流体の温度や流量に依存して熱通過率 K の値は変化する。しかし、設備シミュレーションでは簡易化して熱通過率 K と伝熱面積 A を合わせて固定値として扱うことが多い。なお、KA を伝熱係数と呼ぶ。

シミュレーションを行う際には、式 8.11~式 8.13 に示す p, q, r, R を用いて直接的に平均の温度差 ΔT_m を計算する方法が提案されている[8.2)]。

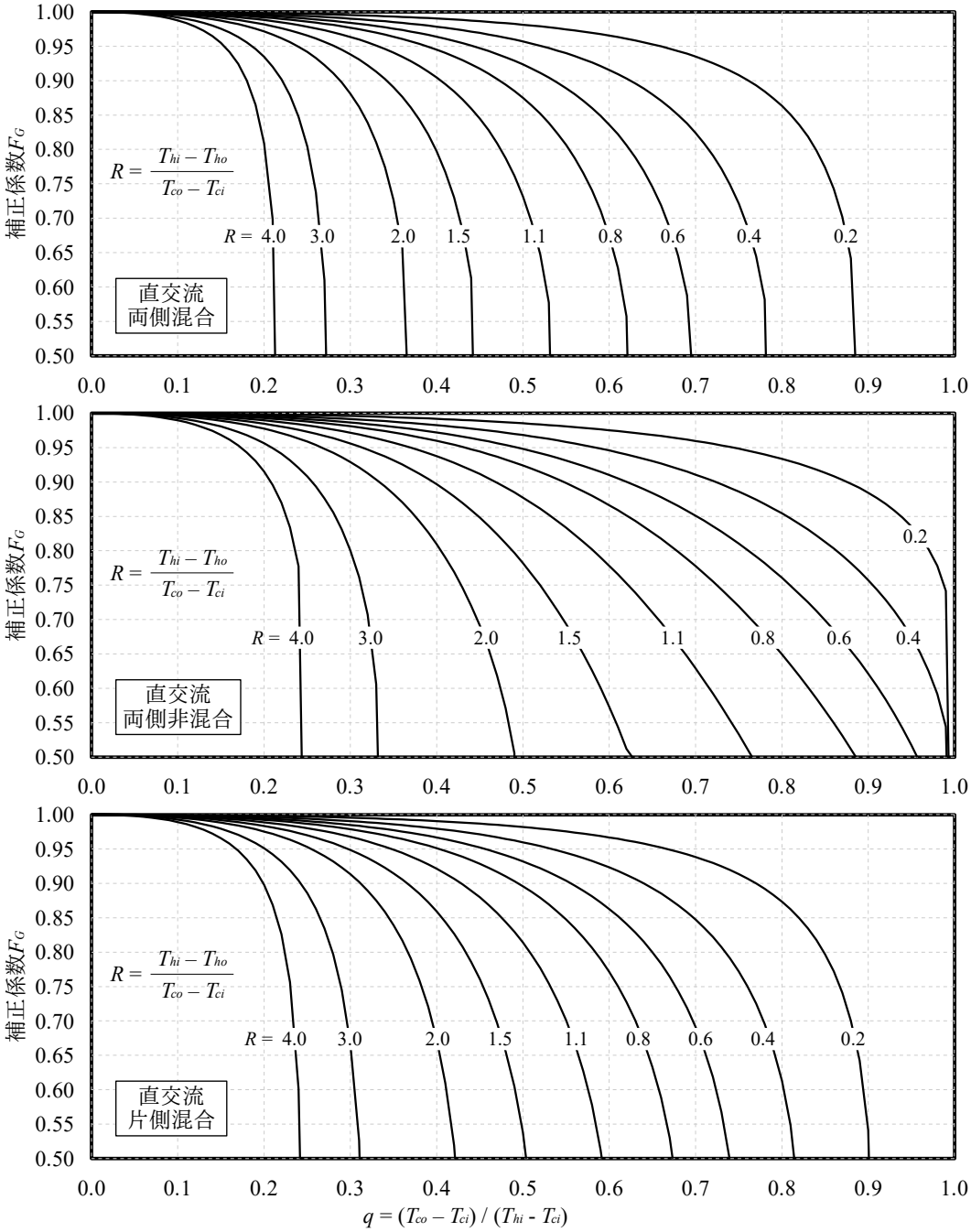


図 8.2　各種熱交換器の補正係数 F_G

$$p=\frac{T_{hi}-T_{ho}}{T_{hi}-T_{ci}} \tag{8.11}$$

$$q=\frac{T_{co}-T_{ci}}{T_{hi}-T_{ci}} \tag{8.12}$$

$$r=\frac{\Delta T_m}{T_{hi}-T_{ci}} \tag{8.13}$$

$$R=\frac{p}{q}=\frac{T_{hi}-T_{ho}}{T_{co}-T_{ci}} \tag{8.14}$$

向流型熱交換器の場合には、p, q, r の関係は式 8.15 で表現される。

（向流型）

$$r=(p-q)\left(\ln\frac{1-q}{1-p}\right)^{-1} \tag{8.15}$$

直交流型熱交換器で、各々の流体が流れと直交方向に混合する場合には、p, q, r の関係は式 8.16 で表現される[†1)]。

（直交流型 ：両流体混合）

$$r=\frac{p}{1-\exp(-p/r)}+\frac{q}{1-\exp(-q/r)}-1 \tag{8.16}$$

直交流型熱交換器で、片方の流体のみが混合する場合には、p, q, r の関係は式 8.17 で表現される。

（直交流型 ：片側流体混合）

$$r=\frac{q}{\ln\dfrac{1}{1-\dfrac{q}{p}\ln\dfrac{1}{1-p}}} \tag{8.17}$$

直交流型熱交換器で、両流体共に混合しない場合には、p, q, r の関係は式 8.18 で表現される[8.4)]。

（直交流型 ：両流体非混合）

$$r=\sum_{u=0}^{\infty}\sum_{v=0}^{\infty}\left\{(-1)^{u+v}\frac{(u+v)!}{u!(u+1)!\,v!(v+1)!}\left(\frac{p}{r}\right)^{u}\left(\frac{q}{r}\right)^{v}\right\} \tag{8.18}$$

式 8.16 と式 8.18 に関しては両辺に r が表れるため、適用にあたっては収束計算が必要となる。

【例題 8.1】

下記の定格性能値を持つプレート型熱交換器について、必要となる伝熱面積を求めよ。

低温冷水入口温度 ：6.0 °C　　低温冷水出口温度 ：10.0 °C
高温冷水入口温度 ：12.0 °C　　高温冷水出口温度 ：7.0 °C
熱交換能力 ：500 kW　　熱通過率 ：4 kW/(m^2·K)

【解】

向流型のプレート式熱交換器を想定することとし、平均温度差としては対数平均温度を使用する。式 8.5~式 8.7 により ΔT_{LM} は、

$\Delta T_1 = 12.0 - 10.0 = 2.0$ °C　　$\Delta T_2 = 7.0 - 6.0 = 1.0$ °C

$\Delta T_{LM} = (2.0 - 1.0) \div \ln(2.0 / 1.0) = 1.4$ °C

である。式 8.4 により、

$500 = A \times 4 \times 1.4$　　$A = 89$ m^2

となる。

8.2.2　熱通過有効度

1）熱通過有効度

実際の計算にあたっては、熱交換器の入口条件が与えられた上で、出口の状態を予測することが問題となることが多い。しかし、対数平均温度差は出口温度の関数であるため、対数平均温度差を利用して出口状態を予測するためには反復計算が必要となる。この反復計算を回避するために、熱通過有効度という概念を用いた計算方法が提案されている[8.1)]。

熱通過有効度を用いた熱交換量の計算式を式 8.19 および式 8.20 に示す。ここで、ε [-]が熱通過有効

†1　空調設備分野での片側流体混合の例としては冷温水コイルが挙げられる。チューブ内を通る冷温水は混合しない一方で、フィンの間を移動する空気は自由に移動する。また、両流体非混合の例としては静止型全熱交換器が挙げられる。両流体混合の直交型熱交換器は思いつかない。

度であり、無次元の係数である。mc [W/K]は熱容量流量であり、流体の質量流量 m [kg/s]と比熱 c [J/(kg·K)]を乗じた値である。

$$Q=\varepsilon\, mc_{min}\left(T_{hi}-T_{ci}\right) \tag{8.19}$$

$$mc_{min}=Min\left(mc_h, mc_c\right) \quad , \quad mc_{max}=Max\left(mc_h, mc_c\right) \tag{8.20}$$

この式の意味は以下のとおりである。伝熱面積が無限大の向流型熱交換器を想定すると、熱交換を行った結果、高温側流体の出口温度 T_{ho} または低温側流体の出口温度 T_{co} のいずれかが、低温側流体の入口温度 T_{ci} または高温側流体の入口温度 T_{hi} に一致する。つまり、流体の出入口のいずれかで、高温側流体と低温側流体の温度が一致する。この場合の温度変化を図 8.3 に示す。

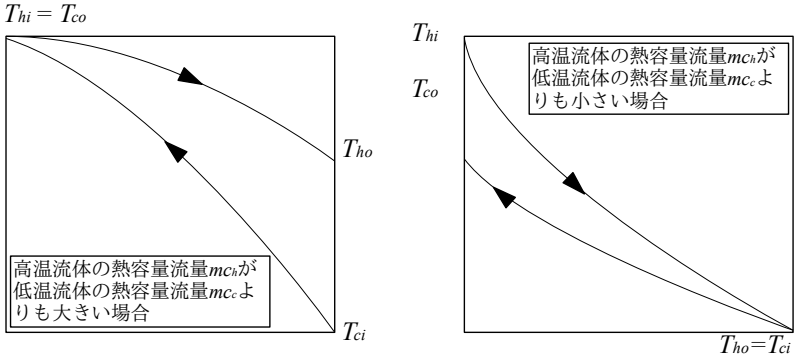


図 8.3　無限大の伝熱面積を持つ向流型熱交換器内の温度分布

熱容量流量の小さい方の流体の出口温度は、熱容量流量の大きい方の流体の入口温度に一致する。図から明らかなように、熱容量流量の小さい方の流体の出入口温度差は T_{hi} - T_{ci} である。温度差に熱容量流量を乗じれば熱交換量が算出されるため、結局のところ、式 8.19 の mc_{min} $(T_{hi} - T_{ci})$ は、無限大の伝熱面積を持つ向流型熱交換器によって交換できる最大の熱量を意味している。熱通過有効度 ε は、このような理想的な熱交換器に比較して、検討対象となっている熱交換器の効率を示す係数であると言える。

問題は熱通過有効度 ε の具体的な算出方法であるが、両流体の熱容量流量の比 R_{mc} [-]と、移動単位数 NTU [-]（Number of Heat Transfer Units）という二つの無次元数で表現できることが知られている[†1)]。それぞれ式 8.21 と式 8.22 で計算する。ただし、mc_{max} および mc_{min} は式 8.20 である。NTU は熱容量流量に対する熱交換の能力であり、熱交換器の大きさの尺度を表している。NTU が 0 の場合には熱通過有効度 ε は 0 である。

$$NTU=KA/mc_{min} \tag{8.21}$$

$$R_{mc}=mc_{min}/mc_{max} \tag{8.22}$$

直膨コイルや蒸気コイルなど、熱交換器内で片側の流体が一定の温度を取る場合には、熱通過有効度 ε は式 8.23 で計算できる。これは、向流型、並流型、直交流型など、熱交換器の種別を問わず同じで、片側の熱容量流量が他方に比較して無限大に大きいとみなせる場合であり、R_{mc}=0 に相当する。

$$\varepsilon=1-\exp\left(-NTU\right) \quad \text{（片側流体温度一定）} \tag{8.23}$$

向流型熱交換器および並流型熱交換器の熱通過有効度 ε は各々、式 8.24 と式 8.25 で計算する。

†1　移動単位数は「伝熱単位数」あるいは「熱交換単位数」と呼ばれることもある。

（向流型）

$$\varepsilon=\begin{cases}\dfrac{1-\exp[(R_{mc}-1)NTU]}{1-R_{mc}\exp[(R_{mc}-1)NTU]} & (0<R_{mc}<1)\\ NTU/(1+NTU) & (R_{mc}=1)\end{cases} \tag{8.24}$$

（並流型）

$$\varepsilon=\begin{cases}\varepsilon=\dfrac{1-\exp[-NTU(1+R_{mc})]}{1+R_{mc}} & (0<R_{mc}<1)\\ 0.5(1-\exp(-2NTU)) & (R_{mc}=1)\end{cases} \tag{8.25}$$

直交流型熱交換器で、各々の流体が流れと直交方向に混合する場合には、式 8.26 で計算する。

（直交流型 ：両流体混合）

$$\varepsilon=\frac{NTU}{\dfrac{NTU}{1-\exp(-NTU)}+\dfrac{R_{mc}NTU}{1-\exp(-R_{mc}NTU)}-1} \tag{8.26}$$

直交流型熱交換器で、片方の流体のみが混合する場合には、式 8.27 および式 8.28 で計算できる。式 8.27 は熱容量流量が小さい側の流体が非混合の場合、式 8.28 は熱容量流量が大きい側の流体が非混合の場合である。

（直交流型 ：片側流体混合, $mc_{mixed}=mc_{min}$）

$$\varepsilon=1-\exp\left(\frac{\exp(-NTU\,R_{mc})-1}{R_{mc}}\right) \tag{8.27}$$

（直交流型 ：片側流体混合, $mc_{mixed}=mc_{max}$）

$$\varepsilon=\frac{1-\exp((\exp(-NTU)-1)R_{mc})}{R_{mc}} \tag{8.28}$$

両流体共に混合しない場合の熱通過有効度 ε は単純には計算できないため、図表を用いるか式 8.29 で示される近似式で計算を行う[8.3)]。

$$\varepsilon=1-\exp\left(\frac{\exp(-R_{mc}NTU^{0.78})-1}{R_{mc}NTU^{-0.22}}\right) \quad \text{（直交流型 ：両流体非混合）} \tag{8.29}$$

図 8.4 に各種の熱交換器の熱通過有効度を示す。コンピュータを用いない場合にはこのような図表を読み取りながら計算を行う[8.5) 8.6) 8.7)]。

2）温度効率

$mc_h=mc_{min}$ の場合には $Q=mc_h(T_{hi}-T_{ho})$ を式 8.19 に代入して式 8.30 が得られる。同様に $mc_c=mc_{min}$ の場合には $Q=mc_c(T_{co}-T_{ci})$ を式 8.19 に代入して式 8.31 が得られる。この時、η_h と η_c はそれぞれ高温側と低温側の温度効率と呼ばれる。モデルによっては温度効率を一定として取り扱うことも場合も多い。

$$\varepsilon=\eta_h=\frac{T_{hi}-T_{ho}}{T_{hi}-T_{ci}} \qquad (mc_h=mc_{min}\text{の場合}) \tag{8.30}$$

$$\varepsilon=\eta_c=\frac{T_{co}-T_{ci}}{T_{hi}-T_{ci}} \qquad (mc_c=mc_{min}\text{の場合}) \tag{8.31}$$

【例題 8.2】

例題 8.1 のプレート型熱交換器について、下記の条件の場合の出口状態を求めよ。

低温冷水入口温度 ：6.0 °C　　高温冷水入口温度 ：12.0 °C

低温冷水流量 ：1,000L/min　　高温冷水流量 ：900 L/min

熱通過率 ：4 kW/(m²·K)　　伝熱面積 ：89 m²

【解】

まず、各流体の熱容量流量を計算する。

低温側：1,000 L/min × 1.0 kg/L ÷ 60 s/min × 4.186 kJ/(kg·K) = 69.8 kW/K

高温側：900 L/min × 1.0 kg/L ÷ 60 s/min × 4.186 kJ/(kg·K) = 62.8 kW/K

従って、熱容量流量の比 R_{mc} は

62.8 ÷ 69.8 = 0.90

である。移動単位数 *NTU* は式 8.21 を用いて、

$NTU = 4\ \mathrm{kW/(m^2 \cdot K)} \times 89\ \mathrm{m^2} \div 62.8\ \mathrm{kW/K} = 5.7$

となる。プレート熱交換器は向流型の熱交換器であるとみなせるため、熱通過有効度は式 8.24 を用いて、

$\exp[(R_{mc}\text{-}1)\ NTU] = \exp[(0.9 - 1.0) \times 5.7] = 0.57$

$\varepsilon = (1 - 0.57) \div (1 - 0.9 \times 0.57) = 0.88$

となる。熱交換量は、式 8.19 により、

$Q = 0.88 \times 62.8 \times (12.0 - 6.0) = 331.6$ kW

となる。従って出口水温は各々、

低温側：6.0 °C + 331.6 kW ÷ 69.8 kW/K = 10.8 °C,　高温側：12.0 °C – 331.6 kW ÷ 62.8 kW/K = 6.7 °C

となる。

なお、例題 8.1 の条件により、低温側と高温側の定格水量は各々、1,792 L/min と 1,433 L/min となる。この水量の場合に上記の計算を行うと、交換熱量が定格能力である 500kW に一致することが確認できる。

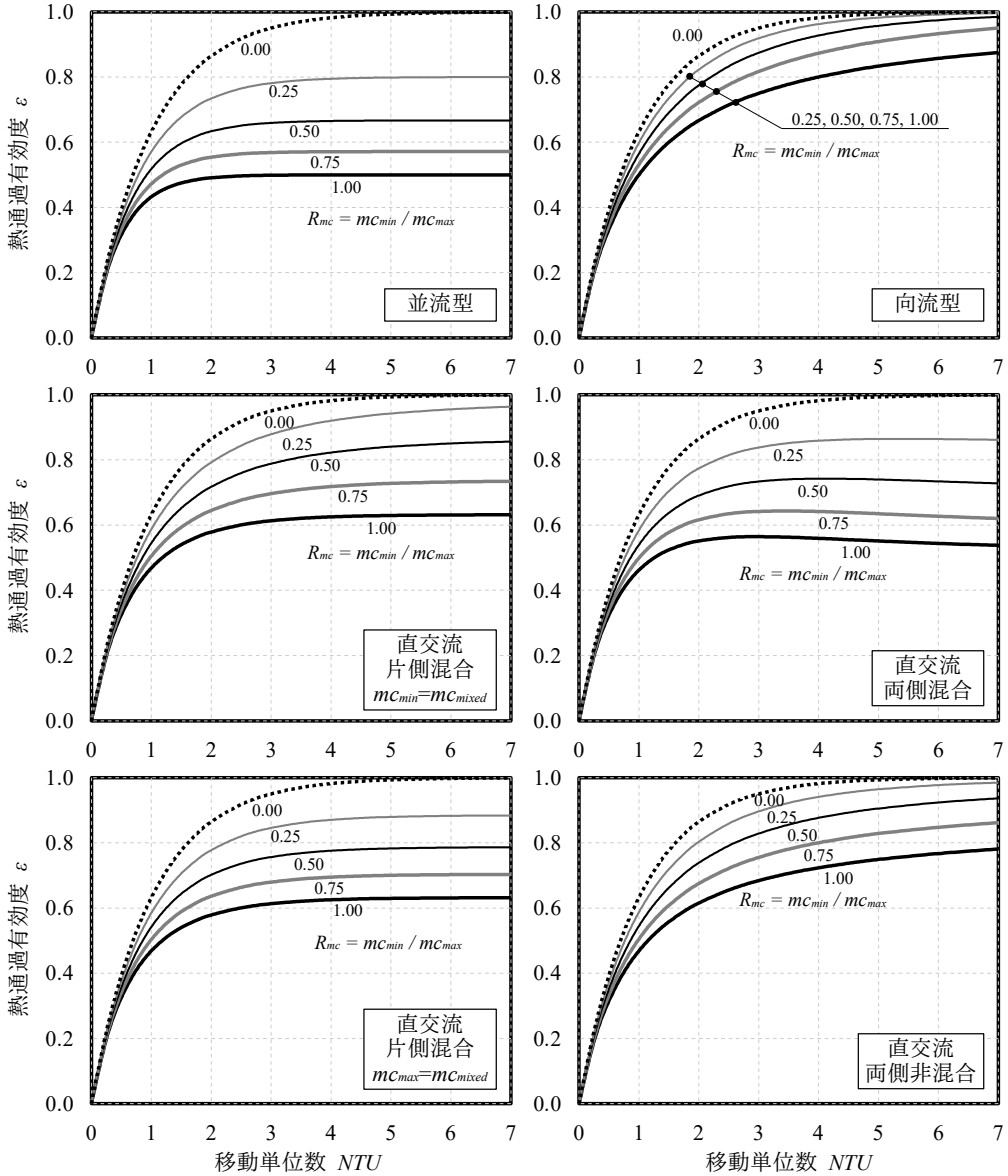


図 8.4　各種熱交換器の熱通過有効度 ε

8.3　計算法

プログラム 8.1 に流れのタイプを表わす列挙型の定義を示す。

プログラム 8.1　流れのタイプの定義

Popolo.HVAC.HeatExchanger.HeatExchange class

```
 1 /// <summary>流れのタイプ</summary>
 2 public enum FlowType
 3 {
 4   /// <summary>向流型</summary>
 5   CounterFlow,
 6   /// <summary>並流型</summary>
 7   ParallelFlow,
 8   /// <summary>直交流型（両流体混合）</summary>
 9   CrossFlow_BothFluidsUnmixed,
10   /// <summary>直交流型（熱容量流量大の流体が混合）</summary>
11   CrossFlow_CmaxMixed,
12   /// <summary>直交流型（熱容量流量小の流体が混合）</summary>
13   CrossFlow_CminMixed,
14   /// <summary>直交流型（両流体非混合）</summary>
15   CrossFlow_BothFluidMixed
16 }
```

8.3.1　平均温度差の計算

プログラム 8.2 に各種の熱交換器の平均温度差の計算処理を示す。

12~28 行と 38~44 行はそれぞれ式 8.5~8.9と式 8.17 の直接的な実装である。一方、式 8.16 と式 8.18 に関しては収束計算処理が必要となるため、45~63 行で黄金探索を行う。式 8.16 と式 8.18 にもとづく誤差関数は 69~77 行と 79~116 行に定義した。いずれも仮定した r の値に対して誤差を出力する関数である。黄金探索を実行するためには解の範囲を絞り込む必要がある。直交流の熱交換器の平均温度差は向流の熱交換器の平均温度差を上回ることは無いため、向流熱交換器の平均温度差（ΔT_{LM}）を前提として求めた r を上限値とする（52 行）。また、出口温度差が平均温度となる値（$\Delta T_m = T_{ho} - T_{co}$）を前提として求めた値を下限値とする（53 行）。グラフを描画するとわかるが、q や p の値によっては式 8.16 と式 8.18 は解を持たない。そこで極小値探索をした結果、値が 0 にならない（解がない）場合には平均温度差=0 とする（62 行）。

79~116 行に定義した式 8.18 の計算はやや複雑である。無限の和で表わされるため、u と v を 1 ずつ上げていき、収束したところで計算を打ち切る。しかし r が p や q と比較して非常に小さい場合には、なかなか収束しない。そこで計算回数の上限値を定めておき（86 行）、これを超える場合には r が極めて小さいものとみなして上記の下限値を出力する（115 行）。

プログラム 8.2　平均温度差 ΔT_m の計算

Popolo.HVAC.HeatExchanger.HeatExchange class

```
 1 /// <summary>出入口流体温度にもとづいて平均温度差[C]を計算する</summary>
 2 /// <param name="hotInletTemperature">高温流体入口温度[C]</param>
 3 /// <param name="coldInletTemperature">低温流体入口温度[C]</param>
 4 /// <param name="hotOutletTemperature">高温流体出口温度[C]</param>
 5 /// <param name="coldOutletTemperature">低温流体出口温度[C]</param>
 6 /// <param name="flowType">流れのタイプ</param>
 7 /// <returns>平均温度差[C]</returns>
 8 public static double GetMeanTemperatureDifference
 9   (double hotInletTemperature, double coldInletTemperature,
10   double hotOutletTemperature, double coldOutletTemperature, FlowType flowType)
11 {
12   //向流の場合
13   if (flowType == FlowType.CounterFlow)
14   {
15     double dt1 = hotInletTemperature - coldOutletTemperature;
16     double dt2 = hotOutletTemperature - coldInletTemperature;
17     if (dt1 == dt2) return dt1;
18     if (dt1 <= 0 || dt2 <= 0) return 0;
19     else return (dt1 - dt2) / Math.Log(dt1 / dt2);
```

```
  }
  //並流の場合
  else if (flowType == FlowType.ParallelFlow)
  {
    double dt1 = hotInletTemperature - coldInletTemperature;
    double dt2 = hotOutletTemperature - coldOutletTemperature;
    if (dt1 <= 0 || dt2 <= 0) return 0;
    return (dt1 - dt2) / Math.Log(dt1 / dt2);
  }
  else
  {
    double p = (hotInletTemperature - hotOutletTemperature) /
      (hotInletTemperature - coldInletTemperature);
    double q = (coldOutletTemperature - coldInletTemperature) /
      (hotInletTemperature - coldInletTemperature);
    double r = 0;

    //直交流（片側混合）の場合
    if (flowType == FlowType.CrossFlow_CminMixed ||
    flowType == FlowType.CrossFlow_CmaxMixed)
    {
      double bf = 1 - q / p * Math.Log(1 / (1 - p));
      if (bf <= 0) r = 0;
      else r = q / Math.Log(1 / bf);
    }
    //直交流（両側混合・非混合）の場合
    else if (flowType == FlowType.CrossFlow_BothFluidMixed ||
       flowType == FlowType.CrossFlow_BothFluidsUnmixed)
    {
      //r の上下限値を計算して解の範囲を特定
      double rMax;  //上限値（対向流の r）
      if (Math.Abs(p - q) < 1e-8) rMax = 1;
      else rMax = (p - q) / Math.Log((1 - q) / (1 - p));
      r = Math.Max(1e-4, 1 - (p + q));  //下限値（出口温度差）

      //極小値を黄金探索
      Minimization.MinimizeFunction mFnc = delegate (double x)
      {
        if (flowType == FlowType.CrossFlow_BothFluidMixed)
          return Math.Abs(eFncBothMixedR(p, q, x));
        else return Math.Abs(eFncBothUnMixedR(p, q, x));
      };
      if (1e-3 < Minimization.GoldenSection(ref r, rMax, mFnc)) r = 0;
    }
    return r * (hotInletTemperature - coldInletTemperature);
  }
  throw new Exception("Not implemented");
}

/// <summary>両流体混合の場合の誤差関数</summary>
/// <param name="p">p</param>
/// <param name="q">q</param>
/// <param name="r">r</param>
/// <returns>誤差</returns>
private static double eFncBothMixedR(double p, double q, double r)
{
  return (p / (1 - Math.Exp(-p / r)) + q / (1 - Math.Exp(-q / r)) - 1) - r;
}

/// <summary>両流体非混合の場合の誤差関数</summary>
/// <param name="p">p</param>
/// <param name="q">q</param>
/// <param name="r">r</param>
/// <returns>誤差</returns>
private static double eFncBothUnMixedR(double p, double q, double r)
{
  const int UVMAX = 50;
  double rSum = 1;
  double pr = p / r;
  double qr = q / r;
  for (int u = 1; u <= UVMAX; u++)
  {
    double dR = 0;
    double pqr = 1;
    for (int i = 1; i <= u; i++) pqr *= pr / (i + 1);
    if (u % 2 == 0) dR += pqr;
    else dR -= pqr;

    for (int v = 1; v <= u; v++)
    {
```

```
100       pqr *= qr * (u + v) / (v * (v + 1));
101       if ((u + v) % 2 == 0) dR += pqr;
102       else dR -= pqr;
103     }
104
105     for (int i = u ; 0 < i; i--)
106     {
107       pqr *= i * (i + 1) / (pr * (u + i));
108       if ((i + u) % 2 == 0) dR -= pqr;
109       else dR += pqr;
110     }
111
112     rSum += dR;
113     if (Math.Abs(dR) < 1e-6) return rSum - r;
114   }
115   return (1 - (p + q)) - r;
116 }
```

【例題 8.3】

図 8.2 に示した特性線図を作成せよ。

【解】

プログラム 8.3 に計算処理を示す。冷水側の出入口温度を 0 ℃と 1 ℃に固定して計算を行う。q と R にもとづいて温水出入口温度を計算する（28, 29 行）。プログラム 8.2 を用いて ΔT_m を計算し、これと ΔT_{LM} の比をとることで F_G を計算する。

プログラム 8.3　補正係数 F_G の計算

```
 1 private static void heatExchangeTest1()
 2 {
 3   int tci = 0;
 4   int tco = 1;
 5   double[] rFac = new double[] { 4.0, 3.0, 2.0, 1.5, 1.1, 0.8, 0.6, 0.4, 0.2 };
 6   HeatExchange.FlowType[] fTypes = new HeatExchange.FlowType[] {
 7     HeatExchange.FlowType.CrossFlow_BothFluidMixed,
 8     HeatExchange.FlowType.CrossFlow_BothFluidsUnmixed,
 9     HeatExchange.FlowType.CrossFlow_CminMixed };
10
11   using (StreamWriter sWriter =
12     new StreamWriter("hexTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
13   {
14     for (int i = 0; i < 9; i++) sWriter.Write("," + rFac[i]);
15     sWriter.WriteLine();
16
17     for (int i = 0; i < fTypes.Length; i++)
18     {
19       sWriter.WriteLine(fTypes[i].ToString());
20       for (int j = 0; j <= 100; j++)
21       {
22         double qq = 0.01 * j;
23         sWriter.Write(qq);
24         for (int k = 0; k < 9; k++)
25         {
26           if (qq == 0) sWriter.Write("," + 1);
27           else
28           {
29             double thi = (tco + tci * (qq - 1)) / qq;
30             double tho = thi - rFac[k] * (tco - tci);
31             double dt1 = thi - tco;
32             double dt2 = tho - tci;
33             if (dt1 <= 0 || dt2 <= 0) sWriter.Write("," + 0);
34             else
35             {
36               double lmtd = (dt1 - dt2) / Math.Log(dt1 / dt2);
37               double tm = HeatExchange.GetMeanTemperatureDifference(thi, tci, tho, tco, fTypes[i]);
38               sWriter.Write("," + tm / lmtd);
39             }
40           }
41         }
42         sWriter.WriteLine();
43       }
44     }
45   }
46 }
```

8.3.2　熱通過有効度の計算

熱通過有効度 ε を計算するプログラムを 8.4 に示す。与えられた流れのタイプに基き、式 8.21 から式 8.29 を用いて計算を行うプログラムである。

プログラム **8.4**　熱通過有効度の計算

Popolo.HVAC.HeatExchanger.HeatExchange class

```
/// <summary>熱通過有効度を計算する</summary>
/// <param name="ntu">移動単位数 NTU[-]</param>
/// <param name="heatCapacityRatio">熱容量流量比[-]</param>
/// <param name="flowType">流れタイプ</param>
/// <returns>熱通過有効度[-]</returns>
public static double GetEffectiveness(double ntu, double heatCapacityRatio, FlowType flowType)
{
  double rMC = heatCapacityRatio;

  //熱容量流量比が 0 の場合
  if (rMC <= 0) return 1 - Math.Exp(-ntu);

  //NTU が 0 の場合
  if (ntu <= 0) return 0;

  double eps;
  switch (flowType)
  {
    case FlowType.CounterFlow:
      if (rMC < 1) return (1 - Math.Exp((rMC - 1) * ntu)) / (1 - rMC * Math.Exp((rMC - 1) * ntu));
      else return ntu / (1 + ntu);

    case FlowType.ParallelFlow:
      if (rMC < 1) return (1 - Math.Exp(-ntu * (rMC + 1))) / (1 + rMC);
      else return 0.5 * (1 - Math.Exp(-2 * ntu));

    case FlowType.CrossFlow_BothFluidMixed:
      eps = ntu / (1 - Math.Exp(-ntu)) + rMC * ntu / (1 - Math.Exp(-rMC * ntu));
      return ntu / (eps - 1);

    case FlowType.CrossFlow_CminMixed:
      return 1 - Math.Exp((Math.Exp(-ntu * rMC) - 1) / rMC);

    case FlowType.CrossFlow_CmaxMixed:
      return (1 - Math.Exp((Math.Exp(-ntu) - 1) * rMC)) / rMC;

    case FlowType.CrossFlow_BothFluidsUnmixed:
      eps = (Math.Exp(-rMC * Math.Pow(ntu, 0.78)) - 1) / (rMC * Math.Pow(ntu, -0.22));
      return 1 - Math.Exp(eps);
  }
  return 0;
}
```

プログラム 8.5 は熱通過有効度 ε から移動単位数 *NTU* を計算するプログラムである。ニュートンラプソン法とプログラム 8.4 を用いて *NTU* について反復収束計算を行う。熱交換器の定格能力および出入口流体条件が既知である場合に *NTU* (および伝熱面積) を求める際には本プログラムを使用する。

プログラム **8.5**　**NTU** の計算

Popolo.HVAC.HeatExchanger.HeatExchange class

```
/// <summary>移動単位数 NTU[-]を計算する</summary>
/// <param name="effectiveness">熱通過有効度[-]</param>
/// <param name="heatCapacityRatio">熱容量流量比[-]</param>
/// <param name="flowType">流れのタイプ</param>
/// <returns>移動単位数 NTU[-]</returns>
public static double GetNTU(double effectiveness, double heatCapacityRatio, FlowType flowType)
{
  Roots.ErrorFunction eFnc = delegate (double ntu)
  { return effectiveness - GetEffectiveness(ntu, heatCapacityRatio, flowType); };
  return Roots.Newton(eFnc, 0, 1e-4, 1e-6, 1e-6, 20);
}
```

【例題 8.4】

図 8.4 に示した特性線図を作成せよ。

【解】

プログラム 8.6 に計算処理を示す。

プログラム **8.6**　熱通過有効度 ε 図表の作成

```
private static void heatExchangeTest2()
{
  using (StreamWriter sWriter =
  new StreamWriter("hexTest2.csv", false, Encoding.GetEncoding("Shift_JIS")))
  {
    double[] ntu = new double[]
```

```
 7     { 0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.5, 3, 3.5, 4, 4.5, 5, 6, 7 };
 8     double[] cmcm = new double[] { 0, 0.25, 0.5, 0.75, 1 };
 9
10     sWriter.WriteLine("NTU, 0.00, 0.25, 0.50, 0.75, 1.00");
11     foreach (HeatExchange.FlowType ft in Enum.GetValues
12       (typeof(HeatExchange.FlowType)))
13     {
14       sWriter.WriteLine(ft.ToString());
15       for (int i = 0; i < ntu.Length; i++)
16       {
17         sWriter.Write(ntu[i].ToString() + ",");
18         for (int j = 0; j < cmcm.Length; j++)
19           sWriter.Write(HeatExchange.GetEffectiveness(ntu[i], cmcm[j], ft) + ",");
20         sWriter.WriteLine();
21       }
22     }
23   }
24 }
```

【例題 8.5】

熱通過有効度 ε を用いて例題 8.1 の熱交換器と同一条件において同一の熱交換能力を持つ直交流（両流体非混合）の熱交換器の伝熱面積を求めるプログラムを作成せよ。

【解】

プログラム 8.7 に伝熱面積の計算処理を示す。出入口条件および熱交換能力は既知であるため、熱通過有効度 ε は式 8.19 を用いて計算できる（12 行）。次に、プログラム 8.5 を用いることで熱通過有効度 ε から NTU を求め、式 8.21 に基いて伝熱面積を計算する。

プログラム 8.7　伝熱面積の計算

```
 1 private static void heatExchangeTest3()
 2 {
 3   double heatTransfer = 500;
 4   double heatTransferCoefficient = 4;
 5   double hotInletTemperature = 12;
 6   double coldInletTemperature = 6;
 7   //熱容量流量[kW/K]
 8   double mh = 500 / (12 - 7);
 9   double mc = 500 / (10 - 6);
10
11   //熱通過有効度の計算
12   double epsilon = heatTransfer / Math.Min(mh, mc) / (hotInletTemperature - coldInletTemperature);
13
14   //NTU の計算
15   double rmc = Math.Min(mh, mc) / Math.Max(mh, mc);
16   double ntu = HeatExchanger.GetNTU(epsilon, rmc, HeatExchanger.FlowType.CrossFlow_BothFluidsUnmixed);
17
18   //伝熱面積の計算
19   double surfaceArea = ntu / heatTransferCoefficient * Math.Min(mh, mc);
20   Console.WriteLine("伝熱面積=" + surfaceArea.ToString("F0") + "m2");
21 }
```

【例題 8.6】

例題 8.1 の熱交換器について、同一の伝熱面積、熱通過率、流量、流体入口温度の条件のもと、流体の流れが直交流（両流体非混合）であった場合の熱交換量を求めるプログラムを作成せよ。

【解】

プログラム 8.8 に熱交換量計算処理を示す。11~25 行が平均温度差 ΔT_m を用いた方法、27~32 行が熱通過有効度 ε を用いた方法である。平均温度差 ΔT_m を用いる場合には収束計算が必要となる。同一条件のもとでは、直交流の場合の交換熱量は向流に比較して小さくなるはずであるから、2.3 節で説明した二分法を用いて、500kW 未満に範囲を指定して求根する。熱通過有効度 ε を用いる場合には収束計算は不要であり、計算は簡便である。本プログラムを実行すると出力として 460kW 程度が得られ、向流の場合（500kW）に比較すると 9 割程度の交換熱量となることが確認できる。

プログラム 8.8　熱交換器の熱交換量の計算

```
 1 private static void heatExchangeTest4()
 2 {
 3     double surfaceArea = 89;
 4     double heatTransferCoefficient = 4;
 5     double hotInletTemperature = 12;
 6     double coldInletTemperature = 6;
 7     //熱容量流量[kW/K]
 8     double mh = 500 / (12 - 7);
 9     double mc = 500 / (10 - 6);
10
```

```
11     //二分法を用いて求根
12     Popolo.Numerics.Roots.ErrorFunction eFnc = delegate(double heatTransfer)
13     {
14         //出口温度[C]を計算
15         double coldOutletTemperature = coldInletTemperature + heatTransfer / mc;
16         double hotOutletTemperature = hotInletTemperature - heatTransfer / mh;
17         //平均温度差[K]を計算
18         double tm = HeatExchanger.GetMeanTemperatureDifference
19           (hotInletTemperature, coldInletTemperature, hotOutletTemperature,
20           coldOutletTemperature, HeatExchanger.FlowType.CrossFlow_BothFluidsUnmixed);
21         //誤差を評価
22         return tm * surfaceArea * heatTransferCoefficient - heatTransfer;
23     };
24     double ht1 = Popolo.Numerics.Roots.Bisection(1, 500, 0.001, eFnc);
25     Console.WriteLine("熱交換量=" + ht1.ToString("F0") + "kW");
26
27     double rmc = Math.Min(mh, mc) / Math.Max(mh, mc);
28     double ntu = 4d * 89d / Math.Min(mh, mc);
29     double epsilon = HeatExchanger.GetEffectiveness
30         (ntu, rmc, HeatExchanger.FlowType.CrossFlow_BothFluidsUnmixed);
31     double ht2 = epsilon * Math.Min(mh, mc) * (hotInletTemperature - coldInletTemperature);
32     Console.WriteLine("熱交換量=" + ht2.ToString("F0") + "kW");
33 }
```

8.3.3 「プレート熱交換器クラス」の作成

前節までに開発した熱交換に関するメソッドを用いて、最も基本的な熱交換器であるプレート熱交換器のクラスを作成する。

通常、プレート熱交換器は図 8.5 に示す構成で制御を行う。T_R [K]は還温度であり、熱交換によってこれを往温度 T_S [K]まで加熱または冷却する。往温度を設定温度に合わせるためには熱源水 T_{HS} [K]の流量を変化させる。図 8.5 に示すようにバイパス制御としても良いし、ポンプの回転数制御としても良い。モデルには、所定の往温度にするために必要な熱源水流量を計算する処理を実装させる。

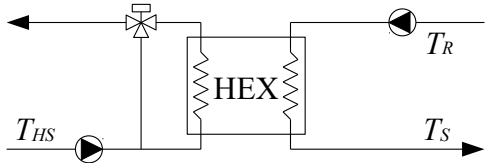


図 8.5　プレート熱交換器の往温度制御

プログラム 8.9 にプレート熱交換器クラスのプロパティおよびコンストラクタを示す。往温度を制御するためには熱源水流量を制御するが、無限大に増加させることはできないため、8 行に示すように上限流量を設ける。コンストラクタの引数は熱源水と供給水のそれぞれの入口温度と流量、熱交換量である。50 行で加熱か冷却かを判定し、51~64 行で伝熱係数を初期化してプロパティに保存する。

プログラム **8.9**　プレート熱交換器クラスのプロパティおよびコンストラクタ

Popolo.HVAC.HeatExchanger.PlateHeatExchanger class

```
 1 /// <summary>過負荷か否か</summary>
 2 public bool IsOverLoad { get; private set; }
 3
 4 /// <summary>伝熱係数[kW/K]を取得する</summary>
 5 public double HeatTransferCoefficient { get; private set; }
 6
 7 /// <summary>熱源水流量上限値[kg/s]を取得する</summary>
 8 public double MaxHeatSourceFlowRate { get; private set; }
 9
10 /// <summary>供給流量上限値[kg/s]を取得する</summary>
11 public double MaxSupplyFlowRate { get; private set; }
12
13 /// <summary>熱源水流量[kg/s]を取得する</summary>
14 public double HeatSourceFlowRate { get; private set; }
15
16 /// <summary>供給流量[kg/s]を取得する</summary>
17 public double SupplyFlowRate { get; private set; }
18
19 /// <summary>熱源水入口温度[C]を取得する</summary>
20 public double HeatSourceInletTemperature { get; private set; }
21
```

```
22 /// <summary>熱源水出口温度[C]を取得する</summary>
23 public double HeatSourceOutletTemperature { get; private set; }
24
25 /// <summary>供給温度[C]を取得する</summary>
26 public double SupplyTemperature { get; private set; }
27
28 /// <summary>還温度[C]を取得する</summary>
29 public double ReturnTemperature { get; private set; }
30
31 /// <summary>供給温度設定値[C]を設定・取得する</summary>
32 public double SupplyTemperatureSetpoint { get; set; }
33
34 /// <summary>熱交換量[kW]を取得する</summary>
35 public double HeatTransfer { get; private set; }
36
37 /// <summary>インスタンスを初期化する</summary>
38 /// <param name="heatTransfer">熱交換量[kW]</param>
39 /// <param name="heatsourceTemperature">熱源水温度[C]</param>
40 /// <param name="heatsourceFlowRate">熱源水流量[kg/s]</param>
41 /// <param name="supplyTemperature">供給温度[C]</param>
42 /// <param name="supplyFlowRate">供給流量[kg/s]</param>
43 public PlateHeatExchanger(double heatTransfer,
44   double heatsourceTemperature, double heatsourceFlowRate, double supplyTemperature, double supplyFlowRate)
45 {
46   MaxHeatSourceFlowRate = heatsourceFlowRate;
47   MaxSupplyFlowRate = supplyFlowRate;
48   HeatSourceInletTemperature = heatsourceTemperature;
49
50   bool isHeating = supplyTemperature < heatsourceTemperature;
51   if (isHeating)
52   {
53     ReturnTemperature = supplyTemperature - heatTransfer / (WATER_SPECIFIC_HEAT * supplyFlowRate);
54     HeatTransferCoefficient = HeatExchange.GetHeatTransferCoefficient
55   (heatsourceTemperature, ReturnTemperature, heatsourceFlowRate * WATER_SPECIFIC_HEAT,
56   supplyFlowRate * WATER_SPECIFIC_HEAT, heatTransfer, HeatExchange.FlowType.CounterFlow);
57   }
58   else
59   {
60     ReturnTemperature = supplyTemperature + heatTransfer / (WATER_SPECIFIC_HEAT * supplyFlowRate);
61     HeatTransferCoefficient = HeatExchange.GetHeatTransferCoefficient
62   (ReturnTemperature, heatsourceTemperature, supplyFlowRate * WATER_SPECIFIC_HEAT,
63   heatsourceFlowRate * WATER_SPECIFIC_HEAT, heatTransfer, HeatExchange.FlowType.CounterFlow);
64   }
65   ShutOff();
66 }
```

プログラム 8.10 にプレート熱交換器の状態更新と停止処理を示す。1~69 行が状態更新処理であり、プロパティで指定する往温度設定値に合致するように熱源水流量を制御する。13 行で加熱運転か冷却運転かを判定し、そもそも温度が逆転して制御不能の場合には 17~22 行で停止・終了する。26~47 行は加熱運転時の処理、48~69 行は冷却運転時の処理である。いずれも処理は同様であり、まず最大水量での熱交換量を求め（29~32 行）、必要な熱交換量を超えるか否かを確かめる。最大水量でも熱交換量が確保できない場合には過負荷であるため、熱源水量に最大値を設定する。熱交換可能な場合には、収束計算で熱源水量を推定する（38~45 行）。71~74 行で往温度と熱交換量をプロパティに保存する。

プログラム 8.10　プレート熱交換器の状態更新と停止処理

Popolo.HVAC.HeatExchanger.PlateHeatExchanger class

```
 1 /// <summary>供給温度[C]を制御する</summary>
 2 /// <param name="heatsourceTemperature">熱源水入口温度[C]</param>
 3 /// <param name="returnTemperature">還水温度[C]</param>
 4 /// <param name="supplyFlowRate">供給流量[kg/s]</param>
 5 public void ControlSupplyTemperature
 6   (double heatsourceTemperature, double returnTemperature, double supplyFlowRate)
 7 {
 8   HeatSourceInletTemperature = heatsourceTemperature;
 9   ReturnTemperature = returnTemperature;
10   SupplyFlowRate = supplyFlowRate;
11
12   //加熱冷却逆転判定
13   bool isHeating = returnTemperature < SupplyTemperatureSetpoint;
14   bool isRev = (heatsourceTemperature < returnTemperature && isHeating) ||
```

```
    (returnTemperature < heatsourceTemperature && !isHeating);

  //流量0以下または加熱冷却逆転の場合には機器を停止
  if (supplyFlowRate <= 0 || isRev)
  {
    ShutOff();
    return;
  }

  double mcSply = supplyFlowRate * WATER_SPECIFIC_HEAT;
  double ql = Math.Abs(returnTemperature - SupplyTemperatureSetpoint) * mcSply;
  //加熱運転
  if (isHeating)
  {
    //最大流量での熱交換量計算
    HeatTransfer = HeatExchange.GetHeatTransfer
      (heatsourceTemperature, returnTemperature, MaxHeatSourceFlowRate * WATER_SPECIFIC_HEAT, mcSply,
      HeatTransferCoefficient, HeatExchange.FlowType.CounterFlow);
    IsOverLoad = HeatTransfer < ql;
    //過負荷判定
    if (IsOverLoad) HeatSourceFlowRate = MaxHeatSourceFlowRate;
    else
    {
      Roots.ErrorFunction eFnc = delegate (double flow) {
        HeatTransfer = HeatExchange.GetHeatTransfer
        (heatsourceTemperature, returnTemperature, flow * WATER_SPECIFIC_HEAT, mcSply,
        HeatTransferCoefficient, HeatExchange.FlowType.CounterFlow);
        return HeatTransfer - ql;
      };
      HeatSourceFlowRate = Roots.Bisection
        (0, MaxHeatSourceFlowRate, -ql, HeatTransfer - ql, 0, MaxHeatSourceFlowRate * 0.001, eFnc, 20);
    }
  }
  //冷却運転
  else
  {
    //最大流量での熱交換量計算
    HeatTransfer = HeatExchange.GetHeatTransfer
      (returnTemperature, heatsourceTemperature, mcSply, MaxHeatSourceFlowRate * WATER_SPECIFIC_HEAT,
      HeatTransferCoefficient, HeatExchange.FlowType.CounterFlow);
    IsOverLoad = HeatTransfer < ql;
    //過負荷判定
    if (IsOverLoad) HeatSourceFlowRate = MaxHeatSourceFlowRate;
    else
    {
      Roots.ErrorFunction eFnc = delegate (double flow) {
        HeatTransfer = HeatExchange.GetHeatTransfer
        (returnTemperature, heatsourceTemperature, mcSply, flow * WATER_SPECIFIC_HEAT,
        HeatTransferCoefficient, HeatExchange.FlowType.CounterFlow);
        return HeatTransfer - ql;
      };
      HeatSourceFlowRate = Roots.Bisection
        (0, MaxHeatSourceFlowRate, -ql, HeatTransfer - ql, 0, MaxHeatSourceFlowRate * 0.001, eFnc, 20);
    }
  }

  //出口水温計算
  SupplyTemperature = ReturnTemperature + HeatTransfer / mcSply;
  HeatSourceOutletTemperature = HeatSourceInletTemperature
    - HeatTransfer / (HeatSourceFlowRate * WATER_SPECIFIC_HEAT);
}

/// <summary>停止させる</summary>
public void ShutOff()
{
  HeatTransfer = 0;
  SupplyTemperature = ReturnTemperature;
  HeatSourceOutletTemperature = HeatSourceInletTemperature;
  SupplyFlowRate = HeatSourceFlowRate = 0;
}
```

【例題 8.7】

例題 8.1 の熱交換器を作成し、供給流量が変化した場合に熱源水所要流量の変化を計算せよ。

【解】

プログラム 8.11 に計算処理を示す。3 行でプレート熱交換器のインスタンスを生成する。5~15 行が冷却運転のテストであり、負荷率（供給流量比）を 5 %刻みとし、冷却水還温度を 10~20 ºC の範囲で揺らす。同様に 17~27 行では加熱運転のテストを行う。結果をグラフ化すると図 8.6 が得られる。

プログラム 8.11　供給流量変化に伴うプレート熱交換器熱源水所要流量変化

```
 1 private static void plateHeatExchangerTest()
 2 {
 3   PlateHeatExchanger pHex = new PlateHeatExchanger(500, 6, 1792d / 60, 7, 1433d / 60);
 4
 5   Console.WriteLine("冷却テスト");
 6   pHex.SupplyTemperatureSetpoint = 7;
 7   for (int trw = 10; trw <= 20; trw += 2)
 8   {
 9     for (int i = 0; i <= 20; i++)
10     {
11       pHex.ControlSupplyTemperature(6, trw, 1433d / 60 * (0.05 * i));
12       Console.Write(pHex.HeatSourceFlowRate.ToString("F2") + ", ");
13     }
14     Console.WriteLine();
15   }
16
17   Console.WriteLine("加熱テスト");
18   pHex.SupplyTemperatureSetpoint = 45;
19   for (int trw = 30; trw <= 40; trw += 2)
20   {
21     for (int i = 0; i <= 20; i++)
22     {
23       pHex.ControlSupplyTemperature(50, trw, 1433d / 60 * (0.05 * i));
24       Console.Write(pHex.HeatSourceFlowRate.ToString("F2") + ", ");
25     }
26     Console.WriteLine();
27   }
28 }
```

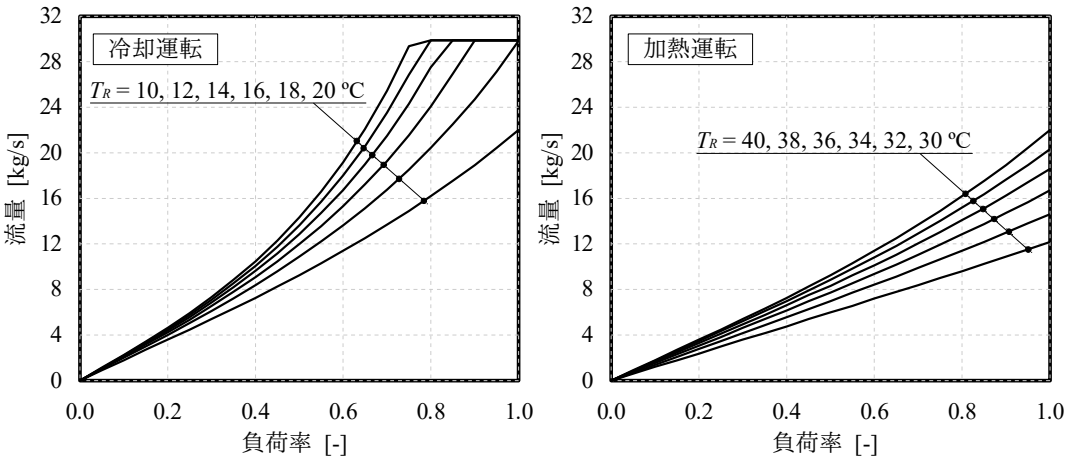


図 8.6　プレート熱交換器の負荷と所要流量の関係

【第 8 章 記号表】

A	：伝熱面積 **[m²]**	Q	：熱流 **[W]**
c	：比熱 **[J/(kg·K)]**	R_{mc}	：熱容量流量比 **[-]**
F_G	：補正係数 **[-]**	T	：絶対温度 **[K]**
K	：熱通過率 **[W/(m²·K)]**	ε	：熱通過有効度 **[-]**
m	：質量流量 **[kg/s]**	ΔT_{LM}	：対数平均温度差 **[°C]**
mc	：熱容量流量 **[W/K]**	ΔT_m	：平均温度差 **[°C]**
NTU	：移動単位数 **[-]**	η	：温度効率 **[-]**
sub scripts			
c	：低温側流体	i	：入口側
h	：高温側流体	o	：出口側

【第 8 章 参考文献】

8.1） Kays, W. M. and London, A.L., Compact Heat Exchangers, 2nd ed., McGraw-Hill, New York, 1964.

8.2） Bowman, R. A., Mueller, A. C., and Nagle, W. M., Mean temperature difference, Trans. ASME. 62, pp.283~294, 1940

8.3） Mason, J. L.: Heat Transfer in Cross-Flow, Proc. Appl. Mechanics, 2nd U.S. Nat. Congress, p.801, 1954

8.4） W. Nusselt, Eine neue Formel für den Wärmedurchgangim Kreuzstrom, Technische Mechanik und Thermodynamik, pp.417~422, 1930

8.5） 伝熱工学ハンドブック JSME Heat Transfer Handbook: 日本機械学会, 丸善出版株式会社, 1993

8.6） 伝熱工学資料 JSME Data Book, Heat Transfer 5th Edition: 日本機械学会, 丸善出版株式会社, 2009

8.7） JSME テキストシリーズ 伝熱工学: 日本機械学会, 丸善出版株式会社, 2005

第 II 編　エネルギー 評価編

第 II 編では、熱源設備、冷却塔、流体機械など、主に設備システムのエネルギー消費に大きな影響を持つ機器類について解説する。熱環境システムは熱交換の連続であり、第 9 章~第 12 章では水、空気、冷媒の間で熱交換を行う具体的な機器を取り上げる。いずれも第 8 章の理論が基礎になる。第 13 章~第 15 章は熱源機器であり、ボイラ、圧縮式冷凍機、吸収式冷凍機を取り上げる。単純な熱交換だけではなく外部から投入される電力やガスなどを含めたエネルギー収支を解く必要があり、それまでの章よりも難易度はやや高い。本書では、いずれの熱源機器も簡単な特性式によるモデルだけではなく理論式にもとづくモデルも示し、機器内部の物理的な挙動を学習できるようにした。熱源機器に流入する水の温度や流量などの変化により、その効率が定性的にどのような影響を受けるのかについて判断できるようになれば、一段階高いレベルの設計が可能となるためである。熱交換を行う熱媒を搬送するためのエネルギーに関しては第 16 章と第 17 章で解説する。第 16 章は主に期間的なエネルギーシミュレーションに用いられることの多い解法であり、第 17 章は制御系のシミュレーションにも対応可能な詳細な解法である。第 18 章と第 19 章では特殊設備の中では採用事例の多い太陽エネルギー設備と蓄熱槽を取り上げる。第 20 章では算出されたエネルギー消費量を経済性という観点から評価する方法について解説する。第 21 章では、熱源設備、冷却塔、ポンプを連成させて年間のエネルギー消費量を計算することで熱源設備システム全体の評価を行う。

第9章　プレートフィン付管熱交換器
(Tube-in-fin Heat Exchanger)

9.1　概要

空調設備システムにおいては、液体と空気の熱交換にはプレートフィン付管熱交換器（コイル）が使われることが多い。空気調和機の内部には通常、冷温水コイルと温水コイルが設置されている。また、ファンコイルユニットは冷温水コイルとファンがユニット化された機器である。近年、採用事例が増加している個別分散型空調システムにおいても、室内機には冷媒と空気の熱交換を行うための直膨コイルが組み込まれている。従って、プレートフィン付管熱交換器の計算は、二次側空調システムを検討するための基本的な技術である。

本章では、加熱コイルと冷却除湿コイルのそれぞれについて計算法を示す。特に冷却除湿コイルは顕熱移動に加えて潜熱移動も生じるため、第 8 章で解説した単純な熱交換器の理論式をそのまま適用することはできない。熱と水分の両方を睨みながら計算を行う必要がある。冷却除湿コイルは空調設計の醍醐味である湿り空気の調整を可能とする機器である。通常はコイルを通過する空気の出口乾球温度が所定の値となるように水量を制御するため、入口条件を設定した上での出口状態の成り行き計算に加え、出口条件を設定した上での所要水量の計算法についても説明する。

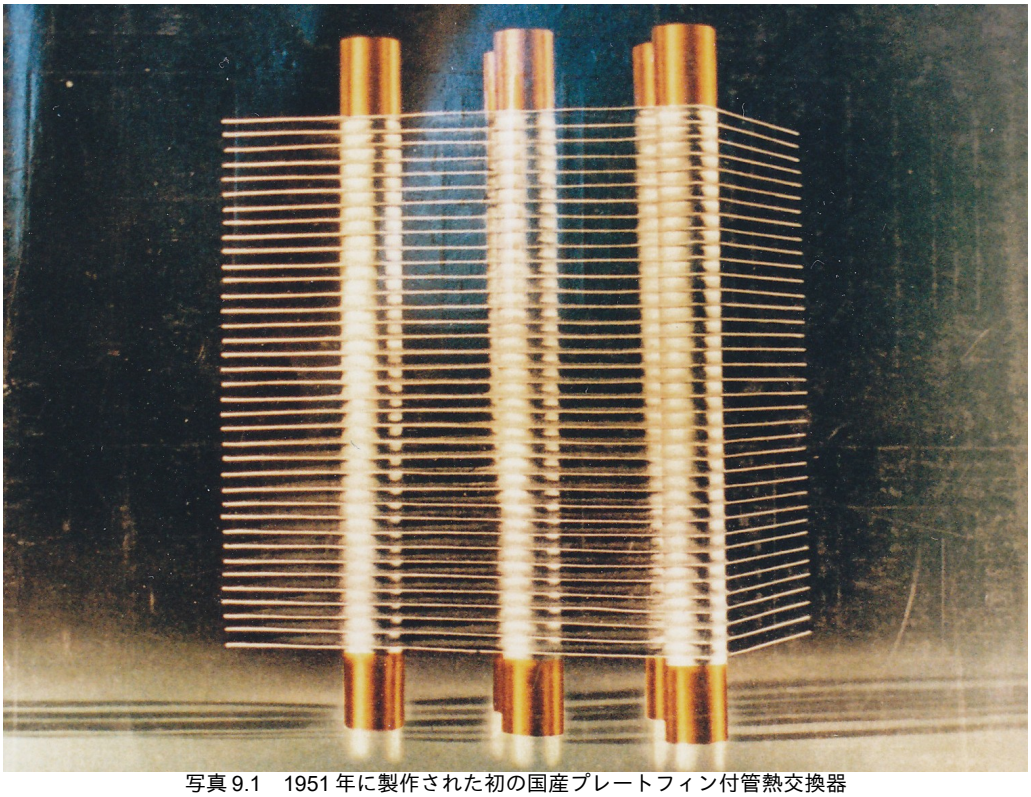


写真 9.1　1951 年に製作された初の国産プレートフィン付管熱交換器

（新晃工業株式会社 50 年史[9.1)]より）

9.2　理論

1）拡大伝熱面

熱交換器で液体と空気の熱交換を行うとする。界壁を通じて熱移動を行うことになるが、液体の対流熱伝達率は数千 W/(m²·K)である一方、空気の対流熱伝達率は精々、数十 W/(m²·K)である[†1)]。即ち、熱交換器全体としての熱抵抗は空気側の対流熱伝達率によって支配されていることになる。従って、熱交換器の性能を上昇させるためには、空気が界壁に接触する面積を拡大させて空気側の熱抵抗を下げることが有効である。これを拡大伝熱面（フィン）と呼び、プレートフィン付管熱交換器の場合には、薄いプレート状のフィンを液体が流れる管の外表面に取り付ける。拡大伝熱面の効果の例を図 9.1 に示す。図の右に記載の水および空気の条件にもとづいて熱伝達率を計算すると水側が 3,000 W/(m²·K)、空気側が 6 W/(m²·K)となり、空気側が著しく小さい。図の左側は拡大伝熱面がない場合の熱流を示しているが、熱貫流率は 5.98 A W/K であり、空気側の小さな熱伝達率に引きづられていることがわかる。図の右側は拡大伝熱面がある場合の熱流であり、空気側の表面積が水側の 100 倍となる。この時の熱貫流率は 500 A W/K であり、拡大伝熱面がない場合に比較して約 80 倍に増加する。

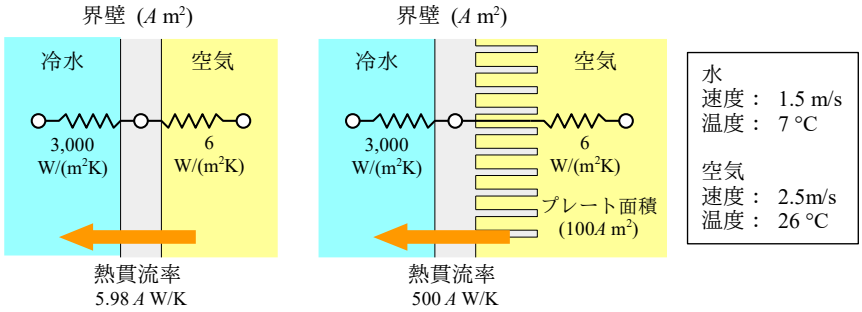


図 9.1　拡大伝熱面の効果

2）流体の流れ

プレートフィンは配管に垂直に取り付けられるため、空気と水は直交方向に流れることになる。しかし、通常は配管は図 9.2 の左に示すような形で折り返されて複数の列（図中の n は列数）で構成される[†2)]。従って、巨視的には空気と逆流する方向に水が流れるため、対向流型の熱交換器に近い挙動を示す。このような折り返しのある熱交換器の熱通過有効度 ε [-]は式 9.1 で計算する[†3)]。また、ε_p [-]は各列での直交流の熱交換器としての熱通過有効度である（計算法は第 8 章で記した）。ただし、ε_p の計算にあたっては入力値として NTU ではなく、NTU を列数で除した NTU / n を与える。

$$\varepsilon = \frac{n\varepsilon_p}{1+(n-1)\varepsilon_p} \tag{9.1}$$

式 9.1 を用いて列数に応じて NTU と熱通過有効度 ε の関係を計算すると図 9.2 右の通りとなる。列数 n の数を増加させるにつれて対向流の熱交換器の特性に漸近していく様子が確認できる。空調用のコイルは通常は列数が 4 以上あり、計算も簡便になるため、対向流の熱交換器とみなして計算を行うことが多い。Elmahdy と Mitalas は 4 列以上のコイルであればこのような仮定を行っても十分な精度

†1　100 °C の温泉には入れないが、100 °C のサウナには入れる。空気は水に比較して対流熱伝達率が遥かに小さいためである。

†2　プレートフィン付管熱交換器は通常「コイル」と呼ばれる。とぐろを巻いた蛇を英語で coiled snake と言い、配管がとぐろ状に配置されていることが語源のようである。

†3　式 9.1 は熱容量流量比 $R = 1$ の場合に成立する。熱容量流量比が異なる場合に関しては参考文献 9.15 を参照。結論はあまり変わらない。

が得られるとしている[9.5]。

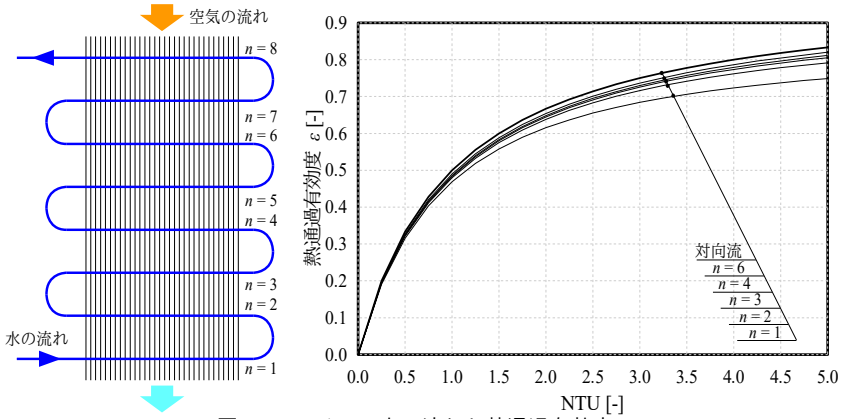


図 9.2　コイルの水の流れと熱通過有効度 ε

9.2.1　加熱コイル

温水と空気で熱交換を行い、空気温度を上げるプレートフィン付管熱交換器を加熱コイルと呼ぶ。後述の冷却除湿コイルとは異なり、熱移動の過程で湿り空気の湿度がかわらないため、熱の移動は単純である。向流型の熱交換器とみなし、第 8 章で解説した熱通過有効度 ε を利用して式 9.2 で熱交換量 Q [kW]を計算する。ただし t_{wi} と t_{ai} はそれぞれ温水と空気の入口温度 [°C]であり、mc_{min} [kW/K]は温水または空気の熱容量流量である。

$$Q = \varepsilon\, mc_{min}\left(t_{wi} - t_{ai}\right) \tag{9.2}$$

【例題 9.1】

定格条件において下記の能力を示す加熱コイルについて伝熱面積を計算せよ。また水量を定格値に比較して半分に低下させた場合の出口空気温度および出口水温を計算せよ。

風量：40,000CMH、水量：497 L/min、入口水温：50 ºC、入口空気温度：22 ºC、

入口絶対湿度：0.01 kg/kg、熱交換量：173.3 kW、熱貫流率：50 W/(m^2·K)

【解】

式 4.7 により湿り空気の比熱 c_{pma} は、

c_{pma} = 1.006 + 1.805 × 0.01 = 1.023 kJ/(kg·K)

である。従って、空気側の熱容量流量 mc_a は、

mc_a = 40,000 m^3/h / 3600 s/h × 1.023 kJ/(kg·K) ×1.2 kg/m^3 = 13.6 kW/K

となる。同様に水側の熱容量流量 mc_w は、

mc_w = 497 L/min / 60 s/min × 4.186 kJ/(kg·K) × 1 kg/L = 34.7 kW/K

である。式 9.2 を熱通過有効度 ε について解き、

ε = 173.3 kW / 13.6 kW/K / (50 ºC – 22 ºC) = 0.454

である。向流型の熱交換器とみなすため、図 8.4 から NTU = 0.67 である。式 8.21 により伝熱面積 S は、

S = 0.67 × 13.6 kW/K ÷ 0.050 kW/(m^2·K) = 183 m^2

と求められる。

水量を半分にした場合の熱容量流量 mc_w は、

mc_w = 248 L/min / 60 s/min × 4.186 kJ/(kg·K) × 1 kg/L = 17.3 kW/K

となる。式 8.21 により NTU は、

NTU = 183 m^2 ÷ 13.6 kW/K × 0.05 kW/(m^2·K) = 0.67

である。図 8.4 を参照して熱通過有効度 ε は 0.418 である。従って、式 9.2 により交換熱量 Q_a は、

Q_a = 0.418 × 13.6 kW/K × (50 ºC – 22 ºC) = 159.1 kW

である。出口水温 t_{wo} と出口空気温度 t_{ao} はそれぞれ、

$t_{w,o}$ = 50 ºC – 159.1 ÷ 17.3 kW/K = 41.3 ºC

$t_{a,o}$ = 22 ºC + 159.1 ÷ 13.6 kW/K = 33.6 ºC

となる。

9.2.2　冷却除湿コイル

冷水と空気で熱交換を行い、空気の温湿度を下げるプレートフィン付管熱交換器を冷却除湿コイルと呼ぶ。空気を露点温度以下に冷却した場合には結露が生じるため、加熱コイルとは異なり熱通過有効度を用いた単純な計算はできない。そこで、結露が生じてコイル表面が濡れている領域（湿りコイル）とそうではない領域（乾きコイル）とに分けて計算を行う。乾き領域、湿り領域、乾湿境界のそれぞれに添字 d（dry）、w（wet）、b（border）をつけて記号表現することとする。

温度 t_{ai} で流入した空気は、まず乾きコイル部分で顕熱冷却されて t_{ab} [°C]となり、その後、湿りコイル部分で冷却除湿されることで t_{ao} [°C]となる。同様に t_{wi} で流入した水は湿りコイル部分で加熱されて t_{wb} [°C]となり、その後、乾きコイル部分でさらに加熱されて t_{wo} [°C]となる。湿りコイルに入った空気の一部はコイルに接触することなく通過（バイパス）する。このバイパス比率をバイパスファクターと呼ぶ。バイパス効果により、湿りコイル出口の空気は相対湿度 100 %には到達せず、ある相対湿度 ϕ_w [%]の線上にのる[†1]。湿り空気線図上で表現すると図 9.3 のとおりである。ϕ_w はコイルの列数や入口空気条件によって変化するが、計算上は 90~95 %で一定値を取るとみなすことが多い。

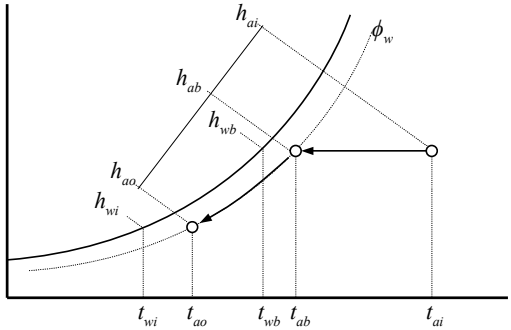


図 9.3　冷却除湿コイル内での湿り空気状態

1）乾きコイル

結露が発生しない乾きコイルに関しては、対数平均温度差 Δt_{LM} [°C]を用いて式 9.3~9.5が成立する。ここで m_a と c_{pma} は湿り空気の質量流量[kg/s]と定圧比熱[kJ/(kg·K)]、m_w と c_{pw} は水の質量流量[kg/s]と定圧比熱[kJ/(kg·K)]である。c_{pma} は絶対湿度によって変化するが、乾きコイルでは絶対湿度一定のため、一定値をとる。熱貫流率 K_d [kW/(m²·K)]および伝熱面積 S_d [m²]の計算については後述する。

$$Q_d = m_a c_{pma}\left(t_{ai} - t_{ab}\right) \tag{9.3}$$

$$Q_d = m_w c_{pw}\left(t_{wo} - t_{wb}\right) \tag{9.4}$$

$$Q_d = K_d S_d \Delta t_{LM} \tag{9.5}$$

$$\Delta t_{LM} = \frac{\left(t_{ai} - t_{wo}\right) - \left(t_{ab} - t_{wb}\right)}{\ln\left\{\left(t_{ai} - t_{wo}\right) / \left(t_{ab} - t_{wb}\right)\right\}} \qquad \text{ただし、}$$

式 9.3~9.5を行列で表現すると式 9.6 となる。3 行 3 列の行列であるため、解析的に解くことが可能であり、式 9.7 となる。

†1　湿り空気線図上に線を引くとわかるが、厳密にはバイパスファクターが一定としても相対湿度は一定にならない。モデルの説明をする場合にはバイパスファクターを一定としたのか、相対湿度を一定としたのか、明確に表現を分ける必要がある。

$$\begin{bmatrix} 1 & 0 & X_d \\ 0 & 1 & Y_d \\ Z_d & 1 & 0 \end{bmatrix} \begin{bmatrix} t_{ab} \\ t_{wo} \\ Q_d \end{bmatrix} = \begin{bmatrix} t_{ai} \\ t_{wb} \\ t_{ai}+Z_d t_{wb} \end{bmatrix} \tag{9.6}$$

$$\begin{bmatrix} t_{ab} \\ t_{wo} \\ Q_d \end{bmatrix} = \frac{1}{W_d} \begin{bmatrix} Y_d & -X_d & X_d \\ -Y_d Z_d & X_d Z_d & Y_d \\ Z_d & 1 & -1 \end{bmatrix} \begin{bmatrix} t_{ai} \\ t_{wb} \\ t_{ai}+Z_d t_{wb} \end{bmatrix} \tag{9.7}$$

ただし、 $X_d = 1/m_a c_{pma}$ 、 $Y_d = -1/m_w c_{pw}$
$Z_d = \exp[K_d S_d (X_d + Y_d)]$ 、 $W_d = Z_d X_d + Y_d$

従って、乾き湿り境界部分での空気温度 t_{ab} と冷水の出口温度 t_{wo} はそれぞれ t_{ai} と t_{wb} を用いて式 9.8 と 9.9 で表現できる。

$$t_{ab} = t_{ai} - V_1 (t_{ai} - t_{wb}) \tag{9.8}$$

$$t_{wo} = t_{ai} - V_2 (t_{ai} - t_{wb}) \tag{9.9}$$

ただし、 $V_1 = X_d (Z_d - 1)/W_d$ 、 $V_2 = Z_d (X_d + Y_d)/W_d$

2）湿りコイル

湿りコイルに関しても同様に、対数平均エンタルピー差を Δh_{LM} [kJ/kg]用いて式 9.10~9.12が成立する[†1)]。ただし、h_{wb} と h_{wo} は乾湿境界とコイル出口における水温と等しい温度の飽和空気のエンタルピー [kJ/kg]であり、式 9.13 で近似する（図 9.3 を参照）。厳密には係数 a および b は水温の関数であるが、入口水温 t_{wi} および入口水温と等温の空気温度の飽和エンタルピー h_{Swi} [kJ/kg]にもとづいて値を求め、一定値と扱う。大温度差空調であっても冷却コイル前後の冷水温度差は精々 10 °C 程度であり、このような仮定による誤差は大きくはない。

$$Q_w = m_a (h_{ab} - h_{ao}) \tag{9.10}$$

$$Q_w = \frac{m_w c_{pw}}{a} (h_{wb} - h_{wi}) \tag{9.11}$$

$$Q_w = K_w S_w \Delta h_{LM} \tag{9.12}$$

$$\Delta h_{LM} = \frac{(h_{ab} - h_{wb}) - (h_{ao} - h_{wi})}{\ln\{(h_{ab} - h_{wb})/(h_{ao} - h_{wi})\}} \qquad \text{ただし、}$$

$$h_w = a t_w + b \tag{9.13}$$

ただし、 $a = \frac{dh_{Swi}}{dt_{wi}}$ 、 $b = h_{Swi} - a t_{wi}$

乾きコイルと同様に行列で表現し、解を求めると式 9.14~9.17となる。なお、式 9.17 は後々の式変形がしやすいように、式 9.13 を用いて左辺を乾湿境界水温 t_{wb} としたものである。

$$\begin{bmatrix} 1 & 0 & X_w \\ 0 & 1 & Y_w \\ Z_w & 1 & 0 \end{bmatrix} \begin{bmatrix} h_{ao} \\ h_{wb} \\ Q_w \end{bmatrix} = \begin{bmatrix} h_{ab} \\ h_{wi} \\ h_{ab}+Z_w h_{wi} \end{bmatrix} \tag{9.14}$$

$$\begin{bmatrix} h_{ao} \\ h_{wb} \\ Q_w \end{bmatrix} = \frac{1}{W_w} \begin{bmatrix} Y_w & -X_w & X_w \\ -Y_w Z_w & X_w Z_w & Y_w \\ Z_w & 1 & -1 \end{bmatrix} \begin{bmatrix} h_{ab} \\ h_{wi} \\ h_{ab}+Z_w h_{wi} \end{bmatrix} \tag{9.15}$$

ただし、 $X_w = 1/m_a$ 、 $Y_w = -a/m_w c_{pw}$
$Z_w = \exp[K_w S_w (X_w + Y_w)]$ 、 $W_w = Z_w X_w + Y_w$

†1 対数平均エンタルピー差の導出に関しては第 12 章を参照。

$$\begin{bmatrix} 1 & 0 & X_w \\ 0 & 1 & Y_w \\ Z_w & 1 & 0 \end{bmatrix} \begin{bmatrix} h_{ao} \\ h_{wb} \\ Q_w \end{bmatrix} = \begin{bmatrix} h_{ab} \\ h_{wi} \\ h_{ab}+Z_w h_{wi} \end{bmatrix} \tag{9.14}$$

$$h_{ao} = V_3 h_{ab} + V_4 h_{wi} \tag{9.16}$$

$$t_{wb} = V_5 t_{wi} - V_6 (h_{ab} - b) \tag{9.17}$$

ただし、　$V_3 = (X_w + Y_w)/W_w$　、　$V_4 = X_w (Z_w - 1)/W_w$

$V_5 = Z_w (X_w + Y_w)/W_w$　、　$V_6 = Y_w (1 - Z_w)/W_w a$

3）全領域の計算

計算にあたっては空気と水の入口条件（t_{ai} と t_{wi}）が与えられることが通常であるが、式 9.6 と 9.14 はいずれも乾湿境界での状態（h_{ab} と t_{wb}）が入力条件となっており不便である。そこで式 9.8、9.17、9.18 を用いて t_{ab} と h_{ab} を消去し、乾湿境界水温 t_{wb} について整理すると式 9.19 が得られる。t_{wb} を 9.8 と 9.9 に代入すれば乾湿境界空気温度 t_{ab} と出口水温 t_{wo} が得られる。さらに式 9.16 と 9.18 を用いて空気の出口エンタルピー h_{ao} を求めることができ、入口条件から乾湿境界および出口における状態値がすべて求まる。

$$h_{ab} - h_{ai} = c_{pma} (t_{ab} - t_{ai}) \tag{9.18}$$

$$t_{wb} = \frac{V_5 t_{wi} + V_6 (h_{ai} - V_1 c_{pma} t_{ai} - b)}{1 - V_1 V_6 c_{pma}} \tag{9.19}$$

乾きコイルの面積比率 R_{dw} [-]を式 9.20 で定義する。式 9.3~9.19は式 9.20 の R_{dw} に 0~1.0 までの間の任意の数値を入れても成立するため、水と空気の状態を確定するためには条件がもう一つ必要である。前述の通り、冷却除湿コイルの乾湿境界においては相対湿度が ϕ_w となるため、条件として式 9.21 を追加する。$f(\cdot)$は相対湿度と絶対湿度から乾球温度を求める関数である。なお、$R_{dw} = 1.0$ で $f(\phi_w, W_{ai}) < t_{ab}$ となる場合には、乾きコイル部分で熱交換が終了しており結露は生じない。

$$R_{dw} = S_d / (S_d + S_w) \tag{9.20}$$

$$t_{ab} = f(\varphi_w, W_{ai}) \tag{9.21}$$

9.2.3　熱貫流率の推定

本節では特性式にもとづく方法と積み上げによる方法の 2 つの方法での熱貫流率の計算法を記す。

1）フロータイプ

配管内の水速は熱交換に大きな影響を与える。水速を把握するためにはフロータイプという概念を知る必要がある。

配管内の水の速度が小さすぎると水の流れが層流となり熱伝達率が低下する。逆に水の速度が大きすぎると抵抗が大きくなり搬送動力が大きくなる。このような観点から、配管内の水速は 0.5 ~ 2.5 m/s 程度の範囲になるように設計されることが通常である。水速を所定の範囲内に納めるためには水の流路の断面積を調整する必要があり、通常はハーフフロー、シングルフロー、ダブルフロー、トリプルフローという 4 種類の流路のタイプを適切に選択することでこれを行う[†1)]。図 9.4 にコイルのフロータイプと水の流れを示す。流路断面積がシングルフローに比較してハーフフローでは 1/2 倍、ダブルフローでは 2 倍、トリプルフローでは 3 倍となっていることがわかる。水速は断面積に反比例す

†1　地下水利用などで極端に水量が少ない場合などには 1/8 フローなどを用いることもある。工場検査などにいくとわかるが、建築設備機器の多くは大量生産品ではなく、目に見える形で職人が制作している。従って、配管流路の仕様なども比較的自由に選択可能であり、設備技術者の力量が直接にシステムの性能に反映される傾向にある。

るため、それぞれ 2 倍、1/2 倍、1/3 倍となる。

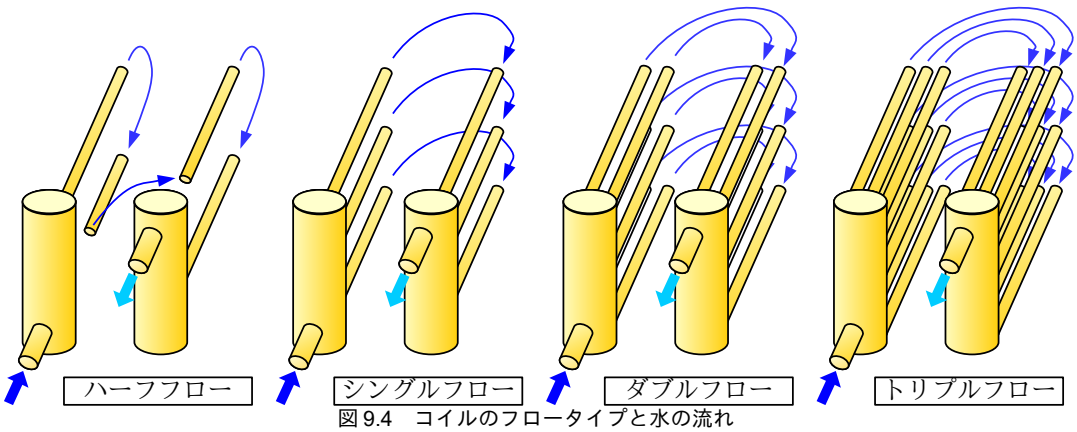


図 9.4　コイルのフロータイプと水の流れ

2）特性式による計算法

この方法は HASP/ACSS において用いられている方法であり、式 9.22 と 9.23 で計算する[9.7]。精度は後述の積み上げ計算による方法に劣るが、水速と風速のみが入力情報であるため、プログラムの実装やパラメータの設定が容易である。K_d [kW/(m²·K)]は乾きコイルの熱貫流率であり、加熱コイルと冷却除湿コイルの両方に適用する。K_w [kW/(m²·(kJ/kg))]は湿りコイルの熱貫流率である。v_wは管内の水速[m/s]、v_{af}は前面面積基準の風速[m/s]である。前面面積基準とコア面積基準の違いは後述する。

$$K_d=\left(4.72+4.91\,v_w^{-0.8}+26.7\,v_{af}^{-0.64}\right)^{-1} \tag{9.22}$$

$$K_w=\left(10.044+10.44\,v_w^{-0.8}+39.6\,v_{af}^{-0.64}\right)^{-1} \tag{9.23}$$

式 9.22 と 9.23 をグラフ化すると図 9.5 が得られる。このような図表に関しては空調機メーカーのカタログや技術資料にも記載があるが、その際には熱貫流率の単位に気をつける必要がある。図 9.5 は伝熱面積あたりの値であるが、カタログには一列あたり、あるいは前面面積あたりの熱貫流率が記載されている場合がある。

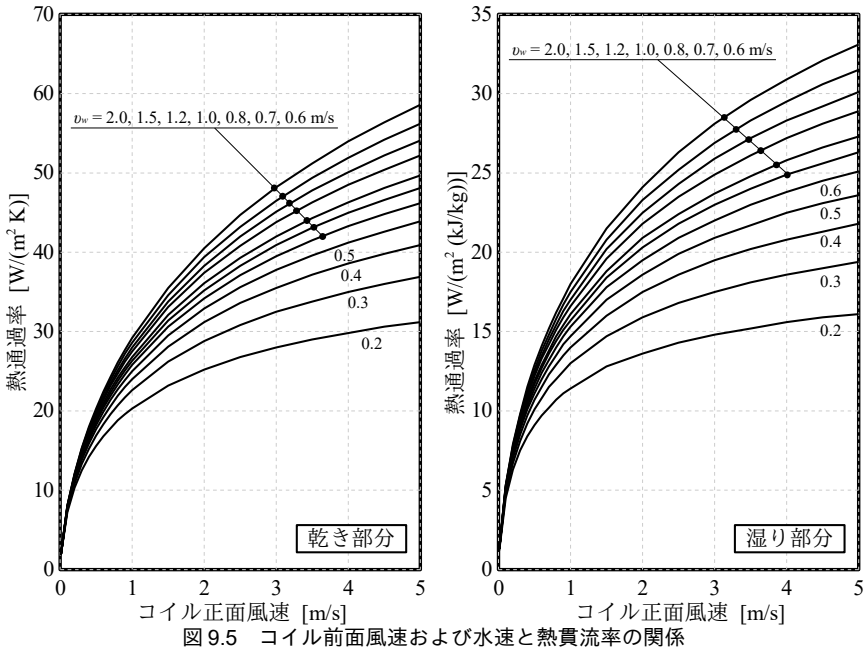


図 9.5　コイル前面風速および水速と熱貫流率の関係

3）積み上げによる計算法[9.9) 9.8)]

特性式にもとづく方法とは異なり、詳細な情報が必要であるが、それぞれのコイルの形状に応じた熱貫流率を算出することができるため、精度は高い。

・乾きコイル

乾きコイルの熱貫流率 K_d は式 9.24 で計算する。

$$\frac{1}{1000\,K_d}=\frac{R_{aw}}{\alpha_w}+\frac{1}{\alpha_{fd}\left\{\phi_d+(1/R_{aw})\right\}} \tag{9.24}$$

R_{aw} [-]は水側の表面積 S_w [m²]に対する空気側の表面積 S_a [m²]であり、式 9.25 で計算する。S_fはフィンの表面積 [m²/列]、S_{to}はチューブの外表面積 [m²/列]、S_{ti}はチューブの内表面積 [m²/列]であり、いずれも 1 列あたりの面積として式 9.26~9.28で計算できる。H [m]、W [m]、D [m]はそれぞれコイルの高さ、幅、奥行きであり、図 9.6 のとおりである。P_f [m/枚]と d_f [m]はフィンのピッチと厚み、N_R [列]と N_D [段]はコイルの列数と段数、d_i [m]と d_i [m]はチューブの内径と外径である。

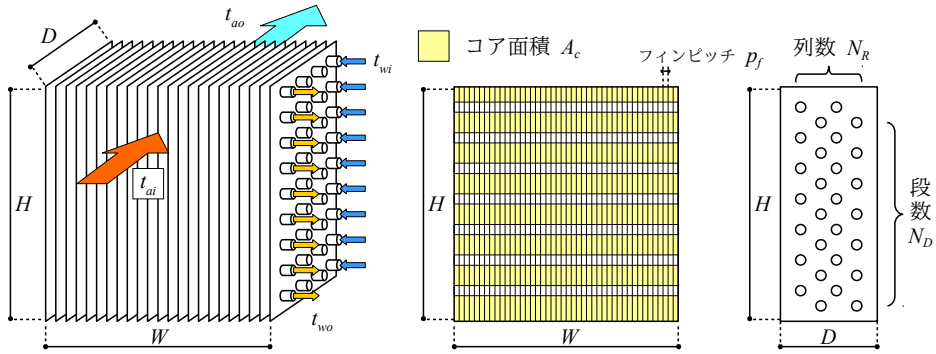


図 9.6　コイルの寸法

$$R_{aw}=\frac{S_a}{S_w}=\frac{N_R\left(S_f+S_{to}\right)}{N_R S_{ti}} \tag{9.25}$$

$$S_f=2\left(\frac{HD}{N_R}-\frac{d_o^2}{4}\pi N_D\right)\frac{W}{P_f} \tag{9.26}$$

$$S_{to}=d_o\pi N_D W\left(1-\frac{d_f}{P_f}\right) \tag{9.27}$$

$$S_{ti}=d_i\pi N_D W \tag{9.28}$$

α_wは管内表面の対流熱伝達率[W/(m²·K)]であり、計算法はすでに例題 3.2 において示した。

α_{fd}は乾き部分のフィン表面の対流熱伝達率[W/(m²·K)]である。新津らの実験によれば、コイルの乾き伝熱面上において、ヌセルト数 Nu_d [-]とレイノルズ数 Re [-]との間には、式 9.29 の関係がある[9.10)]。一方、ヌセルト数は式 9.30 で表現されるため、フィン表面の対流熱伝達率 α_{fd} は式 9.31 で計算できる。v_{ac} [m/s]はコア面積基準の風速であり風量 Q_a [CMH]をコア面積 A_c [m²]で除して式 9.32 で計算する。コア面積とは図 9.6 に示すように単純な前面面積（W×H）からチューブとフィンの投影面積を差し引いた、空気が通過可能な部分の見付面積であり、式 9.33 で計算する。λ_aと ν_aはそれぞれ空気の熱伝導率 [W/(m·K)]と動粘性係数 [m²/s]であり、計算法はすでに第 4 章で示した。d_eは空気の流路の相当直径[†1)] [m]であり、式 9.34 で計算する。

†1　直径でいくらの円管の集合と等しいかを表す、流路の代表長さ [m]。流路の断面積[m²]を流路の周長[m]で除して求める。等価

$$Nu_d = 0.129\,Re_d^{0.64} \quad \text{ただし、レイノルズ数は} \quad Re_d = \frac{\upsilon_{ac} d_e}{\nu_a} \tag{9.29}$$

$$Nu_d = \frac{\alpha_{fd} d_e}{\lambda_a} \tag{9.30}$$

$$\alpha_{fd} = 0.129 \frac{\lambda_a}{d_e} Re^{0.64} \tag{9.31}$$

$$\upsilon_{ac} = \frac{Q_a}{A_c} \tag{9.32}$$

$$A_c = (WH - d_o W N_D)\left(1 - \frac{d_f}{P_f}\right) \tag{9.33}$$

$$d_e = 4\,A_c / (S_a / D) \tag{9.34}$$

ϕ_d [-]はフィン効率であり、式 9.35 で示される相当半径x_e [m]を用いて一定厚さの環状フィンとみなして式 9.36 で計算する[9.14)]。

$$x_e = \sqrt{(D/N_R)(H/N_D)/\pi} \tag{9.35}$$

$$\phi_d = \frac{2}{u_{bd}\left[1 - (u_{ed}/u_{bd})^2\right]} \left[\frac{I_1(u_{bd}) K_1(u_{ed}) - I_1(u_{ed}) K_1(u_{bd})}{I_0(u_{bd}) K_1(u_{ed}) + I_1(u_{ed}) K_0(u_{bd})} \right] \tag{9.36}$$

ただし、 $u_{bd} = \dfrac{w\sqrt{\alpha_{fd}/\lambda_f y_b}}{x_e/x_b - 1}$ 、 $u_{ed} = u_{bd}(x_e/x_b)$

y_b, w, x_bはそれぞれ環状フィンの寸法 [m]を表しており、図 9.7 のとおりである。また、λ_fはフィン材料の熱伝導率 [W/(m·K)]であり、材料としてはアルミ（237 W/(m·K)）または銅（398 W/(m·K)）を用いることが多い。

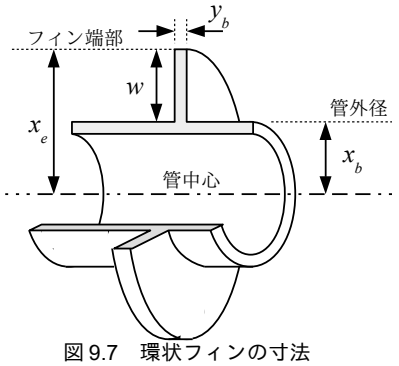


図 9.7　環状フィンの寸法

$I_0(\cdot)$、$I_1(\cdot)$、$K_0(\cdot)$、$K_1(\cdot)$はそれぞれ変形ベッセル関数であり、式 9.37~9.44で近似する[9.11) 9.12)]。

$$\begin{aligned} I_0(u) \approx{} & 1 + 3.5156229(u/3.75)^2 + 3.0899424(u/3.75)^4 + 1.2067492(u/3.75)^6 \\ & + 0.2659732(u/3.75)^8 + 0.0360768(u/3.75)^{10} + 0.0045813(u/3.75)^{12} \end{aligned} \quad (-3.75 \le u < 3.75) \tag{9.37}$$

$$\begin{aligned} I_0(u) u^{0.5} e^{-u} \approx{} & 0.398942280 + 0.013285917(3.75/u) + 0.002253187(3.75/u)^2 \\ & - 0.001575649(3.75/u)^3 + 0.009162808(3.75/u)^4 - 0.020577063(3.75/u)^5 \\ & + 0.026355372(3.75/u)^6 - 0.016476329(3.75/u)^7 + 0.003923767(3.75/u)^8 \end{aligned} \quad (3.75 \le u < \infty) \tag{9.38}$$

$$\begin{aligned} I_1(u)/u \approx{} & 0.87890594(u/3.75)^2 + 0.51498869(u/3.75)^4 + 0.15084934(u/3.75)^6 \\ & + 0.02658733(u/3.75)^8 + 0.00301532(u/3.75)^{10} + 0.00032411(u/3.75)^{12} + 0.5 \end{aligned} \quad (-3.75 \le u < 3.75) \tag{9.39}$$

$$\begin{aligned} I_1(u) u^{0.5} e^{-u} \approx{} & 0.398942280 - 0.039880242(3.75/u) - 0.003620183(3.75/u)^2 \\ & + 0.001638014(3.75/u)^3 - 0.010315550(3.75/u)^4 + 0.022829673(3.75/u)^5 \\ & - 0.028953121(3.75/u)^6 + 0.017876535(3.75/u)^7 - 0.004200587(3.75/u)^8 \end{aligned} \quad (3.75 \le u < \infty) \tag{9.40}$$

直径とも呼び、英語表記は Equivalent Diameter。

$$K_0(u)+\ln(0.5u)I_0(u)\approx -0.57721566 +0.42278420(u/2)^2+0.23069756(u/2)^4+0.03488590(u/2)^6 +0.00262698(u/2)^8+0.00010750(u/2)^{10}+0.00000740(u/2)^{12} \quad (0\le u<2) \tag{9.41}$$

$$K_0(u)u^{0.5}e^u\approx 1.25331414 -0.07832358(2/u)+0.02189568(2/u)^2-0.01062446(2/u)^3 +0.00587872(2/u)^4-0.00251540(2/u)^5+0.00053208(2/u)^6 \quad (2\le u<\infty) \tag{9.42}$$

$$\left\{K_1(u)-\ln(0.5u)I_1(u)\right\}u\approx 1 +0.15443144(u/2)^2-0.67278579(u/2)^4-0.18156897(u/2)^6 -0.01919402(u/2)^8-0.00110404(u/2)^{10}-0.00004686(u/2)^{12} \quad (0\le u<2) \tag{9.43}$$

$$K_1(u)u^{0.5}e^u\approx 1.25331414 +0.23498619(2/u)-0.03655620(2/u)^2+0.01504268(2/u)^3 -0.00780353(2/u)^4+0.00325614(2/u)^5-0.00068245(2/u)^6 \quad (2\le u<\infty) \tag{9.44}$$

・湿りコイル

湿りコイルの熱貫流率 K_w [kW/(m²·(kJ/kg))]は式 9.45 で計算する。

$$\frac{1}{1000K_w}=\frac{aR_{aw}}{\alpha_w}+\frac{1}{k_f\left\{\phi_w+(1/R_{aw})\right\}} \tag{9.45}$$

a は式 9.13 で示した比エンタルピーの近似係数である。

k_fはフィン表面のエンタルピー基準の物質移動係数 [W/(m²·(kJ/kg))]である。新津らの実験によれば、コイルの湿り伝熱面上において、シャーウッド数 Sh [-]とレイノルズ数 Re [-]との間には、式 9.46 の関係がある[9.10)]。一方、シャーウッド数は式 9.47 で表現されるため、湿りフィン表面のエンタルピー基準の物質移動係数 k_fは式 9.48 で計算できる。なお α_{av}は空気中における水蒸気の拡散係数[m²/s]、γ_aは湿り空気の比重量[kg/m³]であり、計算法は第 4 章において記した。

$$Sh=0.0372Re^{0.8} \tag{9.46}$$

$$Sh=0.001\frac{k_f d_e}{\gamma_a \alpha_{av}} \tag{9.47}$$

$$k_f=37.2\gamma_a\alpha_{av}Re^{0.8}/d_e \tag{9.48}$$

ϕ_wは湿りコイルのフィン効率であり、式 9.49 で計算する。

$$\phi_w=\frac{2}{u_{bw}\left[1-(u_{ew}/u_{bw})^2\right]}\left[\frac{I_1(u_{bw})K_1(u_{ew})-I_1(u_{ew})K_1(u_{bw})}{I_0(u_{bw})K_1(u_{ew})+I_1(u_{ew})K_0(u_{bw})}\right] \tag{9.49}$$

ただし、　$u_{bw}=\dfrac{w\sqrt{\alpha_{fw}/\lambda_f y_b}}{x_e/x_b-1}$　、　$u_{ew}=u_{bw}(x_e/x_b)$

$$\alpha_{fw}=\frac{a}{c_{pma}L_e}\alpha_{fd} \quad 、 \quad L_e=3.19Re^{-0.16}$$

ところで、理論としてはコイル外表面の伝熱面積の和である S_f+S_{to}は、式 9.5 と 9.12 における伝熱面積の和 S_w+S_dに一致するはずである。しかし実際にはモデル化の精度の限界により、S_w+S_dに S_f+S_{to}を代入すると定格条件下における能力が必ずしも定格能力と一致しないことが通常である。そこでこの誤差を吸収するために式 9.50 に示す通り、補正係数 α_Sを導入することとする[†1)]。

$$S_w+S_d=\alpha_S(S_f+S_{to}) \tag{9.50}$$

†1　わざわざ演繹的な方法によってあり得べき伝熱性能を計算したにも関わらず、補正係数で出力から調整を加えてしまうことを奇異に感じるかもしれない。しかし、機器単体レベルでのあり得べき性能の解析は機械設計の分野である。建築空調設計の立場からは、機器単体としては当然に仕様通りの性能が表れるものとし、システムとしてのあり得べき性能を追求することが必要である。無論、補正係数 α_Sが 1.0 から異常に離れるような場合にはモデル性能の適否を疑う必要はある。

【例題 9.2】

下記の定格能力を持つ冷却除湿コイルの伝熱面積を計算せよ。

風量：6,750CMH、水量：300 L/min、入口水温：5 ºC、入口空気温度：32 ºC、

入口絶対湿度：0.0125 kg/kg、熱交換量：70 kW、乾湿境界相対湿度：95 %

乾き部分の熱貫流率：60 W/(m^2·K)、湿り部分の熱貫流率：45 W/(m^2·(kJ/kg))

【解】

湿り部分の伝熱面積と乾き部分の伝熱面積をそれぞれ計算し、合算することで熱交換器全体の面積を求める。

まず、湿り空気の出口条件を計算する。空気の質量流量 m_a は

$m_a = 6{,}750 / 3600 \times 1.2 = 2.25$ kg/s

である。入口空気の乾球温度と絶対湿度が与えられているため、入口空気の比エンタルピーを求めることができ、その値は 64 kJ/kg である。熱交換量と入口エンタルピー、空気の質量流量を用いることで出口エンタルピー h_{ao} は、

$h_{ao} = 64 - 70 / 2.25 = 33$ kJ/kg

となる。また、乾湿境界では相対湿度が 95 %であり、入口絶対湿度との交点の空気状態を求めると、比エンタルピー h_{ab} が 50 kJ/kg、乾球温度 t_{ab} が 18.3 ºC である。

以上により、入口、乾湿境界、出口での空気の比エンタルピーが得られたため、空気の質量流量に基づいて乾き部分と湿り部分の熱交換量が求められる。

$Q_d = 2.25 \times (64 - 50) = 31.5$ kW

$Q_w = 70 - 31.5 = 38.5$ kW

熱交換量と入口水温 t_{wi} に基づいて乾湿境界水温 t_{wb} および出口水温 t_{wo} は、

$t_{wb} = 5 + 38.5 \div 4.186 \div (300 / 60) = 6.8$ ºC

$t_{wo} = 6.8 + 31.5 \div 4.186 \div (300 / 60) = 8.3$ ºC

と求められる。入口水温 t_{wi} および乾湿境界での水温 t_{wb} に相当する空気の飽和エンタルピーを求めると、h_{wi} = 19 kJ/kg、h_{wb} = 22 kJ/kg となる。従って、対数平均エンタルピー差は、

$\Delta h_{LM} = ((50 - 22) - (33 - 19)) / \ln((50 - 22)/(33 - 19)) = 20.4$

となる。式 9.12 を用いて湿り部分の伝熱面積 S_w は、

$S_w = 38.5 / 20.4 / 0.045 = 42$m^2

となる。

乾き部分も同様の計算を行い、対数平均温度差は、

$\Delta t_{LM} = ((32 - 8.3) - (18.3 - 6.8)) / \ln((32 - 8.3)/(18.3 - 6.8)) = 16.8$

となる。式 9.5 を用いて乾き部分の伝熱面積 S_d は、

$S_d = 31.5 / 16.8 / 0.060 = 31$m^2

となる。

従って伝熱面積 S は湿り部分と乾き部分を合算し、

$S = 42 + 31 = 73$ m²

である。

9.3 計算法

プレートフィン付管熱交換器の計算に用いる列挙型の定義をプログラム 9.1 に示す。

プログラム 9.1 列挙型の定義

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
/// <summary>フロータイプ</summary>
public enum WaterFlowType
{
  /// <summary>ハーフフロー</summary>
  HalfFlow,
  /// <summary>シングルフロー</summary>
  SingleFlow,
  /// <summary>ダブルフロー</summary>
  DoubleFlow,
  /// <summary>トリプルフロー</summary>
  TripleFlow
}
```

9.3.1　熱貫流率の計算

式 9.22 と 9.23 にもとづき、水速と前面風速から熱貫流率を計算するプログラムを 9.2 に示す。

プログラム 9.2　熱貫流率の計算 1

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
/// <summary>熱貫流率を計算する</summary>
/// <param name="waterSpeed">水速[m/s]</param>
/// <param name="velocity">前面風速[m/s]</param>
/// <param name="dryHeatTransferCoefficient">乾き部分熱貫流率[kW/(m2K)]</param>
/// <param name="wetHeatTransferCoefficient">湿り部分熱貫流率[kW/(m2(kJ/kg))]</param>
public static void GetHeatTransferCoefficient(double waterSpeed, double velocity,
 out double dryHeatTransferCoefficient, out double wetHeatTransferCoefficient)
{
  dryHeatTransferCoefficient = 1 / (4.72 + 4.91 * Math.Pow(waterSpeed, -0.8)
    + 26.7 * Math.Pow(velocity, -0.64));
  wetHeatTransferCoefficient = 1 / (10.044 + 10.44 * Math.Pow(waterSpeed, -0.8)
    + 39.6 * Math.Pow(velocity, -0.64));
}
```

式 9.24 と 9.45 にもとづき、熱交換器の形状をもとに熱貫流率を計算するプログラムを 9.3 に示す。

1~42 行は、熱交換器の幾何学形状を計算するメソッドである。式 9.25~9.28などの熱媒の状態に依存しない値に関しては、ここでまとめて計算を行う。

44~120 行が熱貫流率の計算処理であり、熱媒の状態に加え、先のメソッドで計算された幾何学形状を入力値として受け取る。このような二段階のメソッドとすることで、熱媒の状態が変化する度に再計算を行う必要がなくなる。70~79 行で計算に必要な水および湿り空気の物性を計算する。84~89 行では乾き部分と湿り部分の両方の熱貫流率計算で必要となるレイノルズ数および水側対流熱伝達率を計算する。91~101 行は乾き部分の熱貫流率計算、103~119 行は湿り部分の熱貫流率計算処理である。

122~136 行は式 9.13 で示される飽和エンタルピーの近似係数の計算メソッドである。水温を微小変化させることで微分値を求める。

プログラム 9.3　熱貫流率の計算 2

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
/// <summary>幾何学形状を計算する</summary>
/// <param name="depth">奥行き[m]</param>
/// <param name="width">幅[m]</param>
/// <param name="height">高さ[m]</param>
/// <param name="rowNumber">列数[列]</param>
/// <param name="columnNumber">段数[段]</param>
/// <param name="finPitch">フィンピッチ[m]</param>
/// <param name="finThickness">フィン厚み[m]</param>
/// <param name="innerDiameter">管の内径[m]</param>
/// <param name="outerDiameter">管の外径[m]</param>
/// <param name="airWaterSurfaceRatio">空気側・水側伝熱面積比[-]</param>
/// <param name="coreArea">コア面積[m2]</param>
/// <param name="equivalentFinRadius">環状フィンの相当半径[m]</param>
/// <param name="equivalentDiameter">等価直径[m]</param>
/// <param name="surfaceArea">空気側伝熱面積[m2]</param>
public static void GetGeometricCompfigulation
  (double depth, double width, double height, int rowNumber, int columnNumber,
  double finPitch, double finThickness, double innerDiameter, double outerDiameter,
  out double airWaterSurfaceRatio,out double coreArea, out double equivalentFinRadius,
  out double equivalentDiameter, out double surfaceArea)
{
  //空気側伝熱面積[m2]の計算
  double sf = 2 * (height * depth / rowNumber - outerDiameter * outerDiameter /
    4 * Math.PI * columnNumber) * width / finPitch;
  double sto = outerDiameter * Math.PI * columnNumber * width * (1 - finThickness / finPitch);
  surfaceArea = sf + sto;

  //水側伝熱面積[m2]の計算
  double wSurface = innerDiameter * Math.PI * columnNumber * width;

  //空気側・水側伝熱面積比[-]の計算
  airWaterSurfaceRatio = surfaceArea / wSurface;

```

```
  //コア面積[m2]の計算
  coreArea = (width * height - outerDiameter * width * columnNumber) * (1 - finThickness / finPitch);

  //環状フィンの相当半径[m]の計算
  equivalentFinRadius = Math.Sqrt((depth / rowNumber) * (height / columnNumber) / Math.PI);

  //等価直径[m]の計算
  equivalentDiameter = 4 * coreArea / (surfaceArea * rowNumber / depth);
}

/// <summary>熱貫流率を計算する</summary>
/// <param name="airWaterSurfaceRatio">空気側・水側伝熱面積比[-]</param>
/// <param name="coreArea">コア面積[m2]</param>
/// <param name="equivalentFinRadius">環状フィンの相当半径[m]</param>
/// <param name="equivalentDiameter">等価直径[m]</param>
/// <param name="waterPath">配管の流路数[-]</param>
/// <param name="finThickness">フィン厚み[m]</param>
/// <param name="thermalConductivity">フィン材料の熱伝導率[W/(mK)]</param>
/// <param name="innerDiameter">管の内径[m]</param>
/// <param name="outerDiameter">管の外径[m]</param>
/// <param name="airFlowRate">風量[kg/s]</param>
/// <param name="inletAirTemperature">入口空気温度[C]</param>
/// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
/// <param name="borderRelativeHumidity">乾湿境界での空気相対湿度[%]</param>
/// <param name="waterFlowRate">水量[kg/s]</param>
/// <param name="inletWaterTemperature">入口水温[C]</param>
/// <param name="dryHeatTransferCoefficient">乾き部分の熱貫流率[kW/(m2K)]</param>
///<param name="wetHeatTransferCoefficient">湿り部分の熱貫流率[kW/(m2(kJ/kg))]</param>
public static void GetHeatTransferCoefficient
  (double airWaterSurfaceRatio, double coreArea, double equivalentFinRadius,
  double equivalentDiameter, double waterPath,double finThickness,
  double thermalConductivity, double innerDiameter, double outerDiameter,
  double airFlowRate, double inletAirTemperature, double inletAirHumidityRatio,
  double borderRelativeHumidity, double waterFlowRate, double inletWaterTemperature,
  out double dryHeatTransferCoefficient, out double wetHeatTransferCoefficient)
{
  //湿り空気物性の計算//比熱[kJ/kgK]・動粘性係数[m2/s]・熱伝導率[W/(mK)]
  //比体積[kg/m3]・拡散係数[m2/s]
  double cpma = MoistAir.GetSpecificHeat(inletAirHumidityRatio);
  double dVis = MoistAir.GetDynamicViscosity
    (inletAirTemperature, inletAirHumidityRatio, ATMOSPHERIC_PRESSURE);
  double tCond = MoistAir.GetThermalConductivity(inletAirTemperature);
  double sVol = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
    (inletAirTemperature, inletAirHumidityRatio, ATMOSPHERIC_PRESSURE);
  double difc = MoistAir.GetDiffusionCoefficient
    (inletAirTemperature, ATMOSPHERIC_PRESSURE);

  //実風速の計算[m/s]
  double coreVelocity = airFlowRate * AIR_SPECIFIC_WEIGHT / coreArea;

  //レイノルズ数[-]の計算
  double re = coreVelocity * equivalentDiameter / dVis;

  //水側の対流熱伝達率[W/(m2K)]の計算
  double wfCoefficient = Pipe.GetWaterSideConvectiveHeatTransferCoefficient
    (inletWaterTemperature, innerDiameter, waterFlowRate / waterPath);

  //乾き部分の計算////
  //空気側対流熱伝達率[W/(m2K)]の計算
  double afd = 0.129 * tCond / equivalentDiameter * Math.Pow(re, 0.64);

  //フィン効率[-]の計算
  double fEfficiencyD = HeatExchanger.GetCircularFinEfficiency
    (outerDiameter / 2, equivalentFinRadius, finThickness, afd, thermalConductivity);

  //熱貫流率[kW/(m2K)]の計算
  dryHeatTransferCoefficient = 0.001 / (airWaterSurfaceRatio / wfCoefficient +
    1 / (afd * (fEfficiencyD + 1 / airWaterSurfaceRatio)));

  //湿り部分の計算////
  //フィン表面の物質移動係数[W/(m2(kJ/kg))]の計算
  double kf = 37.2 * difc / (sVol * equivalentDiameter) * Math.Pow(re, 0.8);

  //エンタルピー近似係数の計算
  double a, b;
  getSaturationEnthalpyCoefficients(inletWaterTemperature, out a, out b);

  //フィン効率[-]の計算
  double lewis = 3.19 * Math.Pow(re, -0.16);
  double afw = a / (cpma * lewis) * afd;
```

```
114   double fEfficiencyW = HeatExchanger.GetCircularFinEfficiency
115     (outerDiameter / 2, equivalentFinRadius, finThickness, afw, thermalConductivity);
116
117   //熱貫流率[kW/(m2(kJ/kg))]の計算
118   wetHeatTransferCoefficient = 0.001 / (a * airWaterSurfaceRatio / wfCoefficient +
119     1 / (kf * (fEfficiencyW + 1 / airWaterSurfaceRatio)));
120 }
121
122 /// <summary>乾球温度[C]による飽和エンタルピー[kJ/kg]近似式の係数を計算する</summary>
123 /// <param name="drybulbTemperature">乾球温度[C]</param>
124 /// <param name="a">乾球温度[C]の係数</param>
125 /// <param name="b">切片</param>
126 private static void getSaturationEnthalpyCoefficients
127   (double drybulbTemperature, out double a, out double b)
128 {
129   const double DELTA = 0.001;
130   double hws1 = MoistAir.GetSaturationEnthalpyFromDrybulbTemperature
131     (drybulbTemperature, ATMOSPHERIC_PRESSURE);
132   double hws2 = MoistAir.GetSaturationEnthalpyFromDrybulbTemperature
133     (drybulbTemperature + DELTA, ATMOSPHERIC_PRESSURE);
134   a = (hws2 - hws1) / DELTA;
135   b = hws1 - a * drybulbTemperature;
136 }
```

9.3.2　伝熱面積の計算

プログラム 9.4 に伝熱面積の計算処理を示す。

18~21 行は前処理であり、水と空気の熱容量流量を計算する。

24 行で水と空気の入口温度の高低によって加熱コイルか冷却コイルかを判定し、加熱コイルの場合には 26~34 行、冷却コイルの場合には 39~92 行でそれぞれ伝熱面積を計算する。加熱コイルの伝熱面積の計算は例題 9.1 に示したとおりであり、向流型熱交換器の熱貫流率を用いて計算を行う。

冷却コイルの伝熱面積の計算方法は、熱交換が顕熱のみか潜熱を伴うかによって異なる。39~45 行に示すように、入力条件として与えられた熱交換能力から出口空気の比エンタルピーを求め、乾湿境界相対湿度との交点の絶対湿度を求める。この絶対湿度が入口空気の絶対湿度よりも低い場合には潜熱交換が発生していることになる。52~59 行は顕熱交換のみの場合の伝熱面積の計算であり、式 9.3~9.5を用いて対数平均温度差から伝熱面積を逆算する。63~92 行は潜熱交換を伴う場合の伝熱面積の計算であり、処理内容は例題 9.2 で示した通りである。

プログラム 9.4　伝熱面積の計算

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
 1 /// <summary>伝熱面積[m2]を計算する</summary>
 2 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
 3 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
 4 /// <param name="borderRelativeHumidity">乾湿境界相対湿度[%]</param>
 5 /// <param name="inletWaterTemperature">入口水温[C]</param>
 6 /// <param name="airFlowRate">風量[kg/s]</param>
 7 /// <param name="waterFlowRate">水量[kg/s]</param>
 8 /// <param name="heatTransfer">熱交換量能力[kW]</param>
 9 /// <param name="dryHeatTransferCoefficient">乾き部分の熱貫流率[kW/(m2K)]</param>
10 /// <param name="wetHeatTransferCoefficient">湿り部分の熱貫流率[kW/(m2(kJ/kg))]</param>
11 /// <returns>伝熱面積[m2]</returns>
12 public static double GetSurfaceArea
13   (double inletAirTemperature, double inletAirHumidityRatio, double borderRelativeHumidity,
14   double inletWaterTemperature, double airFlowRate, double waterFlowRate, double heatTransfer,
15   double dryHeatTransferCoefficient, double wetHeatTransferCoefficient)
16 {
17
18   //水と湿り空気の熱容量流量[kW/s]の計算
19   double cpma = MoistAir.GetSpecificHeat(inletAirHumidityRatio);
20   double mca = airFlowRate * cpma;
21   double mcw = waterFlowRate * WATER_SPECIFIC_HEAT;
22
23   //加熱コイルの場合
24   if (inletAirTemperature < inletWaterTemperature)
25   {
26     //NTU値の計算
27     double mcMin = Math.Min(mcw, mca);
```

```
28     double mcMax = Math.Max(mcw, mca);
29
30     //熱通過有効度[-]の計算
31     double eff = heatTransfer / mcMin / (inletWaterTemperature - inletAirTemperature);
32     double ntu = HeatExchange.GetNTU(eff, mcMin / mcMax, HeatExchange.FlowType.CounterFlow);
33
34     return ntu * mcMin / dryHeatTransferCoefficient;
35   }
36   //冷却コイルの場合
37   else
38   {
39     //冷水出口温度[C]の計算
40     double oWaterTemp = inletWaterTemperature + heatTransfer / mcw;
41
42     //空気出入口エンタルピー[kJ/kg]の計算
43     double iAirEnthalpy = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
44       (inletAirTemperature, inletAirHumidityRatio);
45     double oAirEnthalpy = iAirEnthalpy - heatTransfer / airFlowRate;
46
47     //コイルの乾湿境界の計算
48     double oAirHumidRatio = MoistAir.GetHumidityRatioFromEnthalpyAndRelativeHumidity
49       (oAirEnthalpy, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
50
51     //乾きコイルのみの場合
52     if (inletAirHumidityRatio < oAirHumidRatio)
53     {
54       double oAirTemp = inletAirTemperature - heatTransfer / mca;
55       double d1 = inletAirTemperature - oWaterTemp;
56       double d2 = oAirTemp - inletWaterTemperature;
57       double lmtd = (d1 - d2) / Math.Log(d1 / d2);
58       return heatTransfer / lmtd / dryHeatTransferCoefficient;
59     }
60     //乾き+湿りコイルの場合
61     else
62     {
63       //境界点での湿り空気状態の計算
64       double bAirTemp =
65         MoistAir.GetDryBulbTemperatureFromHumidityRatioAndRelativeHumidity
66         (inletAirHumidityRatio, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
67       double bAirEnthalpy =
68         MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(bAirTemp, inletAirHumidityRatio);
69
70       //境界点での水温[C]の計算
71       double htWet = (bAirEnthalpy - oAirEnthalpy) * airFlowRate;
72       double bWaterTemp = inletWaterTemperature + htWet / mcw;
73
74       //水温と等しい温度の空気の飽和エンタルピー[kJ/(kg)]の計算
75       double iWaterEnthalpy =
76         MoistAir.GetSaturationEnthalpyFromDrybulbTemperature
77         (inletWaterTemperature, ATMOSPHERIC_PRESSURE);
78           double bWaterEnthalpy =
79         MoistAir.GetSaturationEnthalpyFromDrybulbTemperature(bWaterTemp, ATMOSPHERIC_PRESSURE);
80
81       //乾きコイルの表面積[m2]の計算
82       double dt1 = inletAirTemperature - oWaterTemp;
83       double dt2 = bAirTemp - bWaterTemp;
84       double lmtd = (dt1 - dt2) / Math.Log(dt1 / dt2);
85       double sd = (heatTransfer - htWet) / lmtd / dryHeatTransferCoefficient;
86
87       double dh1 = bAirEnthalpy - bWaterEnthalpy;
88       double dh2 = oAirEnthalpy - iWaterEnthalpy;
89       double lmhd = (dh1 - dh2) / Math.Log(dh1 / dh2);
90       double sw = htWet / lmhd / wetHeatTransferCoefficient;
91
92       return sd + sw;
93     }
94   }
95 }
```

9.3.3　出口状態の計算

プログラム 9.5 は熱媒の入口条件が与えられた場合の成り行きでの出口状態の計算処理である。

熱媒流量が 0 の場合には 23~32 行の通り、出口状態=入口状態とする。

空気と水の入口温度によって加熱コイルか冷却コイルかの判定を行う。加熱コイルの場合には 39~56 行、冷却コイルの場合には 58~119 行で計算する。

加熱コイルの計算には熱通過有効度を用いるため、熱容量流量などから直接計算する。

冷却コイルに関しては、乾き部分の比率 R_{dw} を未知変数として、収束計算によって出口状態を計算する。まず、結露が始まる乾湿境界での空気温度を求める（61 行）。誤差関数は 80~102 行に示す通り、乾き部分の比率 R_{dw} を引数とする関数として定義する。R_{dw} が与えられれば、9.2.2 節で記したとおり、式 9.19 と式 9.7 により乾湿境界の水温 t_{wb} と空気温度 t_{ab} を求めることができる。一方で、乾湿境界の温度は、入力条件として与えらた入口空気の絶対湿度と乾湿境界の相対湿度からも求めることができ、両者が一致する点が解である。そこで、上記の誤差関数は両者の差分を誤差として出力する。ただし、冷却除湿コイルは必ずしも除湿を伴わない。そこで、乾き部分の比率を 100 %とした場合の計算を行い、出口乾球温度が乾湿境界温度よりも高い場合には除湿がなく乾き部分の比率 R_{dw} を 1.0 として計算を終わらせる。出口乾球温度が乾湿境界温度よりも小さい場合には、Brent 法で乾き部分の比率 R_{dw} を収束計算する（105 行）。

プログラム 9.5　出口状態の計算

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
 1 /// <summary>出口状態を計算する</summary>
 2 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
 3 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
 4 /// <param name="borderRelativeHumidity">乾湿境界での空気相対湿度[%]</param>
 5 /// <param name="inletWaterTemperature">入口水温[C]</param>
 6 /// <param name="airFlowRate">風量[kg/s]</param>
 7 /// <param name="waterFlowRate">水量[kg/s]</param>
 8 /// <param name="dryHeatTransferCoefficient">乾き部分の熱伝達率[kW/(m2K)]</param>
 9 /// <param name="wetHeatTransferCoefficient">湿り部分の熱伝達率[kW/(m2(kJ/kg))]</param>
10 /// <param name="surfaceArea">表面積[m2]</param>
11 /// <param name="outletAirTemperature">出力：出口乾球温度[C]</param>
12 /// <param name="outletAirHumidityRatio">出力：出口絶対湿度[kg/kg]</param>
13 /// <param name="outletWaterTemperature">出力：出口水温[C]</param>
14 /// <param name="dryRate">出力：乾き部分の割合[-]</param>
15 public static void GetOutletState
16   (double inletAirTemperature, double inletAirHumidityRatio,
17   double borderRelativeHumidity, double inletWaterTemperature,
18   double airFlowRate, double waterFlowRate, double dryHeatTransferCoefficient,
19   double wetHeatTransferCoefficient, double surfaceArea,
20   out double outletAirTemperature, out double outletAirHumidityRatio,
21   out double outletWaterTemperature, out double dryRate)
22 {
23   //熱媒流量が 0 の場合は出口状態=入口状態
24   if (airFlowRate <= 0 || waterFlowRate <= 0 ||
25     inletWaterTemperature == inletAirTemperature)
26   {
27     outletAirTemperature = inletAirTemperature;
28     outletAirHumidityRatio = inletAirHumidityRatio;
29     outletWaterTemperature = inletWaterTemperature;
30     dryRate = 1;
31     return;
32   }
33
34   //水と湿り空気の熱容量流量[kW/s]の計算
35   double cpma = MoistAir.GetSpecificHeat(inletAirHumidityRatio);
36   double mca = airFlowRate * cpma;
37   double mcw = waterFlowRate * WATER_SPECIFIC_HEAT;
38
39   //加熱コイルの場合
40   if (inletAirTemperature < inletWaterTemperature)
41   {
42     dryRate = 1.0;
43
44     double mcMin = Math.Min(mcw, mca);
45     double mcMax = Math.Max(mcw, mca);
46     double ntu = dryHeatTransferCoefficient * surfaceArea / mcMin;
47
48     //対向流の熱通過有効度[-]の計算
49     double eff = HeatExchange.GetEffectiveness(ntu, mcMin / mcMax, HeatExchange.FlowType.CounterFlow);
50
51     //交換熱量・出口状態を計算
52     double q = eff * mcMin * (inletWaterTemperature - inletAirTemperature);
53     outletAirTemperature = inletAirTemperature + q / mca;
54     outletWaterTemperature = inletWaterTemperature - q / mcw;
55     outletAirHumidityRatio = inletAirHumidityRatio;
```

```
  }
  //冷却コイルの場合
  else
  {
    //乾湿境界での空気温度[C]の計算
    double ba = MoistAir.GetDryBulbTemperatureFromHumidityRatioAndRelativeHumidity
    (inletAirHumidityRatio, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);

    //入口空気のエンタルピー[kJ/kg]の計算
    double iAirEnthalpy = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
    (inletAirTemperature, inletAirHumidityRatio);

    //エンタルピー近似係数の計算
    double a, b;
    getSaturationEnthalpyCoefficients(inletWaterTemperature, out a, out b);

    double xd = 1 / mca;
    double yd = -1 / mcw;
    double xw = 1 / airFlowRate;
    double yw = -a / mcw;

    double v2, v3, v4, bWaterTemp, bAirTemp;
    v2 = v3 = v4 = bWaterTemp = bAirTemp = 0;

    Roots.ErrorFunction eFnc = delegate (double dRate)
    {
      double zd = Math.Exp(dryHeatTransferCoefficient * surfaceArea * dRate * (xd + yd));
      double wd = zd * xd + yd;
      double v1 = xd * (zd - 1) / wd;
      v2 = zd * (xd + yd) / wd;

      double zw = Math.Exp(wetHeatTransferCoefficient * surfaceArea * (1 - dRate) * (xw + yw));
      double ww = zw * xw + yw;
      v3 = (xw + yw) / ww;
      v4 = xw * (zw - 1) / ww;
      double v5 = zw * (xw + yw) / ww;
      double v6 = yw * (1 - zw) / ww / a;

      //乾湿境界の水温[C]の計算
      bWaterTemp = (v5 * inletWaterTemperature
      + v6 * (iAirEnthalpy - v1 * cpma * inletAirTemperature - b)) / (1 - v1 * v6 * cpma);
      //乾湿境界での空気状態の計算
      bAirTemp = inletAirTemperature - v1 * (inletAirTemperature - bWaterTemp);

      //誤差の評価
      return ba - bAirTemp;
    };
    //結露が生じる場合には乾きコイル面積比を収束計算
    dryRate = 1.0;
    if (0 < eFnc(dryRate)) dryRate = Roots.Brent(0, 1, 0.0001, eFnc);

    //出口水温[C]の計算
    outletWaterTemperature = inletAirTemperature - v2 * (inletAirTemperature - bWaterTemp);
    double bAirEnthalpy = cpma * (bAirTemp - inletAirTemperature) + iAirEnthalpy;
    //出口空気状態の計算
    double iWaterEnthalpy = a * inletWaterTemperature + b;
    double oAirEnthalpy = v3 * bAirEnthalpy + v4 * iWaterEnthalpy;
    if (dryRate < 1.0)
      outletAirHumidityRatio = MoistAir.GetHumidityRatioFromEnthalpyAndRelativeHumidity
      (oAirEnthalpy, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
    else outletAirHumidityRatio = inletAirHumidityRatio;
    outletAirTemperature = MoistAir.GetDryBulbTemperatureFromHumidityRatioAndEnthalpy
      (outletAirHumidityRatio, oAirEnthalpy);
  }
}
```

9.3.4　必要水量の計算

全外気の空調機では、出口乾球温度が一定となるように冷温水二方弁で水量調整を行うことが多い。この場合には出口乾球温度を入力とし、必要水量を出力とする計算が必要となる。また、内部負荷を処理する空調機では還気温度が一定となるように水量を調整することがあるが、少なくとも定常計算では熱負荷、還気温度、風量から必要となる出口乾球温度が確定するため、結局は出口乾球温度を入力とする計算ができれば良い。

適当な水量を与えた場合の出口状態はプログラム 9.5 で計算できるため、二分法を用いて所定の出

口空気温度となる水量を計算する。プログラム 9.6 に計算処理を示す。1~53 行は簡易モデルによる計算、55~122 行は詳細モデルによる計算である。22 行で入口空気温度と水温から冷却か加熱かの判定を行う。出口空気温度設定値と入口空気温度を比較して通水が不要な場合には計算を終了する（25 行）。32, 33 行では最大水量での出口状態を計算し、最大水量でも出口温度設定値に達せない場合（過負荷の場合）には、最大水量での出口状態を出力する。出口温度設定値を実現できる場合には 0 kg/s~最大水量までの範囲のどこかに解があるから、 Brent 法を用いて水量を収束計算する（44~52 行）。以上は簡易モデルの場合の処理であるが、55~122 行の詳細モデルも処理内容は同様である。注意すべきは、簡易モデルであれ、詳細モデルであれ、水量が変化すると熱貫流率が変化するため、反復計算にあたっては熱貫流率も含めた再計算が必要となることである。

プログラム 9.6　必要水量の計算

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
 1 /// <summary>必要な冷温水流量[kg/s]を計算する</summary>
 2 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
 3 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
 4 /// <param name="borderRelativeHumidity">乾湿境界での空気相対湿度[%]</param>
 5 /// <param name="inletWaterTemperature">入口水温[C]</param>
 6 /// <param name="velocity">風速[m/s]</param>
 7 /// <param name="ratedWaterSpeed">定格水速[m/s]</param>
 8 /// <param name="airFlowRate">風量[kg/s]</param>
 9 /// <param name="ratedWaterFlowRate">定格冷温水量[kg/s]</param>
10 /// <param name="maxWaterFlowRate">最大冷温水量[kg/s]</param>
11 /// <param name="surfaceArea">伝熱面積[m2]</param>
12 /// <param name="outletAirTemperatureSetpoint">出口空気乾球温度設定値[C]</param>
13 /// <returns>冷温水流量[kg/s]</returns>
14 public static double GetWaterFlowRate(
15   double inletAirTemperature, double inletAirHumidityRatio,
16   double borderRelativeHumidity, double inletWaterTemperature,
17   double velocity, double ratedWaterSpeed,
18   double airFlowRate, double ratedWaterFlowRate, double maxWaterFlowRate,
19   double surfaceArea, double outletAirTemperatureSetpoint)
20 {
21   //冷却・加熱の判定
22   bool isCooling = (inletWaterTemperature < inletAirTemperature);
23
24   //冷却・加熱不要の場合
25   if (isCooling && inletAirTemperature < outletAirTemperatureSetpoint
26     || !isCooling && outletAirTemperatureSetpoint < inletAirTemperature) return 0;
27
28   double wc = ratedWaterSpeed / ratedWaterFlowRate;
29
30   //最大水量で成り行き出口温度を計算
31   double oat, oah, owt, dr, kd, kw;
32   GetHeatTransferCoefficient(wc * maxWaterFlowRate, velocity, out kd, out kw);
33   GetOutletState(inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity,
34     inletWaterTemperature, airFlowRate, maxWaterFlowRate, kd, kw, surfaceArea,
35     out oat, out oah, out owt, out dr);
36
37   //過負荷の場合には最大水量を出力
38   if ((isCooling && (outletAirTemperatureSetpoint < oat))
39     || (!isCooling && (oat < outletAirTemperatureSetpoint)))
40     return maxWaterFlowRate;
41
42   //負荷が処理可能な場合は水量を Brent 法で収束計算
43   //誤差関数を定義
44   Roots.ErrorFunction eFnc = delegate (double wFlow)
45   {
46     GetHeatTransferCoefficient(wc * wFlow, velocity, out kd, out kw);
47     GetOutletState(inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity,
48       inletWaterTemperature, airFlowRate, wFlow, kd, kw, surfaceArea,
49       out oat, out oah, out owt, out dr);
50     return outletAirTemperatureSetpoint - oat;
51   };
52   return Roots.Brent(0, maxWaterFlowRate, 0.01, eFnc);
53 }
54
55 /// <summary>必要な冷温水流量[kg/s]を計算する</summary>
56 /// <param name="airWaterSurfaceRatio">空気側・水側伝熱面積比[-]</param>
57 /// <param name="coreArea">コア面積[m2]</param>
```

```
 58 /// <param name="equivalentFinRadius">環状フィンの相当半径[m]</param>
 59 /// <param name="equivalentDiameter">等価直径[m]</param>
 60 /// <param name="waterPath">配管の流路数[-]</param>
 61 /// <param name="finThickness">フィン厚み[m]</param>
 62 /// <param name="thermalConductivity">フィン材料の熱伝導率[W/(mK)]</param>
 63 /// <param name="innerDiameter">管の内径[m]</param>
 64 /// <param name="outerDiameter">管の外径[m]</param>
 65 /// <param name="airFlowRate">風量[kg/s]</param>
 66 /// <param name="inletAirTemperature">入口空気温度[C]</param>
 67 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
 68 /// <param name="borderRelativeHumidity">乾湿境界での空気相対湿度[%]</param>
 69 /// <param name="waterFlowRate">水量[kg/s]</param>
 70 /// <param name="inletWaterTemperature">入口水温[C]</param>
 71 /// <param name="maxWaterFlowRate">最大冷温水量[kg/s]</param>
 72 /// <param name="surfaceArea">伝熱面積[m2]</param>
 73 /// <param name="outletAirTemperatureSetpoint">出口空気乾球温度設定値[C]</param>
 74 /// <returns>冷温水流量[kg/s]</returns>
 75 public static double GetWaterFlowRate(
 76   double airWaterSurfaceRatio, double coreArea, double equivalentFinRadius,
 77   double equivalentDiameter, double waterPath,double finThickness,
 78   double thermalConductivity, double innerDiameter, double outerDiameter,
 79   double airFlowRate, double inletAirTemperature, double inletAirHumidityRatio,
 80   double borderRelativeHumidity, double waterFlowRate, double inletWaterTemperature,
 81   double maxWaterFlowRate, double surfaceArea, double outletAirTemperatureSetpoint)
 82 {
 83   //冷却・加熱の判定
 84   bool isCooling = (inletWaterTemperature < inletAirTemperature);
 85
 86   //冷却・加熱不要の場合
 87   if (isCooling && inletAirTemperature < outletAirTemperatureSetpoint
 88     || !isCooling && outletAirTemperatureSetpoint < inletAirTemperature) return 0;
 89
 90   //最大水量で成り行き出口温度を計算
 91   double oat, oah, owt, dr, kd, kw;
 92   GetHeatTransferCoefficient(airWaterSurfaceRatio, coreArea, equivalentFinRadius,
 93     equivalentDiameter, waterPath, finThickness, thermalConductivity,
 94     innerDiameter, outerDiameter, airFlowRate, inletAirHumidityRatio,
 95     inletAirHumidityRatio, borderRelativeHumidity, maxWaterFlowRate,
 96     inletWaterTemperature, out kd, out kw);
 97   GetOutletState(inletAirTemperature, inletAirHumidityRatio,
 98     borderRelativeHumidity, inletWaterTemperature,
 99     airFlowRate, maxWaterFlowRate, kd, kw, surfaceArea,
100     out oat, out oah, out owt, out dr);
101
102   //過負荷の場合には最大水量を出力
103   if ((isCooling && (outletAirTemperatureSetpoint < oat))
104     || (!isCooling && (oat < outletAirTemperatureSetpoint)))
105     return maxWaterFlowRate;
106
107   //負荷が処理可能な場合は水量を Brent 法で収束計算
108   //誤差関数を定義
109   Roots.ErrorFunction eFnc = delegate(double wFlow)
110   {
111     GetHeatTransferCoefficient(airWaterSurfaceRatio, coreArea, equivalentFinRadius,
112       equivalentDiameter, waterPath, finThickness, thermalConductivity,
113       innerDiameter, outerDiameter, airFlowRate, inletAirHumidityRatio,
114       inletAirHumidityRatio, borderRelativeHumidity, wFlow,
115       inletWaterTemperature, out kd, out kw);
116     GetOutletState(inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity,
117       inletWaterTemperature, airFlowRate, wFlow, kd, kw, surfaceArea,
118       out oat, out oah, out owt, out dr);
119     return outletAirTemperatureSetpoint - oat;
120   };
121   return Roots.Brent(0, maxWaterFlowRate, 0.01, eFnc);
122 }
```

【例題 9.3】

下記仕様の冷却除湿コイルについて、1）冷水量を変化させた場合の処理熱量の変化、2）出口空気乾球温度を変化させた場合の必要水量の変化、をそれぞれ計算せよ。

定格条件：

風量：6,750 CMH、水量：300 L/min、入口水温：5 ºC、入口空気温度：32 ºC

入口絶対湿度：0.0125 kg/kg、熱交換量：70 kW、乾湿境界相対湿度：95 %

【解】

プログラムを 9.7 に示す。4~12 行は計算に用いる変数、15~21 行は定格条件と能力である。26~32 行では、定格条件における熱貫流率を計算し、伝熱面積を求める。33~50 行で水温変化に対する感度解析を

行っている。簡易モデルの場合には水温が変化しても熱貫流率に変化は無いため、単純に水温を変化させ、成り行きでの出口状態を計算する。51~78 行は出口空気乾球温度に対する感度解析である。60 行で必要な水量を求め、64~71 行で当該水量における出口状態を計算する。

プログラム 9.7　簡易コイルモデルの感度解析

```
 1 /// <summary>簡易モデルで感度解析を行う</summary>
 2 private static void PlateFinHeatExchangerTest1()
 3 {
 4   //モデルパラメータ
 5   double kd, kw;  //熱貫流率
 6   double sArea;   //伝熱面積
 7
 8   //出力
 9   double outletAirTemperature;
10   double outletAirHumidity;
11   double outletWaterTemperature;
12   double dryRate;
13
14   //定格条件と能力
15   const double inletAirTemperature = 32;         //C
16   const double inletAirHumidityRatio = 0.0125;   //kg/kg
17   const double borderRelativeHumidity = 90;      //%
18   const double inletWaterTemperature = 5;        //C
19   const double heatExchange = 70;                //kW
20   const double waterFlowRate = 300d / 60;        //kg/s
21   const double airFlowRate = 6750d * 1.2 / 3600; //kg/s
22
23   //簡易モデルの計算/////////////////////////////////////////////
24   double velocity = 2.5;
25   double waterSpeed = 1.5;
26   //熱貫流率の計算
27   CrossFinHeatExchanger.GetHeatTransferCoefficient(waterSpeed, velocity, out kd, out kw);
28   //伝熱面積の計算
29   sArea = CrossFinHeatExchanger.GetSurfaceArea
30     (inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity,
31     inletWaterTemperature, airFlowRate, waterFlowRate, heatExchange, kd, kw);
32   //感度解析1（入口水温に対する出口状態の変化）
33   Console.WriteLine("簡易モデル（水温変化）");
34   Console.WriteLine("入口水温[C], 出口空気温度[C], 出口水温[C], 熱交換量[kW]");
35   for (int i = 0; i < 10; i++)
36   {
37     double iwt = 3 + 0.5 * i;
38     //簡易モデルでは温度変化によって熱貫流率は変わらない
39     CrossFinHeatExchanger.GetOutletState
40       (inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity, iwt,
41       airFlowRate, waterFlowRate, kd, kw, sArea, out outletAirTemperature,
42       out outletAirHumidity, out outletWaterTemperature, out dryRate);
43     //熱量計算[kW]
44     double hl = (outletWaterTemperature - iwt) * 4.186 * waterFlowRate;
45     //書き出し処理
46     Console.WriteLine(iwt.ToString("F2") + ", " + outletAirTemperature.ToString("F2") + ", " +
48       outletWaterTemperature.ToString("F2") + ", " + hl.ToString("F2"));
49   }
50   //感度解析2（空気温度設定値に対する必要水量の変化）
51   Console.WriteLine();
52   Console.WriteLine("簡易モデル（空気温度設定値変化）");
53   Console.WriteLine("空気温度設定値[kW], 出口空気温度[C], 水量[L/min], 熱交換量[kW]");
54   for (int i = 0; i < 10; i++)
55   {
56     double oats = 12 + 0.5 * i;
57
58     //必要水量[L/min]の計算
59     double wf = CrossFinHeatExchanger.GetWaterFlowRate
60       (inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity, inletWaterTemperature,
61       velocity, waterSpeed, airFlowRate, waterFlowRate, waterFlowRate, sArea, oats);
62     //熱貫流率の計算
63     CrossFinHeatExchanger.GetHeatTransferCoefficient
64       (waterSpeed * (wf / waterFlowRate), velocity, out kd, out kw);
65     //出口状態の計算
66     CrossFinHeatExchanger.GetOutletState
67       (inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity,
68       inletWaterTemperature, airFlowRate, wf, kd, kw, sArea, out outletAirTemperature,
69       out outletAirHumidity, out outletWaterTemperature, out dryRate);
70     //熱量計算[kW]
71     double hl = (outletWaterTemperature - inletWaterTemperature) * 4.186 * wf;
72     //書き出し処理
73     Console.WriteLine(oats.ToString("F2") + ", " + outletAirTemperature.ToString("F2") + ", " +
74       (wf * 60).ToString("F2") + ", " + hl.ToString("F2"));
75   }
```

```
47 }
```

計算結果をグラフ化すると図 9.8 が得られる。

左は入口水温に対する感度解析結果であり、出口温度と交換熱量はほぼ線形に変化することがわかる。右は出口空気温度に対する結果である。定格能力である 12.5 °C を下回る出口空気温度は実現できず、最大水量である 300 L/min で運転する。また、必要水量は出口空気温度設定値に対して非線形に減少する。

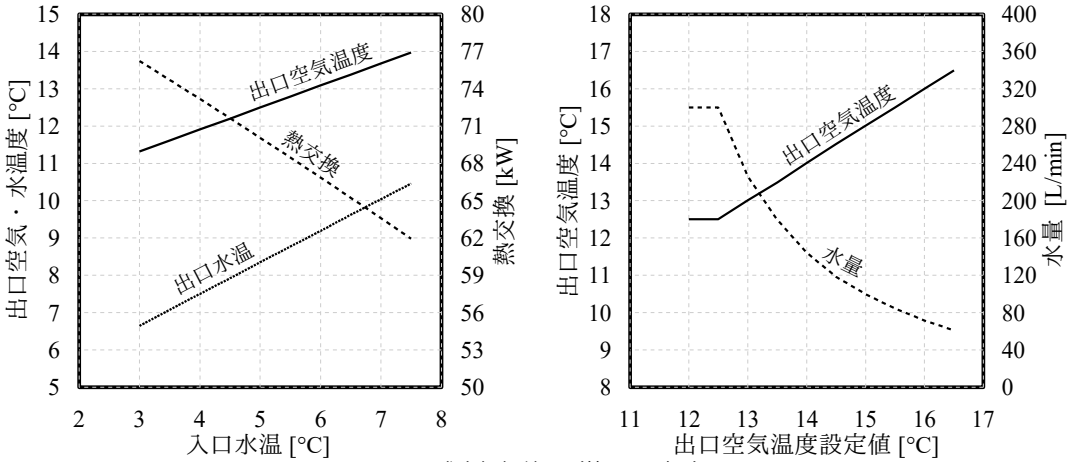


図 9.8　感度解析結果（簡易モデル）

9.3.5　「プレートフィン付管熱交換器クラス」の作成

プログラム 9.7 に示したとおり、静的関数のみを用いて諸々の解析を行うと、多くのパラメータを管理する必要があり、プログラムが読みづらくなる。パラメータ数が多い詳細モデルは尚更である。そこで、プレートフィン付管熱交換器のクラスを作成し、パラメータをインスタンス変数として管理する。

プログラム 9.8 にインスタンス変数およびプロパティを示す。インスタンス変数はいずれもコンストラクタで初期化した後に変更しない値であるため、修飾子として readonly を設定する。

プログラム 9.8　プレートフィン付管熱交換器クラスのインスタンス変数

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
/// <summary>詳細モデルか否か</summary>
private readonly bool isDetailedModel;

/// <summary>乾湿境界の空気相対湿度[%]</summary>
private readonly double borderRelativeHumidity;

/// <summary>詳細モデルのコイル仕様</summary>
private readonly int rowNumber, columnNumber;
private readonly double airWaterSurfaceRatio, coreArea,
  equivalentFinRadius, equivalentDiameter, waterPath,
  finThickness, thermalConductivity, innerDiameter, outerDiameter;

/// <summary>簡易モデルのコイル仕様</summary>
private readonly double ratedVelocity, ratedWaterSpeed;

/// <summary>最大水量[kg/s]を取得する</summary>
public double MaxWaterFlowRate { get; private set; }

/// <summary>定格水量[kg/s]を取得する</summary>
public double RatedWaterFlowRate { get; private set; }

/// <summary>水量[kg/s]を取得する</summary>
public double WaterFlowRate { get; private set; }

/// <summary>風量[kg/s]を取得する</summary>
public double AirFlowRate { get; private set; }

/// <summary>定格風量[kg/s]を取得する</summary>
public double RatedAirFlowRate { get; private set; }

/// <summary>入口空気の乾球温度[C]を取得する</summary>
```

```
32 public double InletAirTemperature { get; private set; }
33
34 /// <summary>入口空気の絶対湿度[kg/kg]を取得する</summary>
35 public double InletAirHumidityRatio { get; private set; }
36
37 /// <summary>出口空気の乾球温度[C]を取得する</summary>
38 public double OutletAirTemperature { get; private set; }
39
40 /// <summary>出口空気の絶対湿度[kg/kg]を取得する</summary>
41 public double OutletAirHumidityRatio { get; private set; }
42
43 /// <summary>入口水温[C]を取得する</summary>
44 public double InletWaterTemperature { get; private set; }
45
46 /// <summary>出口水温[C]を取得する</summary>
47 public double OutletWaterTemperature { get; private set; }
48
49 /// <summary>伝熱面積[m2]を取得する</summary>
50 public double SurfaceArea { get; private set; }
51
52 /// <summary>乾きコイルの比率[-]を取得する</summary>
53 public double DryRate { get; private set; }
54
55 /// <summary>乾きコイルの熱貫流率[kW/(m2K)]を取得する</summary>
56 public double DryHeatTransferCoefficient { get; private set; }
57
58 /// <summary>湿りコイルの熱貫流率[kW/(m2(kJ/kg))]を取得する</summary>
59 public double WetHeatTransferCoefficient { get; private set; }
60
61 /// <summary>表面積補正係数[-]を取得する</summary>
62     public double CorrectionFactor { get; private set; }
63
64 /// <summary>交換熱量[kW]を取得する</summary>
65 public double HeatTransfer
66 {
67   get
68   {
69     return (OutletWaterTemperature - InletWaterTemperature) * WaterFlowRate * WATER_SPECIFIC_HEAT;
70   }
71 }
```

コンストラクタをプログラム 9.9 に示す。オーバーロードを行い、22~84 行に示す詳細モデル用コンストラクタと 97~128 行に示す簡易モデル用コンストラクタの 2 種類を定義する。32 行では、簡易モデルか詳細モデルかの情報を保存する。34~51 行では与えられた情報からコイルの幾何学形状を計算して保存する。61~83 行では定格条件での熱貫流率を計算し、定格能力から伝熱面積を逆算して保存する。簡易モデルのコンストラクタも同様であり、106~116 行で各種の仕様を保存し、118~127 行で伝熱面積を逆算する。

プログラム 9.9　プレートフィン付管熱交換器クラスのコンストラクタ

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
 1 /// <summary>コンストラクタ（詳細モデル）</summary>
 2 /// <param name="depth">奥行き[m]</param>
 3 /// <param name="width">幅[m]</param>
 4 /// <param name="height">高さ[m]</param>
 5 /// <param name="rowNumber">列数[列]</param>
 6 /// <param name="columnNumber">段数[段]</param>
 7 /// <param name="finPitch">フィンピッチ[m]</param>
 8 /// <param name="finThickness">フィン厚み[m]</param>
 9 /// <param name="thermalConductivity">フィン材料の熱伝導率[W/(mK)]</param>
10 /// <param name="innerDiameter">管の内径[m]</param>
11 /// <param name="outerDiameter">管の外径[m]</param>
12 /// <param name="ratedAirFlowRate">定格風量[kg/s]</param>
13 /// <param name="ratedInletAirTemperature">定格入口空気乾球温度[C]</param>
14 /// <param name="ratedInletAirHumidityRatio">定格入口空気絶対湿度[kg/kg]</param>
15 /// <param name="borderRelativeHumidity">乾湿境界での空気相対湿度[%]</param>
16 /// <param name="ratedWaterFlowRate">定格水量[kg/s]</param>
17 /// <param name="maxWaterFlowRate">最大水量[kg/s]</param>
18 /// <param name="ratedInletWaterTemperature">定格入口水温[C]</param>
19 /// <param name="flowFactor">フロー倍率</param>
20 /// <param name="heatTransfer">定格能力[kW]</param>
21 /// <param name="useCorrectionFactor">修正係数を用いるか否か</param>
22 public CrossFinHeatExchanger(double depth, double width, double height,
23   int rowNumber, int columnNumber, double finPitch, double finThickness,
24   double thermalConductivity, double innerDiameter, double outerDiameter,
25   double ratedAirFlowRate, double ratedInletAirTemperature,
26   double ratedInletAirHumidityRatio, double borderRelativeHumidity,
```

```
27   double ratedWaterFlowRate, double maxWaterFlowRate,
28   double ratedInletWaterTemperature, double flowFactor,
29   double heatTransfer, bool useCorrectionFactor)
30 {
31   //詳細モデルによる初期化
32   isDetailedModel = true;
33
34   //コイルの幾何学形状を計算
35   double asr, car, eqr, eqd, asa;
36   GetGeometricCompfigulation(depth, width, height, rowNumber, columnNumber,
37     finPitch, finThickness, innerDiameter, outerDiameter,
38     out asr, out car, out eqr, out eqd, out asa);
39
40   //コイル仕様を保存
41   this.airWaterSurfaceRatio = asr;
42   this.coreArea = car;
43   this.equivalentFinRadius = eqr;
44   this.equivalentDiameter = eqd;
45   this.waterPath = flowFactor * columnNumber;
46   this.rowNumber = rowNumber;
47   this.columnNumber = columnNumber;
48   this.finThickness = finThickness;
49   this.thermalConductivity = thermalConductivity;
50   this.innerDiameter = innerDiameter;
51   this.outerDiameter = outerDiameter;
52
53   //その他のコイル仕様を保存
54   this.RatedAirFlowRate = ratedAirFlowRate;
55   this.RatedWaterFlowRate = ratedWaterFlowRate;
56   this.MaxWaterFlowRate = maxWaterFlowRate;
57
58   //乾湿境界での空気の相対湿度を保存
59   this.borderRelativeHumidity = borderRelativeHumidity;
60
61   if (useCorrectionFactor)
62   {
63     //熱貫流率を計算する
64     double kd, kw;
65     GetHeatTransferCoefficient(airWaterSurfaceRatio, coreArea, equivalentFinRadius,
66       equivalentDiameter, waterPath, finThickness, thermalConductivity, innerDiameter,
67       outerDiameter, RatedAirFlowRate, ratedInletAirTemperature,
68       ratedInletAirHumidityRatio, borderRelativeHumidity, RatedWaterFlowRate,
69       ratedInletWaterTemperature, out kd, out kw);
70     DryHeatTransferCoefficient = kd;
71     WetHeatTransferCoefficient = kw;
72
73     //伝熱面積[m2]を取得する
74     SurfaceArea = GetSurfaceArea(ratedInletAirTemperature, ratedInletAirHumidityRatio,
75       borderRelativeHumidity, ratedInletWaterTemperature, ratedAirFlowRate,
76       ratedWaterFlowRate, heatTransfer, kd, kw);
77     CorrectionFactor = SurfaceArea / (asa * rowNumber);
78   }
79   else
80   {
81     SurfaceArea = asa * rowNumber;
82     CorrectionFactor = 1.0d;
83   }
84 }
85
86 /// <summary>コンストラクタ（簡易モデル）</summary>
87 /// <param name="ratedAirFlowVolume">定格風量[kg/s]</param>
88 /// <param name="ratedVelocity">定格前面風速[m/s]</param>
89 /// <param name="ratedInletAirTemperature">定格入口空気乾球温度[C]</param>
90 /// <param name="ratedInletAirHumidityRatio">定格入口空気絶対湿度[kg/kg]</param>
91 /// <param name="borderRelativeHumidity">乾湿境界相対湿度[%]</param>
92 /// <param name="ratedWaterFlowRate">定格水量[kg/s]</param>
93 /// <param name="ratedWaterSpeed">定格水速[m/s]</param>
94 /// <param name="maxWaterFlowRate">最大水量[kg/s]</param>
95 /// <param name="ratedInletWaterTemperature">定格入口水温[C]</param>
96 /// <param name="heatTransfer">定格能力[kW]</param>
97 public CrossFinHeatExchanger(
98   double ratedAirFlowVolume, double ratedVelocity, double ratedInletAirTemperature,
99   double ratedInletAirHumidityRatio, double borderRelativeHumidity,
100  double ratedWaterFlowRate, double ratedWaterSpeed, double maxWaterFlowRate,
101  double ratedInletWaterTemperature, double heatTransfer)
102 {
103   //簡易モデルによる初期化
104   isDetailedModel = false;
105
106   //簡易モデルのコイル仕様を保存
```

```
107   this.ratedVelocity = ratedVelocity;
108   this.ratedWaterSpeed = ratedWaterSpeed;
109
110   //その他のコイル仕様を保存
111   this.RatedAirFlowVolume = ratedAirFlowVolume;
112   this.RatedWaterFlowRate = ratedWaterFlowRate;
113   this.MaxWaterFlowRate = maxWaterFlowRate;
114
115   //乾湿境界での空気の相対湿度を保存
116   this.borderRelativeHumidity = borderRelativeHumidity;
117
118   //熱貫流率を計算する
119   double kd, kw;
120   GetHeatTransferCoefficient(ratedWaterSpeed, ratedVelocity, out kd, out kw);
121   DryHeatTransferCoefficient = kd;
122   WetHeatTransferCoefficient = kw;
123
124   //伝熱面積[m2]を取得する
125   SurfaceArea = GetSurfaceArea(ratedInletAirTemperature, ratedInletAirHumidityRatio,
126     borderRelativeHumidity, ratedInletWaterTemperature, ratedAirFlowVolume,
127     ratedWaterFlowRate, heatTransfer, kd, kw);
128 }
```

詳細モデルのコンストラクタのパラメータ数は非常に多く、常にこれを用いることは不便である。そこで、いくつかのパラメータを自動設定する簡略化したコンストラクタを定義する。プログラム 9.10 にコンストラクタを示す。

プログラム 9.10　プレートフィン付管熱交換器クラスのコンストラクタ 2

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
 1 /// <summary>インスタンスを初期化する（詳細モデル：自動初期化）</summary>
 2 /// <param name="width">幅[m]</param>
 3 /// <param name="height">高さ[m]</param>
 4 /// <param name="rowNumber">列数[列]</param>
 5 /// <param name="columnNumber">段数[段]</param>
 6 /// <param name="ratedAirFlowRate">定格風量[kg/s]</param>
 7 /// <param name="ratedInletAirTemperature">定格入口空気乾球温度[C]</param>
 8 /// <param name="ratedInletAirHumidityRatio">定格入口空気絶対湿度[kg/kg]</param>
 9 /// <param name="borderRelativeHumidity">乾湿境界での空気相対湿度[%]</param>
10 /// <param name="ratedWaterFlowRate">定格水量[kg/s]</param>
11 /// <param name="maxWaterFlowRate">最大水量[kg/s]</param>
12 /// <param name="ratedInletWaterTemperature">定格入口水温[C]</param>
13 /// <param name="flowType">フロータイプ</param>
14 /// <param name="heatTransfer">定格能力[kW]</param>
15 /// <param name="useCorrectionFactor">修正係数を用いるか否か</param>
16 public CrossFinHeatExchanger(
17   double width, double height, int rowNumber, int columnNumber, double ratedAirFlowRate,
18   double ratedInletAirTemperature, double ratedInletAirHumidityRatio, double borderRelativeHumidity,
19   double ratedWaterFlowRate, double maxWaterFlowRate, double ratedInletWaterTemperature,
20   WaterFlowType flowType, double heatTransfer, bool useCorrectionFactor)
21   : this(rowNumber * 0.0329, width, height, rowNumber, columnNumber, 0.0029, 0.0002, 237, 0.0146, 0.0158,
22   ratedAirFlowRate, ratedInletAirTemperature, ratedInletAirHumidityRatio, borderRelativeHumidity,
23   ratedWaterFlowRate, maxWaterFlowRate, ratedInletWaterTemperature, getFlowFactor(flowType),
24   heatTransfer, useCorrectionFactor)
25 { }
26
27 /// <summary>フロータイプからフロー倍率を計算する</summary>
28 /// <param name="wType">フロータイプ</param>
29 /// <returns>フロー倍率</returns>
30 private static double getFlowFactor(WaterFlowType wType)
31 {
32   if (wType == WaterFlowType.HalfFlow) return 0.5;
33   else if (wType == WaterFlowType.SingleFlow) return 1.0;
34   else if (wType == WaterFlowType.DoubleFlow) return 2.0;
35   else return 3.0;
36 }
```

プログラム 9.11 に出口状態の計算を示す。各種のパラメータがインスタンス変数として保存されているため、プログラム 9.5 に比較して引数がはるかに少ないことがわかる。これがインスタンスを作成するメリットである。27~49 行は熱貫流率の計算処理であり、簡易モデルか詳細モデルかの別に応じて呼び出す静的メソッドを切り替える。53 行で出口状態を計算し、結果をプロパティに保存する。

プログラム **9.11**　出口状態の計算（インスタンスメソッド）

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
/// <summary>出口状態を計算する</summary>
/// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
/// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
/// <param name="inletWaterTemperature">入口水温[C]</param>
/// <param name="airFlowRate">風量[kg/s]</param>
/// <param name="waterFlowRate">水量[kg/s]</param>
public void UpdateOutletState(double inletAirTemperature, double inletAirHumidityRatio,
  double inletWaterTemperature, double airFlowRate, double waterFlowRate)
{
  //入力値を保存
  InletAirTemperature = inletAirTemperature;
  InletAirHumidityRatio = inletAirHumidityRatio;
  InletWaterTemperature = inletWaterTemperature;
  AirFlowRate = airFlowRate;
  WaterFlowRate = waterFlowRate;

  //熱媒が流れていない場合
  if (AirFlowRate <= 0 || WaterFlowRate <= 0)
  {
    OutletAirTemperature = InletAirTemperature;
    OutletAirHumidityRatio = InletAirHumidityRatio;
    OutletWaterTemperature = InletWaterTemperature;
    DryRate = 1.0;
    return;
  }

  //熱貫流率を計算
  if (isDetailedModel)
  {
    //詳細モデル
    double kd, kw;
    GetHeatTransferCoefficient(airWaterSurfaceRatio, coreArea, equivalentFinRadius, equivalentDiameter,
      waterPath, finThickness, thermalConductivity, innerDiameter, outerDiameter, AirFlowRate,
      InletAirTemperature, InletAirHumidityRatio, BorderRelativeHumidity, WaterFlowRate,
      InletWaterTemperature, out kd, out kw);
    DryHeatTransferCoefficient = kd;
    WetHeatTransferCoefficient = kw;
  }
  else
  {
    //簡易モデル
    double kd, kw;
    //水速と風速を計算//風量と水量に比例
    double velocity = (AirFlowRate / RatedAirFlowRate) * ratedVelocity;
    double waterSpeed = (WaterFlowRate / RatedWaterFlowRate) * ratedWaterSpeed;
    GetHeatTransferCoefficient(waterSpeed, velocity, out kd, out kw);
    DryHeatTransferCoefficient = kd;
    WetHeatTransferCoefficient = kw;
  }

  //出口状態を計算
  double ta, xa, tw, dr;
  GetOutletState(InletAirTemperature, InletAirHumidityRatio, BorderRelativeHumidity,
    InletWaterTemperature, AirFlowRate, WaterFlowRate, DryHeatTransferCoefficient,
    WetHeatTransferCoefficient, SurfaceArea, out ta, out xa, out tw, out dr);
  OutletAirTemperature = ta;
  OutletAirHumidityRatio = xa;
  OutletWaterTemperature = tw;
  DryRate = dr;
}
```

プログラム 9.12 に必要水量の計算を示す。出口状態の計算と同様、パラメータがインスタンス変数に保存されているため、静的メソッドに比較して引数の数が少ない。中身の処理はプログラム 9.6 とほぼ同様である。

プログラム **9.12**　必要水量の計算（インスタンスメソッド）

Popolo.HVAC.HeatExchanger.CrossFinHeatExchanger class

```
/// <summary>出口空気温度を制御する</summary>
/// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
/// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
/// <param name="inletWaterTemperature">入口水温[C]</param>
/// <param name="airFlowRate">風量[kg/s]</param>
/// <param name="outletAirTemperatureSetpoint">出口空気乾球温度設定値[C]</param>
/// <returns>制御可能か否か</returns>
public bool ControlOutletAirTemperature(double inletAirTemperature, double inletAirHumidityRatio,
```

```
 9   double inletWaterTemperature, double airFlowRate, double outletAirTemperatureSetpoint)
10 {
11   //入力値を保存
12   InletAirTemperature = inletAirTemperature;
13   InletAirHumidityRatio = inletAirHumidityRatio;
14   InletWaterTemperature = inletWaterTemperature;
15   AirFlowRate = airFlowRate;
16
17   //冷却・加熱の判定
18   bool isCooling = (inletWaterTemperature < inletAirTemperature);
19
20   //冷却・加熱不要の場合
21   if (isCooling && inletAirTemperature < outletAirTemperatureSetpoint + 1e-3 ||
22     !isCooling && outletAirTemperatureSetpoint < inletAirTemperature + 1e-3)
23   {
24     ShutOff();
25     return false;
26   }
27
28   //最大水量で成り行き出口温度を計算
29   UpdateOutletState
30     (InletAirTemperature, InletAirHumidityRatio, InletWaterTemperature, AirFlowRate, MaxWaterFlowRate);
31
32   //過負荷の場合には最大能力での成り行き状態を出力
33   if ((isCooling && (outletAirTemperatureSetpoint < OutletAirTemperature))
34     || (!isCooling && (OutletAirTemperature < outletAirTemperatureSetpoint)))
35     return false;
36
37   //負荷が処理可能な場合は水量をBrent法で収束計算
38   //誤差関数を定義
39   Roots.ErrorFunction eFnc = delegate(double wFlow)
40   {
41     UpdateOutletState
42     (InletAirTemperature, InletAirHumidityRatio, InletWaterTemperature, AirFlowRate, wFlow);
43     return outletAirTemperatureSetpoint - OutletAirTemperature;
44   };
45   double wf = Roots.Brent(0, MaxWaterFlowRate, MaxWaterFlowRate * 0.001, eFnc);
46   UpdateOutletState(InletAirTemperature, InletAirHumidityRatio, InletWaterTemperature, AirFlowRate, wf);
47   OutletAirTemperature = outletAirTemperatureSetpoint;
48   return true;
49 }
50
51 /// <summary>停止させる</summary>
52 public void ShutOff()
53 {
54   OutletAirTemperature = InletAirTemperature;
55   OutletAirHumidityRatio = InletAirHumidityRatio;
56   OutletWaterTemperature = InletWaterTemperature;
57   WaterFlowRate = 0;
58   DryRate = 1;
59 }
```

【例題 9.4】

下記仕様の冷却除湿コイルについて冷水量を変化させた場合の処理熱量の変化を計算せよ。ただし、列数に関しては4列、6列、8列のそれぞれについて計算するものとする。

形状：

高さ：750 mm、幅：1,000 mm、段数：20 段、奥行き：列数×32.9 mm

フィン厚み：0.2 mm、フィンピッチ：2.9 mm、材料：アルミ（237 W/(m·K)）

管外径：15.8 mm、管内径：14.6 mm

定格条件：

風量：6,750 CMH、水量：300 L/min、入口水温：5 ºC、入口空気温度：32 ºC

入口絶対湿度：0.0125 kg/kg、乾湿境界相対湿度：95 %

熱交換量：52 kW（4列）、70 kW（6列）、79 kW（8列）

【解】

プログラムを 9.13 に示す。4~21 行は定格条件、パラメータなどである。24~35 行でプレート熱交換器のインスタンス（4列、6列、8列）を作成する。47 行で出口状態を更新して、49 行で出力処理を行う。プログラム 9.7 に比較すると、各種のパラメータがインスタンス変数として保存されているため、プログラムコードが極めて簡易なことが確認できる[†1]。計算結果をグラフ化すると図 9.9 が得られる。

†1　本例では4列、6列、8列の3種のプレートフィン付管形熱交換器を作成したが、このように複数のモデルを取り扱う際に、静的メソッドのみを用いて計算を行うことを想像してもらいたい。パラメータ管理が如何に大変になるかがわかる。

プログラム **9.13**　水量に対する感度解析

```
/// <summary>詳細モデルで感度解析を行う</summary>
private static void PlateFinHeatExchangerTest2()
{
  //定格条件・能力等
  const double thermalConductivity = 237; //熱伝導率[W/(mK)]
  const double inletAirTemperature = 32;  //C
  const double inletAirHumidityRatio = 0.0125;//kg/kg
  const double borderRelativeHumidity = 90;   //%
  const double inletWaterTemperature = 5; //C
  const double waterFlowRate = 300d / 60; //kg/s
  const double airFlowRate = 6750d * 1.2 / 3600; //kg/s
  double[] cap = new double[] { 52, 70, 79 }; //kW

  //形状
  const double width = 1.0; //m
  const double height = 0.75;   //m
  const int columnNumber = 20;  //段
  const double finPitch = 0.0029;   //m
  const double finThickness = 0.0002;   //m
  const double innerDiameter = 0.0146;  //m
  const double outerDiameter = 0.0158;  //m

  //プレート熱交換器をインスタンス化
  CrossFinHeatExchanger[] pl = new CrossFinHeatExchanger[3];
  for (int i = 0; i < 3; i++)
  {
    int rNum = 4 + i * 2;
    double depth = rNum * 0.0329;
    pl[i] = new CrossFinHeatExchanger
      (depth, width, height, rNum, columnNumber, finPitch, finThickness,
      thermalConductivity, innerDiameter, outerDiameter, airFlowRate,
      inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity,
      waterFlowRate, waterFlowRate, inletWaterTemperature,
      CrossFinHeatExchanger.WaterFlowType.SingleFlow, cap[i], true);
  }

  Console.WriteLine("水量[L/min], 熱交換量(4列)[kW], 出口温度(4列)[C]," +
    "熱交換量(6列)[kW], 出口温度(6列)[C],熱交換量(8列)[kW], 出口温度(8列)[C]");
  for (int i = 0; i <= 30; i++)
  {
    //水量[L/min]を計算
    double wf = 10 * i;
    Console.Write(wf.ToString("F0") + ", ");
    //出口状態を更新して書き出し
    for (int j = 0; j < 3; j++)
    {
      pl[j].GetOutletState(inletAirTemperature,
        inletAirHumidityRatio, inletWaterTemperature, airFlowRate, wf / 60d);
      Console.Write(pl[j].HeatTransfer.ToString("F1") + ", " +
    pl[j].OutletAirTemperature.ToString("F1") + ", ");
    }
    Console.WriteLine();
  }
}
```

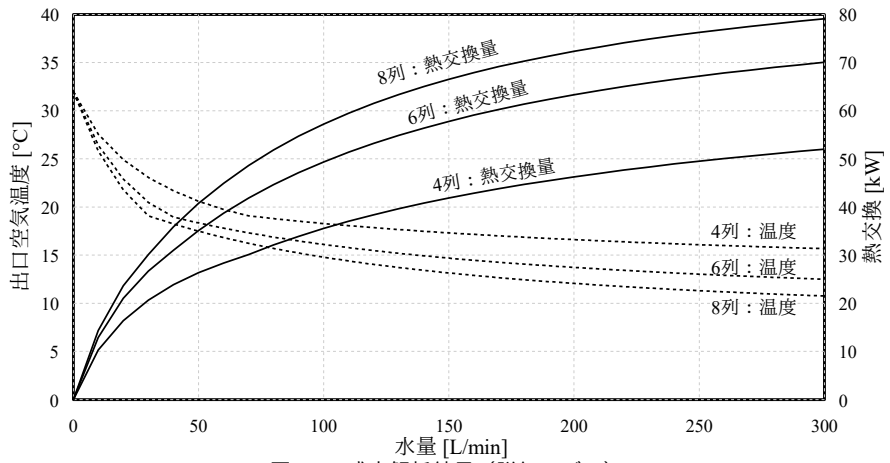


図 9.9　感度解析結果（詳細モデル）

【第 9 章 記号表】

A_c	：コア面積 **[m²]**	Q	：交換熱量 **[kW]**
a	：比エンタルピー近似係数 **1 [kJ/(kg·°C)]**	R_{dw}	：乾きコイル部分の比率 **[-]**
b	：比エンタルピー近似係数 **2 [kJ/kg]**	Re	：レイノルズ数 **[-]**
c_{pma}	：湿り空気の定圧比熱 **[kJ/(kg·K)]**	S	：伝熱面積 **[m²]**
D	：奥行き **[m]**	Sh	：シャーウッド数 **[-]**
d_e	：空気流路の相当直径 **[m]**	t	：温度 **[°C]**
d_i	：配管内径 **[m]**	W	：幅 **[m]**
d_o	：配管外径 **[m]**	W	：絶対湿度 **[kg/kg]**
H	：高さ **[m]**	x_e	：管状フィンの相当半径 **[m]**
h	：比エンタルピー **[kJ/kg]**	α_{av}	：拡散係数 **[m²/s]**
K	：熱貫流率 **[kW/(m²·K)]** **[kW/(m²·(kJ/kg))]**	α_f	：フィン側対流熱伝達率 **[W/(m²·K)]**
		α_S	：面積補正係数 **[-]**
k_f	：物質移動係数 **[W/(m²·(kJ/kg))]**	α_w	：水側対流熱伝達率 **[W/(m²·K)]**
Le	：ルイス数 **[-]**	γ_a	：比重量 **[kg/m³]**
m	：質量流量 **[kg/s]**	ε	：熱通過有効度 **[-]**
mc	：熱容量流量 **[kW/K]**	ε_p	：**1** 列の熱通過有効度 **[-]**
mc_{min}	：熱容量流量（小さい側） **[kW/K]**	λ	：熱伝導率 **[W/(m·K)]**
N_D	：段数 **[段]**	ν	：動粘性係数 **[m²/s]**
N_R	：列数 **[列]**	v_{ac}	：風速（コア面積基準） **[m/s]**
NTU	：移動単位数 **[-]**	v_{af}	：風速（正面面積基準） **[m/s]**
Nu	：ヌセルト数 **[-]**	v_w	：水速 **[m/s]**
P_f	：フィンピッチ **[m]**	ϕ	：フィン効率 **[-]**

添字：

a	：空気側	o	：出口
b	：乾湿境界	ti	：配管内表面
d	：乾き部分	to	：配管外表面
f	：フィン	w	：水側
i	：入口	w	：湿り部分
LM	：対数平均		

【第 9 章 参考文献】

9.1） 新晃工業 50 年史: 新晃工業株式会社, 2000

9.2） 石野久彌, 郡公子: 冷却コイルの詳細熱解析とその基本的応用に関する研究, 空気調和・衛生工学会論文集, Vol.23, pp.57-69, 1983

9.3） 石野久彌, 郡公子: 種々の冷却コイルの詳細熱解析と設計への応用に関する研究, 空気調和・衛生工学会論文集, Vol.32 , pp.1-12, 1986

9.4） 山口弘雅, 吉田治典, 丹羽英治, 渡邊剛, 宮田征門, 小田久人, 塩谷正樹: コミッショニングのための冷却コイル特性実験とモデル精度の検証, 空気調和・衛生工学会論文集, Vol.143, pp.61-70, 2009

9.5） Elmahdy, A. H. et al: A Simple Model for Cooling and Dehumidifying Coils for Use in Calculating Energy Requirements for Buildings, ASHRAE Transactions, Paper No. 2456, Vol. 83 Part 2, 1977

9.6） HVACSIM+(J) マニュアル, TYPE 602 冷却/除湿コイル, pp.6.103-6.118

9.7） 空調システム標準シミュレーションプログラム HASP/ACSS/8502 プログラム解説書, pp.105-110, 日本建築設備士協会

9.8） 井上宇一: 空気調和ハンドブック, 丸善株式会社, 2008

9.9） 李春夫: 空気調和システムのシミュレーション及び評価に関する研究, 早稲田大学博士論文, 昭和 57 年 5 月

9.10） 新津靖, 内藤和夫: フィン付き熱交換器の性能とその設計に関する研究（6）, 空気調和衛生工学, 41 巻, 6 号, pp.615-625, 1967

9.11） E. E. Allen: Polynomial approximations to some modified Bessel functions, Mathematics of Computation, 10, pp.162-164, 1956

9.12） Milton Abramowitz and Irene A. Stegun: Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, National Bureau of Standards Applied Mathematics Series 55, June, 1964

9.13） 伝熱ハンドブック, 日本機械学会, 1993

9.14） Gardner, K.A.: Efficiency of extended surface, Transactions of the American Society of Mechanical Engineers, Journal of Heat Transfer, No.65: pp.621-631, 1945

9.15） Kays, W. M. and London, A.L., Compact Heat Exchangers, 2nd ed., McGraw-Hill, New York, 1964

第10章　蒸発器 / 凝縮器
(Evaporator / Condensor)

10.1　概要

蒸発器と凝縮器は、冷媒と空気または水との熱交換を行う機器である。冷媒を状態変化させて熱交換を行う機器であるため、冷媒温度は概ね一定に保たれる。従って、液体との熱交換を行う機器に関しては、片側流体の温度が一定の場合の熱交換器として取り扱えば、第 8 章の理論式を用いて比較的容易に計算ができる。一方、湿り空気との熱交換を行う蒸発器の場合には、第 9 章で記したプレートフィン付管形熱交換器と同様に、湿り空気側で水分凝縮が発生するため、やや計算が複雑になる。特に蒸発器の場合には冷媒温度を氷点下に下げることもあり、この場合には水分凝縮に加え着霜も生じうる。また、湿り空気との熱交換を行う凝縮器の空気側は顕熱交換のみとなるが、近年、フィンに水を散水・噴霧して伝熱を促進する機器が生産されており、このような機器の計算のためには凝縮器フィン表面での水分蒸発を考慮する必要がある。そこで本章では、湿り空気との熱交換を行うプレートフィン付管形を取り上げることとし、冷却・除湿・着霜が生じる蒸発器、加熱・水分蒸発が生じる凝縮器の計算法をそれぞれ記す。水との熱交換を行うシェルアンドチューブ形の蒸発器および凝縮器の計算法に関しては、第 14 章の圧縮式冷凍機で解説する。

写真 10.1　パッケージ空調機室外機の凝縮器/蒸発器

10.2　理論

10.2.1　蒸発器

蒸発器は熱交換器内で冷媒を蒸発させて冷却を行う機器であり、冷媒を完全に蒸発させるものを乾式蒸発器、熱交換器内の大部分が液状体にあるものを満液式蒸発器と呼ぶ。前者はパッケージ冷凍機、後者はターボ冷凍機などの大型機器に採用されることが多い。満液式蒸発器の場合には出口冷媒は飽和蒸気であるが、乾式蒸発器の場合にはやや過熱状態となる。また、熱交換の対象からの分類も可能であり、水やブラインなど、液体を冷却する場合にはシェルアンドチューブ形、空気を冷却する場合にはプレートフィン付管形が用いられる。

パッケージ空調機で冷房を行う場合には、室内機に設置された熱交換器を蒸発器として用い、冷媒を蒸発させることで室内空気から熱を奪う。冷水やブラインなどの中間熱媒体を用いず、被冷却体を冷媒で直接的に冷却するため、この場合の蒸発器を直接膨張式コイル（直膨コイル: DX (Direct Expansion)コイル）と呼ぶこともある。また、直膨コイルをエアハンドリングユニットに組み込む場合もあり、このような機器を直膨エアハンと呼ぶ。室内の冷房を行う場合には、蒸発器は 20°C 程度の湿り空気と熱交換を行うため、冷却および除湿現象を解けばよく、第 9 章で記したプレートフィン付管形熱交換器の計算と同様の計算法を用いることができる。一方、室内の暖房を行う場合には、屋外設置の熱交換器を蒸発器として用い、外気から熱を奪うことが必要になる。厳冬期においては外気は 0°C 近辺にあり、水分凝縮に加えて凝固が生じるため、計算が複雑になる。このようにフィン表面に湿り空気の水分が凝固して氷が発生することを着霜と呼ぶ。着霜はフィンの伝熱性能を低下させ、最終的にはフィンの閉塞につながるため、これを取り除く運転（除霜運転と呼ぶ）が必要となる。除霜運転によって加熱効率は大きく影響を受けるため、除霜に必要なエネルギーを評価できるモデルとする必要がある。

冷却除湿着霜コイル内での湿り空気状態を図 10.1 に示す。

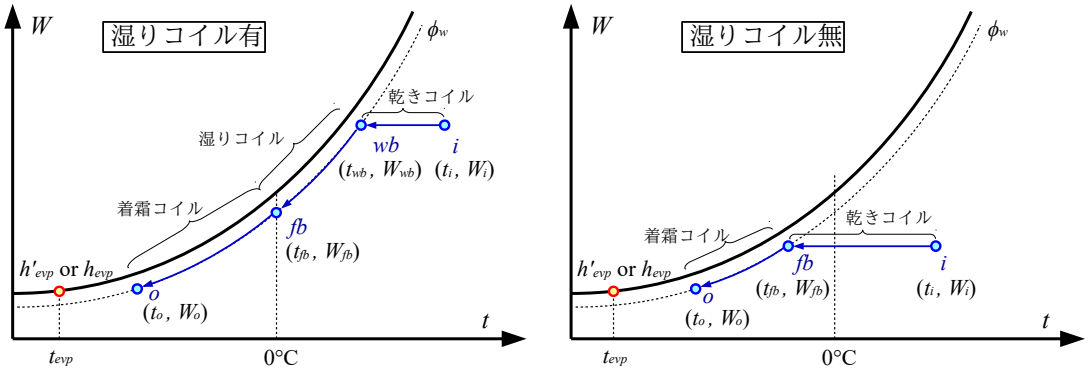


図 10.1　冷却除湿着霜コイル内での湿り空気状態

基本的な考え方は第 9 章のプレートフィン付管形熱交換器と同様である。バイパス効果により、湿り空気は相対湿度一定の線上を移動する。乾球温度が 0°C を下回ると凝縮した水分は凝固する。このコイルを着霜コイルと呼ぶことにする。図 10.1 に示すように、入口湿り空気の状態に応じて二種類の経路が考えられる。左は凝固点に達する前に露点に達する場合であり、乾きコイル、湿りコイル、着霜コイルの 3 つの領域に分けて解く必要がある。右は凝固点を下回ってから露点に達する場合であり、乾きコイルと着霜コイルに分けて解く必要がある。領域を区別するために、乾きコイルから湿り

コイルに移る点を添字の *wb* (Wet border)、湿りコイルから着霜コイルに移る点を添え字の *fb* (Frost border)で表現することにする。式 10.1 で示されるように 3 つの領域における熱交換量の合算が、蒸発器全体での熱交換量 Q_{evp} [kW]であり、各領域で基礎式を解く必要がある。

$$Q_{evp}=Q_d+Q_w+Q_f \tag{10.1}$$

1）乾きコイル

式 10.2 と式 10.3 に乾きコイルの基礎式を示す。式 10.2 は湿り空気の出入口温度差による熱交換量であり、Q_d は熱交換量[kW]、mc_a は湿り空気の熱容量流量[kW/K]である。式 10.3 は熱通過有効度にもとづく熱交換量であり、t_{evp} は蒸発温度[°C]である。片側温度一定（蒸発温度一定）の熱交換器のため、熱通過有効度 ε_d [-]は式 10.4 で解析的に求めることができる。K_d と S_d はそれぞれ乾きコイルの熱通過率[kW/(m²·K)]と伝熱面積[m²]である。

$$Q_d=mc_a\left(t_i-t_{wb}\right) \tag{10.2}$$

$$Q_d=\varepsilon_d mc_a\left(t_i-t_{evp}\right) \tag{10.3}$$

$$\varepsilon_d=1-\exp\left(-K_d S_d/mc_a\right) \tag{10.4}$$

乾きコイルの熱通過率 K_d は式 10.5 で計算する。v_a はコイル前面風速[m/s]、v_{sa} は基準前面風速[m/s]である。式 10.5 の係数は基準前面風速を 2.0m/s として、井上らの調査結果[10.2)]から回帰したものである。前面風速は 1.5~2.5m/s で計画されることが多く、また湿り空気の密度変化は小さく無視できると考えれば、式 10.6 で計算しても大きな誤差は生じないと予想する。ここで m_a は湿り空気の質量流量[kg/s]、m_{aN} は定格の質量流量[kg/s]である。

$$K_d=0.0236\left(v_a/v_{sa}\right)^{0.5479} \tag{10.5}$$

$$K_d=0.0236\left(m_a/m_{aN}\right)^{0.5479} \tag{10.6}$$

2）湿りコイル

式 10.7 と式 10.8 に湿りコイルの基礎式を示す[†1)]。式 10.7 は湿り空気の出入口エンタルピー差による熱交換量 Q_w の計算式である。式 10.3 は熱通過有効度 ε_w にもとづく熱交換量であり、ε_w [-]は式 10.9 で求める。$h_{S,evp}$[kJ/kg]は温度 t_{evp} における飽和空気の比エンタルピーであり式 10.10 で計算する。c_{pa} は乾き空気の定圧比熱（=1.006 kJ/(kg·K)）、γ_{v0} は 0°C の水の蒸発潜熱（2,501 kJ/kg）、c_{pv} は水蒸気の定圧比熱（1.805 kJ/(kg·K)）、$W_{S,evp}$ は温度 t_{evp} における飽和絶対湿度[kg/kg]である。K_w と S_w はそれぞれ湿りコイルの比エンタルピー基準の熱通過率[kW/(m²·(kJ/kg))]と伝熱面積[m²]である。

$$Q_w=m_a\left(h_{wb}-h_{fb}\right) \tag{10.7}$$

$$Q_w=\varepsilon_w m_a\left(h_{wb}-h_{S,evp}\right) \tag{10.8}$$

$$\varepsilon_w=1-\exp\left(-K_{wf} S_w/m_a\right) \tag{10.9}$$

$$h_{S,evp}=c_{pa} t_{evp}+\left(\gamma_{v0}+c_{pv} t_{evp}\right)W_{S,evp} \tag{10.10}$$

湿りコイルの比エンタルピー基準の熱通過率 K_w は、熱移動と物質移動のアナロジーにより、式 10.11 で計算する。c_{pma} は湿り空気の定圧比熱[kJ/(kg·K)]であるが、絶対湿度によって値が変化するため、式 10.12 に示すように乾湿境界点 wb と凝固点 fb での値の平均値を採用することにする。

†1　対数平均エンタルピー差による計算の理論については、第 12 章 冷却塔を参照。

$$K_{wf}=K_d/(c_{pa}+c_{pv}W)=K_d/c_{pma} \tag{10.11}$$

$$c_{pma,w}=0.5(c_{pma,wb}+c_{pma,fb}) \tag{10.12}$$

3）着霜コイル

着霜コイルの基礎式は湿りコイルとほぼ同様である。ただし、比エンタルピー h[kJ/kg]のかわりに式 10.16 で示される固体基準の比エンタルピー h'[kJ/kg]を用いる。ここで γ_{s0} は 0°C の氷の昇華潜熱（=2,837 kJ/kg）、c_{ps} は氷の定圧比熱（2.09 kJ/(kg·K)）である。また、着霜によってフィンの伝熱性能が低下するため式 10.15 に示すように、熱通過率 K_{wf} に修正係数 R_f を乗じる。着霜時の熱通過率に関しては青木らによる詳細な計算法が提案されているが[10.1]、フィンピッチや配管サイズなどを明らかにする必要があり、情報の入手が困難である。そこで本書では R_f=0.6 として計算を行う†1)。基準となる熱通過率 K_{wf} の計算は湿りコイル領域と同様に式 10.11 を用いるが、湿り空気比熱 c_{pma} としては、凝固点 fb での湿り空気比熱 $c_{pma,fb}$ を採用する。

$$Q_f=m_a(h'_{fb}-h'_o) \tag{10.13}$$

$$Q_f=\varepsilon_f m_a(h'_{fb}-h'_{S,evp}) \tag{10.14}$$

$$\varepsilon_f=1-\exp(-R_f K_{wf} S_f/m_a) \tag{10.15}$$

$$h'=c_{pa}t+(\gamma_{s0}+c_{ps}t)W \tag{10.16}$$

$$h'_{S,evp}=c_{pa}t_{evp}+(\gamma_{s0}+c_{ps}t_{evp})W_{S,evp} \tag{10.17}$$

フィンに着霜した水分は融点まで加熱して融解させる必要があり、このために必要な除霜運転の熱量は式 10.18 で計算できる。

$$Q_{df}=m_a(\gamma_{s0}-c_{ps}t_o)(W_{fb}-W_o) \tag{10.18}$$

【例題 10.1】

下記条件の蒸発器の必要伝熱面積を計算せよ。

蒸発温度　：-10 °C	吸込乾球温度：7 °C	風量	：1 kg/s
熱交換能力：15 kW	吸込湿球温度：6 °C	乾湿境界相対湿度	：95 %

【解】

乾きコイル、湿りコイル、着霜コイル、それぞれの伝熱面積を計算して合算することで必要伝熱面積を求める。

式 10.6 で m_a=m_{aN} として、乾きコイルの熱通過率 K_d は 0.0236 kW/(m²·K)である。入口空気条件により湿り空気の比熱 c_{pma} は 1.015 kJ/(kg·K)であり、これに質量流量を乗じて熱容量流量 m_{ca} は 1.015 kW/K となる。乾湿境界点（図 10.1 の wb）における湿り空気の状態は、吸込み空気の絶対湿度と乾湿境界相対湿度 95 %により計算ができ、5.7°C / 95 %である。従って、この点まで冷却した場合の熱交換量 Q_d は、式 10.2 を用いて、Q_d=(7.0 – 5.7)×1.015=1.32 kW である。Q_d<必要熱交換能力(15 kW)であるため、熱交換は乾きコイルで終了しない。式 10.3 を用いて乾きコイルの ε_d を計算すると、ε_d=Q_d÷(1.015×(7.0-(-10)))=0.077 である。式 10.4 に ε_d を代入すると乾きコイルの伝熱面積 S_d は、S_d=-ln(1-0.077)×1.015÷0.0236=3.43m² となる。

乾湿境界における湿り空気の比エンタルピー h_{wb} は 19.3 kJ/kg である。また、着霜開始点（図 10.1 の fb）における湿り空気の比エンタルピー h_{fb} は、0°C / 95 %の条件により 9.0 kJ/kg となる。従って、この点まで冷却した場合の熱交換量 Q_w は、式 10.7 を用いて、Q_d=(19.3 – 9.0)×1.0=10.29 kW である。乾きコイルと湿りコイルにおける熱交換量 1.32+10.29=11.61<必要熱交換能力(15 kW)であるため、熱交換は湿りコイルでも終了しない。式 10.10 により $h_{S,evp}$ は -6.1 kJ/kg である。従って、湿りコイルの熱通過有効度は、ε_w=10.29÷(1.0×(19.3-(-6.1)))=0.41 となる。また、式 10.11 により湿りコイルの熱通過率は K_w=0.0236÷1.015=0.0233 kW/(m²·(kJ/kg))であり、これを式 10.8 に代入し、湿りコイルの伝熱面積は S_w=-ln(1-0.41)×1.0÷0.0233=22.37m² となる。

†1　青木らの研究報告から逆算すると 1mm の着霜で v_a=5~15m/s の条件で R_f=0.5~0.6 となる。低風速の方が R_f が大きい傾向があったため、本書では 0.6 を採用した。

着霜コイルにおける熱交換量は、必要能力から冷却除湿コイルの熱交換量を差し引き、Q_f=15.0-1.32-10.29=3.39 kW である。式 10.16 と式 10.17 を用いて、h'_{fb} と $h'_{S,evp}$ はそれぞれ 10.17 kJ/kg と-5.55 kJ/kg となる。これらを式 10.14 に代入すると着霜コイルの熱通過有効度は、ε_f=3.39÷(1.0×(10.17-(-5.55)))=0.216 である。さらに式 10.15 により伝熱面積は、S_f=-ln(1-0.216)×1.0÷(0.0233×0.6)=17.35m^{2}となる。

乾きコイル、湿りコイル、着霜コイルの伝熱面積を合算し、必要伝熱面積は S=3.43+22.37+17.35=43.15 m^{2}となる。

10.2.2　凝縮器

凝縮器は冷媒蒸気を凝縮させて加熱を行う機器である。蒸発器と同様に熱交換の対象が液体の場合にはシェルアンドチューブ形、空気の場合にはプレートフィン付管形が用いられる。熱交換の対象である空気を加熱する側に働くため、蒸発器と異なり、基本的には湿り空気の状態変化は考慮する必要が無い。従って、基礎式は式 10.19 と式 10.20 に示すように蒸発器の乾きコイルと同様である。Q_{cnd} は熱交換量[kW]、mc_a は湿り空気の熱容量流量[kW/K]、t_{cnd} は凝縮温度[°C]、ε_{cnd} は熱通過有効度[-]、K_{cnd} は熱通過率[kW/(m^2·K)]、S_{cnd} は伝熱面積[m^2]である。熱通過率 K_{cnd} は式 10.21 で計算する†1)。

$$Q_{cnd}=mc_a\left(t_o-t_i\right) \tag{10.19}$$

$$Q_{cnd}=\varepsilon_{cnd}\,mc_a\left(t_i-t_{cnd}\right) \tag{10.20}$$

$$\varepsilon_{cnd}=1-\exp\left(-K_{cnd}S_{cnd}/mc_a\right) \tag{10.21}$$

$$K_{cnd}=0.0688\left(m_a/m_{aN}\right)^{0.3187} \tag{10.22}$$

夏季に凝縮器における熱交換量を増大させるための工夫として、凝縮器のフィン表面に水を噴霧する方法が提案されている[10.4) 10.5) 10.6) 10.7)]。この場合の湿り空気線図上の動きを図 10.2 に示す。凝縮温度は t_{cnd} [°C]であり、水噴霧がない場合には入口乾球温度 t_i [°C]から絶対湿度一定で t_{cnd} へ向かい、出口乾球温度は t_o [°C]となる。水噴霧がある場合には飽和温度 t_S [°C]に向かって等湿球温度線上を進み、入口乾球温度は t_{i2} [°C]になる。さらに凝縮器内の熱交換によって絶対湿度一定で t_{cnd} に向かい、出口乾球温度は t_{o2} [°C]となる。温度アプローチが大きい分、熱交換の効率が高くなるため、水噴霧が無い場合の出入口比エンタルピー差 Δh_{io} [kJ/kg]に比較して、水噴霧がある場合の出入口比エンタルピー差 Δh_{io2} [kJ/kg]は大きな値となる。

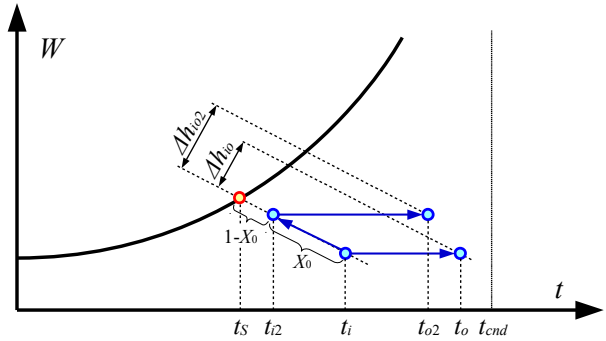


図 10.2　水噴霧による湿り空気の動きの変化

飽和温度 t_S に対してどの程度まで近づけるかを温度低減効果率 X_0 と表現すると、補正入口温度 t_{i2} は式 10.23 で計算できる。X_0 は 0~1 までの値を取り、水噴霧の量、噴霧の方式、外気条件などに依存する。中川らはパッケージ空調機の室外機について実測を行い、0.4~X_0~0.5 であったと報告している[10.6)]。また、山口らはヒートポンプチラーについて実測を行っており、図からは 0.6~X_0~0.7 程度と読み取れる[10.4) 10.5)]。

†1　文献 10.3 に記載の図から回帰した係数であり、冷媒は R410A、溝付管、管径 9.52mm、冷媒流量 0.015kg/s での値である。

$$t_{i2}=t_{i1}-X_0\left(t_i-t_S\right) \tag{10.23}$$

水噴霧によってフィンの表面が完全に湿潤に保たれるのであれば、湿り空気は冷却塔における熱交換[†1]と同様に、飽和線に平行して右上方に移動しそうであるが、実際にはこのような動きとはならない。現在発売されている水噴霧の機種は写真 10.2 に示すように外気を吸い込むフィンの側面に噴出口が設置されているため、入口付近から離れたフィンについては通常の顕熱交換が行われているものと推定する。

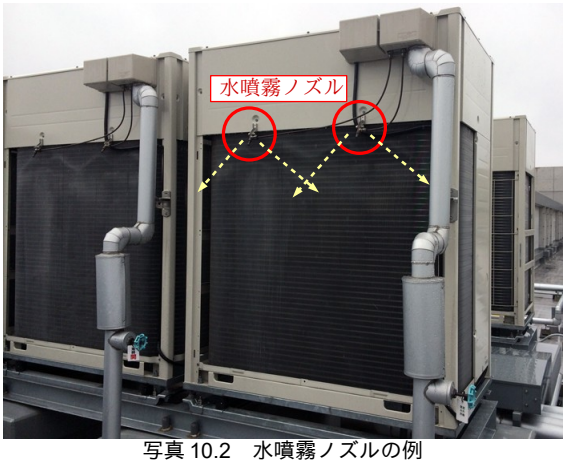


写真 10.2　水噴霧ノズルの例

山口らの実測結果によれば、フィンに直接に水を噴霧する機種の場合には、給水量のほぼ全てが蒸発している[10.4) 10.5)]。従って、水噴霧による水使用量 m_s [kg/s]は空気の出入口絶対湿度 W_i [kg/kg]、W_o [kg/kg]と風量 m_a [kg/s]を用いて式 10.24 で計算できる。

$$m_s=m_a\left(W_o-W_i\right)=m_a X_0\left(W_S-W_i\right) \tag{10.24}$$

10.3　計算

蒸発器および凝縮器のシミュレーション上、必要となる主な計算は、1）定格条件から伝熱面積を推定する計算、2）蒸発温度あるいは凝縮温度を指定した場合の成り行きでの熱交換量の計算、3）熱交換量を指定した場合の蒸発温度あるいは凝縮温度の計算、である。以下、蒸発器と凝縮機のそれぞれについて上記の 3 種類の計算法を示す。

10.3.1　蒸発器の計算

プログラム 10.1 に定数宣言を示す。

プログラム 10.1　蒸発器計算の定数宣言

Popolo.HVAC.HeatExchanger.CrossFinEvaporator class

```
/// <summary>氷の昇華潜熱[kJ/kg]</summary>
private const double SUBLIMINATION_LATENT_HEAT = 2837;

/// <summary>0℃の水の蒸発潜熱[kJ/kg]</summary>
private const double VAPORIZATION_LATENT_HEAT = 2501;

/// <summary>乾き空気の定圧比熱[kJ/kg-K]</summary>
private const double DRYAIR_ISOBARIC_SPECIFIC_HEAT = 1.005;

/// <summary>水蒸気の定圧比熱[kJ/kg-K]</summary>
private const double VAPOR_ISOBARIC_SPECIFIC_HEAT = 1.805;

/// <summary>氷の定圧比熱[kJ/kg-K]</summary>
private const double ICE_ISOBARIC_SPECIFIC_HEAT = 2.090;

```

†1　第 12 章参照。

```
/// <summary>乾きコイルの熱貫流率[kW/(m2K)]計算係数</summary>
/// <remarks>プレートフィン2.0m/s基準:空気調和ハンドブックより</remarks>
private const double CF_A = 0.0236;
private const double CF_B = 0.5479;

/// <summary>着霜による熱貫流率効率低下係数</summary>
private const double F_PENALTY = 0.6;
```

1）伝熱面積の計算

伝熱面積の計算をプログラム 10.2 に示す。計算の流れは例題 10.1 と同様であり、乾きコイル、湿りコイル、着霜コイルの順で必要面積を計算していき、最後に合算する。ただし、図 10.1 の右の場合に対応するために、43 行で条件分岐を行う。84 行は例外処理であり、出口空気温度が蒸発温度に一致してもなお、熱交換能力を満足できない場合である。90~100 行は固体基準の比エンタルピーの計算関数であり、式 10.16 の実装である。81, 82 行に示すように着霜コイルの計算に用いる。

プログラム 10.2　伝熱面積の計算

Popolo.HVAC.HeatExchanger.CrossFinEvaporator class

```
/// <summary>伝熱面積[m2]を計算する</summary>
/// <param name="evpTemperature">蒸発温度[C]</param>
/// <param name="heatTransfer">熱交換能力[kW]</param>
/// <param name="airFlowRate">風量[kg/s]</param>
/// <param name="nominalAirFlowRate">定格風量[kg/s]</param>
/// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
/// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
/// <param name="borderRelativeHumidity">乾湿境界相対湿度[%]</param>
/// <returns>伝熱面積[m2]</returns>
public static double GetSurfaceArea
  (double evpTemperature, double heatTransfer, double airFlowRate, double nominalAirFlowRate,
  double inletAirTemperature, double inletAirHumidityRatio, double borderRelativeHumidity)
{
  double epsilon;
  double kD = CF_A * Math.Pow(airFlowRate / nominalAirFlowRate, CF_B);

  //乾湿境界判定
  double rh = MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
    (inletAirTemperature, inletAirHumidityRatio, ATMOSPHERIC_PRESSURE);
  borderRelativeHumidity = Math.Max(rh, borderRelativeHumidity);

  //湿り空気比熱の計算
  double cpmaWB = MoistAir.GetSpecificHeat(inletAirHumidityRatio);

  //乾きコイル面積の計算
  double mca = cpmaWB * airFlowRate;
  double tWB = MoistAir.GetDryBulbTemperatureFromHumidityRatioAndRelativeHumidity
    (inletAirHumidityRatio, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
  double qD = (inletAirTemperature - tWB) * mca;

  //乾きコイルで伝熱が終了する場合
  if (heatTransfer < qD)
  {
    epsilon = heatTransfer / (mca * (inletAirTemperature - evpTemperature));
    return -Math.Log(1 - epsilon) * mca / kD;
  }
  //湿りコイルまで到達する場合
  epsilon = qD / (mca * (inletAirTemperature - evpTemperature));
  double sD = -Math.Log(1 - epsilon) * mca / kD;

  double qW, sW, xFB, tFB, cpmaFB;
  //湿りコイルがある場合
  if (0 < tWB)
  {
    tFB = 0;
    //湿りコイル面積の計算
    xFB = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
      (0, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
    cpmaFB = MoistAir.GetSpecificHeat(xFB);
    double hWB = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
      (tWB, inletAirHumidityRatio);
    double hEvp = MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
      (evpTemperature, 100, ATMOSPHERIC_PRESSURE);
    double hFB = MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
      (0, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
    qW = (hWB - hFB) * airFlowRate;
```

```
57     double kW = kD / (0.5 * (cpmaWB + cpmaFB));
58
59     //湿りコイルで伝熱が終了する場合
60     if (heatTransfer - qD < qW)
61     {
62       epsilon = (heatTransfer - qD) / (airFlowRate * (hWB - hEvp));
63       return -Math.Log(1 - epsilon) * airFlowRate / kW + sD;
64     }
65     //着霜コイルまで到達する場合
66     epsilon = qW / (airFlowRate * (hWB - hEvp));
67     sW = -Math.Log(1 - epsilon) * airFlowRate / kW;
68   }
69   else
70   {
71     tFB = tWB;
72     xFB = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
73       (tWB, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
74     cpmaFB = MoistAir.GetSpecificHeat(xFB);
75     qW = 0;
76     sW = 0;
77   }
78
79   //着霜コイル面積の計算
80   double kF = kD / cpmaFB * F_PENALTY;
81   double hdF = getHD(tFB, borderRelativeHumidity);
82   double hdEvp = getHD(evpTemperature, 100);
83   epsilon = (heatTransfer - qD - qW) / (airFlowRate * (hdF - hdEvp));
84   if (1 <= epsilon) throw new Exception("初期化不能エラー");
85   double sF = -Math.Log(1 - epsilon) * airFlowRate / kF;
86
87   return sF + sD + sW;
88 }
89
90 /// <summary>乾球温度[C]から固体基準の比エンタルピー[kW]を計算する</summary>
91 /// <param name="temperature">乾球温度[C]</param>
92 /// <param name="relativeHumidity">相対湿度[%]</param>
93 /// <returns>固体基準の比エンタルピー[kW]</returns>
94 private static double getHD(double temperature, double relativeHumidity)
95 {
96   double x = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
97     (temperature, relativeHumidity, ATMOSPHERIC_PRESSURE);
98   return DRYAIR_ISOBARIC_SPECIFIC_HEAT * temperature
99     + (ICE_ISOBARIC_SPECIFIC_HEAT * temperature + SUBLIMINATION_LATENT_HEAT) * x;
100 }
```

2）熱交換量の計算

蒸発温度 t_{evp} が与えられた場合の熱交換量の計算フローを図 10.3 に示す。伝熱面積計算のときと同様に、乾きコイル、湿りコイル、着霜コイルの順で計算を進め、出口空気の範囲を絞り込む。

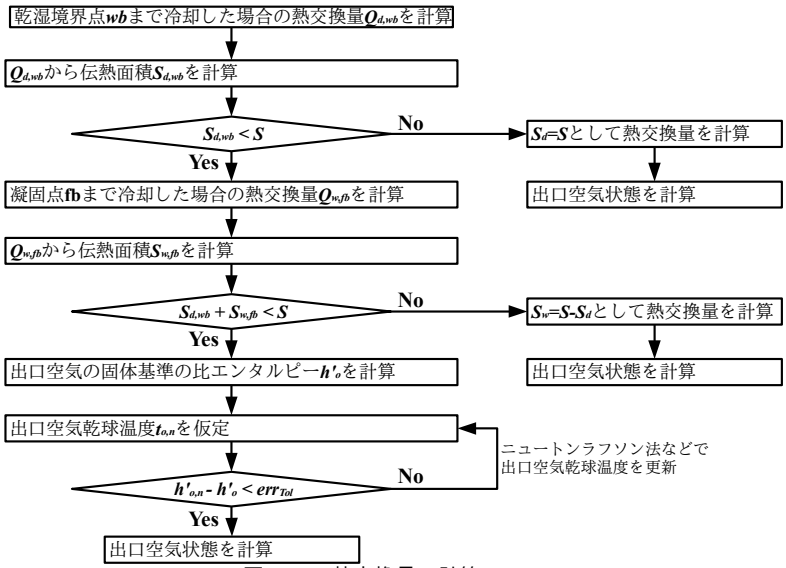


図 10.3　熱交換量の計算フロー

プログラム 10.3 に熱交換量の計算処理を示す。40 行までで、乾湿境界点まで冷却した場合の熱交

換量と必要伝熱面積を計算する。必要伝熱面積が実際の伝熱面積よりも大きい場合には乾湿境界まで辿り着かずに熱交換が終了するということであるから、46~59 行に示すように式 10.4 の S_d に S を設定して熱交換量を求め、計算を終了させる。

S_d<S の場合には湿りコイルに到達する（61 行以降の処理）。ただし、乾湿境界点の乾球温度が 0 ℃未満である場合には湿りコイルは無く、着霜コイルの計算（107 行以降）に移る。この場合には t_{fb} が 0 ℃ではないため、98~105 行で fb 点の絶対湿度と湿り空気比熱を計算する。湿りコイルが存在する場合には、65~78 行で凝固点まで冷却した場合の熱交換量と必要伝熱面積を計算する。必要伝熱面積が乾きコイル面積を差し引いた残りの伝熱面積よりも大きい場合には、凝固点まで辿り着かない。80~95 行で式 10.9 の S_w に S を設定して熱交換量を求め、計算を終了させる。

107~114 行で出口空気の固体基準の比エンタルピー h'_o を計算する。116~129 行で、この比エンタルピー h'_o に合致する乾球温度をニュートン・ラフソン法で収束計算する。126~133 行で乾球温度から絶対湿度を求め、式 10.18 を適用して除霜負荷を計算する。136 行で各領域の熱交換量を合算し、計算を終了する。

プログラム 10.3　熱交換量の計算

Popolo.HVAC.HeatExchanger.CrossFinEvaporator class

```
  1 /// <summary>交換熱量[kW]を計算する</summary>
  2 /// <param name="evpTemperature">蒸発温度[C]</param>
  3 /// <param name="airFlowRate">風量[kg/s]</param>
  4 /// <param name="nominalAirFlowRate">定格風量[kg/s]</param>
  5 /// <param name="surfaceArea">伝熱面積[m2]</param>
  6 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
  7 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
  8 /// <param name="borderRelativeHumidity">乾湿境界相対湿度[%]</param>
  9 /// <param name="heatTransfer">出力：交換熱量[kW]</param>
 10 /// <param name="outletAirTemperature">出力：出口空気乾球温度[C]</param>
 11 /// <param name="outletAirHumidityRatio">出力：出口空気絶対湿度[kg/kg]</param>
 12 /// <param name="sD">出力：乾きコイル面積[m2]</param>
 13 /// <param name="sW">出力：湿りコイル面積[m2]</param>
 14 /// <param name="defrostLoad">出力：除霜負荷[kW]</param>
 15 public static void GetHeatTransfer
 16   (double evpTemperature, double airFlowRate, double nominalAirFlowRate, double surfaceArea,
 17   double inletAirTemperature, double inletAirHumidityRatio, double borderRelativeHumidity,
 18   out double heatTransfer, out double outletAirTemperature, out double outletAirHumidityRatio,
 19   out double sD, out double sW, out double defrostLoad)
 20 {
 21   //乾きコイルの熱通過率[kW/m2K]
 22   double kD = CF_A * Math.Pow(airFlowRate / nominalAirFlowRate, CF_B);
 23
 24   //乾湿境界判定
 25   double rh = MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
 26     (inletAirTemperature, inletAirHumidityRatio, ATMOSPHERIC_PRESSURE);
 27   borderRelativeHumidity = Math.Max(rh, borderRelativeHumidity);
 28   double tWB = MoistAir.GetDryBulbTemperatureFromHumidityRatioAndRelativeHumidity
 29     (inletAirHumidityRatio, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
 30
 31   //湿り空気比熱[kJ/kgK]の計算
 32   double cpmaWB = MoistAir.GetSpecificHeat(inletAirHumidityRatio);
 33   double mca = cpmaWB * airFlowRate;
 34
 35   //乾きコイルの計算
 36   //露点まで冷却するために必要な面積を計算
 37   double qD = mca * (inletAirTemperature - tWB);
 38   double epsilonD = qD / (mca * (inletAirTemperature - evpTemperature));
 39   if (epsilonD < 1) sD = -Math.Log(1 - epsilonD) * mca / kD;
 40   else sD = surfaceArea;
 41
 42   double hWB = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(tWB, inletAirHumidityRatio);
 43   double hEvp =
 44     MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity(evpTemperature, 100, ATMOSPHERIC_PRESSURE);
 45
 46   //乾きコイルのみで伝熱が終了する場合
 47   if (surfaceArea <= sD || 1 <= epsilonD || hWB < hEvp)
 48   {
 49     sD = surfaceArea;
```

```
    sW = 0;
    defrostLoad = 0;
    outletAirHumidityRatio = inletAirHumidityRatio;

    epsilonD = 1 - Math.Exp(-kD * sD / mca);
    qD = epsilonD * mca * (inletAirTemperature - evpTemperature);
    outletAirTemperature = inletAirTemperature - qD / mca;
    heatTransfer = qD;
    return;
  }

  //湿りコイルがある場合
  double tFB, qW, xFB, cpmaFB;
  if (0 < tWB)
  {
    tFB = 0;
    xFB = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
      (0, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
    cpmaFB = MoistAir.GetSpecificHeat(xFB);

    //凝固点 (0C) まで冷却するために必要な面積を計算
    double hFB = MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
      (0, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);

    qW = (hWB - hFB) * airFlowRate;
    double kW = kD / (0.5 * (cpmaWB + cpmaFB));
    double epsilonW = qW / (airFlowRate * (hWB - hEvp));
    if (epsilonW < 1) sW = -Math.Log(1 - epsilonW) * airFlowRate / kW;
    else sW = surfaceArea - sD;

    //湿りコイルで伝熱が終了する場合
    if (surfaceArea <= sW + sD || 1 <= epsilonW)
    {
      sW = surfaceArea - sD;
      defrostLoad = 0;

      epsilonW = 1 - Math.Exp(-kW * sW / airFlowRate);
      qW = epsilonW * airFlowRate * (hWB - hEvp);
      double ho = hWB - qW / airFlowRate;
      outletAirHumidityRatio = MoistAir.GetHumidityRatioFromEnthalpyAndRelativeHumidity
        (ho, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
      outletAirTemperature = MoistAir.GetDryBulbTemperatureFromHumidityRatioAndEnthalpy
        (outletAirHumidityRatio, ho);
      heatTransfer = qD + qW;
      return;
    }
  }
  else
  {
    qW = 0;
    sW = 0;
    tFB = tWB;
    xFB = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
      (tWB, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);
    cpmaFB = MoistAir.GetSpecificHeat(xFB);
  }

  //着霜コイルの計算
  double kF = kD / cpmaFB * F_PENALTY;
  double hdFB = getHD(tFB, borderRelativeHumidity);
  double hdEvp = getHD(evpTemperature, 100);
  double sF = surfaceArea - sD - sW;
  double epsilonF = 1 - Math.Exp(-kF * sF / airFlowRate);
  double qF = epsilonF * airFlowRate * (hdFB - hdEvp);
  double hdo = hdFB - qF / airFlowRate;

  //出口空気温度を収束計算
  double to = tFB;
  double err1 = Math.Abs(getHD(to, borderRelativeHumidity) - hdo);
  const double DELTA = 0.001;
  while (0.01 < err1)
  {
    double err2 = Math.Abs(getHD(to + DELTA, borderRelativeHumidity) - hdo);
    to -= DELTA * err1 / (err2 - err1);
    err1 = Math.Abs(getHD(to, borderRelativeHumidity) - hdo);
  }
  outletAirTemperature = to;
  outletAirHumidityRatio = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
    (outletAirTemperature, borderRelativeHumidity, ATMOSPHERIC_PRESSURE);

```

```
130   //除霜負荷を計算
131   defrostLoad = airFlowRate * (xFB - outletAirHumidityRatio)
132     * (SUBLIMINATION_LATENT_HEAT - ICE_ISOBARIC_SPECIFIC_HEAT * outletAirTemperature);
133
134   //交換熱量[kW]を集計
135   heatTransfer = qD + qW + qF;
136 }
```

3）蒸発温度の計算

熱交換量が与えられた場合の蒸発温度 t_{evp} の計算をプログラム 10.4 に示す。23, 24 行で入口湿り空気比熱から蒸発温度を粗く推定し、プログラム 10.3 を用いてニュートン・ラフソン法で解を得る。

プログラム 10.4　蒸発温度の計算

Popolo.HVAC.HeatExchanger.CrossFinEvaporator class

```
  1 /// <summary>蒸発温度[C]を計算する</summary>
  2 /// <param name="heatTransfer">交換熱量[kW]</param>
  3 /// <param name="airFlowRate">風量[kg/s]</param>
  4 /// <param name="nominalAirFlowRate">定格風量[kg/s]</param>
  5 /// <param name="surfaceArea">伝熱面積[m2]</param>
  6 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
  7 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
  8 /// <param name="borderRelativeHumidity">乾湿境界相対湿度[%]</param>
  9 /// <param name="deductDefrostLoad">除霜負荷を差し引くか否か</param>
 10 /// <param name="evaporatingTemperature">出力：蒸発温度[C]</param>
 11 /// <param name="outletAirTemperature">出力：出口空気乾球温度[C]</param>
 12 /// <param name="outletAirHumidityRatio">出力：出口空気絶対湿度[kg/kg]</param>
 13 /// <param name="sD">出力：乾きコイル面積[m2]</param>
 14 /// <param name="sW">出力：湿りコイル面積[m2]</param>
 15 /// <param name="defrostLoad">出力：除霜負荷[kW]</param>
 16 public static void GetEvaporatingTemperature
 17   (double heatTransfer, double airFlowRate, double nominalAirFlowRate,
 18   double surfaceArea, double inletAirTemperature, double inletAirHumidityRatio, double borderRelativeHumidity,
 19   bool deductDefrostLoad, out double evaporatingTemperature, out double outletAirTemperature,
 20   out double outletAirHumidityRatio, out double sD, out double sW, out double defrostLoad)
 21 {
 22   //蒸発温度を仮定
 23   double cpma = MoistAir.GetSpecificHeat(inletAirHumidityRatio);
 24   evaporatingTemperature = inletAirTemperature - heatTransfer / (airFlowRate * cpma);
 25
 26   Roots.ErrorFunction eFnc = delegate (double eTemp)
 27   {
 28     double ht, ot, oa, sd, sw, dl;
 29     GetHeatTransfer(eTemp, airFlowRate, nominalAirFlowRate, surfaceArea, inletAirTemperature,
 30       inletAirHumidityRatio, borderRelativeHumidity, out ht, out ot, out oa, out sd, out sw, out dl);
 31     if (deductDefrostLoad) return ht - heatTransfer - dl;
 32     else return ht - heatTransfer;
 33   };
 34   evaporatingTemperature = Roots.Brent(evaporatingTemperature - 10, evaporatingTemperature + 10, 0.00001,eFnc);
 35   double hTransfer;
 36   GetHeatTransfer(evaporatingTemperature, airFlowRate, nominalAirFlowRate, surfaceArea,
 37     inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity,
 38     out hTransfer, out outletAirTemperature, out outletAirHumidityRatio, out sD, out sW, out defrostLoad);
 39 }
```

4）蒸発器クラスの作成

プログラム 10.5 に蒸発器クラスのプロパティ宣言を示す。

プログラム 10.5　プロパティ宣言

Popolo.HVAC.HeatExchanger.CrossFinEvaporator class

```
 1 /// <summary>乾湿境界の相対湿度[%]</summary>
 2 private double borderRelativeHumidity;
 3
 4 /// <summary>伝熱面積[m2]を取得する</summary>
 5 public double SurfaceArea { get; private set; }
 6
 7 /// <summary>乾き伝熱面積[m2]を取得する</summary>
 8 public double DrySurfaceArea { get; private set; }
 9
10 /// <summary>湿り伝熱面積[m2]を取得する</summary>
11 public double WetSurfaceArea { get; private set; }
12
13 /// <summary>着霜伝熱面積[m2]を取得する</summary>
14 public double FrostSurfaceArea
15 {
16   get { return SurfaceArea - (DrySurfaceArea + WetSurfaceArea); }
```

```
17 }
18
19 /// <summary>定格風量[kg/s]を取得する</summary>
20 public double NominalAirFlowRate { get; private set; }
21
22 /// <summary>風量[kg/s]を取得する</summary>
23 public double AirFlowRate { get; private set; }
24
25 /// <summary>蒸発温度[C]を取得する</summary>
26 public double EvaporatingTemperature { get; private set; }
27
28 /// <summary>乾湿境界の相対湿度[%]を設定・取得する</summary>
29 public double BorderRelativeHumidity
30 {
31   get { return borderRelativeHumidity; }
32   set { borderRelativeHumidity = Math.Max(50, value); }
33 }
34
35 /// <summary>入口空気乾球温度[C]を設定・取得する</summary>
36 public double InletAirTemperature { get; set; }
37
38 /// <summary>入口空気絶対温度[kg/kg]を設定・取得する</summary>
39 public double InletAirHumidityRatio { get; set; }
40
41 /// <summary>出口空気乾球温度[C]を取得する</summary>
42 public double OutletAirTemperature { get; set; }
43
44 /// <summary>出口空気絶対温度[kg/kg]を取得する</summary>
45 public double OutletAirHumidityRatio { get; set; }
46
47 /// <summary>交換熱量[kW]を取得する</summary>
48 public double HeatTransfer { get; private set; }
49
50 /// <summary>デフロスト負荷[kW]を取得する</summary>
51 public double DefrostLoad { get; private set; }
52
53 /// <summary>停止しているか否か</summary>
54 public bool IsShutOff { get; private set; }
```

プログラム 10.6 にコンストラクタを示す。プログラム 10.2 を用いて定格能力から伝熱面積を推定する。27~38 行は機器の停止処理である。

プログラム 10.6　コンストラクタ

Popolo.HVAC.HeatExchanger.CrossFinEvaporator class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="evpTemperature">蒸発温度[C]</param>
 3 /// <param name="heatTransfer">熱交換能力[kW]</param>
 4 /// <param name="airFlowRate">風量[kg/s]</param>
 5 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
 6 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
 7 /// <param name="borderRelativeHumidity">乾湿境界相対湿度[%]</param>
 8 public CrossFinEvaporator(double evpTemperature, double heatTransfer, double airFlowRate,
 9   double inletAirTemperature, double inletAirHumidityRatio, double borderRelativeHumidity)
10 {
11   //プロパティ初期化
12   NominalAirFlowRate = airFlowRate;
13   AirFlowRate = airFlowRate;
14   BorderRelativeHumidity = borderRelativeHumidity;
15   InletAirTemperature = inletAirTemperature;
16   InletAirHumidityRatio = inletAirHumidityRatio;
17   shutOff();
18
19   //伝熱面積を初期化する
20   SurfaceArea = GetSurfaceArea(evpTemperature, heatTransfer, airFlowRate, airFlowRate,
21     inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity);
22 }
23
24 /// <summary>機器を停止させる</summary>
25 private void shutOff()
26 {
27   AirFlowRate = 0;
28   OutletAirTemperature = InletAirTemperature;
29   OutletAirHumidityRatio = InletAirHumidityRatio;
30   DrySurfaceArea = SurfaceArea;
31   WetSurfaceArea = 0;
32   HeatTransfer = 0;
33   EvaporatingTemperature = 0;
34   DefrostLoad = 0;
```

```
35   IsShutOff = true;
36 }
```

プログラム 10.7 に熱交換量と蒸発温度の計算を示す。プログラム 10.3 と 10.4 を用いて計算を行い、結果をプロパティに保存する。19 行と 54 行に示すように、風量が 0 以下、入口乾球温度が蒸発温度以下、交換熱量が 0 の場合には機器を停止させる。

プログラム 10.7　熱交換量と蒸発温度の計算

Popolo.HVAC.HeatExchanger.CrossFinEvaporator class

```
 1 /// <summary>交換熱量[kW]を計算する</summary>
 2 /// <param name="evpTemperature">蒸発温度[C]</param>
 3 /// <param name="airFlowRate">風量[kg/s]</param>
 4 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
 5 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
 6 /// <returns>交換熱量[kW]</returns>
 7 public double GetHeatTransfer
 8   (double evpTemperature, double airFlowRate, double inletAirTemperature, double inletAirHumidityRatio)
 9 {
10   //プロパティ設定
11   EvaporatingTemperature = evpTemperature;
12   AirFlowRate = airFlowRate;
13   InletAirTemperature = inletAirTemperature;
14   InletAirHumidityRatio = inletAirHumidityRatio;
15
16   //運転判定
17   if (airFlowRate <= 0 || inletAirTemperature <= evpTemperature)
18   {
19     shutOff();
20     return 0;
21   }
22
23   double ht, to, wo, sd, sw, dfl;
24   GetHeatTransfer(evpTemperature, airFlowRate, NominalAirFlowRate, SurfaceArea, inletAirTemperature,
25     inletAirHumidityRatio, borderRelativeHumidity, out ht, out to, out wo, out sd, out sw, out dfl);
26   OutletAirTemperature = to;
27   OutletAirHumidityRatio = wo;
28   DrySurfaceArea = sd;
29   WetSurfaceArea = sw;
30   HeatTransfer = ht;
31   DefrostLoad = dfl;
32   return ht;
33 }
34
35 /// <summary>蒸発温度[C]を計算する</summary>
36 /// <param name="heatTransfer">交換熱量[kW]</param>
37 /// <param name="airFlowRate">風量[kg/s]</param>
38 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
39 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
40 /// <param name="deductDefrostLoad">除霜負荷を差し引くか否か</param>
41 /// <returns>蒸発温度[C]</returns>
42 public double GetEvaporatingTemperature(double heatTransfer, double airFlowRate,
43   double inletAirTemperature, double inletAirHumidityRatio, bool deductDefrostLoad)
44 {
45   //プロパティ設定
46   HeatTransfer = heatTransfer;
47   AirFlowRate = airFlowRate;
48   InletAirTemperature = inletAirTemperature;
49   InletAirHumidityRatio = inletAirHumidityRatio;
50
51   //運転判定
52   if (airFlowRate <= 0 || heatTransfer <= 0)
53   {
54     shutOff();
55     return 0;
56   }
57
58   double te, to, wo, sd, sw, dfl;
59   GetEvaporatingTemperature(heatTransfer, airFlowRate, NominalAirFlowRate, SurfaceArea,
60     inletAirTemperature, inletAirHumidityRatio, borderRelativeHumidity, deductDefrostLoad,
61     out te, out to, out wo, out sd, out sw, out dfl);
62   OutletAirTemperature = to;
63   OutletAirHumidityRatio = wo;
64   DrySurfaceArea = sd;
65   WetSurfaceArea = sw;
66   EvaporatingTemperature = te;
67   DefrostLoad = dfl;
68   return te;
```

69 }

【例題 10.2】

下記の定格能力を持つ蒸発器について、乾球温度を変化させた場合の 1.熱交換能力、2.除霜負荷、3.出口乾球温度、を蒸発温度別（相対湿度は 85 %に固定）、相対湿度別（蒸発温度は-10 °C に固定）に計算せよ。

蒸発温度	：2 °C	吸込乾球温度	：7 °C	風量	：167 m³/h
熱交換能力	：13 kW	吸込湿球温度	：6 °C	乾湿境界相対湿度	：95 %

【解】

計算処理をプログラム 10.8 に示す。5, 6 行は初期化処理であり、コンストラクタに定格能力を与える。11~31 行が蒸発温度別の計算、37~50 行が相対湿度別の計算である。出力をグラフ化した結果を図 10.4 に示す。上段が蒸発温度別、下段が相対湿度別の結果である。蒸発温度が低いほど、熱交換量と除霜負荷が大きく、出口空気温度が低い傾向が確認できる。除霜負荷は+1~2°C 近辺で極大値を持つことがわかる。また、吸込み空気の相対湿度が高いほど、熱交換量、除霜負荷、出口空気温度が高くなる。

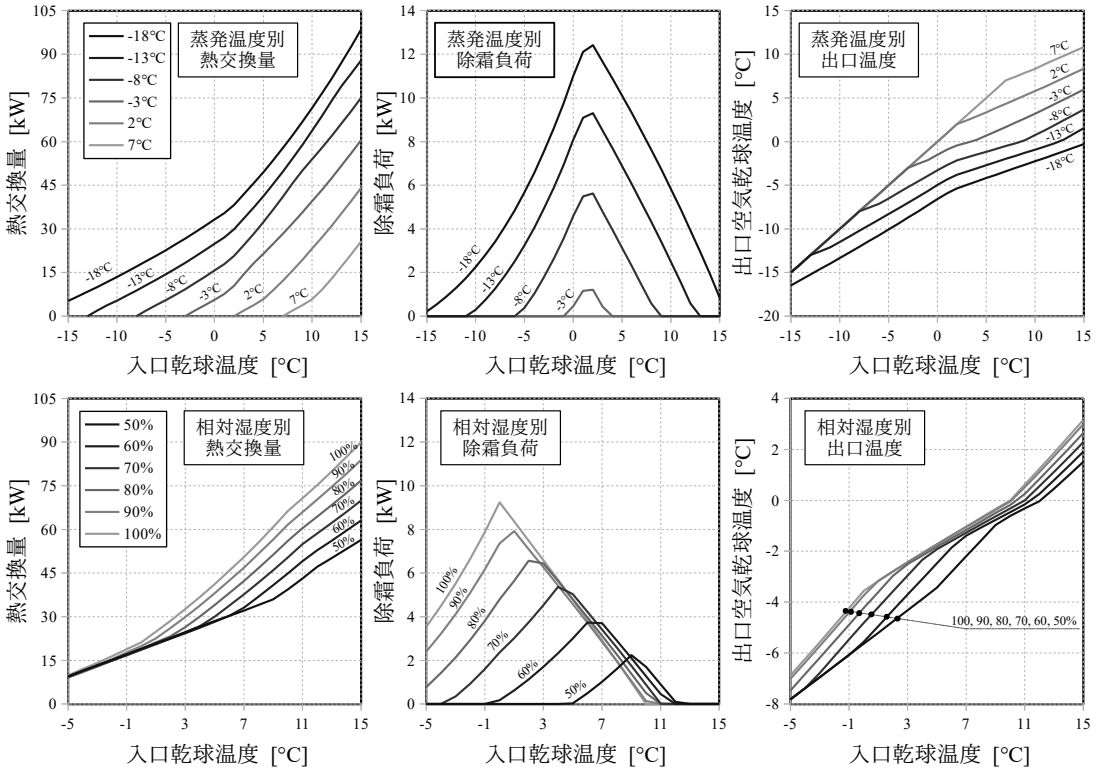


図 10.4　蒸発器の感度解析結果

プログラム 10.8　蒸発器の感度解析

```
 1 /// <summary>冷却・除湿・着霜コイルの感度解析を行う</summary>
 2 private static void CrossFinEvaporatorTest()
 3 {
 4   //蒸発器初期化
 5   double hr = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndWetBulbTemperature(7, 6, 101.325);
 6   CrossFinEvaporator evp = new CrossFinEvaporator(2, 13, 167.0 / 60 * 1.2, 7, hr, 95);
 7
 8   using (StreamWriter sWriter = new StreamWriter
 9     ("CrossFinEvaporatorTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
10   {
11     //乾球温度・蒸発温度・能力の関係
12     double[] te = new double[] { -18, -13, -8, -3, 2, 7 };
13     sWriter.WriteLine("乾球温度・蒸発温度・能力の関係");
14     sWriter.Write("乾球温度[C]");
15     for (int i = 0; i < te.Length; i++)
16       sWriter.Write("," + te[i] + "C 交換熱量," + te[i] + "C 除霜," + te[i] + "C 出口温度");
17     sWriter.WriteLine();
18
19     for (int i = -15; i <= 15; i++)
20     {
```

```
21        sWriter.Write(i);
22        for (int j = 0; j < te.Length; j++)
23        {
24          hr = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity(i, 85, 101.325);
25          sWriter.Write("," + evp.GetHeatTransfer(te[j], evp.NominalAirFlowRate, i, hr));
26          sWriter.Write("," + evp.DefrostLoad + "," + evp.OutletAirTemperature);
27        }
28        sWriter.WriteLine();
29      }
30
31      sWriter.WriteLine();
32      //乾球温度・相対湿度・能力の関係
33      double[] rh = new double[] { 50, 60, 70, 80, 90, 100 };
34      sWriter.WriteLine("乾球温度・蒸発温度・能力の関係");
35      sWriter.Write("乾球温度[C]");
36      for (int i = 0; i < rh.Length; i++)
37        sWriter.Write("," + rh[i] + "% 交換熱量," + rh[i] + "% 除霜," + rh[i] + "% 出口温度");
38      sWriter.WriteLine();
39
40      for (int i = -5; i <= 15; i++)
41      {
42        sWriter.Write(i);
43        for (int j = 0; j < rh.Length; j++)
44        {
45          hr = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity(i, rh[j], 101.325);
46          sWriter.Write("," + evp.GetHeatTransfer(-10, evp.NominalAirFlowRate, i, hr));
47          sWriter.Write("," + evp.DefrostLoad + "," + evp.OutletAirTemperature);
48        }
49        sWriter.WriteLine();
50      }
51    }
52 }
```

10.3.2　凝縮器の計算

プログラム 10.9 に定数宣言を示す。

プログラム 10.9　定数宣言

Popolo.HVAC.HeatExchanger.CrossFinCondensor class

```
1 /// <summary>乾きコイルの熱貫流率[kW/(m2K)]計算係数</summary>
2 private const double CF_A = 0.0688;
3 private const double CF_B = 0.3187;
```

1）伝熱面積の計算

伝熱面積の計算をプログラム 10.10 に示す。凝縮器では湿り空気の状態変化が無いため、蒸発器に比較すると容易に伝熱面積を求めることができる。

プログラム 10.10　伝熱面積の計算

Popolo.HVAC.HeatExchanger.CrossFinCondensor class

```
 1 /// <summary>伝熱面積[m2]を計算する</summary>
 2 /// <param name="cndTemperature">凝縮温度[C]</param>
 3 /// <param name="heatTransfer">熱交換能力[kW]</param>
 4 /// <param name="airFlowRate">風量[kg/s]</param>
 5 /// <param name="nominalAirFlowRate">定格風量[kg/s]</param>
 6 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
 7 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
 8 /// <returns>伝熱面積[m2]</returns>
 9 public static double GetSurfaceArea
10   (double cndTemperature, double heatTransfer, double airFlowRate,
11   double nominalAirFlowRate, double inletAirTemperature, double inletAirHumidityRatio)
12 {
13   double cpma = MoistAir.GetSpecificHeat(inletAirHumidityRatio);
14   double mca = cpma * airFlowRate;
15   double epsilon = heatTransfer / (mca * (cndTemperature - inletAirTemperature));
16   double kCnd = CF_A * Math.Pow(airFlowRate / nominalAirFlowRate, CF_B);
17   return -Math.Log(1 - epsilon) * mca / kCnd;
18 }
```

2）熱交換量の計算

プログラム 10.11 に熱交換量の計算を示す。26~36 行に示すように、熱通過有効度を用いて収束計算無しに出口状態を計算可能である。ただし、水噴霧がある場合には入口空気状態を補正する必要がある。39~57 行に空気状態を修正する関数を定義しており、21 行でこれを呼び出している。式 10.23

と式 10.24 を用いて空気状態の修正および水噴霧量を計算する。

プログラム 10.11　熱交換量の計算

Popolo.HVAC.HeatExchanger.CrossFinCondensor class

```
/// <summary>交換熱量[kW]を計算する</summary>
/// <param name="cndTemperature">凝縮温度[C]</param>
/// <param name="airFlowRate">風量[kg/s]</param>
/// <param name="nominalAirFlowRate">定格風量[kg/s]</param>
/// <param name="surfaceArea">伝熱面積[m2]</param>
/// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
/// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
/// <param name="sprayEffectiveness">水噴霧の温度低減効果率[-]</param>
/// <param name="heatTransfer">出力：交換熱量[kW]</param>
/// <param name="outletAirTemperature">出力：出口空気乾球温度[C]</param>
/// <param name="outletAirHumidityRatio">出力：出口空気絶対湿度[kg/kg]</param>
/// <param name="waterSupply">出力：水消費量[kg/s]</param>
public static void GetHeatTransfer
  (double cndTemperature, double airFlowRate, double nominalAirFlowRate,
  double surfaceArea, double inletAirTemperature, double inletAirHumidityRatio,
  double sprayEffectiveness, out double heatTransfer, out double outletAirTemperature,
  out double outletAirHumidityRatio, out double waterSupply)
{
  //水噴霧がある場合
  if (0 < sprayEffectiveness)
    waterSupply = getWaterSupply
      (ref inletAirTemperature, ref inletAirHumidityRatio, sprayEffectiveness, airFlowRate);
  //水噴霧がない場合
  else waterSupply = 0;

  //熱通過率[kW/m2K]
  double kCnd = CF_A * Math.Pow(airFlowRate / nominalAirFlowRate, CF_B);
  //湿り空気比熱[kJ/kgK]
  double cpma = MoistAir.GetSpecificHeat(inletAirHumidityRatio);
  double mca = cpma * airFlowRate;

  double epsilon = 1 - Math.Exp(-kCnd * surfaceArea / mca);
  double q = epsilon * mca * (cndTemperature - inletAirTemperature);
  outletAirTemperature = inletAirTemperature + q / mca;
  outletAirHumidityRatio = inletAirHumidityRatio;
  heatTransfer = q;
}

/// <summary>水噴霧による水消費量[kg/s]を計算する</summary>
/// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
/// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
/// <param name="sprayEffectiveness">水噴霧の温度低減効果率[-]</param>
/// <param name="airFlowRate">風量[kg/s]</param>
private static double getWaterSupply
  (ref double inletAirTemperature, ref double inletAirHumidityRatio,
  double sprayEffectiveness, double airFlowRate)
{
  double twb = MoistAir.GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio
  (inletAirTemperature, inletAirHumidityRatio, ATMOSPHERIC_PRESSURE);
  double ts = MoistAir.GetDryBulbTemperatureFromWetBulbTemperatureAndRelativeHumidity
    (twb, 100, ATMOSPHERIC_PRESSURE);
  double ws = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
    (ts, 100, ATMOSPHERIC_PRESSURE);
  inletAirTemperature -= sprayEffectiveness * (inletAirTemperature - ts);
  inletAirHumidityRatio += sprayEffectiveness * (ws - inletAirHumidityRatio);
  return airFlowRate * sprayEffectiveness * (ws - inletAirHumidityRatio);
}
```

3）熱交換量の計算

プログラム 10.12 に凝縮温度の計算を示す。熱交換量の計算と同様、まず 19~24 行で水噴霧による入口空気状態の修正を行った後、26~35 行で熱通過有効度を用いて凝縮温度を求める。

プログラム 10.12　凝縮温度の計算

Popolo.HVAC.HeatExchanger.CrossFinCondensor class

```
/// <summary>凝縮温度[C]を計算する</summary>
/// <param name="heatTransfer">交換熱量[kW]</param>
/// <param name="airFlowRate">風量[kg/s]</param>
/// <param name="nominalAirFlowRate">定格風量[kg/s]</param>
/// <param name="surfaceArea">伝熱面積[m2]</param>
/// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
/// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
/// <param name="sprayEffectiveness">水噴霧の温度低減効果率[-]</param>
```

```
 9 /// <param name="condensingTemperature">出力：凝縮温度[C]</param>
10 /// <param name="outletAirTemperature">出力：出口空気乾球温度[C]</param>
11 /// <param name="outletAirHumidityRatio">出力：出口空気絶対湿度[kg/kg]</param>
12 /// <param name="waterSupply">出力：水消費量[kg/s]</param>
13 public static void GetCondensingTemperature
14   (double heatTransfer, double airFlowRate, double nominalAirFlowRate,
15   double surfaceArea, double inletAirTemperature, double inletAirHumidityRatio,
16   double sprayEffectiveness, out double condensingTemperature,
17   out double outletAirTemperature, out double outletAirHumidityRatio, out double waterSupply)
18 {
19   //水噴霧がある場合
20   if (0 < sprayEffectiveness)
21     waterSupply = getWaterSupply(ref inletAirTemperature,
22       ref inletAirHumidityRatio, sprayEffectiveness, airFlowRate);
23   //水噴霧がない場合
24   else waterSupply = 0;
25
26   //熱通過率[kW/m2K]
27   double kCnd = CF_A * Math.Pow(airFlowRate / nominalAirFlowRate, CF_B);
28   //湿り空気比熱[kJ/kgK]
29   double cpma = MoistAir.GetSpecificHeat(inletAirHumidityRatio);
30   double mca = cpma * airFlowRate;
31
32   outletAirTemperature = inletAirTemperature + heatTransfer / mca;
33   outletAirHumidityRatio = inletAirHumidityRatio;
34   double epsilon = 1 - Math.Exp(-kCnd * surfaceArea / mca);
35   condensingTemperature = inletAirTemperature + heatTransfer / (epsilon * mca);
36 }
```

4）凝縮器クラスの作成

プログラム 10.13 に凝縮器クラスのインスタンス変数およびプロパティ宣言を示す。水噴霧関連の情報を除き、蒸発器クラスと同様である。

プログラム **10.13**　凝縮器クラスのインスタンス変数およびプロパティ宣言

Popolo.HVAC.HeatExchanger.CrossFinCondensor class

```
 1 /// <summary>水噴霧の温度低減効果率[-]</summary>
 2 /// <remarks>0.4～0.5程度の値。0は水噴霧無し</remarks>
 3 private double sprayEffectiveness = 0.0;
 4
 5 /// <summary>伝熱面積[m2]を取得する</summary>
 6 public double SurfaceArea { get; private set; }
 7
 8 /// <summary>定格風量[kg/s]を取得する</summary>
 9 public double NominalAirFlowRate { get; private set; }
10
11 /// <summary>風量[kg/s]を取得する</summary>
12 public double AirFlowRate { get; private set; }
13
14 /// <summary>凝縮温度[C]を取得する</summary>
15 public double CondensingTemperature { get; private set; }
16
17 /// <summary>入口空気乾球温度[C]を設定・取得する</summary>
18 public double InletAirTemperature { get; set; }
19
20 /// <summary>入口空気絶対温度[kg/kg]を設定・取得する</summary>
21 public double InletAirHumidityRatio { get; set; }
22
23 /// <summary>出口空気乾球温度[C]を取得する</summary>
24 public double OutletAirTemperature { get; set; }
25
26 /// <summary>出口空気絶対温度[kg/kg]を取得する</summary>
27 public double OutletAirHumidityRatio { get; set; }
28
29 /// <summary>交換熱量[kW]を取得する</summary>
30 public double HeatTransfer { get; private set; }
31
32 /// <summary>水噴霧の温度低減効果率[-]を設定・取得する</summary>
33 public double SprayEffectiveness
34 {
35   get { return sprayEffectiveness; }
36   set { sprayEffectiveness = Math.Max(0, Math.Min(value, 1.0)); }
37 }
38
39 /// <summary>水噴霧による水使用量[kg/s]を取得する</summary>
40 public double WaterSupply { get; private set; }
41
42 /// <summary>停止しているか否か</summary>
```

```
43 public bool IsShutOff { get; private set; }
44
45 /// <summary>水噴霧を用いるか否か</summary>
46 public bool UseWaterSpray { get; set; }
```

プログラム 10.14 に凝縮器クラスのコンストラクタを示す。

プログラム 10.14　凝縮器クラスのコンストラクタ

Popolo.HVAC.HeatExchanger.CrossFinCondensor class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="cndTemperature">凝縮温度[C]</param>
 3 /// <param name="heatTransfer">熱交換能力[kW]</param>
 4 /// <param name="airFlowRate">風量[kg/s]</param>
 5 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
 6 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
 7 public CrossFinCondensor
 8   (double cndTemperature, double heatTransfer, double airFlowRate,
 9   double inletAirTemperature, double inletAirHumidityRatio)
10 {
11   //プロパティ初期化
12   NominalAirFlowRate = airFlowRate;
13   AirFlowRate = airFlowRate;
14   InletAirTemperature = inletAirTemperature;
15   InletAirHumidityRatio = inletAirHumidityRatio;
16   shutOff();
17
18   //伝熱面積を初期化する
19   SurfaceArea = GetSurfaceArea(cndTemperature, heatTransfer,
20     airFlowRate, airFlowRate, inletAirTemperature, inletAirHumidityRatio);
21 }
22
23 /// <summary>機器を停止させる</summary>
24 private void shutOff()
25 {
26   AirFlowRate = 0;
27   OutletAirTemperature = InletAirTemperature;
28   OutletAirHumidityRatio = InletAirHumidityRatio;
29   CondensingTemperature = InletAirTemperature;
30   WaterSupply = 0;
31   IsShutOff = true;
32 }
```

プログラム 10.15 に熱交換量および凝縮温度の計算を示す。プログラム 10.11 および 10.12 を呼び出し、必要に応じてプロパティに値を設定する。蒸発器と同様、風量や入口空気温度と凝縮温度との関係によっては、19 行と 61 行に示すように機器を停止させる。

プログラム 10.15　熱交換量および凝縮温度の計算

Popolo.HVAC.HeatExchanger.CrossFinCondensor class

```
 1 /// <summary>交換熱量[kW]を計算する</summary>
 2 /// <param name="cndTemperature">凝縮温度[C]</param>
 3 /// <param name="airFlowRate">風量[kg/s]</param>
 4 /// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
 5 /// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
 6 /// <returns>交換熱量[kW]</returns>
 7 public double GetHeatTransfer
 8   (double cndTemperature, double airFlowRate, double inletAirTemperature, double inletAirHumidityRatio)
 9 {
10   //プロパティ設定
11   CondensingTemperature = cndTemperature;
12   AirFlowRate = airFlowRate;
13   InletAirTemperature = inletAirTemperature;
14   InletAirHumidityRatio = inletAirHumidityRatio;
15
16   //運転判定
17   if (airFlowRate <= 0 || cndTemperature <= inletAirTemperature)
18   {
19     shutOff();
20     return 0;
21   }
22
23   double ht, to, wo, ws;
24   GetHeatTransfer(cndTemperature, airFlowRate, NominalAirFlowRate, SurfaceArea,
25     inletAirTemperature, inletAirHumidityRatio, sprayEffectiveness, out ht, out to, out wo, out ws);
26   OutletAirTemperature = to;
27   OutletAirHumidityRatio = wo;
28   HeatTransfer = ht;
```

```
  WaterSupply = ws;
  return ht;
}

/// <summary>凝縮温度[C]を計算する</summary>
/// <param name="heatTransfer">交換熱量[kW]</param>
/// <param name="airFlowRate">風量[kg/s]</param>
/// <param name="inletAirTemperature">入口空気乾球温度[C]</param>
/// <param name="inletAirHumidityRatio">入口空気絶対湿度[kg/kg]</param>
/// <returns>凝縮温度[C]</returns>
public double GetCondensingTemperature
  (double heatTransfer, double airFlowRate, double inletAirTemperature, double inletAirHumidityRatio)
{
  //プロパティ設定
  HeatTransfer = heatTransfer;
  AirFlowRate = airFlowRate;
  InletAirTemperature = inletAirTemperature;
  InletAirHumidityRatio = inletAirHumidityRatio;

  //運転判定
  if (airFlowRate <= 0 || heatTransfer <= 0)
  {
    shutOff();
    return 0;
  }

  double tc, to, wo, ws;
  GetCondensingTemperature(heatTransfer, airFlowRate, NominalAirFlowRate, SurfaceArea,
    inletAirTemperature, inletAirHumidityRatio, sprayEffectiveness, out tc, out to, out wo, out ws);
  OutletAirTemperature = to;
  OutletAirHumidityRatio = wo;
  WaterSupply = ws;
  return tc;
}
```

【例題 10.3】

下記の定格能力を持つ凝縮器について、乾球温度を変化させた場合の熱交換能力と水噴霧量を相対湿度別に計算せよ。

凝縮温度　：45 °C　　吸込乾球温度：35 °C　　風量：167 m³/h

熱交換能力：25 kW　　吸込相対湿度：55 %

【解】

計算処理をプログラム 10.16 に示す。図 10.5 に計算結果をグラフ化した結果を示す。水噴霧を行わない場合には、能力は相対湿度に対して殆ど変化しないため、水噴霧無しの計算結果は 1 つだけ図示した。乾球温度が低いほど温度アプローチが大きいため、熱交換量も大きくなる。また、相対湿度が低いと水噴霧が効果的になり、水使用量は増えるが、熱交換能力は高くなる。

プログラム 10.16　水噴霧形凝縮器の感度解析

```
/// <summary>水噴霧形凝縮器の感度解析を行う</summary>
private static void CrossFinCondensorTest()
{
  //凝縮器初期化
  double hr = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity(35, 55, 101.325);
  CrossFinCondensor cnd = new CrossFinCondensor(45, 25, 167d / 60 * 1.2, 35, hr);
  cnd.SprayEffectiveness = 0;

  using (StreamWriter sWriter = new StreamWriter
    ("CrossFinCondenserTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
  {
    //乾球温度・凝縮温度・能力の関係
    double[] rh = new double[] { 35, 55, 75 };
    sWriter.WriteLine("乾球温度・凝縮温度・能力の関係");
    sWriter.Write("乾球温度[C]");
    for (int i = 0; i < rh.Length; i++)
      sWriter.Write("," + rh[i] + "% 交換熱量1," + rh[i] + "% 交換熱量2," + rh[i] + "% 水噴霧");
    sWriter.WriteLine();

    for (int i = 20; i <= 40; i++)
    {
      sWriter.Write(i);
      for (int j = 0; j < rh.Length; j++)
      {
        hr = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity(i, rh[j], 101.325);
        cnd.SprayEffectiveness = 0.0;
        sWriter.Write("," + cnd.GetHeatTransfer(45, cnd.NominalAirFlowRate, i, hr));
        cnd.SprayEffectiveness = 0.6;
```

```
29          sWriter.Write("," + cnd.GetHeatTransfer(45, cnd.NominalAirFlowRate, i, hr));
30          sWriter.Write("," + cnd.WaterSupply * 3600);
31        }
32        sWriter.WriteLine();
33      }
34    }
35 }
```

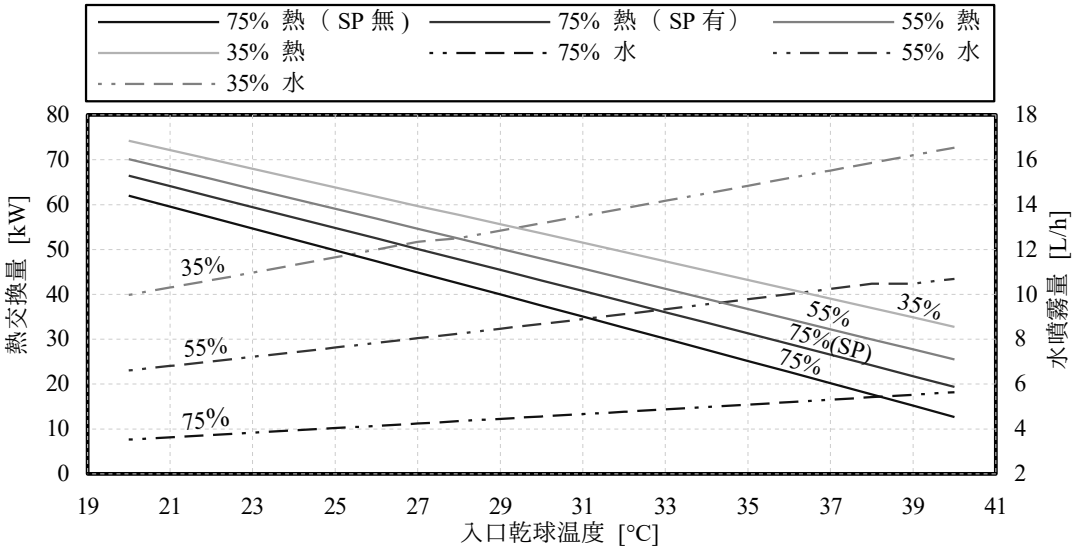


図 10.5　凝縮器の感度解析結果

【第 10 章 記号表】

c_{pa}	: 乾き空気の定圧比熱 **(1.006 kJ/(kg·K))**	R_f	: 着霜による熱通過率修正係数 **[-]**
c_{ps}	: 氷の定圧比熱 **(2.09 kJ/(kg·K))**	S	: 伝熱面積 **[m²]**
c_{pv}	: 水蒸気の定圧比熱 **(1.805 kJ/(kg·K))**	t	: 乾球温度 **[°C]**
h	: 比エンタルピー **[kJ/kg]**	v_a	: 風速 **[m/s]**
h'	: 固体基準の比エンタルピー **[kJ/kg]**	W	: 絶対湿度 **[kg/kg]**
K	: 熱通過率 **[kW/(m²·K)]**	X_0	: 水噴霧による温度低減効果率 **[-]**
m_a	: 空気の質量流量 **[kg/s]**	ε	: 熱通過有効度 **[-]**
mc_a	: 湿り空気の熱容量流量 **[kW/K]**	γ_{s0}	: **0 °C** の氷の昇華潜熱 **(2,837 kJ/kg)**
m_s	: 水噴霧のための給水量 **[kg/s]**	γ_{v0}	: **0 °C** の水の蒸発潜熱 **(2,501 kJ/kg)**
Q	: 熱交換量 **[kW]**		
添字：			
d	: 乾きコイル	*N*	: 定格
evp	: 蒸発器	*o*	: 出口
f	: 着霜コイル	*S*	: 飽和
fb	: 凝固点	*w*	: 湿りコイル
i	: 入口	*wb*	: 乾湿境界点

【第 10 章 参考文献】

10.1） 青木和夫, 服部賢, 伊藤武: 着霜を伴うフィン付管群形熱交換器の特性に関する研究 第 1 報 平均熱伝達特性について, 日本機械学会論文集 B 編, 51 巻, 469 号, pp.3048-3054, 1985

10.2） 井上宇市: 空気調和ハンドブック改訂 5 版, 第 7 章 空気調和機, pp.296, 丸善株式会社

10.3） SI による初級 冷凍受験テキスト 第 7 次改訂, 第 6 章 凝縮器, 図 6.11, pp.71, 日本冷凍空調学会

10.4） 山口弘雅, 丹羽英治, 三浦光城, 鳴海大典: 水噴霧による空気熱源ヒートポンプチラーの効率向上効果に関する実測研究, 空気調和・衛生工学会大会学術講演論文集, p.1745-1748, 2005

10.5） 山口弘雅, 丹羽英治, 三浦光城, 鳴海大典, 中澤和弘: 水噴霧による空気熱源ヒートポンプチラーの効率向上効果に関する実測研究 第 2 報 噴霧方式および噴霧水量による効果の差異の分析, 空気調和・衛生工学会大会学術講演論文集, p.1725-1728, 2006

10.6） 中川善博, 長澤浩司, 塩地純夫, 佐藤孝輔, 丹羽英治: 水噴霧による個別分散空調システムの効率向上効果を予測する技術に関する研究, 空気調和・衛生工学会大会学術講演論文集, p.2257-2260, 2011

10.7） 川島実: 外調機のドレン水の散水による空冷ヒートポンプの効率向上, 日本建築学会大会学術講演梗概集, p.1407-1408, 2006

第11章　空気対空気 熱交換器
(Air to Air Heat Exchanger)

11.1　概要

本章では空気から空気へと直接に熱交換を行う機器を取り扱う。空調設備分野において、空気対空気の熱交換器は、外気処理にかかる負荷を削減する目的で導入されることが多い。特に人員密度が高い集会場などで換気量が大きい場合には、建物が消費するエネルギーの多くを外気負荷が占めるため、排気からの熱回収は非常に有効である。また、近年ではクールビズ空調の実現やドラフト感の防止を目的に、吹出し空気の温度を調整する目的で熱交換器を用いる事例もあらわれており、様々な応用方法がある。

空気対空気の代表的な熱交換器は回転型熱交換器と静止型熱交換器である。回転型の熱交換器は熱交換器が持つ熱容量を用いて熱交換を行うものであり、回転する蓄熱体が給気と排気と交互に接触することで熱を移動させる。静止型は隔壁を通して直接に熱を伝播させるものである。両者はさらに顕熱のみを交換する顕熱交換器と、水分も同時に交換する全熱交換器に分けられる。本書では回転型、静止型、それぞれの計算方法を解説する。

写真 11.1　AHU 内に設置された回転型熱交換器

11.2　理論

換気を行う際の室内排気を外気と比較すると、夏の冷房運転時であれば涼しく、冬の暖房運転時であれば温かい。従って、冷房運転時であれば外気から排気に熱を移動させることで、また、暖房運転時であれば排気から外気に熱を移動させることで、外気を冷却あるいは加熱するためのエネルギーを削減することができる。蓄熱体との接触によって熱交換を行う蓄熱式熱交換器と伝熱壁の熱通過によって熱交換を行う換熱式熱交換器とに大別される。空調設備分野では、前者の代表として回転型熱交換器、後者の代表として静止型熱交換器が用いられることが多い。いずれの熱交換器も第 8 章の基礎理論に従う。

11.2.1　回転型熱交換器

1）仕組み

回転型熱交換器は比較的大風量の熱交換器に適しており、冷温水コイルや加湿器などとともに外気処理空調機に組み込まれることが多い。顕熱のみを交換する顕熱交換器と、顕熱と潜熱を交換する全熱交換器があるが、通常は熱回収量の最大化を図るために全熱交換器を組み込む。図 11.1 に回転型熱交換器の構造を示す。

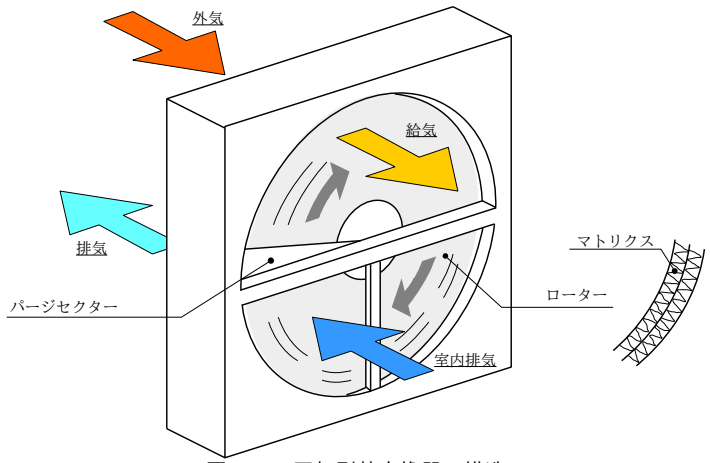


図 11.1　回転型熱交換器の構造

ローターと呼ばれる回転体が連続的に回転しており、この中を外気と室内排気が通過する。図 11.1 では、外気が上部を、排気が下部をそれぞれ通過する。ローターは熱容量をもった素材（マトリクスと呼ぶ）により構成されており、ローターを通過する空気との間で熱交換を行う。例えば冬季あれば熱交換器の下部において排気によって暖められたマトリクスが、ローターの回転により熱交換器の上部に移り、冷たい外気に熱を与えることになる。全熱交換器の場合にはマトリクスにシリカゲルやゼオライトなどの吸着材[†1)]を含浸させ、水分の吸放出も行う。また、排気が室内に逆流しないように、空気の流れが逆転する位置にパージセクターと呼ばれるじゃま板が設けられている。

2）理論

・熱交換の基礎式

一般の熱交換器と同様に第 8 章で解説した熱通過有効度 ε [-]を用いて計算を行う。即ち、式 11.1 に示されるように熱交換器による顕熱交換熱量 Q_S [W]は、排気と給気の入口乾球温度の差$|t_{EAi} - t_{SAi}|$ [K]

†1　汚れなど、人体に有害な物質も吸放出されないのかと不思議に思うかもしれない。ホルムアルデヒドや臭いのもととなる硫化水素などは水分子に比較して分子サイズが大きい。そこで、吸着材の孔径を 3Å 程度にすることで水分子を選択的に吸着するという工夫がされている。

に熱容量流量 mc_{min} [W/K]と熱通過有効度 ε を乗じて計算する。

$$Q_S=\varepsilon\, mc_{min}\left|t_{EAi}-t_{SAi}\right| \tag{11.1}$$

顕熱交換熱量 Q_Sは式 11.2 と式 11.3 で表現することもできる。式 11.2 の ε_{SA}は無限大の伝熱面積を持つ熱交換器における流体の出入口温度差に対する、現実の給気側の出入口温度差の大きさを表している。空調する給気側空気のどれだけの熱量を回収した熱で賄えたかという視点での効率である。逆に式 11.3 は排気側からみた効率であり、排気からどれだけ熱を回収できたかという視点での効率である。ε_{SA}と ε_{EA}は風量バランスに対して逆の傾向を示す†1)ため、どちらの意味で効率を表現しているのかを注意する必要がある。国土交通省の建築設備設計基準や JIS では ε_{SA}を熱交換器の熱交換効率と表現している[11.8) 11.9)]。なお、式 11.1 の ε は mc_{min}=Min(mc_{SA}, mc_{EA})とした場合の ε である。カタログなどで入手できる熱交換効率は通常は ε_{SA}であるため、mc_{EA} = mc_{min} の場合には、式 11.4 を用いて熱通過有効度に変換する必要がある。ε_{SA}と ε_{EA}が得られれば、熱交換器の出口空気温度 T_{SAo}と T_{EAo}は式 11.5 と式 11.6 で計算できる。

$$Q_S=\varepsilon_{SA}\, mc_{SA}\left|t_{SAi}-t_{EAi}\right| \tag{11.2}$$

$$Q_S=\varepsilon_{EA}\, mc_{EA}\left|t_{EAi}-t_{SAi}\right| \tag{11.3}$$

$$\varepsilon=\varepsilon_{SA}\frac{mc_{SA}}{mc_{EA}} \qquad (mc_{EA}=mc_{min}\text{の場合}) \tag{11.4}$$

$$t_{SAo}=t_{SAi}-\varepsilon_{SA}\left(t_{SAi}-t_{EAi}\right) \tag{11.5}$$

$$t_{EAo}=t_{EAi}-\varepsilon_{EA}\left(t_{EAi}-t_{SAi}\right) \tag{11.6}$$

・熱通過有効度の計算

最も単純なモデルは、流体の条件に関わらず熱通過有効度 ε を定格性能値で一定とみなす方法である。HASP/ACSS ではこのモデルを採用している。この場合には熱通過率は、式 11.7 に示すようにローターの回転数比に比例するとみなす。ただし Ω は回転数[1/s = rps]†2)、添字の N は定格性能・定格条件を表わす。

$$\varepsilon=\frac{\Omega}{\Omega_N}\varepsilon_N \tag{11.7}$$

厳密には熱通過有効度 ε は流体の流量比に影響を受け、式 11.8 に示されるように修正移動単位数 NTU_0、熱容量流量比 R_{mc}、マトリクスと流体との熱容量比 R_r、流れ方向伝導係数 λ†3)という 4 つの無次元数で表現できる[11.1) 11.2)]。

$$\varepsilon=f\left(NTU_{0,}R_{mc},R_r,\lambda\right) \tag{11.8}$$

NTU_0は式 11.9 で計算でき、mc は質量流量[kg/s]と比熱[J/(kg·K)]を乗じた熱容量流量 [W/K]、α は対流熱伝達率 [W/(m^2·K)]、A_sは流体とマトリクスの間で温度および物質移動を行う伝熱面積 [m^2]である。熱容量流量の大きい側の流体に添字の max を、小さい側の流体に添字の min を付している。

$$NTU_0=\frac{1}{mc_{min}}\left[\frac{1}{1/(\alpha A_s)_{min}+1/(\alpha A_s)_{max}}\right] \tag{11.9}$$

R_{mc}と R_rはそれぞれ式 11.10 と式 11.11 で計算する。mc_rはマトリクスの熱容量回転量[W/K]であり式 11.12 で計算する。m_rと c_{pr}はマトリクスの質量[kg]と比熱[J/(kg·K)]である。m_{cr}は回転数 Ω に比例する

†1　例えば給気に対して排気の風量が大きい場合には、ε_{SA}は大きく、ε_{EA}は小さくなる。

†2　15 rpm = 0.25 rps 程度で回転することが多い。

†3　対応する日本語が不明のため筆者の造語である。英語では longitudinal conduction parameter と呼ぶ。

ことがわかる。なお、マトリクスの熱容量はカタログ等から把握することが困難であるため、その他の定格性能値および仕様から逆算する。

$$R_{mc}=mc_{min}/mc_{max} \tag{11.10}$$

$$R_r=mc_r/mc_{min} \tag{11.11}$$

$$mc_r=m_r c_{pr}\Omega \tag{11.12}$$

λは熱交換を行う流体を分ける隔壁の内部において流体と同一方向に生じる熱伝導の影響を評価する係数であり式 11.13 で計算する。流路が短く熱交換効率が高い熱交換器ではこの影響が大きく、回転型熱交換器はその代表格である†1)。λ_mは隔壁の熱伝導率[W/(m·K)]、A_kは隔壁の見付面積[m²]、Lはローターの流路の長さ[m]である。λが大きくなるにつれて熱通過有効度は低下する。逆にλ=0 の場合（隔壁の熱伝導率・断面積が小さい、流路が長い、流体の熱容量流量が大きい場合）には、隔壁の熱伝導による影響は無い。

$$\lambda=\frac{\lambda_m A_k}{L\,mc_{min}} \tag{11.13}$$

ローターの回転数が無限大（$R_r=\infty$）で理想的な熱交換が可能な場合の熱通過有効度 ε_∞は通常の対向流型熱交換器と同じように式 11.14 で計算できる。

$$\varepsilon_\infty=\begin{cases}\dfrac{1-\exp[(R_{mc}-1)NTU_0]}{1-R_{mc}\exp[(R_{mc}-1)NTU_0]} & (0<R_{mc}<1, R_r=\infty)\\ NTU_0/(1+NTU_0) & (R_{mc}=1, R_r=\infty)\end{cases} \tag{11.14}$$

現実（$R_r\neq\infty$）の熱交換器の熱通過有効度 εは、理想的な熱交換器の熱通過有効度 ε_∞の関数として表現することができる。熱交換を行う両流体の熱容量流量が等しい場合（$R_{min}=R_{max}$, R_{mc}=1）には式 11.15~11.17で計算できる 11.4)。

$$\varepsilon=\varepsilon_\infty\left(1-\frac{1}{9R_r^{1.93}}\right)(1-C_\lambda) \tag{11.15}$$

$$C_\lambda=\frac{1}{1+NTU_0(1+\lambda\Phi)/(1+\lambda NTU_0)}-\frac{1}{1+NTU_0} \tag{11.16}$$

$$\Phi=\left(\frac{\lambda NTU_0}{1+\lambda NTU_0}\right)^{0.5}\tanh\left\{\frac{NTU_0}{[\lambda NTU_0/(1+\lambda NTU_0)]^{0.5}}\right\}\approx\left(\frac{\lambda NTU_0}{1+\lambda NTU_0}\right)^{0.5}\ for\ NTU_0\geq 3 \tag{11.17}$$

熱容量流量が異なる場合（$0<R_{mc}<1$）には、式 11.18 と 11.19 を用いて NTU_0と R_rを補正する 11.6)。

$$NTU_{0,m}=\frac{2NTU_0 R_{mc}}{1+R_{mc}} \tag{11.18}$$

$$R_{r,m}=\frac{2R_r R_{mc}}{1+R_{mc}} \tag{11.19}$$

得られた $NTU_{0,m}$と $R_{r,m}$を用いて式 11.20 と式 11.21 で計算する。ただし、$C_{\lambda,m}$は式 11.16 と 11.17 において NTU_0を $NTU_{0,m}$に置き換えた値である。

$$\varepsilon=\frac{1-\exp\left\{\varepsilon_m(R_{mc}^2-1)/[2R_{mc}(1-\varepsilon_m)]\right\}}{1-R_{mc}\exp\left\{\varepsilon_m(R_{mc}^2-1)/[2R_{mc}(1-\varepsilon_m)]\right\}} \tag{11.20}$$

†1　回転型全熱交換器は熱交換効率 70%以上、厚み 500mm 以下で設計されることが多い。

$$\varepsilon_m = \frac{NTU_{0,m}}{1+NTU_{0,m}}\left(1-\frac{1}{9R_{r,m}^{1.93}}\right)\left(1-\frac{C_{\lambda,m}}{2-R_{mc}}\right) \tag{11.21}$$

・幾何学形状の想定

実際に式 11.9 の α や A_s、式 11.13 の A_k などを計算するためには、マトリクスの断面形状を把握する必要がある。田中らの報告によればマトリクスの断面形状は図 11.2 の通りで、流路断面積 A_f は 2.584 mm^2、マトリクス断面積 A_m は 0.196 mm^2、周長 L_m は 5.76×10^{-3} m、等価直径 d_e は 1.795×10^{-3} m である。

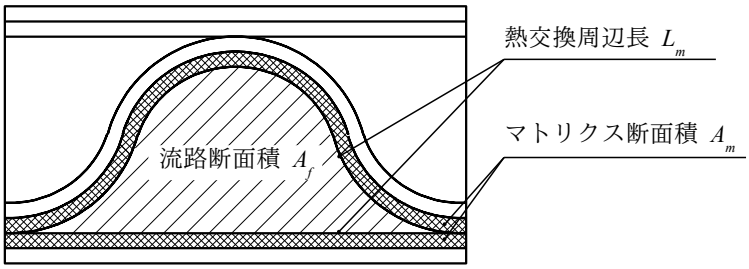


図 11.2　マトリクスの断面形状

対流熱伝達率 α は Schack による空気の管内流の簡易式[11.10]を用いて等価直径 d_e [m]、風速 v [m/s]、乾球温度 t_a [°C]の関数として式 11.22 で計算する。

$$\alpha = \left(4.13+0.195\frac{t_a}{100}\right)\frac{v^{0.75}}{d_e^{0.25}} \tag{11.22}$$

・消費電力の計算

回転型全熱交換器ではローターを回転させるためのモーターの電力が必要となる。多くの運転時間では定格回転数で動作するため、一定の消費電力を計上するモデルでも誤差は小さいと予想できる。しかし、外気温度が室温よりもやや低い中間期において、全熱交換器による熱回収効果と室内内部発熱の影響により冷房負荷が発生する場合があり、この場合には厳密には消費電力が変化しうる。冷房負荷の発生を回避するためには熱回収量を低下させる必要があり、このためには「ローターの発停制御」「ローターの回転数制御」「バイパス制御」という手段がある。

ロータの発停制御は間欠的にローターを回転させることで熱回収量を低下させる方法である。間欠運転する時間の比率を R_{run} [-]とすると、時間平均の出口温度 $\overline{T_{SAo}}$ [K]は、式 11.5 を応用して式 11.23 で計算できる。これを R_{run} について解くと式 11.24 が得られる。停止時の消費電力を 0 kW とすれば、ローターの消費電力 E [kW]は、定格消費電力 E_N [kW]に R_{run} を乗じて式 11.25 で計算できる。

$$\overline{t_{SAo}} = R_{run}t_{SAo}+(1-R_{run})t_{SAi} = R_{run}\left|t_{SAi}-\varepsilon_{SA}(t_{SAi}-t_{EAi})\right|+(1-R_{run})t_{SAi} \tag{11.23}$$

$$R_{run} = \frac{\overline{t_{SAo}}-t_{SAi}}{\varepsilon_{SA}(t_{EAi}-t_{SAi})} \tag{11.24}$$

$$E = R_{run}E_N \tag{11.25}$$

ローターの回転数制御はインバータにより回転数 Ω を制御する方法である。回転数 Ω とモータの消費電力はほぼ比例的だが、これまでに示した数式から明らかなように回転数 Ω と給気温度 T_{SAo} は比例的ではない。従って厳密には T_{SAo} から回転数 Ω を逆算する必要があるが、そもそもロータの消費電力はファンの消費電力などに比較して非常に小さく[†1]、この要素に過度な精度を求めてもあまり益は無いように思う。実用的には式 11.25 の近似で十分であろう。

†1　メーカーカタログによれば 50,000 CMH 程度の風量で精々 1 kW 程度である。必要静圧にもよるが給排気ファンの搬送動力合算値に比較して大きくとも 1/10~1/20 程度の消費電力と捉えて良いだろう。

バイパス制御は一部の空気をロータに通さずバイパスさせることで回収熱量を低下させる方式である。この場合には回転数は変わらないため、消費電力は定格値で変わらない†1)。

・潜熱交換効率

上で解説した熱通過有効度は顕熱交換に関する効率であり、潜熱交換に関してはやや異なる値を取るが、本書のモデルでは顕熱交換効率=潜熱交換効率として扱う。従って、絶対湿度および潜熱交換に関しては式 11.26~式 11.29 が成立する。Q_L [W]は潜熱交換量、γ_0 は水の蒸発潜熱（=2,501×10^3 J/kg）である。

$$Q_L=\varepsilon_{SA} m_{SA} \gamma_0 |W_{SAi}-W_{EAi}| \tag{11.26}$$

$$Q_L=\varepsilon_{EA} m_{EA} \gamma_0 |W_{EAi}-W_{SAi}| \tag{11.27}$$

$$W_{SAo}=W_{SAi}-\varepsilon_{SA}(W_{SAi}-W_{EAi}) \tag{11.28}$$

$$W_{EAo}=W_{EAi}-\varepsilon_{EA}(W_{EAi}-W_{SAi}) \tag{11.29}$$

出口絶対湿度を制御対象とする場合には、式 11.30 および式 11.31 で間欠運転時間比率を計算する。

$$\overline{W_{SAo}}=R_{run} W_{SAo}+(1-R_{run}) W_{SAi}=R_{run}\left| W_{SAi}-\varepsilon_{SA}(W_{SAi}-W_{EAi})\right|+(1-R_{run}) W_{SAi} \tag{11.30}$$

$$R_{run}=\frac{\overline{W_{SAo}}-W_{SAi}}{\varepsilon_{SA}(W_{EAi}-W_{SAi})} \tag{11.31}$$

出口比エンタルピーを制御対象とする場合はやや複雑である。比エンタルピーの定義により式 11.32 が成立するため、これに式 11.23 と式 11.30 を代入し、式 11.33~11.36を得る。解の公式を用いて $0 \le R_{run} \le 1$ となる解を選択する。

$$\overline{h_{SAo}}=c_{pa}\overline{t_{SAo}}+(\gamma_0+c_{pv}\overline{t_{SAo}})\overline{W_{SAo}} \tag{11.32}$$

$$0=a_{run} R_{run}^2+b_{run} R_{run}+c_{run} \tag{11.33}$$

$$a_{run}=\varepsilon_{SA}^2 c_{pv}(t_{EAi}-t_{SAi})(W_{EAi}-W_{SAi}) \tag{11.34}$$

$$b_{run}=\varepsilon_{SA}\left|(h_{EAi}-h_{SAi})-c_{pv}(t_{EAi}-t_{SAi})(W_{EAi}-W_{SAi})\right| \tag{11.35}$$

$$c_{run}=h_{SAi}-\overline{h_{SAo}} \tag{11.36}$$

【例題 11.1】

図 11.2 を参考にローターの直径が 2,150 mm 、奥行きが 400mm の全熱交換器の伝熱面積 A_s [m^2]およびマトリクスの見付面積 A_k[m^2]を求めよ。

【解】

ローターの見付け面積は (2.15 / 2)^2×3.14 = 3.63m^2。図 11.2 により、1 流路あたりの断面積は A_f+A_m=2.584+0.196=2.78 mm^2 であるため、流路の数は 3.63÷(2.78×10^{-6}) =1.31×10^6 となる。1 流路あたりの周長は 5.76×10^{-3}m であるため、ローター全体での周長は、(5.76×10^{-3})×(1.31×10^6)=7.55×10^3 m となる。これに奥行を乗じて伝熱面積 A_s は、(7.55×10^3)×0.40=3,020 m^2 と求められる。

図 11.2 により、マトリクス断面積比率は A_m / (A_f+A_m) = 0.196 / 2.78 = 7.1%である。従ってマトリクス断面積 A_k は 3.63 m² × 7.1% = 0.26 m^2 となる。

【例題 11.2】

例題 11.1 の全熱交換器の定格性能および仕様が下記の通りであったとき、マトリクスの熱容量 mc_r を計算せよ。また、ローター回転数が半分になった時の熱交換効率を計算せよ。ただし空気の密度は 1.2 kg/m^3、比熱は 1006 J/(kg·K)とする。また、外気と排気が通過するローターの面積比は 1:1 とする。

外気風量：10,000 CMH　　外気温湿度：35 ºC , 19.5 g/kg

†1　空気搬送システム全体としては、VAV 制御などで INV ファンが搭載されている場合には、理屈としては、バイパス制御によって低下した抵抗の大きさに応じてファンの回転数が絞られて搬送動力が減少するように思う。しかし、現実には VAV の要求風量と開度情報にもとづき、予め用意したテーブルで回転数を決定するという話を計装業者より聞いたことがあり、このような制御方法であれば必ずしもバイパス制御による搬送動力の低下は生じないようにも思う。

排気風量： 8,000 CMH　　排気温湿度：26 ºC , 10.5 g/kg
熱交換効率 ε_{SA}：72%　　マトリクス（アルミ）熱伝導率：210 W/(m·K)

【解】

両流体の熱容量流量はそれぞれ、

$mc_{SA} = 10{,}000 \div 3{,}600 \times 1.2 \times 1006 = 3.35 \times 10^3$ W/K

$mc_{EA} = 8{,}000 \div 3{,}600 \times 1.2 \times 1006 = 2.68 \times 10^3$ W/K

である。従って熱容量流量比 R_{mc} は

$R_{mc} = 2.68 \times 10^3 \div 3.35 \times 10^3 = 0.8$

となる。$mc_{EA} = mc_{min}$ であるため、式 11.4 を用いて、

$\varepsilon = 72\% \times (3.35 \times 10^3) \div (2.68 \times 10^3) = 90\%$

である。例題 11.2 により、ローター見付断面積に対する流路の面積は 3.63-0.26=3.37m^2 あり、給気（SA）と排気（EA）の風速 v と対流熱伝達率 α は、

$v_{SA} = 10{,}000 \div 3600 \times 1.2 \div (3.37 \div 2) = 1.98$ m/s

$v_{EA} = 8{,}000 \div 3600 \times 1.2 \div (3.37 \div 2) = 1.58$ m/s

$\alpha_{SA} = (4.13 + 0.195 \times 35 \div 100) \times 1.98^{0.75} \div (1.795 \times 10^{-3})^{0.25} = 34.0$ W/(m^2·K)

$\alpha_{EA} = (4.13 + 0.195 \times 26 \div 100) \times 1.58^{0.75} \div (1.795 \times 10^{-3})^{0.25} = 28.6$ W/(m^2·K)

となる。式 11.9 を用いて修正移動単位数 NTU_0 は、

$NTU_0 = 1 / (2.67 \times 10^3) \div (1/(34.0 \times 3{,}020/2) + 1/(28.6 \times 3{,}020/2)) = 8.7$

となる。式 11.18 により補正を行い、

$NTU_{0,m} = (2 \times 8.7 \times 0.8) \div (1 + 0.8) = 7.7$

を得る。式 11.13 により

$\lambda = (210 \times 0.26) \div (0.4 \times 2.68 \times 10^3) = 0.05$

となる。式 11.16 と式 11.17 を用いて、

$\Phi_m = \{(0.05 \times 8.7) / (1 + 0.05 \times 8.7)\}^{0.5} = 0.53$

$C_{\lambda,m} = 1 / (1 + 8.7\ (1 + 0.05 \times 0.53)/(1 + 0.05 \times 8.7)) - 1 / (1 + 87) = 0.035$

となる。式 11.20 を ε_m について解くと式 11.37 が得られ、これに $R_{mc} = 0.8$ と $\varepsilon = 90\%$ を代入すると $\varepsilon_m = 0.82$ となる。

$$\varepsilon_m = \frac{2R_{mc}\ln\left((1-\varepsilon)/(1-\varepsilon R_{mc})\right)}{2R_{mc}\ln\left((1-\varepsilon)/(1-\varepsilon R_{mc})\right) + R_{mc}^2 - 1} \tag{11.37}$$

式 11.21 に $\varepsilon_m = 0.82$、$NTU_{0,m} = 7.7$、$R_{mc} = 0.8$、$C_{\lambda,m} = 0.035$ を代入して $R_{r,m}$ について解くと $R_{r,m} = 1.58$ が得られる。式 11.19 を R_r について解き、

$R_r = 1.58 \times (1 + 0.8) \div (2 \times 0.8) = 1.78$

となる。式 11.11 によりマトリクスの熱容量回転量 mc_r は

$mc_r = 1.78 \times (2.68 \times 10^3) = 4.78 \times 10^3$ W/K

と求められる。

式 11.12 によりローター回転数 Ω と熱容量回転量 mc_r は比例の関係にあるため、ローターの回転数が半分になった場合の $R_{r,m}$ は 1.78×50 % = 0.79 である。流体の流量は変わらないため、$NTU_{0,m} = 7.7$、$R_{mc} = 0.8$、$C_{\lambda,m} = 0.035$ であり、これらと $R_r = 0.52$ を式 11.21 に代入すると $\varepsilon_m = 0.71$ が得られる。$\varepsilon_m = 0.71$ を式 11.20 に代入すると $\varepsilon = 0.79$ となる。式 11.4 を ε_{SA} について解き、

$\varepsilon_{SA} = 0.79 \times (2.68 \times 10^3) \div (3.35 \times 10^3) = 0.63$

である。

・定格熱交換効率の推定

定格条件における熱交換効率 ε_{SA} はカタログ等から入手するが、設備システムに含まれる全熱交換器の台数は多く、すべてについてカタログを参照することはかなりの作業量になる。そこで面風速 v と風量比(m_{SA}/m_{EA})から熱交換効率を推定するための近似式を予め作っておく。式 11.38 は S 社のカタログから作成した定格条件における熱交換効率近似式である。図 11.3 は近似結果であり、プロットがカタログ記載値、実践が近似値である。平均誤差は 0.5 %、最大誤差は 1.9 %程度である。

$$\varepsilon_{SA}=0.0623\frac{m_{SA}}{m_{EA}}-0.0608\,v-0.3093\left(\frac{m_{SA}}{m_{EA}}\right)^2+0.0752\left(\frac{m_{SA}}{m_{EA}}\right)^3+0.0022\,v\frac{m_{SA}}{m_{EA}}+1.0884 \tag{11.38}$$

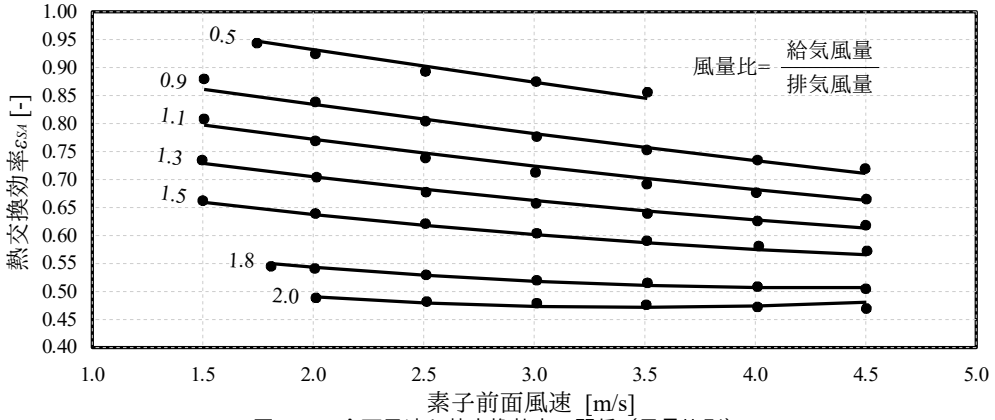


図 11.3　全面風速と熱交換効率の関係（風量比別）

カタログにもとづき、国内 2 社の 5 製品について素子前面風速が 3.0 m/s となるようにローター（直径 d_r [m]）を選定すると図 11.4 が得られる。なお、設計上、推奨される流速は 2~4 m/s 程度である。

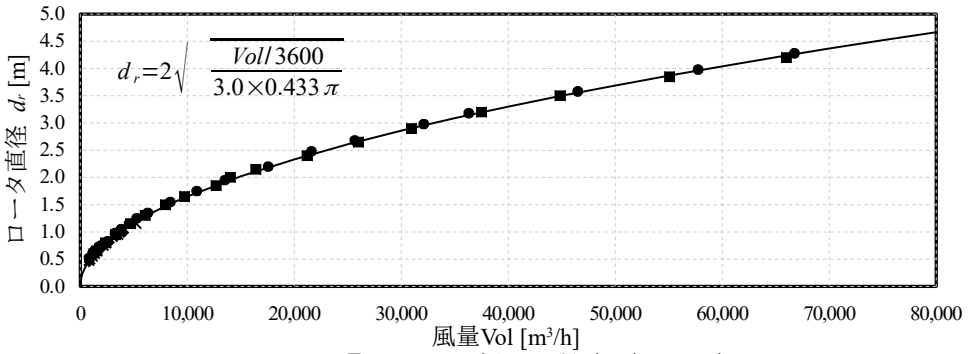


図 11.4　風量とローター直径の関係（風速 3.0 m/s）

11.2.2　静止型熱交換器

1）仕組み

静止型熱交換器の構造を図 11.5 に示す。流路の異なるハニカム状のプレートが交互に重ねられており、排気と給気がプレートの隔壁を介して熱交換を行う。図 11.5 の右にあるように、給気およびファンを組み込み、ユニットとして販売されている製品もある。

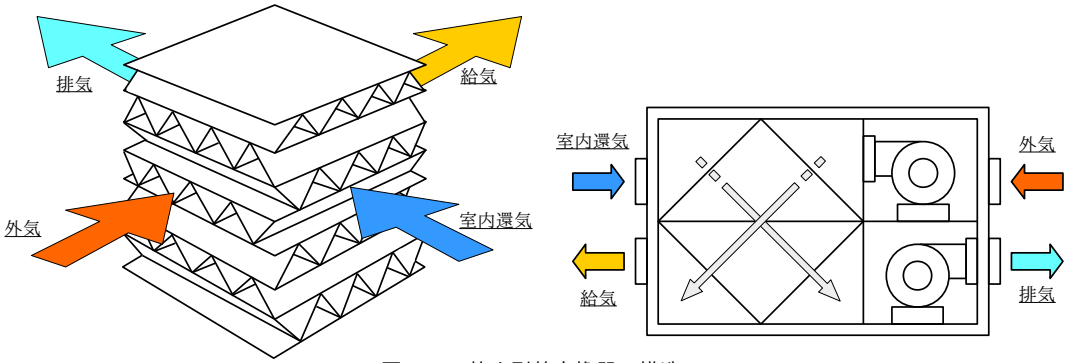


図 11.5　静止型熱交換器の構造 1

図 11.6 に示すように流路を長くとった機種もあり、この場合には直交流型ではなく向流型の熱交換器の傾向を示す。所要スペースは直交型の方が小さいが、熱交換効率は向流型の方が大きい。

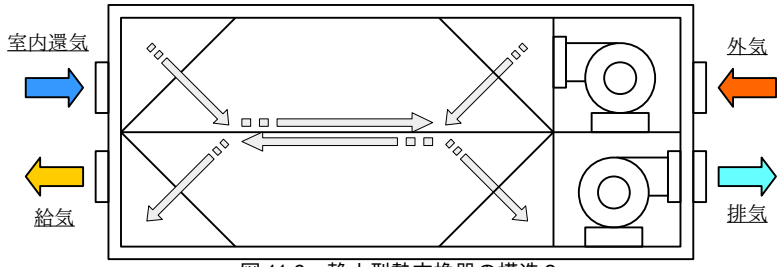


図 11.6　静止型熱交換器の構造 2

顕熱のみを交換する機種は、熱伝導率の高いアルミ等を隔壁の材料として用いる。顕熱に加えて水分も交換する機種は、多孔質の材料を隔壁とすることで、水分移動を可能としている [11.12)]。静止型全熱交換器は回転型熱交換器に比較して構造が単純であるため、小風量の小型装置に向いており、100 CMH 程度の風量から製品が販売されている。

2）理論

熱交換量 Q は、回転型熱交換器と同じように熱通過有効度 ε を用いて式 11.1 で計算する。また、熱通過有効度 ε は、両流体が混合しない直交流または向流の熱交換器であるため、第 8 章において示した式 8.24 と式 8.29 を用いて計算する。

$$Q=\varepsilon mc_{min}\left|T_{EAi}-T_{SAi}\right| \qquad \text{再(11.1)}$$

$$\varepsilon=\begin{cases}\dfrac{1-\exp[(R_{mc}-1)NTU]}{1-R_{mc}\exp[(R_{mc}-1)NTU]} & (0<R_{mc}<1)\\ NTU/(1+NTU) & (R_{mc}=1)\end{cases} \quad \text{（向流型）} \qquad \text{再(8.24)}$$

$$\varepsilon=1-\exp\left(\frac{\exp(-R_{mc}NTU^{0.78})-1}{R_{mc}NTU^{-0.22}}\right) \quad \text{（直交流 ：両流体非混合）} \qquad \text{再(8.29)}$$

顕熱に関しては単純に式 8.24 と式 8.29 を適用すれば計算ができる。潜熱に関しては、顕熱に関する熱通過有効度の式から類推して計算を行う。熱交換器の絶対湿度基準の潜熱貫流率を K_L [kg/(m²·(kg/kg))]とすると、水分移動量 Q_L [kg/s]は式 11.39 で計算できる。顕熱の場合の移動単位数 NTU に類推させて、潜熱基準の移動単位数 NTU_L は式 11.40 で表現できる。

$$Q_L=K_L A_s(x_{SA}-x_{EA}) \qquad (11.39)$$

$$NTU_L=\frac{K_L A_s}{Min(m_{SA}c_{L,SA},m_{EA}c_{L,EA})} \qquad (11.40)$$

顕熱の場合には c は空気の比熱[J/(kg·K)]であり、1 単位の空気に対して、温度を 1 単位上昇させるために必要な熱量であった。一方、潜熱の場合には 1 単位の空気に対して、絶対湿度を 1 単位上昇させるために必要な水分量である。従って、絶対湿度の定義により c_L は常に 1 であり、$c_{L,SA}=c_{L,EA}=1$ となり、式 11.40 は式 11.41 となる。同様に顕熱の熱容量流量比に相当する概念である $R_{mc,L}$ も、式 11.42 に表現されるように単なる質量流量比となる。

$$NTU_L=\frac{K_L A_s}{Min(m_{SA},m_{EA})} \qquad (11.41)$$

$$R_{mc,L}=\frac{Min\left(m_{SA}c_{L,SA},m_{EA}c_{L,EA}\right)}{Max\left(m_{SA}c_{L,SA},m_{EA}c_{L,EA}\right)}=\frac{Min\left(m_{SA},m_{EA}\right)}{Max\left(m_{SA},m_{EA}\right)} \tag{11.42}$$

【例題 11.3】

下記の温湿度条件において、全熱（エンタルピー）交換効率が 68%、顕熱（温度）交換効率が 74%の全熱交換器について、潜熱交換効率を求めよ。

外気条件：34.5 ºC , 75.0 % , 26.3 g/kg , 102.0 kJ/kg , 1.15 kg/m^3

室内条件：26.5 ºC , 64.5 % , 14.0 g/kg ,　62.4 kJ/kg , 1.10 kg/m^3

【解】

式 11.5 にもとづき、熱交換効率を用いて出口空気状態を計算する。

t_{SAo} = 34.5 – 74 % (34.5 – 26.5) = 28.6 ºC

h_{SAo} = 102.0 – 68 % (102.0 – 62.4) = 75.1 kg/kg

乾球温度 28.6 ºC、比エンタルピー 75.1 kg/kg の空気の絶対湿度は 18.1 g/kg である。式 11.5 を用いて、潜熱交換効率 $\varepsilon_{SA,L}$ は

$\varepsilon_{SA,L}$ = (26.3 – 18.1) / (26.3 – 14.0) = 67 %

となる。

【例題 11.4】

例題 11.3 の全熱交換器について、潜熱熱貫流率 K_LA_s [kg/(kg/kg)]を計算せよ。ただし、風量は給排気ともに 150 CMH の直交流の熱交換器とする。また、風量を 110 CMH に変更した場合の潜熱交換効率を計算せよ。

【解】

質量流量はそれぞれ、

m_{SA} = 150 / 3600 × 1.10 = 0.046 kg/s

m_{EA} = 150 / 3600 × 1.15 = 0.048 kg/s

である。従って質量流量比は $m_{SA} \div m_{EA}$ = 0.956 となる。$m_{min} = m_{SA}$ のため、$\varepsilon = \varepsilon_{SA}$ = 67 %である。図 8.4 または式 8.29 を用いて移動単位数を計算すると NTU=2.55 が得られる。式 11.41 を K_LA_s について解き

K_LA_s = 2.55 × 0.046 = 0.117 kg/(kg/kg)

となる。

風量を 110 CMH に変更した場合の質量流量は m_{SA} = 0.034 kg/s である。温湿度条件が同一のため、質量流量比は 0.956 で変わらない。移動単位数は NTU = 0.117 ÷ 0.034 = 3.48 となる。図 8.4 または式 8.29 を用いて、熱通過有効度 ε は 72 %である。$m_{min} = m_{SA}$ のため、ε_{SA} = であり、潜熱交換効率は 72 %である。

11.3　計算法

11.3.1　「回転型熱交換器クラス」の作成

1）クラスメソッドの定義

プログラム 11.1 に回転型熱交換器クラスの定数宣言を示す。図 11.2 で説明したマトリクスの構造を所与とする。

プログラム 11.1　回転型熱交換器クラスの定数宣言

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
/// <summary>SA と EA の流路の比率は 1:1 とする</summary>
private const double SA_AREA_RATE = 0.5;

//マトリクスの 1 単位の構造
//田中宏史:夏期条件での回転式全熱交換器の性能解析および実験
//SHASE 論文集 vol.59, 1995 より
/// <summary>単位マトリクスの流路断面積[m2/流路]</summary>
private const double SA_AIRSTREAM = 2.584e-6;

/// <summary>単位マトリクスの断面積[m2/流路]</summary>
private const double SA_MATRIX = 0.196e-6;

/// <summary>単位マトリクスの周長[m/流路]</summary>
private const double PERIMETER = 5.76e-3;

/// <summary>マトリクスの等価直径[m]</summary>
```

```
17 private const double E_DIAMETER = 1.795e-3;
```

プログラム 11.2 にローターの幾何学形状の計算処理を示す。例題 11.1 の実装であり、プログラム 11.1 で宣言した各種の定数を用いて計算を行う。

プログラム 11.2　ローターの幾何学形状の計算

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
 1 /// <summary>ローターの幾何学形状を計算する</summary>
 2 /// <param name="diameter">ローター直径[m]</param>
 3 /// <param name="length">ローター奥行き[m]</param>
 4 /// <param name="frontalArea">ロータ前面見付面積[m2]</param>
 5 /// <param name="airStreamArea">流路断面積[m2]</param>
 6 /// <param name="matrixArea">マトリクス断面積[m2]</param>
 7 /// <param name="heatTransferArea">伝熱面積[m2]</param>
 8 public static void GetRotaryGeometrics
 9   (double diameter, double length, out double frontalArea, out double airStreamArea,
10   out double matrixArea, out double heatTransferArea)
11 {
12   //見付面積[m2]の計算
13   frontalArea = diameter * diameter * Math.PI / 4;
14   //流路面積[m2]の計算
15   airStreamArea = frontalArea * SA_AIRSTREAM / (SA_AIRSTREAM + SA_MATRIX);
16   //マトリクス面積[m2]の計算
17   matrixArea = frontalArea - airStreamArea;
18   //流路数の計算
19   double airStreamNumber = frontalArea / (SA_AIRSTREAM + SA_MATRIX);
20   //伝熱面積[m2]の計算
21   heatTransferArea = airStreamNumber * PERIMETER * length;
22 }
```

プログラム 11.3 に無次元数の計算処理を示す。プログラム 11.2 で計算したローターの幾何学形状、流体の温湿度条件および流量にもとづいて、修正移動単位数 $NTU_{0,m}$、熱容量流量 mc_{min}、熱容量流量比 R_{mc}、流路方向の熱伝導に関する補正係数 C_{λ} を計算する。

プログラム 11.3　無次元数の計算

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
 1 /// <summary>無次元数を計算する</summary>
 2 /// <param name="frontalArea">ロータ前面見付面積[m2]</param>
 3 /// <param name="airStreamArea">流路断面積[m2]</param>
 4 /// <param name="matrixArea">マトリクス断面積[m2]</param>
 5 /// <param name="heatTransferArea">伝熱面積[m2]</param>
 6 /// <param name="length">ローター奥行き[m]</param>
 7 /// <param name="thermalConductivity">マトリクスの熱伝導率[W/(mK)]</param>
 8 /// <param name="saFlowVolume">給気風量[CMH]</param>
 9 /// <param name="eaFlowVolume">排気風量[CMH]</param>
10 /// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
11 /// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
12 /// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
13 /// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
14 /// <param name="mcMin">熱容量流量[W/m]</param>
15 /// <param name="rmc">熱容量流量比[-]</param>
16 /// <param name="ntu0m">修正移動単位数[-]</param>
17 /// <param name="cLambda">流路方向の熱伝導補正係数[-]</param>
18 /// <param name="isMcMinSASide">給気側の熱容量流量が小さい場合に true</param>
19 public static void GetDimensionLessVariables
20   (double frontalArea, double airStreamArea, double matrixArea, double heatTransferArea, double length,
21   double thermalConductivity, double saFlowVolume, double eaFlowVolume, double inletSADrybulbTemperature,
22   double inletSAHumidityRatio, double inletEADrybulbTemperature, double inletEAHumidityRatio,
23   out double mcMin, out double rmc, out double ntu0m, out double cLambda, out bool isMcMinSASide)
24 {
25   //熱容量流量比[-]の計算
26   double svSA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
27     (inletSADrybulbTemperature, inletSAHumidityRatio, ATMOSPHERIC_PRESSURE);
28   double svEA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
29     (inletEADrybulbTemperature, inletEAHumidityRatio, ATMOSPHERIC_PRESSURE);
30   double mSA = saFlowVolume / 3600d / svSA;
31   double mEA = eaFlowVolume / 3600d / svEA;
32   double mcSA = mSA * MoistAir.GetSpecificHeat(inletSAHumidityRatio) * 1000;
33   double mcEA = mEA * MoistAir.GetSpecificHeat(inletEAHumidityRatio) * 1000;
34   mcMin = Math.Min(mcSA, mcEA);
35   double mcMax = Math.Max(mcSA, mcEA);
36   rmc = mcMin / mcMax;
37   isMcMinSASide = (mcSA == mcMin);
38
39   //風速[m/s]の計算
```

```
40   double vSA = saFlowVolume / 3600d / (airStreamArea * SA_AREA_RATE);
41   double vEA = eaFlowVolume / 3600d / (airStreamArea * (1 - SA_AREA_RATE));
42   //対流熱伝達率[W/(m2K)]の計算
43   double ed25 = 1d / Math.Pow(E_DIAMETER, 0.25);
44   double alphaSA = (4.13 + 0.195 * inletSADrybulbTemperature / 100) * Math.Pow(vSA, 0.75) * ed25;
45   double alphaEA = (4.13 + 0.195 * inletEADrybulbTemperature / 100) * Math.Pow(vEA, 0.75) * ed25;
46   //修正移動単位数[-]の計算
47   double ha = 1 / (alphaSA * heatTransferArea * SA_AREA_RATE)
48     + 1 / (alphaEA * heatTransferArea * (1 - SA_AREA_RATE));
49   double ntu0 = 1 / mcMin / ha;
50   ntu0m = (2 * ntu0 * rmc) / (1 + rmc);
51
52   //流路方向の熱伝導補正係数[-]の計算
53   double lambda = (thermalConductivity * matrixArea) / (length * mcMin);
54   double phi = Math.Sqrt((lambda * ntu0m) / (1 + lambda * ntu0m));
55   phi = phi * Math.Tanh(ntu0m / phi);
56   cLambda = 1 / (1 + ntu0m * (1 + lambda * phi) / (1 + lambda * ntu0m)) - 1 / (1 + ntu0m);
57 }
```

プログラム 11.4 にマトリクスの熱容量回転量の計算処理を示す。例題 11.2 の実装である。まず、25 行でプログラム 11.3 を用いて定格条件における無次元数を計算する。入力としては給気側の熱交換効率 ε_{SA} が与えられるため、31~33 行で熱容量流量の大小を確認し、必要に応じて式 11.4 を用いて熱通過率 ε を計算する。熱容量流量がほぼ等しい場合には、式 11.15 を用いて R_r を計算する（36~43 行）。熱容量流量が異なる場合には、式 11.18~11.21などを用いて R_r を計算する（44~53 行）。

プログラム 11.4　マトリクスの熱容量回転量の計算

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
 1 /// <summary>マトリクスの熱容量回転量[W/K]を計算する</summary>
 2 /// <param name="efficiency">熱交換効率 εSA[-]</param>
 3 /// <param name="frontalArea">ロータ前面見付面積[m2]</param>
 4 /// <param name="airStreamArea">流路断面積[m2]</param>
 5 /// <param name="matrixArea">マトリクス断面積[m2]</param>
 6 /// <param name="heatTransferArea">伝熱面積[m2]</param>
 7 /// <param name="length">ローター奥行き[m]</param>
 8 /// <param name="thermalConductivity">マトリクスの熱伝導率[W/(mK)]</param>
 9 /// <param name="supplyAirFlowVolume">給気風量[CMH]</param>
10 /// <param name="exhaustAirFlowVolume">排気風量[CMH]</param>
11 /// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
12 /// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
13 /// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
14 /// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
15 /// <returns>マトリクスの熱容量回転量[W/K]</returns>
16 public static double GetMatrixHeatCapacity
17   (double efficiency, double frontalArea, double airStreamArea, double matrixArea,
18   double heatTransferArea, double length, double thermalConductivity, double supplyAirFlowVolume,
19   double exhaustAirFlowVolume, double inletSADrybulbTemperature, double inletSAHumidityRatio,
20   double inletEADrybulbTemperature, double inletEAHumidityRatio)
21 {
22   //無次元数を取得
23   double ntu0m, rmc, cLambda, mcMin;
24   bool isMcMinSASide;
25   GetDimensionLessVariables
26     (frontalArea, airStreamArea, matrixArea, heatTransferArea, length, thermalConductivity,
27     supplyAirFlowVolume, exhaustAirFlowVolume, inletSADrybulbTemperature, inletSAHumidityRatio,
28     inletEADrybulbTemperature, inletEAHumidityRatio,
29     out mcMin, out rmc, out ntu0m, out cLambda, out isMcMinSASide);
30
31   double effectiveness;
32   if (isMcMinSASide) effectiveness = efficiency;
33   else effectiveness = efficiency / rmc;
34
35   double rr;
36   if (0.99 < rmc)
37   {
38     //熱容量流量がほぼ等しい場合
39     double ee = ntu0m / (1 + ntu0m);
40     rr = effectiveness / ee / (1 - cLambda);
41     if (1 <= rr) rr = 100;
42     else rr = Math.Pow(1 / (9 * (1 - rr)), 1 / 1.93);
43   }
44   else
45   {
```

```
46     //熱容量流量が異なる場合
47     double em = 2 * rmc * Math.Log((1 - effectiveness) / (1 - effectiveness * rmc));
48     em = em / (em + rmc * rmc - 1);
49     double rm = em * ((1 + ntu0m) / ntu0m) / (1 - cLambda / (2 - rmc));
50     if (1 <= rm) rm = 100;
51     else rm = Math.Pow(1 / (9 * (1 - rm)), 1 / 1.93);
52     rr = rm * (1 + rmc) / (2 * rmc);
53   }
54   return rr * mcMin;
55 }
```

プログラム 11.5 に熱通過有効度の計算処理を示す。入力として各種の無次元数を与える。熱容量流量が等しいか否かによって条件分岐を行い、プログラム 11.4 とは逆の計算で熱通過有効度を求める。

プログラム 11.5　熱通過有効度の計算

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
 1 /// <summary>熱通過有効度[-]を計算する</summary>
 2 /// <param name="mcMin">熱容量流量[W/m]</param>
 3 /// <param name="rmc">熱容量流量比[-]</param>
 4 /// <param name="ntu0m">修正移動単位数[-]</param>
 5 /// <param name="cLambda">流路方向の熱伝導補正係数[-]</param>
 6 /// <param name="rr">熱容量回転量比[-]</param>
 7 /// <param name="supplyAirFlowVolume">給気風量[CMH]</param>
 8 /// <param name="exhaustAirFlowVolume">排気風量[CMH]</param>
 9 /// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
10 /// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
11 /// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
12 /// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
13 /// <returns>熱通過有効度[-]</returns>
14 public static double GetEffectiveness
15   (double mcMin, double rmc, double ntu0m, double cLambda, double rr, double supplyAirFlowVolume,
16   double exhaustAirFlowVolume, double inletSADrybulbTemperature, double inletSAHumidityRatio,
17   double inletEADrybulbTemperature, double inletEAHumidityRatio)
18 {
19   double e;
20   if (0.99 < rmc)
21   {
22     //熱容量流量がほぼ等しい場合
23     double ee = ntu0m / (1 + ntu0m);
24     e = ee * (1 - 1 / (9 * Math.Pow(rr, 1.93))) * (1 - cLambda);
25   }
26   else
27   {
28     //熱容量流量が異なる場合
29     double rrm = 2 * rr * rmc / (1 + rmc);
30     double em = ntu0m / (1 + ntu0m) * (1 - 1 / (9 * Math.Pow(rrm, 1.93)));
31     em *= 1 - cLambda / (2 - rmc);
32     e = Math.Exp(em * (rmc * rmc - 1) / (2 * rmc * (1 - em)));
33     e = (1 - e) / (1 - rmc * e);
34   }
35   return Math.Min(1, Math.Max(e, 0));
36 }
```

2）回転型熱交換器クラスの作成

プログラム 11.6 に回転型熱交換器クラスのインスタンス変数とプロパティを示す。

プログラム 11.6　回転型熱交換器クラスのプロパティおよびインスタンス変数

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
 1 /// <summary>マトリクスの熱伝導率[W/(mK)]</summary>
 2 private double thermalConductivity;
 3
 4 /// <summary>ロータ前面見付面積[m2]</summary>
 5 private double frontalArea;
 6
 7 /// <summary>流路断面積[m2]</summary>
 8 private double airStreamArea;
 9
10 /// <summary>マトリクス断面積[m2]</summary>
11 private double matrixArea;
12
13 /// <summary>伝熱面積[m2]</summary>
14 private double heatTransferArea;
15
16 /// <summary>マトリクスの熱容量回転量[W/K]</summary>
```

```
17 private double matrixHeatCapacity;
18
19 /// <summary>定格の熱通過率[-]</summary>
20 private double nominalEpsilon;
21
22 /// <summary>詳細モデルか否かを取得する</summary>
23 /// <remarks>簡易モデルは風量によらず熱通過有効度一定</remarks>
24 public bool IsDetailedModel { get; private set; }
25
26 /// <summary>全熱交換器か否かを取得する</summary>
27 public bool IsDesiccantWheel { get; private set; }
28
29 /// <summary>ローターの直径[m]を取得する</summary>
30 public double Diameter { get; private set; }
31
32 /// <summary>ローターの奥行き[m]を取得する</summary>
33 public double Depth { get; private set; }
34
35 /// <summary>給気風量[CMH]を取得する</summary>
36 public double SupplyAirFlowVolume { get; private set; }
37
38 /// <summary>排気風量[CMH]を取得する</summary>
39 public double ExhaustAirFlowVolume { get; private set; }
40
41 /// <summary>給気入口乾球温度[C]を取得する</summary>
42 public double SupplyAirInletDrybulbTemperature { get; private set; }
43
44 /// <summary>排気入口乾球温度[C]を取得する</summary>
45 public double ExhaustAirInletDrybulbTemperature { get; private set; }
46
47 /// <summary>給気出口乾球温度[C]を取得する</summary>
48 public double SupplyAirOutletDrybulbTemperature { get; private set; }
49
50 /// <summary>排気出口乾球温度[C]を取得する</summary>
51 public double ExhaustAirOutletDrybulbTemperature { get; private set; }
52
53 /// <summary>給気入口絶対湿度[kg/kg]を取得する</summary>
54 public double SupplyAirInletHumidityRatio { get; private set; }
55
56 /// <summary>排気入口絶対湿度[kg/kg]を取得する</summary>
57 public double ExhaustAirInletHumidityRatio { get; private set; }
58
59 /// <summary>給気出口絶対湿度[kg/kg]を取得する</summary>
60 public double SupplyAirOutletHumidityRatio { get; private set; }
61
62 /// <summary>排気出口絶対湿度[kg/kg]を取得する</summary>
63 public double ExhaustAirOutletHumidityRatio { get; private set; }
64
65 /// <summary>給気側熱交換効率 εSA[-]を取得する</summary>
66 public double Efficiency { get; private set; }
67
68 /// <summary>定格消費電力[kW]を設定・取得する</summary>
69 public double NominalElectricity { get; set; }
70
71 /// <summary>消費電力[kW]を取得する</summary>
72 public double Electricity { get; private set; }
```

プログラム 11.7 に回転型熱交換器クラスの初期化処理を示す。1~12 行は一定の熱通過率を用いる簡易モデルのコンストラクタであり、引数は少ない。IsDetailedModel を偽に設定する。63~104 行は詳細モデルの初期化処理である。プログラム 11.2 を用いてローターの直径および奥行きから幾何学形状を算出し、インスタンス変数などに保存する。また、これらの情報を用いて 90~95 行で熱容量回転量を計算する。最後に 102 行で成行計算（後述）を行って状態値を初期化する。14~35 行はこの初期化処理をそのまま適用したコンストラクタである。モデル化作業を軽減するため、図 11.3 と図 11.4 を用いて熱通過率とローター直径の設定を自動化したコンストラクタを 37~61 行に定義する。

プログラム 11.7　回転型熱交換器クラスのコンストラクタ

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
1 /// <summary>インスタンスを初期化する</summary>
2 /// <param name="nominalEpsilon">定格の熱通過率[-]</param>
3 /// <param name="isDesiccantWheel">デシカントローターの場合には true</param>
4 /// <param name="nominalElectricity">定格消費電力[kW]</param>
```

```
/// <remarks>風量によらず熱通過有効度を一定とする簡易モデル</remarks>
public RotaryRegenerator(double nominalEpsilon, bool isDesiccantWheel, double nominalElectricity)
{
  IsDetailedModel = false;
  this.nominalEpsilon = nominalEpsilon;
  IsDesiccantWheel = isDesiccantWheel;
  NominalElectricity = nominalElectricity;
}

/// <summary>インスタンスを初期化する</summary>
/// <param name="efficiency">定格の給気側熱交換効率 εSA[-]</param>
/// <param name="diameter">ローター直径[m]</param>
/// <param name="depth">ローター奥行き[m]</param>
/// <param name="thermalConductivity">マトリクスの熱伝導率[W/(mK)]</param>
/// <param name="saFlowVolume">給気風量[CMH]</param>
/// <param name="eaFlowVolume">排気風量[CMH]</param>
/// <param name="isDesiccantWheel">デシカントローターの場合には true</param>
/// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
/// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
/// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
/// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
/// <param name="nominalElectricity">定格消費電力</param>
public RotaryRegenerator
  (double efficiency, double diameter, double depth, double thermalConductivity, double saFlowVolume,
  double eaFlowVolume, bool isDesiccantWheel, double inletSADrybulbTemperature, double inletSAHumidityRatio,
  double inletEADrybulbTemperature, double inletEAHumidityRatio, double nominalElectricity)
{
  initialize(efficiency, diameter, depth, thermalConductivity, saFlowVolume, eaFlowVolume,
    isDesiccantWheel, inletSADrybulbTemperature, inletSAHumidityRatio, inletEADrybulbTemperature,
    inletEAHumidityRatio, nominalElectricity);
}

/// <summary>インスタンスを初期化する</summary>
/// <param name="depth">ローター奥行き[m]</param>
/// <param name="saFlowVolume">給気風量[CMH]</param>
/// <param name="eaFlowVolume">排気風量[CMH]</param>
/// <param name="isDesiccantWheel">デシカントローターの場合には true</param>
/// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
/// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
/// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
/// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
public RotaryRegenerator
  (double depth, double saFlowVolume, double eaFlowVolume, bool isDesiccantWheel,
  double inletSADrybulbTemperature, double inletSAHumidityRatio,
  double inletEADrybulbTemperature, double inletEAHumidityRatio)
{
  const double VEL = 3.0;
  double diameter = 2 * Math.Sqrt(saFlowVolume / 3600 / (VEL * 0.433 * Math.PI));
  double mr = saFlowVolume / eaFlowVolume;
  double eff = 1.0884 - 0.0608 * VEL + mr * ((0.0623 + 0.0022 * VEL) + mr * (-0.3093 + mr * 0.0752));
  double nomElec = 1e-5 * saFlowVolume + 0.09;
  IsDesiccantWheel = isDesiccantWheel;

  initialize
    (eff, diameter, depth, 210, saFlowVolume, eaFlowVolume, IsDesiccantWheel, inletSADrybulbTemperature,
    inletSAHumidityRatio, inletEADrybulbTemperature, inletEAHumidityRatio, nomElec);
}

/// <summary>インスタンスを初期化する</summary>
/// <param name="efficiency">定格の給気側熱交換効率 εSA[-]</param>
/// <param name="diameter">ローター直径[m]</param>
/// <param name="depth">ローター奥行き[m]</param>
/// <param name="thermalConductivity">マトリクスの熱伝導率[W/(mK)]</param>
/// <param name="saFlowVolume">給気風量[CMH]</param>
/// <param name="eaFlowVolume">排気風量[CMH]</param>
/// <param name="isDesiccantWheel">デシカントローターの場合には true</param>
/// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
/// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
/// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
/// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
/// <param name="nominalElectricity">定格消費電力</param>
private void initialize
  (double efficiency, double diameter, double depth, double thermalConductivity,
  double saFlowVolume, double eaFlowVolume, bool isDesiccantWheel,
  double inletSADrybulbTemperature, double inletSAHumidityRatio,
  double inletEADrybulbTemperature, double inletEAHumidityRatio, double nominalElectricity)
{
  IsDetailedModel = true;

  //幾何学形状を保存・計算
```

```
 85   Depth = depth;
 86   Diameter = diameter;
 87   GetRotaryGeometrics
 88     (diameter, depth, out frontalArea, out airStreamArea, out matrixArea, out heatTransferArea);
 89
 90   //マトリクスの熱伝導率[W/(mK)]を保存して熱容量を逆算
 91   this.thermalConductivity = thermalConductivity;
 92   matrixHeatCapacity = GetMatrixHeatCapacity
 93     (efficiency, frontalArea, airStreamArea, matrixArea, heatTransferArea, depth, thermalConductivity,
 94     saFlowVolume, eaFlowVolume, inletSADrybulbTemperature, inletSAHumidityRatio,
 95     inletEADrybulbTemperature, inletEAHumidityRatio);
 96
 97   //ローターの種類と消費電力を保存
 98   IsDesiccantWheel = isDesiccantWheel;
 99   NominalElectricity = nominalElectricity;
100
101   //定格条件で成り行き計算
102   UpdateState(saFlowVolume, eaFlowVolume, 1.0, inletSADrybulbTemperature, inletSAHumidityRatio,
103     inletEADrybulbTemperature, inletEAHumidityRatio);
104 }
```

プログラム 11.8 に熱交換効率の計算処理を示す。1~44 行は詳細モデルの計算である。プログラム 11.3 を使って 19 行で無次元数を求め、26, 29 行で回転数制御の効果を含めた熱通過有効度を計算する。給気側の熱交換効率を出力するため、式 11.4 にもとづいて熱容量流量の大小関係に応じて 33~43 行で換算を行う。46~84 行は簡易モデルの計算である。熱通過率は一定とするモデルであるため、回転数の影響と熱容量流量の大小関係だけを補正する。

プログラム 11.8　熱交換効率の計算

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
  1 /// <summary>熱交換効率[-]を計算する（詳細モデル）</summary>
  2 /// <param name="supplyAirFlowVolume">給気風量[CMH]</param>
  3 /// <param name="exhaustAirFlowVolume">排気風量[CMH]</param>
  4 /// <param name="rotatingRate">ローター回転率[-]</param>
  5 /// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
  6 /// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
  7 /// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
  8 /// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
  9 /// <param name="effSA">出力:SA 側熱交換効率 εSA[-]</param>
 10 /// <param name="effEA">出力:EA 側熱交換効率 εSA[-]</param>
 11 private void getEfficiency_Detailed
 12   (double supplyAirFlowVolume, double exhaustAirFlowVolume, double rotatingRate,
 13   double inletSADrybulbTemperature, double inletSAHumidityRatio,
 14   double inletEADrybulbTemperature, double inletEAHumidityRatio, out double effSA, out double effEA)
 15 {
 16   //無次元数を計算
 17   double ntu0m, rmc, cLambda, mcMin;
 18   bool isMcMinSASide;
 19   GetDimensionLessVariables
 20     (frontalArea, airStreamArea, matrixArea, heatTransferArea, Depth,
 21     thermalConductivity, supplyAirFlowVolume, exhaustAirFlowVolume,
 22     inletSADrybulbTemperature, inletSAHumidityRatio, inletEADrybulbTemperature, inletEAHumidityRatio,
 23     out mcMin, out rmc, out ntu0m, out cLambda, out isMcMinSASide);
 24
 25   //回転数制御の効果を計算
 26   double rr = (matrixHeatCapacity * rotatingRate) / mcMin;
 27
 28   //熱通過有効度[-]を計算
 29   double eff = GetEffectiveness
 30     (mcMin, rmc, ntu0m, cLambda, rr, supplyAirFlowVolume, exhaustAirFlowVolume,
 31     inletSADrybulbTemperature, inletSAHumidityRatio, inletEADrybulbTemperature, inletEAHumidityRatio);
 32
 33   //熱交換効率[-]を計算
 34   if (isMcMinSASide)
 35   {
 36     effSA = eff;
 37     effEA = eff * rmc;
 38   }
 39   else
 40   {
 41     effSA = eff * rmc;
 42     effEA = eff;
 43   }
 44 }
 45
 46 /// <summary>熱交換効率[-]を計算する（簡易モデル）</summary>
```

```
47 /// <param name="supplyAirFlowVolume">給気風量[CMH]</param>
48 /// <param name="exhaustAirFlowVolume">排気風量[CMH]</param>
49 /// <param name="rotatingRate">ローター回転率[-]</param>
50 /// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
51 /// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
52 /// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
53 /// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
54 /// <param name="effSA">出力:SA 側熱交換効率 εSA[-]</param>
55 /// <param name="effEA">出力:EA 側熱交換効率 εSA[-]</param>
56 private void getEfficiency_Simplified
57   (double supplyAirFlowVolume, double exhaustAirFlowVolume, double rotatingRate,
58   double inletSADrybulbTemperature, double inletSAHumidityRatio,
59   double inletEADrybulbTemperature, double inletEAHumidityRatio, out double effSA, out double effEA)
60 {
61   //熱容量流量比[-]の計算
62   double svSA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
63     (inletSADrybulbTemperature, inletSAHumidityRatio, ATMOSPHERIC_PRESSURE);
64   double svEA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
65     (inletEADrybulbTemperature, inletEAHumidityRatio, ATMOSPHERIC_PRESSURE);
66   double mSA = supplyAirFlowVolume / 3600d / svSA;
67   double mEA = exhaustAirFlowVolume / 3600d / svEA;
68   double mcSA = mSA * MoistAir.GetSpecificHeat(inletSAHumidityRatio) * 1000;
69   double mcEA = mEA * MoistAir.GetSpecificHeat(inletEAHumidityRatio) * 1000;
70   double mcMin = Math.Min(mcSA, mcEA);
71   double mcMax = Math.Max(mcSA, mcEA);
72   double rmc = mcMin / mcMax;
73
74   if (mcSA == mcMin)
75   {
76     effSA = nominalEpsilon * rotatingRate;
77     effEA = nominalEpsilon * rotatingRate * rmc;
78   }
79   else
80   {
81     effSA = nominalEpsilon * rotatingRate * rmc;
82     effEA = nominalEpsilon * rotatingRate;
83   }
84 }
```

プログラム 11.9 に回転型熱交換器の成り行き計算処理を示す。風量や回転数が 0 の場合には 71~80 行のメソッドで機器を停止する（27~32 行）。モデルの種別（簡易・詳細）の別に応じて 34~49 行で熱交換効率を計算し、式 11.5 と式 11.6 で出口空気の状態を計算する（51~55 行）。デシカントローターの場合には式 11.28 と式 11.29 で出口空気状態を計算し（56~63 行）、顕熱ローターの場合には出口空気絶対湿度=入口空気絶対湿度とする（64~68 行）。

プログラム 11.9　回転型熱交換器の成り行き計算

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
 1 /// <summary>成り行き状態を計算する</summary>
 2 /// <param name="supplyAirFlowVolume">給気風量[CMH]</param>
 3 /// <param name="exhaustAirFlowVolume">排気風量[CMH]</param>
 4 /// <param name="rotatingRate">ローター回転率[-]</param>
 5 /// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
 6 /// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
 7 /// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
 8 /// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
 9 public void UpdateState
10   (double supplyAirFlowVolume, double exhaustAirFlowVolume, double rotatingRate,
11   double inletSADrybulbTemperature, double inletSAHumidityRatio,
12   double inletEADrybulbTemperature, double inletEAHumidityRatio)
13 {
14   //風量を保存
15   SupplyAirFlowVolume = supplyAirFlowVolume;
16   ExhaustAirFlowVolume = exhaustAirFlowVolume;
17
18   //入口空気状態を保存
19   SupplyAirInletDrybulbTemperature = inletSADrybulbTemperature;
20   SupplyAirInletHumidityRatio = inletSAHumidityRatio;
21   ExhaustAirInletDrybulbTemperature = inletEADrybulbTemperature;
22   ExhaustAirInletHumidityRatio = inletEAHumidityRatio;
23
24   //消費電力は回転率に比例
25   Electricity = NominalElectricity * rotatingRate;
26
27   //風量・回転数が 0 の場合
```

```
28   if (supplyAirFlowVolume <= 0 || exhaustAirFlowVolume <= 0 || rotatingRate <= 0)
29   {
30     ShutOff();
31     return;
32   }
33
34   double effSA, effEA;
35   if (IsDetailedModel)
36   {
37     getEfficiency_Detailed
38       (supplyAirFlowVolume, exhaustAirFlowVolume, rotatingRate,
39       inletSADrybulbTemperature, inletSAHumidityRatio,
40       inletEADrybulbTemperature, inletEAHumidityRatio, out effSA, out effEA);
41   }
42   else
43   {
44     getEfficiency_Simplified
45       (supplyAirFlowVolume, exhaustAirFlowVolume, rotatingRate,
46       inletSADrybulbTemperature, inletSAHumidityRatio,
47       inletEADrybulbTemperature, inletEAHumidityRatio, out effSA, out effEA);
48   }
49   Efficiency = effSA;
50
51   //出口空気状態の計算
52   SupplyAirOutletDrybulbTemperature = SupplyAirInletDrybulbTemperature -
53     effSA * (SupplyAirInletDrybulbTemperature - ExhaustAirInletDrybulbTemperature);
54   ExhaustAirOutletDrybulbTemperature = ExhaustAirInletDrybulbTemperature -
55     effEA * (ExhaustAirInletDrybulbTemperature - SupplyAirInletDrybulbTemperature);
56   //水分交換
57   if (IsDesiccantWheel)
58   {
59     SupplyAirOutletHumidityRatio = SupplyAirInletHumidityRatio -
60       effSA * (SupplyAirInletHumidityRatio - ExhaustAirInletHumidityRatio);
61     ExhaustAirOutletHumidityRatio = ExhaustAirInletHumidityRatio -
62       effEA * (ExhaustAirInletHumidityRatio - SupplyAirInletHumidityRatio);
63   }
64   else
65   {
66     SupplyAirOutletHumidityRatio = SupplyAirInletHumidityRatio;
67     ExhaustAirOutletHumidityRatio = ExhaustAirInletHumidityRatio;
68   }
69 }
70
71 /// <summary>停止させる</summary>
72 public void ShutOff()
73 {
74   Efficiency = 0;
75   SupplyAirOutletDrybulbTemperature = SupplyAirInletDrybulbTemperature;
76   ExhaustAirOutletDrybulbTemperature = ExhaustAirInletDrybulbTemperature;
77   SupplyAirOutletHumidityRatio = SupplyAirInletHumidityRatio;
78   ExhaustAirOutletHumidityRatio = ExhaustAirInletHumidityRatio;
79   SupplyAirFlowVolume = ExhaustAirFlowVolume = 0;
80   Electricity = 0;
81 }
```

プログラム 11.10 に出口空気乾球温度制御の計算処理を示す。まず 17 行で成行で計算を行い、熱流の向きが設定値とは逆で意味が無い場合には 24~30 行で機器を停止させる。例えば冷却運転時に成行での出口温度が設定値を下回る場合など、制御によって出口温度を設定値に調整できるならばには、式 11.24 で運転時間比率を求めて消費電力を調整する（38~46 行）。出口絶対湿度または出口比エンタルピーで制御する場合の処理は省略するが、手順は同じである。

プログラム 11.10　出口空気乾球温度制御の計算

Popolo.HVAC.HeatExchanger.RotaryRegenerator class

```
 1 /// <summary>給気出口温度を制御する</summary>
 2 /// <param name="supplyAirFlowVolume">給気風量[CMH]</param>
 3 /// <param name="exhaustAirFlowVolume">排気風量[CMH]</param>
 4 /// <param name="rotatingRate">ローター回転率[-]</param>
 5 /// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
 6 /// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
 7 /// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
 8 /// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
 9 /// <param name="outletSADrybulbTemperatureSP">給気出口乾球温度設定値[C]</param>
10 /// <returns>制御可能か否か</returns>
11 public bool ControlOutletTemperature
12   (double supplyAirFlowVolume, double exhaustAirFlowVolume, double rotatingRate,
```

```
13   double inletSADrybulbTemperature, double inletSAHumidityRatio,
14   double inletEADrybulbTemperature, double inletEAHumidityRatio, double outletSADrybulbTemperatureSP)
15 {
16   //成り行き計算実行
17   UpdateState
18     (supplyAirFlowVolume, exhaustAirFlowVolume, rotatingRate,
19     inletSADrybulbTemperature, inletSAHumidityRatio, inletEADrybulbTemperature, inletEAHumidityRatio);
20
21   //冷却運転か否か
22   bool cl = SupplyAirOutletDrybulbTemperature < SupplyAirInletDrybulbTemperature;
23
24   //熱交換が無駄な場合は停止
25   if ((cl && (SupplyAirInletDrybulbTemperature < outletSADrybulbTemperatureSP)) ||
26     (!cl && (outletSADrybulbTemperatureSP < SupplyAirInletDrybulbTemperature)))
27   {
28     ShutOff();
29     return false;
30   }
31
32   //処理可能な場合には消費電力と出口条件を修正
33   bool canSolve =
34     (cl && SupplyAirOutletDrybulbTemperature <= outletSADrybulbTemperatureSP
35     && outletSADrybulbTemperatureSP <= SupplyAirInletDrybulbTemperature) ||
36     (!cl && outletSADrybulbTemperatureSP < SupplyAirOutletDrybulbTemperature
37     && SupplyAirInletDrybulbTemperature <= outletSADrybulbTemperatureSP);
38   if (canSolve)
39   {
40     double rRun = (1d / Efficiency) * (outletSADrybulbTemperatureSP - SupplyAirInletDrybulbTemperature)
41       / (ExhaustAirInletDrybulbTemperature - SupplyAirInletDrybulbTemperature);
42     Electricity *= rRun;
43     SupplyAirOutletDrybulbTemperature = outletSADrybulbTemperatureSP;
44     SupplyAirOutletHumidityRatio = rRun * SupplyAirOutletHumidityRatio
45       + (1 - rRun) * SupplyAirInletHumidityRatio;
46   }
47   return canSolve;
48 }
```

【例題 11.5】

例題 11.1, 11.2の熱交換器について、外気風量と回転数の変化に対する熱交換効率の特性を計算せよ。

【解】

プログラムを 11.11 に示す。外気風量、回転数ともに 50%まで減らして熱交換効率を計算している。計算結果をグラフ化すると図 11.7 が得られる。回転数の上昇に伴って熱交換効率は上昇する。外気風量が減少すると外気の出入口温度差が大きく取れるため、熱交換効率は上昇する。ただし、これはあくまでも熱交換効率の上昇であって、交換熱量の絶対量が増加しているわけではないことには注意する必要がある。

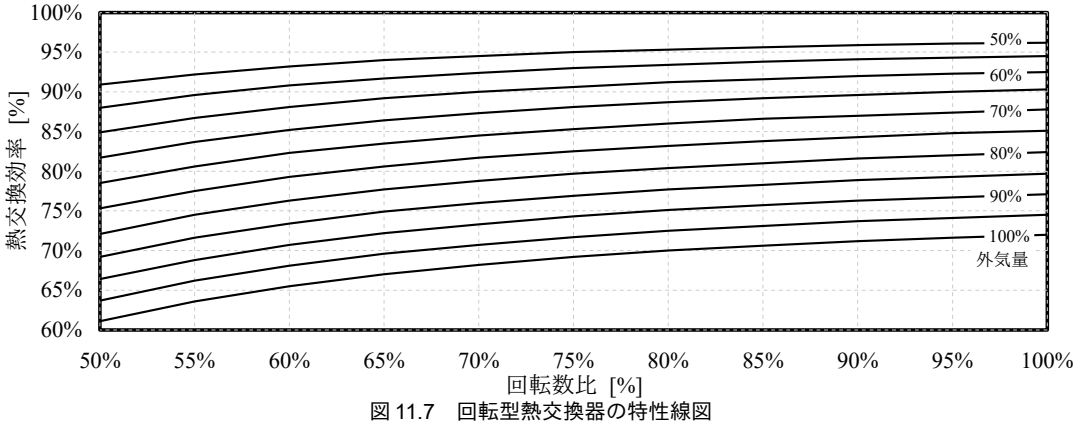


図 11.7　回転型熱交換器の特性線図

プログラム **11.11**　回転型熱交換器の特性の計算

```
1 private static void RotaryRegeneratorTest()
2 {
3   RotaryRegenerator rg =new RotaryRegenerator(0.72, 2.15, 0.4, 210, 10000, 8000, true, 35, 0.0195, 26, 0.0105);
4   using (StreamWriter sWriter = new StreamWriter
5     ("RotaryRegenerator.csv", false, Encoding.GetEncoding("Shift_JIS")))
6   {
7     //タイトル行
8     //流量変更
9     for (int i = 0; i <= 10; i++) sWriter.Write("," + (100 - 5 * i) + "%");
10    sWriter.WriteLine();
```

```
11
12     //回転数変更
13     for (int i = 0; i <= 10; i++)
14     {
15       sWriter.Write((100 - 5 * i) + "%, ");
16       double rotatingRate = 1 - 0.05 * i;
17       //流量変更
18       for (int j = 0; j <= 10; j++)
19       {
20         double oaf = 10000 * (1 - 0.05 * j);
21         rg.UpdateState(oaf, 8000, rotatingRate, 35, 0.0195, 26, 0.0105);
22         sWriter.Write(rg.Efficiency.ToString("F3") + ", ");
23       }
24       sWriter.WriteLine();
25     }
26   }
27 }
```

11.3.2　「静止型熱交換器クラス」の作成

1）クラスメソッドの定義

プログラム 11.12 に静止型熱交換器クラスの列挙型定義を示す。

プログラム 11.12　静止型熱交換器クラスの列挙型定義

Popolo.HVAC.HeatExchanger.AirToAirFlatPlateHeatExchanger class

```
1 /// <summary>空気の流れ</summary>
2 public enum AirFlow
3 {
4   /// <summary>向流</summary>
5   CounterFlow,
6   /// <summary>直交流</summary>
7   CrossFlow
8 }
```

プログラム 11.13 に顕熱貫流率 KA [kW/K]および潜熱貫流率 K_LA [kg/(kg/kg)]の計算処理を示す。貫流率に面積を乗じた値を直接求めている点に注意する。51~72 行は潜熱貫流率の計算処理であり、例題 11.4 の実装である。モデルの初期化を行う際に用いる。

プログラム 11.13　顕熱貫流率および潜熱貫流率の計算

Popolo.HVAC.HeatExchanger.AirToAirFlatPlateHeatExchanger class

```
 1 /// <summary>顕熱貫流率[kW/K]を計算する</summary>
 2 /// <param name="supplyAirMassFlowRate">給気質量流量[kg/s]</param>
 3 /// <param name="exhaustAirMassFlowRate">排気質量流量[kg/s]</param>
 4 /// <param name="supplyAirDrybulbTemperature">給気乾球温度[C]</param>
 5 /// <param name="supplyAirHumidityRatio">給気絶対湿度[kg/kg]</param>
 6 /// <param name="exhaustAirDrybulbTemperature">排気乾球温度[C]</param>
 7 /// <param name="exhaustAirHumitidyRatio">排気絶対湿度[kg/kg]</param>
 8 /// <param name="efficiency">顕熱熱交換効率[-]</param>
 9 /// <param name="flow">空気の流れのタイプ</param>
10 /// <returns>顕熱貫流率[kW/K]</returns>
11 public static double GetSensibleHeatTransferCoefficient
12   (double supplyAirMassFlowRate, double exhaustAirMassFlowRate,
13   double supplyAirDrybulbTemperature, double supplyAirHumidityRatio,
14   double exhaustAirDrybulbTemperature, double exhaustAirHumitidyRatio,
15   double efficiency, AirFlow flow)
16 {
17   //熱容量流量[kW/K]の計算
18   double cpSA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
19     (supplyAirDrybulbTemperature, supplyAirHumidityRatio, ATMOSPHERIC_PRESSURE);
20   double cpEA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
21     (exhaustAirDrybulbTemperature, exhaustAirHumitidyRatio, ATMOSPHERIC_PRESSURE);
22   double mcSA = supplyAirMassFlowRate * cpSA;
23   double mcEA = exhaustAirMassFlowRate * cpEA;
24   double mcMin = Math.Min(mcSA, mcEA);
25
26   //熱容量流量比[-]の計算
27   double capacityRatio = mcMin / Math.Max(mcSA, mcEA);
28
29   //熱通過有効度[-]を計算
30   double effectiveness;
31   if (mcSA < mcEA) effectiveness = efficiency;
32   else effectiveness = efficiency / capacityRatio;
33
34   //移動単位数を計算
35   HeatExchanger.FlowType fType;
```

```
36   if (flow == AirFlow.CounterFlow) fType = HeatExchanger.FlowType.CounterFlow;
37   else fType = HeatExchanger.FlowType.CrossFlow_BothFluidsUnmixed;
38   double ntu = HeatExchanger.GetNTU(effectiveness, capacityRatio, fType);
39
40   return ntu * mcMin;
41 }
42
43 /// <summary>潜熱貫流率[kg/(kg/kg)]を計算する</summary>
44 /// <param name="supplyAirMassFlowRate">給気質量流量[kg/s]</param>
45 /// <param name="exhaustAirMassFlowRate">排気質量流量[kg/s]</param>
46 /// <param name="supplyAirHumidityRatio">給気絶対湿度[kg/kg]</param>
47 /// <param name="exhaustAirHumitidyRatio">排気絶対湿度[kg/kg]</param>
48 /// <param name="efficiency">潜熱熱交換効率[-]</param>
49 /// <param name="flow">空気の流れのタイプ</param>
50 /// <returns>潜熱貫流率[kg/(kg/kg)]</returns>
51 public static double GetLatentHeatTransferCoefficient
52   (double supplyAirMassFlowRate, double exhaustAirMassFlowRate,
53   double supplyAirHumidityRatio, double exhaustAirHumitidyRatio,
54   double efficiency, AirFlow flow)
55 {
56   //質量流量比[-]の計算
57   double mcMin = Math.Min(supplyAirMassFlowRate, exhaustAirMassFlowRate);
58   double massFlowRatio = mcMin / Math.Max(supplyAirMassFlowRate, exhaustAirMassFlowRate);
59
60   //熱通過有効度[-]を計算
61   double effectiveness;
62   if (supplyAirMassFlowRate < exhaustAirMassFlowRate) effectiveness = efficiency;
63   else effectiveness = efficiency / massFlowRatio;
64
65   //移動単位数を計算
66   HeatExchanger.FlowType fType;
67   if (flow == AirFlow.CounterFlow) fType = HeatExchanger.FlowType.CounterFlow;
68   else fType = HeatExchanger.FlowType.CrossFlow_BothFluidsUnmixed;
69   double ntu = HeatExchanger.GetNTU(effectiveness, massFlowRatio, fType);
70
71   return ntu * mcMin;
72 }
```

プログラム 11.14 に熱通過有効度の計算処理を示す。プログラム 11.13 の逆算処理である。

プログラム 11.14　熱通過有効度の計算

Popolo.HVAC.HeatExchanger.AirToAirFlatPlateHeatExchanger class

```
 1 /// <summary>顕熱熱通過有効度[-]を計算する</summary>
 2 /// <param name="supplyAirMassFlowRate">給気質量流量[kg/s]</param>
 3 /// <param name="exhaustAirMassFlowRate">排気質量流量[kg/s]</param>
 4 /// <param name="supplyAirDrybulbTemperature">給気乾球温度[C]</param>
 5 /// <param name="supplyAirHumidityRatio">給気絶対湿度[kg/kg]</param>
 6 /// <param name="exhaustAirDrybulbTemperature">排気乾球温度[C]</param>
 7 /// <param name="exhaustAirHumitidyRatio">排気絶対湿度[kg/kg]</param>
 8 /// <param name="heatTransferCoefficient">顕熱貫流率[kW/K]</param>
 9 /// <param name="flow">空気の流れのタイプ</param>
10 /// <param name="effectiveness">顕熱熱通過有効度[-]</param>
11 /// <param name="mcMin">小さい側の熱容量流量[kg/s]</param>
12 /// <param name="capacityRatio">質量流量比[-]</param>
13 /// <param name="isMcMinSASide">給気側の熱容量流量が小さい場合に true</param>
14 public static void GetSensibleEffectiveness
15   (double supplyAirMassFlowRate, double exhaustAirMassFlowRate,
16   double supplyAirDrybulbTemperature, double supplyAirHumidityRatio,
17   double exhaustAirDrybulbTemperature, double exhaustAirHumitidyRatio,
18   double heatTransferCoefficient, AirFlow flow, out double effectiveness,
19   out double mcMin, out double capacityRatio, out bool isMcMinSASide)
20 {
21   //熱容量流量[kW/K]の計算
22   double cpSA = MoistAir.GetSpecificHeat(supplyAirHumidityRatio);
23   double cpEA = MoistAir.GetSpecificHeat(exhaustAirHumitidyRatio);
24   double mcSA = supplyAirMassFlowRate * cpSA;
25   double mcEA = exhaustAirMassFlowRate * cpEA;
26   mcMin = Math.Min(mcSA, mcEA);
27   isMcMinSASide = (mcSA == mcMin);
28
29   //熱容量流量比[-]の計算
30   capacityRatio = mcMin / Math.Max(mcSA, mcEA);
31
32   //熱通過有効度[-]の計算
33   double ntu = heatTransferCoefficient / mcMin;
34   HeatExchanger.FlowType fType;
35   if (flow == AirFlow.CounterFlow) fType = HeatExchanger.FlowType.CounterFlow;
36   else fType = HeatExchanger.FlowType.CrossFlow_BothFluidsUnmixed;
37   effectiveness = HeatExchanger.GetEffectiveness(ntu, capacityRatio, fType);
38 }
```

```
39
40 /// <summary>潜熱熱通過有効度[-]を計算する</summary>
41 /// <param name="supplyAirMassFlowRate">給気質量流量[kg/s]</param>
42 /// <param name="exhaustAirMassFlowRate">排気質量流量[kg/s]</param>
43 /// <param name="supplyAirHumidityRatio">給気絶対湿度[kg/kg]</param>
44 /// <param name="exhaustAirHumitidyRatio">排気絶対湿度[kg/kg]</param>
45 /// <param name="heatTransferCoefficient">潜熱貫流率[kg/(kg/kg)]</param>
46 /// <param name="flow">空気の流れのタイプ</param>
47 /// <param name="effectiveness">潜熱熱通過有効度[-]</param>
48 /// <param name="mMin">小さい側の質量流量[kg/s]</param>
49 /// <param name="capacityRatio">質量流量比[-]</param>
50 public static void GetLatentEffectiveness
51  (double supplyAirMassFlowRate, double exhaustAirMassFlowRate,
52  double supplyAirHumidityRatio, double exhaustAirHumitidyRatio,
53  double heatTransferCoefficient, AirFlow flow, out double effectiveness,
54  out double mMin, out double capacityRatio)
55 {
56   //質量流量比[-]の計算
57   mMin = Math.Min(supplyAirMassFlowRate, exhaustAirMassFlowRate);
58   capacityRatio = mMin / Math.Max(supplyAirMassFlowRate, exhaustAirMassFlowRate);
59
60   //熱通過有効度[-]の計算
61   double ntu = heatTransferCoefficient / mMin;
62   HeatExchanger.FlowType fType;
63   if (flow == AirFlow.CounterFlow) fType = HeatExchanger.FlowType.CounterFlow;
64   else fType = HeatExchanger.FlowType.CrossFlow_BothFluidsUnmixed;
65   effectiveness = HeatExchanger.GetEffectiveness(ntu, capacityRatio, fType);
66 }
```

2）静止型熱交換器クラスの作成

プログラム 11.15 に静止型熱交換器クラスのインスタンス変数およびプロパティを示す。回転型全熱交換器クラスとほぼ同様であるが、40~47 行に示すように熱交換効率は顕熱と潜熱の両方に対して定義する。また直交流と向流の可能性があるため、空気の流れのタイプを保持する。

プログラム 11.15　静止型熱交換器クラスのインスタンス変数およびプロパティ

Popolo.HVAC.HeatExchanger.AirToAirFlatPlateHeatExchanger class

```
 1 /// <summary>顕熱熱貫流率[kW/K]</summary>
 2 private double sensibleHeatTransferCoefficient;
 3
 4 /// <summary>潜熱熱貫流率[kg/(kg/kg)]</summary>
 5 private double latentHeatTransferCoefficient;
 6
 7 /// <summary>全熱交換器か否かを取得する</summary>
 8 public bool IsTotalHeatExchanger { get; private set; }
 9
10 /// <summary>給気風量[CMH]を取得する</summary>
11 public double SupplyAirFlowVolume { get; private set; }
12
13 /// <summary>排気風量[CMH]を取得する</summary>
14 public double ExhaustAirFlowVolume { get; private set; }
15
16 /// <summary>給気入口乾球温度[C]を取得する</summary>
17 public double SupplyAirInletDrybulbTemperature { get; private set; }
18
19 /// <summary>排気入口乾球温度[C]を取得する</summary>
20 public double ExhaustAirInletDrybulbTemperature { get; private set; }
21
22 /// <summary>給気出口乾球温度[C]を取得する</summary>
23 public double SupplyAirOutletDrybulbTemperature { get; private set; }
24
25 /// <summary>排気出口乾球温度[C]を取得する</summary>
26 public double ExhaustAirOutletDrybulbTemperature { get; private set; }
27
28 /// <summary>給気入口絶対湿度[kg/kg]を取得する</summary>
29 public double SupplyAirInletHumidityRatio { get; private set; }
30
31 /// <summary>排気入口絶対湿度[kg/kg]を取得する</summary>
32 public double ExhaustAirInletHumidityRatio { get; private set; }
33
34 /// <summary>給気出口絶対湿度[kg/kg]を取得する</summary>
35 public double SupplyAirOutletHumidityRatio { get; private set; }
36
37 /// <summary>排気出口絶対湿度[kg/kg]を取得する</summary>
38 public double ExhaustAirOutletHumidityRatio { get; private set; }
39
40 /// <summary>顕熱交換効率[-]を取得する</summary>
```

```
41 public double SensibleEfficiency { get; private set; }
42
43 /// <summary>潜熱交換効率[-]を取得する</summary>
44 public double LatentEfficiency { get; private set; }
45
46 /// <summary>空気の流れのタイプ</summary>
47 public AirFlow Flow { get; private set; }
```

プログラム 11.16 に静止型熱交換器クラスのコンストラクタを示す。プログラム 11.13 を用いて、31~41 行において定格仕様から熱貫流率を逆算する。

プログラム 11.16　静止型熱交換器クラスのコンストラクタ

Popolo.HVAC.HeatExchanger.AirToAirFlatPlateHeatExchanger class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="supplyAirFlowVolume">給気風量[CMH]</param>
 3 /// <param name="exhaustAirFlowVolume">排気風量[CMH]</param>
 4 /// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
 5 /// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
 6 /// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
 7 /// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
 8 /// <param name="sensibleEfficiency">顕熱交換効率[-]</param>
 9 /// <param name="latentEfficiency">潜熱交換効率[-]</param>
10 /// <param name="flow">空気の流れのタイプ</param>
11 /// <param name="isTotalHeatExchanger">全熱交換器の場合には true</param>
12 public AirToAirFlatPlateHeatExchanger
13   (double supplyAirFlowVolume, double exhaustAirFlowVolume,
14   double inletSADrybulbTemperature, double inletSAHumidityRatio,
15   double inletEADrybulbTemperature, double inletEAHumidityRatio,
16   double sensibleEfficiency, double latentEfficiency,
17   AirFlow flow, bool isTotalHeatExchanger)
18 {
19   //機器情報を保存
20   IsTotalHeatExchanger = isTotalHeatExchanger;
21   Flow = flow;
22
23   //空気の質量流量を計算
24   double svSA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
25     (inletSADrybulbTemperature, inletSAHumidityRatio, ATMOSPHERIC_PRESSURE);
26   double svEA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
27     (inletEADrybulbTemperature, inletEAHumidityRatio, ATMOSPHERIC_PRESSURE);
28   double mSA = supplyAirFlowVolume / (3600 * svSA);
29   double mEA = supplyAirFlowVolume / (3600 * svEA);
30
31   //顕熱貫流率を計算
32   sensibleHeatTransferCoefficient = GetSensibleHeatTransferCoefficient
33     (mSA, mEA, inletSADrybulbTemperature, inletSAHumidityRatio,
34     inletEADrybulbTemperature, inletEAHumidityRatio, sensibleEfficiency, flow);
35
36   //全熱交換器の場合には潜熱貫流率[kg/(kg/kg)]を計算
37   if (IsTotalHeatExchanger)
38   {
39     latentHeatTransferCoefficient = GetLatentHeatTransferCoefficient
40       (mSA, mEA, inletSAHumidityRatio, inletEAHumidityRatio, latentEfficiency, flow);
41   }
42
43   //定格条件で成り行き計算
44   UpdateState(supplyAirFlowVolume, exhaustAirFlowVolume, inletSADrybulbTemperature,
45     inletSAHumidityRatio, inletEADrybulbTemperature, inletEAHumidityRatio);
46 }
```

プログラム 11.17 に静止型熱交換器の出口状態の計算処理を示す。

プログラム 11.17　静止型熱交換器の出口状態の計算

Popolo.HVAC.HeatExchanger.AirToAirFlatPlateHeatExchanger class

```
 1 /// <summary>出口状態を更新する</summary>
 2 /// <param name="supplyAirFlowVolume">給気風量[CMH]</param>
 3 /// <param name="exhaustAirFlowVolume">排気風量[CMH]</param>
 4 /// <param name="inletSADrybulbTemperature">給気入口乾球温度[C]</param>
 5 /// <param name="inletSAHumidityRatio">給気入口絶対湿度[kg/kg]</param>
 6 /// <param name="inletEADrybulbTemperature">排気入口乾球温度[C]</param>
 7 /// <param name="inletEAHumidityRatio">排気入口絶対湿度[kg/kg]</param>
 8 public void UpdateState
 9   (double supplyAirFlowVolume, double exhaustAirFlowVolume,
10   double inletSADrybulbTemperature, double inletSAHumidityRatio,
11   double inletEADrybulbTemperature, double inletEAHumidityRatio)
12 {
13   //風量を保存
```

```
14  SupplyAirFlowVolume = supplyAirFlowVolume;
15  ExhaustAirFlowVolume = exhaustAirFlowVolume;
16
17  //入口空気状態を保存
18  SupplyAirInletDrybulbTemperature = inletSADrybulbTemperature;
19  SupplyAirInletHumidityRatio = inletSAHumidityRatio;
20  ExhaustAirInletDrybulbTemperature = inletEADrybulbTemperature;
21  ExhaustAirInletHumidityRatio = inletEAHumidityRatio;
22
23  //風量が 0 の場合
24  if (supplyAirFlowVolume <= 0 || exhaustAirFlowVolume <= 0)
25  {
26    SensibleEfficiency = 0;
27    SupplyAirOutletDrybulbTemperature = SupplyAirInletDrybulbTemperature;
28    ExhaustAirOutletDrybulbTemperature = ExhaustAirInletDrybulbTemperature;
29    SupplyAirOutletHumidityRatio = SupplyAirInletHumidityRatio;
30    ExhaustAirOutletHumidityRatio = ExhaustAirInletHumidityRatio;
31    return;
32  }
33
34  //空気の質量流量を計算
35  double svSA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
36    (inletSADrybulbTemperature, inletSAHumidityRatio, ATMOSPHERIC_PRESSURE);
37  double svEA = MoistAir.GetSpecificVolumeFromDryBulbTemperatureAndHumidityRatio
38    (inletEADrybulbTemperature, inletEAHumidityRatio, ATMOSPHERIC_PRESSURE);
39  double mSA = supplyAirFlowVolume / (3600 * svSA);
40  double mEA = exhaustAirFlowVolume / (3600 * svEA);
41
42  //熱通過有効度[-]を計算
43  double effectiveness, mcMin, capacityRate;
44  bool isMcMinSA;
45  GetSensibleEffectiveness
46    (mSA, mEA, inletSADrybulbTemperature, inletSAHumidityRatio,
47    inletEADrybulbTemperature, inletEAHumidityRatio,
48    sensibleHeatTransferCoefficient, Flow,
49    out effectiveness, out mcMin, out capacityRate, out isMcMinSA);
50
51  //熱交換効率[-]を計算
52  double eff2;
53  if (isMcMinSA)
54  {
55    SensibleEfficiency = effectiveness;
56    eff2 = effectiveness * capacityRate;
57  }
58  else
59  {
60    SensibleEfficiency = effectiveness * capacityRate;
61    eff2 = effectiveness;
62  }
63
64  //出口空気状態の計算
65  SupplyAirOutletDrybulbTemperature =
66    SupplyAirInletDrybulbTemperature - SensibleEfficiency *
67    (SupplyAirInletDrybulbTemperature - ExhaustAirInletDrybulbTemperature);
68  ExhaustAirOutletDrybulbTemperature = ExhaustAirInletDrybulbTemperature -
69    eff2 * (ExhaustAirInletDrybulbTemperature - SupplyAirInletDrybulbTemperature);
70
71  //水分交換
72  if (IsTotalHeatExchanger)
73  {
74    //熱通過有効度[-]を計算
75    GetLatentEffectiveness
76      (mSA, mEA, inletSAHumidityRatio, inletEAHumidityRatio, latentHeatTransferCoefficient, Flow,
77      out effectiveness, out mcMin, out capacityRate);
78
79    //熱交換効率[-]を計算
80    if (mcMin == mSA)
81    {
82      LatentEfficiency = effectiveness;
83      eff2 = effectiveness * capacityRate;
84    }
85    else
86    {
87      LatentEfficiency = effectiveness * capacityRate;
88      eff2 = effectiveness;
89    }
90
91    SupplyAirOutletHumidityRatio = SupplyAirInletHumidityRatio -
92      LatentEfficiency * (SupplyAirInletHumidityRatio - ExhaustAirInletHumidityRatio);
93    ExhaustAirOutletHumidityRatio = ExhaustAirInletHumidityRatio -
```

```
 94       eff2 * (ExhaustAirInletHumidityRatio - SupplyAirInletHumidityRatio);
 95   }
 96   else
 97   {
 98     SupplyAirOutletHumidityRatio = SupplyAirInletHumidityRatio;
 99     ExhaustAirOutletHumidityRatio = ExhaustAirInletHumidityRatio;
100   }
101 }
```

プログラム 11.18 に静止型熱交換器の全熱交換効率の計算処理を示す。プログラム 11.17 で計算された出口温湿度条件にもとづき、出入口の比エンタルピーを計算して全熱交換効率を求める。

プログラム 11.18　静止型熱交換器の全熱交換効率の計算

Popolo.HVAC.HeatExchanger.AirToAirFlatPlateHeatExchanger class

```
 1 /// <summary>全熱交換効率[-]を計算する</summary>
 2 /// <returns>全熱交換効率[-]</returns>
 3 public double GetTotalEfficiency()
 4 {
 5   //空気の出入口エンタルピーの計算
 6   double hSAi = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
 7     (SupplyAirInletDrybulbTemperature, SupplyAirInletHumidityRatio);
 8   double hSAo = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
 9     (SupplyAirOutletDrybulbTemperature, SupplyAirOutletHumidityRatio);
10   double hEAi = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
11     (ExhaustAirInletDrybulbTemperature, ExhaustAirInletHumidityRatio);
12
13   return (hSAi - hSAo) / (hSAi - hEAi);
14 }
```

【例題 11.6】

例題 11.3 と例題 11.4 の熱交換器について、外気風量と排気風量を変化させた場合の潜熱交換効率の特性線図を作成せよ。

【解】

プログラムを 11.19 に示す。外気風量、排気風量ともに 80~150CMH の範囲で変化させて潜熱交換効率を計算している。計算結果をグラフ化すると図 11.8 が得られる。グラフの点線表記は風量を等しく変化させた場合の特性であり、機器カタログにはこの特性が表示されることが多い。排気風量に比較して外気風量が相対的に小さくなるにつれて潜熱交換効率が上昇する傾向となる。

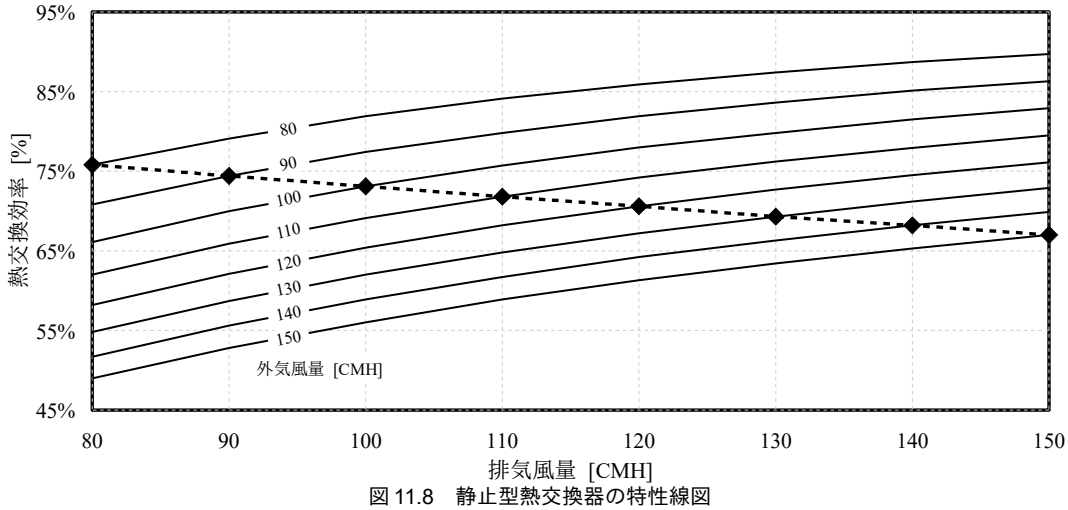


図 11.8　静止型熱交換器の特性線図

プログラム 11.19　静止型熱交換器の風量特性の計算

```
 1 private static void AirToAirFlatPlateHeatExchangerTest()
 2 {
 3   //初期化処理
 4   AirToAirFlatPlateHeatExchanger aHex = new AirToAirFlatPlateHeatExchanger
 5     (150, 150, 34.5, 0.0263, 26.5, 0.014, 0.74, 0.67, AirToAirFlatPlateHeatExchanger.AirFlow.CrossFlow, true);
 6
 7   using (StreamWriter sWriter = new StreamWriter
 8     ("AToAFlatPlateHeatExchanger.csv", false, Encoding.GetEncoding("Shift_JIS")))
 9   {
10     //タイトル行
11     for (int i = 0; i <= 7; i++) sWriter.Write("," + (150 - (10 * i)));
```

```
12      sWriter.WriteLine();
13
14      for (int i = 0; i <= 7; i++)
15      {
16        //排気風量を変更 (50%～100%)
17        double afEA = (150 - (10 * i));
18        sWriter.Write(afEA.ToString());
19        for (int j = 0; j <= 7; j++)
20        {
21          //給気風量を変更 (50%～100%)
22          double afSA = (150 - (10 * j));
23          aHex.UpdateState(afSA, afEA, 34.5, 0.0263, 26.5, 0.014);
24          sWriter.Write(", " + aHex.LatentEfficiency.ToString("F3"));
25        }
26        sWriter.WriteLine();
27      }
28    }
29 }
```

【第 11 章 記号表】

A_f	：流路断面積 **[m²]**	NTU	：移動単位数 **[-]**
A_k	：マトリクス隔壁の見付面積 **[m²]**	NTU_0	：修正移動単位数 **[-]**
A_S	：伝熱面積 **[m²]**	Q	：熱交換量 **[kW]**
C_λ	：流路方向の隔壁内伝熱を評価するための補正係数	R_{mc}	：熱容量流量比 **[-]**
		R_r	：熱容量回転量比 **[-]**
c_p	：定圧比熱 **[kJ/(kg·K)]**	R_{run}	：間欠運転時間比率 **[-]**
d_e	：等価直径 **[m]**	T	：乾球温度 **[K]**
E	：モータ消費電力 **[kW]**	t	：乾球温度 **[°C]**
E_E	：モータ定格消費電力 **[kW]**	v	：風速 **[m/s]**
K_L	：潜熱貫流率 **[kg/(m²·kg/kg)]**	α	：対流熱伝達率 **[W/(m²·K)]**
K_S	：顕熱貫流率 **[kW/(m²·K)]**	ε	：熱通過有効度 **[-]**
L	：ローターの流路方向長さ **[m]**	ε_∞	：伝熱面積無限の場合の熱通過有効度 **[-]**
L_m	：周長 **[m]**	λ	：流れ方向伝導係数 **[-]**
m	：質量流量 **[kg/s]**	Ω	：回転数 **[rps]**

添字：

EA	：排気側	min	：小さい側
i	：入口	o	：出口
L	：潜熱	r	：ローター
m	：補正値	S	：顕熱
max	：大きい側	SA	：給気側

【第 11 章 参考文献】

11.1） Kays, W. M. and London, A.L.: Compact Heat Exchangers, 2nd ed., McGraw-Hill, New York, 1964.

11.2） Ramesh K. Shah, Dusan P. Sekulic: Fundamentals of Heat Exchanger Design, Wiley, 2002

11.3） D. C. Swanepoel, D. G. Kroger: Rotary Regenerator Design Theory and Optimisation, R&D Journal, 1996, 12(3), pp.90-97

11.4） Bahnke G. D. & C. P. Howard: The Effect of Longitudinal Heat Conduction on Periodic-Flow Heat Exchanger Performance, Trans. ASME, Vol.86A, 1964, pp105-120

11.5） Lambertson, T. J.: Performance Factors of a Periodic-Flow Heat Exchanger, Trans. ASME, Vol.80, p.586, 1958

11.6） Razelos P.: An Analytical Solution to the Electric Analog Simulation of the Regenerative Heat Exchanger with Time-Varying Fluid inlet Temperatures, Warme und Stoffubertrag, 12, 1979, pp.59-71

11.7） 田中宏史, 粟田好治, 隈利実: 夏期条件での回転式全熱交換器の性能解析および実験, 空気調和・衛生工学会論文集 (59), pp.41-47, 1995.10.25, 社団法人空気調和・衛生工学会

11.8） 公共建築工事標準仕様書（機械設備工事編） 第 9 節 全熱交換器: 国土交通省 大臣官房 官庁営繕部 設備・環境課

11.9） 日本工業規格 JIS B8628 2003 全熱交換器

11.10） Schack, A.: Der Industrielle Warmeubergang, 1929, Stahleisen m. b. H., Dusseldorf

11.11） Abdulmajeed S. Al-Ghamdi: Analysis of Air-to-air Rotary Energy Wheels, Ohio University, Ph.D Thesis, 2006

11.12） 高橋建造, 草川英昭, 田中修: 繊維性多孔質材料の気体透過特性, 化学工学会 化学工学論文集 5(4), pp.391^396, 1979

第12章　冷却塔
(Cooling Tower)

12.1　概要

冷却塔は、水と空気との間で熱交換を行う熱交換器である。建物で発生した熱を最終的に大気に放出する役割を担う。

代表的な熱源である吸収式冷凍機と圧縮式冷凍機は、通常は冷却水を用いて放熱を行い、その効率は冷却水の温度に影響を受ける。特に近年ではインバータを用いた熱源機が開発されており、冷却水温度最適化による熱源効率向上の効果は従来よりも大きくなってきている。

冷却塔は水分蒸発を利用して放熱を行う機器であるため、その能力は冷却水の流量と温度だけではなく、外気の湿球温度に大きく影響を受ける。従って、冷却塔の最適運用のためには、これらの条件が変化した場合に冷却水温度がどのように変動するのかを予測できる必要がある。そこで本書では湿り空気状態にもとづいて冷却塔能力を予想する物理モデルについて解説する。また、経済性に大きな影響を与える水の消費量に関する計算法についても説明する。

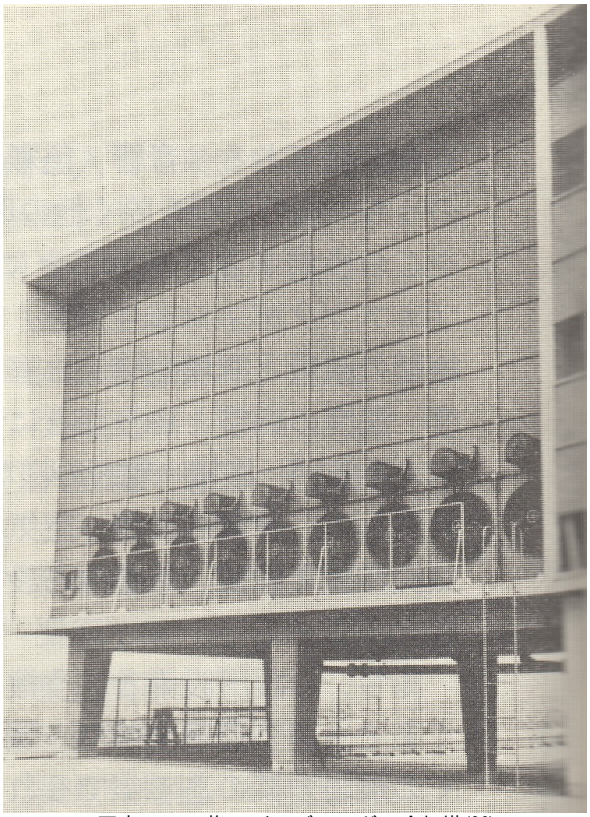


写真 12.1　梅田ビルディングの冷却塔[12.8)]

竹中工務店による昭和 30 年竣工のオフィスビルである。
後に地盤沈下が社会問題になるまで、冷却水の放熱先は井水が主流であったが、
本建物はこの時期に既にすべての放熱を冷却塔で賄うシステムとしていた

12.2　理論

12.2.1　冷却塔の分類

空調用途に使われる冷却塔としては、1. 水と空気の接触が直接的か間接的か(密閉式と開放式)、2. 水と空気の流れが対向する方向か直交する方向か(対向流と直交流)、という観点から分類することが出来る。

1）開放式と密閉式

開放式冷却塔は冷却の対象である水を空気に直接に接触させて熱交換を行う方式であり、多くの場合は、この種類の冷却塔が使われる。一方、例えば海岸に近くて大気に塩分が含まれる場合など、冷却水の汚染が懸念される場合には、水と空気を間接的に接触させて熱交換を行う密閉式冷却塔が使われる。

密閉式冷却塔は冷却水を配管内で通水させながら熱交換を行うため、開放式に比べて配管内での圧力損失が生じる。一方で開放式は、冷却塔上部で冷却水を開放して冷却培の下部まで自然流下させるため、冷却塔の高さ分の実揚程が必要となる。両者の大小関係は冷却塔の設計に依存するため一概には言えないが、同一冷却能力の場合には後者の方がやや小さく設計される場合が多い。

開放式は冷却水を直接に空気に接触させるため、冷却水の蒸発が生じる。従って、開放式では顕熱だけではなく、潜熱の交換も行われるため、冷却水を外気温度以下に冷却することが可能である。一方、密閉式は、冷却水が直接に空気に接触しないため、冷却水の蒸発は生じない。そこで密閉式においては、開放式と同じ様に潜熱を利用した熱交換を実現させるために、冷却水とは別に蒸発用の水を循環させて散水する仕組を用意している。

以上に述べた通り、冷却塔自体の必要場程が大きいことと、散水ポンプの動力が必要であることにより、密閉式のエネルギー消費量は開放式に比較して大きくなる傾向にある。

2）対向流と直交流

冷却水を冷却塔の上部から滴下することに関しては、対向流の冷却塔も直交流の冷却塔も同じである。両者の違いは空気の風向にあり、対向流の場合には、冷却塔の下部から空気を吸込むことで、冷却水とは向かい合う方向に空気の流れを作り出す。一方、直交流の場合には、冷却塔の側面から空気を吸い込むため、冷却水と空気の流れの向きが直交する。熱交換器の特性から明らかなように、風量および水量が同一ならば、対向流の冷却塔の方が大きな熱交換能力を持つため、効率が高い。一方で、所要スペースに関しては直交流の方が小さいため、容量の大きな場合には直交流の冷却塔を採用することが多い。

12.2.2　熱交換の基礎式

冷却塔内の微小部分での熱交換量 dq [kW]は、式 12.1 で表現することができる。第 2 辺は水からみた熱交換量であり、水の微小温度変化量 dt [K]に、水の質量流量 m_w [kg/s]と定圧比熱 c_{pw} [kJ/(kg·K)]を乗じることで熱交換量を算出している。第 3 辺は空気からみた熱交換量であり、空気の微小エンタルピー変化量 dh [kJ/kg]に空気の質量流量 m_a [kg/s]を乗じることで熱交換量を算出している。

$$dq = m_w c_{pw} dt = m_a dh = K\left(h_{sw} - h_a\right) a\, dV \tag{12.1}$$

第 4 辺の導出方法はやや複雑である。図 12.1 に水と空気との間の熱交換を示す。水の温度が t_w [K]の時、水に接触する空気の温度も t_w [K]となり、そのエンタルピーと絶対湿度はそれぞれ温度 t_w [K]に

おける飽和エンタルピー h_{sw} [kJ/kg]と飽和絶対湿度 x_{sw} [kg/kg]となる。水から空気への熱移動としては、水の蒸発による潜熱の移動と、空気との間の対流熱交換による顕熱の移動がある。

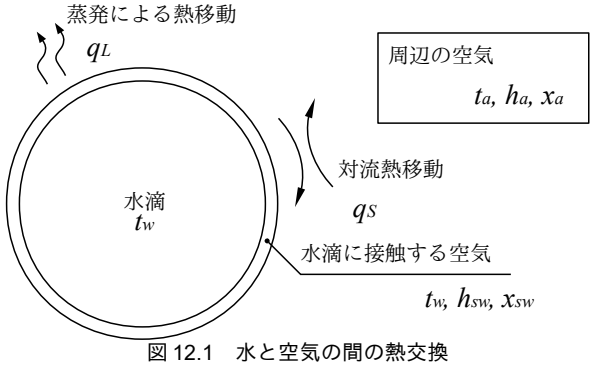


図 12.1　水と空気の間の熱交換

水の蒸発による潜熱移動量 dq_L [kW]は式 12.2 で表現することができる。ここで、γ_0は水の蒸発潜熱(=2,501 kJ/kg)、K_x [kg/(s·m²·(kg/kg))]は物質伝達率、a [m²/m³]は体積あたりの水の表面積、dV [m³]は微小体積である。

$$dq_L=\gamma_0 K_x(x_{sw}-x_a)a\,dV \tag{12.2}$$

空気との間の対流熱交換量 dq_S [kW]は式 12.3 で表現することができる。ここで、K_G [kW/(m²·K)]は対流熱伝達率である。

$$dq_S=K_G(t_w-t_a)a\,dV \tag{12.3}$$

水から空気への熱移動量 dq [kW]は、顕熱移動と潜熱移動を合計して式 12.4 で表現できる。K_xとK_Gは式 12.5 のルイスの関係式で表現できるが、強制対流の場合はルイス数 Le=1 であるため、結局、K_G=K_x c_{pma}となる。これを式 12.4 に代入すると式 12.6 が得られ、K_x=Kと置けば式 12.1 第 4 辺となる。ここで、c_{pma} [kJ/(kg·K)]は湿り空気の比熱である。なお、式 8.3 および式 8.4 から類推できるが、式 12.4 を積分すると対数平均温度差と同様の式形状を持つ対数平均エンタルピー差によって熱交換器内の平均的なエンタルピー差を表現することができる。

$$dq=dq_L+dq_S=\gamma_0 K_x(x_{sw}-x_a)a\,dV+K_G(t_w-t_a)a\,dV \tag{12.4}$$

$$K_G=Le\cdot K_x c_{pma} \tag{12.5}$$

$$dq=K_x\{\gamma_0(x_{sw}-x_a)+c_{pm}(t_w-t_a)\}a\,dV=K_x(h_{sw}-h_a)a\,dV \tag{12.6}$$

12.2.3　基礎式の解き方

式 12.1 の第 2 辺と第 4 辺を移行して変形すると式 12.7 が得られる。

$$\int\frac{c_{pw}}{(h_{sw}-h_a)}dt=\frac{KaV}{m_w} \tag{12.7}$$

ここで、水温の変化量を dt、水温に相当する飽和空気のエンタルピーの変化量を dh として c_s=dh/dtとおくと、式 12.7 は式 12.8 に変形でき[†1]、これを NTU_wと呼ぶことにする。

$$\int\frac{1}{(h_{sw}-h_a)}dh=\frac{KaV}{m_w(c_{pw}/c_s)}=NTU_w \tag{12.8}$$

同様に、式 12.1 の第 3 辺と第 4 辺を移行して変形し、式 12.9 を得る。

$$\int\frac{1}{(h_{sw}-h_a)}dh=\frac{KaV}{m_a}=NTU_a \tag{12.9}$$

式 12.8 と式 12.9 の $m_w(c_{pw}/c_s)$、m_a、NTU_w、NTU_aを用いると、冷却塔の熱交換量 Q [kW]は式 12.10~

†1　c_sはtの関数であるが、飽和空気のエンタルピーと温度の関係はほぼ線形のため、定数として扱う。

式 12.14 で表現できる。

$$Q=\varepsilon\, m_{min}(h_{swi}-h_{ai}) \tag{12.10}$$

$$\varepsilon=f(NTU_{min}, m_{min}/m_{max}) \tag{12.11}$$

$$m_{min}=Min(m_w(c_{pw}/c_s), m_a) \tag{12.12}$$

$$m_{max}=Max(m_w(c_{pw}/c_s), m_a) \tag{12.13}$$

$$NTU_{min}=\begin{cases} NTU_w & (m_w(c_{pw}/c_s)<m_a) \\ NTU_a & (m_w(c_{pw}/c_s)\geq m_a) \end{cases} \tag{12.14}$$

問題は、*NTU* をどのように求めるかであるが、式 12.15 に示す通り、NTU_w に関しては、実験結果により $(m_a/m_w)^n$ に比例することが知られており [12.1)]、予め定格性能値から特性係数 c を推定しておけば計算できる（例題 12.1 参照）。なお、n は-0.35~-1.1 程度であり、平均的には-0.6 とされている [12.1)]。

$$NTU_w=c\left(\frac{m_w}{m_a}\right)^n \tag{12.15}$$

ε は第 8 章に記した熱通過有効度と呼ばれる無次元の係数であり、無限の面積を持つ向流型熱交換器の熱交換量を基準とした場合の、検討対象である熱交換器の熱交換量の比率を示している。式 12.11 の具体的な計算方法は後述する。

熱交換量 Q [kW]が算出できれば、以下の方法により水と空気の出口状態を計算することができる。まず、式 12.1 の第 2 辺と第 3 辺を積分する。水の質量流量 m_w [kg/s]、水の比熱 c_{pw} [kJ/(kg·K)]、空気の質量流量 m_a [kg/s]はそれぞれ一定値であるため、各々、式 12.16 および式 12.17 となる[†1)]。熱交換量 Q [kW]は既知であるため、式 12.16 からは水の出口状態を、式 12.17 からは空気の出口状態をそれぞれ求めることができる。

$$Q=m_w\, c_{pw}(t_{wi}-t_{wo}) \tag{12.16}$$

$$Q=m_a(h_{ao}-h_{ai}) \tag{12.17}$$

【例題 12.1】

下記の定格性能を持つ冷却塔について、質量流量比 m_w/m_a から NTU_w を計算するための係数を求めよ。ただし冷却塔は直交流タイプとする。

冷却水入口温度 : 37.0 °C	入口空気湿球温度 : 27.0 °CWB
冷却水出口温度 : 32.0 °C	風量 : 1,783 m^3/min
冷却水流量 : 2,600L/min	能力 : 907 kW

【解】

$NTU_w = c\,(m_w/m_a)^{-0.6}$ であるため、定格条件における質量流量比 m_w/m_a と NTU_w を各々求めることで c を推定する。まず、冷却水流量と風量を質量流量に変換する。

2,600 L/min×(1/60) = 43.3 kg/s,　　1,783 m^3/min×(1/60)×1.2kg/m^3=35.7 kg/s

従って、質量流量比 m_w/m_a は 43.3÷35.7 = 1.22 となる。冷却水出入口水温に相当する空気の飽和エンタルピーは各々以下の通りである。

37.0 °C の空気の飽和エンタルピー : 142.8 kJ/kg

32.0 °C の空気の飽和エンタルピー : 110.6 kJ/kg

また、入口空気の飽和エンタルピーは 85.0 kJ/kg である。

冷却水の出入口温度差は 37.0 – 32.0 = 5.0 °C であり、飽和エンタルピー差は 142.8 – 110.6 = 32.2 kJ/kg であるため、平均的な比熱 c_s は 32.2 ÷ 5.0 = 6.43 kJ/(kg·K)となる。従って、m_{min}= Min(43.3×4.186÷6.43, 35.7) = Min(28.2, 35.7) = 28.2 kg/s である。同様に m_{max}=35.7 と求められ、m_{min} / m_{max} = 079 となる。また、式 12.10 に

†1　冷却塔内を通過する水の一部は蒸発するため、実際には m_w は一定ではないが、水量に対して数%と微量であるため誤差とみなしている。

より、907 = ε×28.2 × (142.8 – 85.0)であるから、ε=0.56 となる。

m_{min} / m_{max}と ε が与えられれば、直交流の熱交換器の熱通過有効度の線図（第 8 章の図 8.4）から NTU_{min} を求めることができ、NTU_{min} = 1.31 とわかる。本問題では、m_w (c_{pw}/c_s) < m_a であるため、NTU_w=NTU_{min}=1.31 となる。

$NTU_w = c\ (m_w/m_a)^{0.6}$ であり、上記で求めた NTU_w=1.31 と m_w/m_a=1.22 を代入して c について解くと、c=1.16 となる。

12.2.4　水の消費量

冷却塔における水の消費量 M [kg/s]（=補給水量）は、蒸発による水の消費量 E [kg/s]、飛散による水の消費 W [kg/s]、ブローによる水の消費 B [kg/s]を合計した値である。

$$M = E + W + B \tag{12.18}$$

この内、蒸発による水の消費量 E [kg/s]は、熱交換の過程において空気側に蒸発した水の総量であるため、空気の出入口絶対湿度差と風量が与えられれば式 12.19 によって計算できる。

$$E = m_a(x_{ao} - x_{ai}) \tag{12.19}$$

ブロー水量とは、冷却水の水質を保つために強制的に排水する水量である。冷却水内部の不純物の濃度を C_R [kg/kg]、補給水の不純物の濃度を C_M [kg/kg]とおくと、冷却水に内部に供給される不純物と外部に排出される不純物の収支は式 12.20 で表現できる。蒸発による水消費の場合には不純物が外部に排出されず、このために冷却水の不純物濃度は補給水の不純物濃度よりも高い値になる。

$$C_M M = C_R(W + B) \tag{12.20}$$

過度な不純物濃度は冷却塔や冷凍機、冷却水配管など、冷却水に接触する機器の腐食等に繋がるため、強制的な排水を行うことで不純物濃度を低下させる必要がある。このような水質管理を行うための尺度として、冷却水の不純物濃度が補給水に対して何倍になっているかという濃縮倍数 N を導入する。定義により、濃縮倍数 N は式 12.21 で表現することができる。

$$N = C_R / C_M \tag{12.21}$$

式 12.18、式 12.20、式 12.21 を整理すると、式 12.22 が得られる。濃縮倍数を大きく設定すればブロー水量が小さく、また、濃縮倍数を小さく設定すればブロー水量が大きくなる関係性がわかる。

$$B = \frac{E}{N-1} - W \tag{12.22}$$

飛散水量 W [kg/s]は循環水量の 0.05%~0.2%程度とされている。また、通過風量と線形の関係にあると仮定し、式 12.23 で計算する。ただし $m_{a,rated}$ [kg/s]は定格点における空気の質量流量、d_r [-]は非散水量の循環水量に対する比率である。

$$W = m_w \cdot dr\left(\frac{m_a}{m_{a,\,rated}}\right) \tag{12.23}$$

飛散量が非常に大きい場合には、形式的には式 12.24 で B がマイナスの値をとることになるが、実運転上は濃縮倍数の低下が生じるだけなので、正確にはブロー水量は式 12.24 で表現される。

$$B = max\left(0, \left(\frac{E}{N-1}\right) - W\right) \tag{12.24}$$

【例題 12.2】

例題 12.1 で示した冷却塔について、濃縮倍数を 4 倍と 8 倍に設定した際の定格運転時の補給水量を求めよ。ただし、外気条件は 35 °CDB、27 °CWB、54 %、19.3×10^{-3} kg/kg、84.7kJ/kg とする。また、飛散量は循環水量に対して 0.2 %とする。

【解】

式 12.17 を用いると、出口空気のエンタルピーは、

$907 = (h_{ao} - 85.0) \times 35.7 \quad \therefore \quad h_{ao} = 110.5$ kJ/kg

と求められる。出口空気は飽和しているとみなせるため、絶対湿度は 30.6×10^{-3} kg/kg である。式 12.19 により、蒸発量 E [kg/s]は

$E = (30.6\text{-}19.3) \times 10^{-3} \times 35.7 = 0.4$ kg/s

となる。

飛散量 M [kg/s]は循環水量の 0.2%であるから、

$M = 43.3$ kg/s×0.002=0.087 kg/s

となる。

ブロー水量 B [kg/s]は N=4 のとき、B=0.4 / (4-1) – 0.087 = 0.047kg/s となる。

従って、蒸発、飛散、ブローを合計すると、0.4 + 0.087 + 0.047 = 0.534 kg/s となり、これは循環水量の約 1.2 %に相当する。

濃縮倍数 N=8 倍のとき、ブロー水量は、

B=0.4 / (8-1) – 0.087 = – 0.03 kg/s

となる。マイナスの値となっているが、これは、飛散に対応するための補給によって十分に希釈効果が得られるために、濃縮倍数が 8 倍以下に押えられることを意味する。従って、ブロー水量は 0 であり、補給水量は蒸発量 E と飛散量 M を加算した 0.487kg/s となる。

なお、式 12.20 と式 12.21 においてブロー水量 B [kg/s]を 0 とし、濃縮倍数 N について解くと $N=M / W$ であるため、成行状態での濃縮倍数は N=0.4 / 0.087 = 4.6 倍となる。

12.2.5　冷却塔ファンの消費電力

冷却塔ファンの制御方法としては発停とインバータ制御がある。風量が m_a [kg/s]であった場合の冷却塔ファンの消費電力 P_{ct} [kW]は各々、式 12.25 と式 12.26 で計算できる。ただし、$P_{ct,rated}$ [kW]は定格点におけるファンの消費電力である。

$$P_{ct} = P_{ct,rated} \frac{m_a}{m_{a,rated}} \tag{12.25}$$

$$P_{ct} = P_{ct,rated} \left(\frac{m_a}{m_{a,rated}} \right)^3 \tag{12.26}$$

12.3　計算法

12.3.1　冷却塔パラメータの推定

列挙型の定義をプログラム 12.1 に示す。

プログラム 12.1　列挙型の定義

Popolo.HVAC.HeatExchanger.CoolingTower class

```
1 /// <summary>流れのタイプ</summary>
2 public enum AirFlowDirection
3 {
4   /// <summary>向流型</summary>
5   CounterFlow,
6   /// <summary>直交流型</summary>
7   CrossFlow
8 }
```

様々な入力条件に対する冷却塔の応答を知るためには、まず、冷却塔パラメータを推定する必要がある。プログラム 12.2 は定格条件および能力から冷却塔の特性係数 c_s を算出する処理であり、例題 12.1 を実装したものである。空気の流れが直交流か向流かによって熱通過有効度 ε の値が異なるため、先に定義した列挙型を用いて 36~38 行において条件分岐を行う。43 行は ε から NTU を求める計算処理であり、詳細は第 8 章で記した。NTU_w を式 12.15 に代入し、46 行で特性係数 c を計算する。

プログラム **12.2**　冷却塔の特性係数 c_s の計算

Popolo.HVAC.HeatExchanger.CoolingTower class

```
 1 /// <summary>冷却塔の特性係数c[-]を求める</summary>
 2 /// <param name="inletWaterTemperature">冷却水入口温度[C]</param>
 3 /// <param name="outletWaterTemperature">冷却水出口温度[C]</param>
 4 /// <param name="wetbulbTemperature">空気入口湿球温度[C]</param>
 5 /// <param name="waterFlowRate">冷却水の質量流量[kg/s]</param>
 6 /// <param name="airFlowRate">空気の質量流量[kg/s]</param>
 7 /// <param name="airFlowType">流れのタイプ</param>
 8 /// <returns>冷却塔の特性係数c[-]</returns>
 9 public static double GetCoolingTowerCoefficient
10   (double inletWaterTemperature, double outletWaterTemperature, double wetbulbTemperature,
11   double waterFlowRate, double airFlowRate, AirFlowDirection airFlowType)
12 {
13   //熱交換量[kW]を計算
14   double capacity = (inletWaterTemperature - outletWaterTemperature) * waterFlowRate * WATER_SPECIFIC_HEAT;
15
16   //冷却水温度に相当する空気の飽和エンタルピーを計算
17   double hswi = MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
18     (inletWaterTemperature, 100, ATMOSPHERIC_PRESSURE);
19   double hswo = MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
20     (outletWaterTemperature, 100, ATMOSPHERIC_PRESSURE);
21   //入口空気の飽和エンタルピーを計算
22   double hai = MoistAir.GetEnthalpyFromWetBulbTemperatureAndRelativeHumidity
23     (wetbulbTemperature, 100, ATMOSPHERIC_PRESSURE);
24
25   //平均的な比熱で熱容量流量比を計算
26   double cs = (hswi - hswo) / (inletWaterTemperature - outletWaterTemperature);
27   double mwc = waterFlowRate * WATER_SPECIFIC_HEAT / cs;
28   double mMin = Math.Min(mwc, airFlowRate);
29   double mMax = Math.Max(mwc, airFlowRate);
30   double rmc = mMin / mMax;
31
32   //熱通過有効度とNTUの計算
33   double epsilon = capacity / mMin / (hswi - hai);
34   double ntuMin;
35   HeatExchange.FlowType aft;
36   if (airFlowType == AirFlowDirection.CrossFlow) aft = HeatExchange.FlowType.CrossFlow_BothFluidsUnmixed;
37   else aft = HeatExchange.FlowType.CounterFlow;
38   ntuMin = HeatExchange.GetNTU(epsilon, rmc, aft);
39
40   //NTUwの計算
41   double ntuW;
42   if (mwc == mMin) ntuW = ntuMin;
43   else ntuW = ntuMin * mwc / airFlowRate;
44
45   //冷却塔の特性係数c[-]を出力
46   return ntuW / Math.Pow(waterFlowRate / airFlowRate, -0.6);
47 }
```

12.3.2　除去熱量の計算

流体の入口状態および流量が与えられた場合に熱除去量を計算するためには、基本的にはプログラム 12.2 の逆の操作を行う。ただし、水の出口温度が不明のため、式 12.8 の c_s を与えることができない。そこで水の出口温度を仮定して反復計算を行う。除去熱量の計算処理をプログラム 12.3 に示す。23~46 行は、冷却水出口水温にもとづいて誤差評価を行う関数である。50 行でニュートン・ラフソン法を用いて冷却水出口水温を収束計算する。

プログラム **12.3**　冷却塔による除去熱量の計算

Popolo.HVAC.HeatExchanger.CoolingTower class

```
 1 /// <summary>除去熱量[kW]を計算する</summary>
 2 /// <param name="inletWaterTemperature">冷却水入口温度[C]</param>
 3 /// <param name="wetbulbTemperature">空気入口湿球温度[C]</param>
 4 /// <param name="waterFlowRate">冷却水の質量流量[kg/s]</param>
 5 /// <param name="airFlowRate">空気の質量流量[kg/s]</param>
 6 /// <param name="coolingTowerCoefficient">冷却塔の特性係数[-]</param>
 7 /// <param name="airFlowType">流れのタイプ</param>
 8 /// <returns>除去熱量[kW]</returns>
 9 public static double GetHeatRejection
10   (double inletWaterTemperature, double wetbulbTemperature, double waterFlowRate, double airFlowRate,
11   double coolingTowerCoefficient, AirFlowDirection airFlowType)
12 {
13   //冷却水温度に相当する空気の飽和エンタルピーを計算
14   double hswi = MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
```

```
15     (inletWaterTemperature, 100, ATMOSPHERIC_PRESSURE);
16   //入口空気の飽和エンタルピーを計算
17   double hai = MoistAir.GetEnthalpyFromWetBulbTemperatureAndRelativeHumidity
18     (wetbulbTemperature, 100, ATMOSPHERIC_PRESSURE);
19   //NTUw を計算
20   double ntuw = coolingTowerCoefficient * Math.Pow(waterFlowRate / airFlowRate, -0.6);
21
22   //誤差関数を定義
23   Roots.ErrorFunction eFnc = delegate (double waterOutletTemperature)
24   {
25     //平均的な比熱を計算
26     double hswo = MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
27     (waterOutletTemperature, 100, ATMOSPHERIC_PRESSURE);
28     double cs = (hswi - hswo) / (inletWaterTemperature - waterOutletTemperature);
29
30     //冷却水の換算流量を計算
31     double mwc = waterFlowRate * WATER_SPECIFIC_HEAT / cs;
32
33     //熱通過有効度を計算
34     double ntu;
35     if (airFlowRate < mwc) ntu = ntuw * (mwc / airFlowRate);
36     else ntu = ntuw;
37     HeatExchange.FlowType ft;
38     if (airFlowType == AirFlowDirection.CounterFlow) ft = HeatExchange.FlowType.CounterFlow;
39     else ft = HeatExchange.FlowType.CrossFlow_BothFluidsUnmixed;
40     double epsilon = HeatExchange.GetEffectiveness
41     (ntu, Math.Min(airFlowRate, mwc) / Math.Max(airFlowRate, mwc), ft);
42
43     //除去熱量を計算して誤差を評価
44     double hReject = epsilon * (hswi - hai) * Math.Min(airFlowRate, mwc);
45     return hReject - (inletWaterTemperature - waterOutletTemperature) * waterFlowRate * WATER_SPECIFIC_HEAT;
46   };
47
48   //冷却水出口温度を収束計算
49   double wOutT = inletWaterTemperature - 1;
50   wOutT = Roots.Newton(eFnc, wOutT, 0.0001, 0.01, 0.001, 10);
51
52   return (inletWaterTemperature - wOutT) * waterFlowRate * WATER_SPECIFIC_HEAT;
53 }
```

12.3.3　必要風量の計算

実際のシステムにおいて、冷却塔の出口温度を制御する際には冷却塔ファンを発停またはインバータ制御することが多い。この制御を評価するためには、ある除去熱量を実現するために必要となる風量を計算する必要がある。空気と冷却水の入口温度および除去熱量が与えられるため、式 12.10 により εm_{min} を計算することができる。ただし εm_{min} は風量 m_a の関数であるが m_a について解くことはできないため、適当な風量を仮定して反復収束計算を行う。

プログラム 12.4 に必要風量の計算処理を示す。入力条件から水と空気の比エンタルピーを求め（15~22 行）、水を冷却することができない場合には風量を 0 とする（24~29 行）。31~37 行で εm_{min} と冷却水の換算流量を計算して反復計算に備える。39 行~58 行は誤差関数であり、式 12.11~12.15を用いることで任意の風量 m_a に対する εm_{min} を計算して上記の εm_{min} との差を算出する。最大風量でも除去熱量が達成できない場合には最大風量を出力する（61 行）。熱除去が可能な場合には、0~最大風量までの間に解が存在するはずであるから、第 2 章で説明した二分法を用いて解を求める。

プログラム 12.4　必要風量の計算

Popolo.HVAC.HeatExchanger.CoolingTower class

```
1 /// <summary>必要風量[kg/s]を計算する</summary>
2 /// <param name="inletWaterTemperature">冷却水入口温度[C]</param>
3 /// <param name="outletWaterTemperature">冷却水出口温度[C]</param>
4 /// <param name="wetbulbTemperature">空気入口湿球温度[C]</param>
5 /// <param name="waterFlowRate">冷却水の質量流量[kg/s]</param>
6 /// <param name="maxAirFlowRate">空気の最大の質量流量[kg/s]</param>
7 /// <param name="coolingTowerCoefficient">冷却塔の特性係数 c[-]</param>
8 /// <param name="airFlow">流れのタイプ</param>
9 /// <returns>除去熱量[kW]</returns>
10 public static double GetAirFlowRate
11     (double inletWaterTemperature, double outletWaterTemperature,
```

```
12     double wetbulbTemperature, double waterFlowRate, double maxAirFlowRate,
13     double coolingTowerCoefficient, AirFlowDirection airFlow)
14 {
15   //冷却水温度に相当する空気の飽和エンタルピーを計算
16   double hswi = MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
17       (inletWaterTemperature, 100, ATMOSPHERIC_PRESSURE);
18   double hswo = MoistAir.GetEnthalpyFromDryBulbTemperatureAndRelativeHumidity
19       (outletWaterTemperature, 100, ATMOSPHERIC_PRESSURE);
20   //入口空気の飽和エンタルピーを計算
21   double hai = MoistAir.GetEnthalpyFromWetBulbTemperatureAndRelativeHumidity
22       (wetbulbTemperature, 100, ATMOSPHERIC_PRESSURE);
23
24   //加熱してしまう条件の場合には風量0とする
25    if (hswi < hai)
26   {
27     isOverLoad = true;
28     return 0;
29   }
30
31   //熱通過有効度×質量流量：εmの計算
32   double emmi = waterFlowRate * WATER_SPECIFIC_HEAT
33     * (inletWaterTemperature - outletWaterTemperature) / (hswi - hai);
34
35   //平均的な比熱で冷却水の換算流量を計算
36   double cs = (hswi - hswo) / (inletWaterTemperature - outletWaterTemperature);
37   double mwc = waterFlowRate * WATER_SPECIFIC_HEAT / cs;
38
39   //誤差関数を定義
40   Roots.ErrorFunction eFnc = delegate(double afRate)
41   {
42     //NTUを計算
43     double ntuw = coolingTowerCoefficient *
44     Math.Pow(waterFlowRate / afRate, -0.6);
45     double ntu;
46     if (afRate < mwc) ntu = ntuw * (mwc / afRate);
47     else ntu = ntuw;
48
49     //熱通過有効度を計算
50     double mmin = Math.Min(mwc, afRate);
51     double mmax = Math.Max(mwc, afRate);
52     HeatExchanger.FlowType ft;
53     if (airFlow == AirFlowDirection.CounterFlow) ft = HeatExchanger.FlowType.CounterFlow;
54     else ft = HeatExchanger.FlowType.CrossFlow_BothFluidsUnmixed;
55     double epsilon = HeatExchanger.GetEffectiveness(ntu, mmin / mmax, ft);
56
57     return emmi - epsilon * mmin;
58   };
59
60   //風量不足の場合には最大風量を出力する
61   if (0 < eFnc(maxAirFlowRate)) return maxAirFlowRate;
62
63   //風量比0.1%で過剰処理の場合には0.1%とする
64   isOverLoad = 0 < eFnc(0.001 * maxAirFlowRate);
65   if (!isOverLoad) return 0.001 * maxAirFlowRate;
66
67   //二分法で収束計算//誤差率0.1%未満
68   isOverLoad = false;
69   return Roots.Bisection(eFnc, 0.001 * maxAirFlowRate, maxAirFlowRate, 0.001, 0.001 * maxAirFlowRate, 20);
70 }
```

12.3.4　ファンの消費電力および水の消費量の計算

プログラム 12.5 にファンの消費電力量の計算処理を示す。式 12.25 および 12.26 にもとづき、インバータ制御と発停制御の別に応じて計算を行う。

プログラム 12.5　ファンの消費電力量の計算

Popolo.HVAC.HeatExchanger.CoolingTower class

```
1 /// <summary>ファン消費電力[kW]を計算する</summary>
2 /// <param name="airFlowRate">空気の質量流量[kg/s]</param>
3 /// <param name="nominalAirFlowRate">定格風量[kg/s]</param>
4 /// <param name="nominalPowerConsumption">定格消費電力[kW]</param>
5 /// <param name="hasInverter">インバータ制御か否か</param>
6 /// <returns>ファン消費電力[kW]</returns>
7 public static double GetPowerConsumption
8   (double airFlowRate, double nominalAirFlowRate, double nominalPowerConsumption, bool hasInverter)
9 {
10   if (hasInverter) return nominalPowerConsumption * Math.Pow(airFlowRate / nominalAirFlowRate, 3);
11   else return nominalPowerConsumption * airFlowRate / nominalAirFlowRate;
12 }
```

プログラム 12.6 に水消費量の計算処理を示す。式 12.18~12.24にもとづき、蒸発、飛散、ブローによる水消費量をそれぞれ計算する。

プログラム 12.6　水消費量の計算

Popolo.HVAC.HeatExchanger.CoolingTower class

```
/// <summary>水消費量[kg/s]を計算する</summary>
/// <param name="heatRejection">除去熱量[kW]</param>
/// <param name="airEnthalpy">入口空気のエンタルピー[kJ/kg]</param>
/// <param name="airHumidityRatio">入口空気の絶対湿度[kg/kg]</param>
/// <param name="waterFlowRate">冷却水の質量流量[kg/s]</param>
/// <param name="airFlowRate">空気の質量流量[kg/s]</param>
/// <param name="nominalAirFlowRate">定格点での空気の質量流量[kg/s]</param>
/// <param name="driftWaterRate">飛散水量比[-]</param>
/// <param name="concentrationRatio">濃縮倍率[-]</param>
/// <param name="evaporationWater">出力：蒸発による水消費量[kg/s]</param>
/// <param name="driftWater">出力：飛散による水消費量[kg/s]</param>
/// <param name="blowDownWater">出力：ブローによる水消費量[kg/s]</param>
public static void GetMakeupWater
  (double heatRejection, double airEnthalpy, double airHumidityRatio, double waterFlowRate,
  double airFlowRate, double nominalAirFlowRate, double driftWaterRate, double concentrationRatio,
  out double evaporationWater, out double driftWater, out double blowDownWater)
{
  double outletHRatio;  //処理熱量から蒸発水を計算
  if (heatRejection <= 0 || airFlowRate <= 0) outletHRatio = 0;
  else outletHRatio = MoistAir.GetHumidityRatioFromEnthalpyAndRelativeHumidity
    (airEnthalpy + heatRejection / airFlowRate, 100, ATMOSPHERIC_PRESSURE);
  evaporationWater = airFlowRate * (outletHRatio - airHumidityRatio);

  //循環量から飛散水を計算
  driftWater = waterFlowRate * driftWaterRate * (airFlowRate / nominalAirFlowRate);

  //0kg/s 以上になるようにブロー水量を調整
  blowDownWater = Math.Max(0, evaporationWater / (concentrationRatio - 1) - driftWater);
}
```

12.3.5　「冷却塔クラス」の作成

プログラム 12.7 にインスタンス変数とプロパティの定義を示す。濃縮倍率と飛散水量比は 0 よりも大きな値に制限する（40~52 行）。

プログラム 12.7　インスタンス変数とプロパティの定義

Popolo.HVAC.HeatExchanger.CoolingTower class

```
/// <summary>特性係数 c[-]</summary>
private double coefC;

/// <summary>飛散水量比[-]</summary>
private double driftWaterRate = 0.002;

/// <summary>濃縮倍率[-]</summary>
private double concentrationRatio = 4;

/// <summary>水量[kg/s]を設定・取得する</summary>
public double WaterFlowRate { get; set; }

/// <summary>上限水量[kg/s]を取得する</summary>
public double MaxWaterFlowRate { get; private set; }

/// <summary>風量[kg/s]を取得する</summary>
public double AirFlowRate { get; private set; }

/// <summary>最大風量[kg/s]を取得する</summary>
public double MaxAirFlowRate { get; private set; }

/// <summary>外気湿球温度[C]を取得する</summary>
public double OutdoorWetbulbTemperature { get; private set; } = 27;

/// <summary>外気絶対湿度[kg/kg]を取得する</summary>
public double OutdoorHumidityRatio { get; private set; } = 0.0105;

/// <summary>入口水温[C]を取得する</summary>
public double InletWaterTemperature { get; private set; }

/// <summary>出口水温[C]を取得する</summary>
public double OutletWaterTemperature { get; private set; }

/// <summary>出口水温設定値[C]を設定・取得する</summary>
```

```
35 public double OutletWaterSetPointTemperature { get; set; }
36
37 /// <summary>除去熱量[kW]を取得する</summary>
38 public double HeatRejection { get; private set; }
39
40 /// <summary>濃縮倍率[-]を設定・取得する</summary>
41 public double ConcentrationRatio
42 {
43   get { return concentrationRatio; }
44   set { concentrationRatio = Math.Max(0, value); }
45 }
46
47 /// <summary>飛散水量比[-]を設定・取得する</summary>
48 public double DriftWaterRate
49 {
50   get { return driftWaterRate; }
51   set { driftWaterRate = Math.Min(1, Math.Max(0, value)); }
52 }
53
54 /// <summary>インバータ制御か否か</summary>
55 public bool HasInverter { get; private set; }
56
57 /// <summary>空気の流れのタイプを取得する</summary>
58 public AirFlowDirection AirFlowType { get; private set; }
59
60 /// <summary>定格消費電力[kW]を取得する</summary>
61 public double NominalPowerConsumption { get; private set; }
62
63 /// <summary>消費電力[kW]を取得する</summary>
64 public double ElectricConsumption { get; private set; }
65
66 /// <summary>蒸発による水消費量[kg/s]を取得する</summary>
67 public double EvaporationWater { get; private set; }
68
69 /// <summary>飛散による水消費量[kg/s]を取得する</summary>
70 public double DriftWater { get; private set; }
71
72 /// <summary>ブローによる水消費量[kg/s]を取得する</summary>
73 public double BlowDownWater { get; private set; }
74
75 /// <summary>水消費量（蒸発+飛散+ブロー）[kg/s]を取得する</summary>
76 public double WaterConsumption
77 { get { return EvaporationWater + DriftWater + BlowDownWater; } }
78
79 /// <summary>過負荷か否か</summary>
80 public bool IsOverLoad { get; private set; }
```

プログラム 12.8 にコンストラクタを示す。14~20 行で定格仕様をプロパティに保存した後、23 行で特性係数 c_s を求め、27 行で出口状態を計算して初期化する。

冷却塔のカタログには風量に関する情報は記載されていないことが多い。いくつかの事例を参照すると、質量流量としては水量に対して 0.8 程度の風量とする機器が多いため、この情報を用いて風量情報を引数としないコンストラクタのオーバーロードを 30~44 行で定義する。ファンの消費電力も 46~59 行のメソッドを用いて能力から推定する。

プログラム 12.8　コンストラクタの定義

Popolo.HVAC.HeatExchanger.CoolingTower class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="inletWaterTemperature">入口水温[C]</param>
 3 /// <param name="outletWaterTemperature">出口水温[C]</param>
 4 /// <param name="wetbulbTemperature">外気湿球温度[C]</param>
 5 /// <param name="waterFlowRate">冷水流量[kg/s]</param>
 6 /// <param name="airFlowRate">風量[kg/s]</param>
 7 /// <param name="airFlowType">流れのタイプ</param>
 8 /// <param name="powerConsumption">消費電力[kW]</param>
 9 /// <param name="hasInverter">インバータ制御か否か</param>
10 public CoolingTower
11   (double inletWaterTemperature,double outletWaterTemperature, double wetbulbTemperature, double waterFlowRate,
12   double airFlowRate, AirFlowDirection airFlowType, double powerConsumption, bool hasInverter)
13 {
14   //定格条件を保存
15   OutletWaterSetPointTemperature = outletWaterTemperature;
16   MaxAirFlowRate = AirFlowRate = airFlowRate;
17   MaxWaterFlowRate = WaterFlowRate = waterFlowRate;
```

```
18   HasInverter = hasInverter;
19   AirFlowType = airFlowType;
20   NominalPowerConsumption = powerConsumption;
21
22   //定格条件にもとづき特性係数 c[-]を初期化
23   coefC = GetCoolingTowerCoefficient
24     (inletWaterTemperature,outletWaterTemperature,wetbulbTemperature, waterFlowRate, airFlowRate, airFlowType);
25
26   //出口状態を初期化
27   Update(inletWaterTemperature, false);
28 }
29
30 /// <summary>インスタンスを初期化する</summary>
31 /// <param name="inletWaterTemperature">入口水温[C]</param>
32 /// <param name="outletWaterTemperature">出口水温[C]</param>
33 /// <param name="wetbulbTemperature">外気湿球温度[C]</param>
34 /// <param name="waterFlowRate">冷水流量[kg/s]</param>
35 /// <param name="airFlowType">流れのタイプ</param>
36 /// <param name="hasInverter">インバータ制御か否か</param>
37 public CoolingTower
38   (double inletWaterTemperature, double outletWaterTemperature, double wetbulbTemperature,
39   double waterFlowRate, AirFlowDirection airFlowType, bool hasInverter)
40   : this(inletWaterTemperature, outletWaterTemperature, wetbulbTemperature,
41     waterFlowRate, waterFlowRate * 0.8, airFlowType,
42     getFunPower((inletWaterTemperature - outletWaterTemperature) * waterFlowRate * WATER_SPECIFIC_HEAT),
43     hasInverter)
44 { }
45
46 /// <summary>定格能力[kW]から定格ファン動力[kW]を推定する</summary>
47 /// <param name="load">定格能力[kW]</param>
48 /// <returns>定格ファン動力[kW]</returns>
49 private static double getFunPower(double load)
50 {
51   if (load < 30) return 0.2;
52   else if (load < 35) return 0.4;
53   else if (load < 100) return 0.75;
54   else if (load < 210) return 1.5;
55   else if (load < 350) return 2.2;
56   else if (load < 530) return 5.5;
57   else if (load < 800) return 7.5;
58   else throw new Exception("Out of range");
59 }
```

外気条件の設定と状態更新処理をプログラム 12.9 に示す。1~8 行は外気条件設定処理である。10~41 行のメソッドは任意の入口水温と風量について状態を更新する。クラスの外部からは 43~50 行のメソッドで呼び出す。冷却塔の制御方法としては、1）風量を制御して出口冷却水温度を制御する、2）風量を固定して出口冷却水温度を成り行きにする、の 2 種類が考えられる。52~75 行のメソッドはこの 2 つの制御に対応したメソッドである。出口冷却水温度を制御する場合には、60~73 行で所要風量を求める。ただし、入口水温が既に出口水温設定値よりも低い場合には機器を停止させる。停止処理は 77~83 行であり、出口水温=入口水温としてエネルギー消費や水消費を 0 にする。

プログラム 12.9　成り行き状態の計算

Popolo.HVAC.HeatExchanger.CoolingTower class

```
 1 /// <summary>外気条件を設定する</summary>
 2 /// <param name="wetbulbTemperature">湿球温度[C]</param>
 3 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
 4 public void SetOutdoorAirState(double wetbulbTemperature, double humidityRatio)
 5 {
 6   OutdoorWetbulbTemperature = wetbulbTemperature;
 7   OutdoorHumidityRatio = humidityRatio;
 8 }
 9
10 /// <summary>状態を更新する</summary>
11 /// <param name="inletWaterTemperature">入口水温[C]</param>
12 /// <param name="airFlowRate">風量[kg/s]</param>
13 private void update(double inletWaterTemperature, double airFlowRate)
14 {
15   //入力条件設定
16   InletWaterTemperature = inletWaterTemperature;
17   AirFlowRate = airFlowRate;
```

```
18
19   //除去熱量から冷却水出口温度を計算
20   if (AirFlowRate <= 0 || WaterFlowRate <= 0)
21   {
22     HeatRejection = 0;
23     OutletWaterTemperature = InletWaterTemperature;
24   }
25   else
26   {
27     HeatRejection = GetHeatRejection
28       (InletWaterTemperature, OutdoorWetbulbTemperature, WaterFlowRate, AirFlowRate, coefC, AirFlowType);
29     OutletWaterTemperature = InletWaterTemperature - HeatRejection / (WATER_SPECIFIC_HEAT * WaterFlowRate);
30
31   //電力と水の消費量を計算
32   ElectricConsumption = GetPowerConsumption(AirFlowRate, MaxAirFlowRate, NominalPowerConsumption, HasInverter);
33   double ew, dw, bw;
34   double hOA = MoistAir.GetEnthalpyFromHumidityRatioAndWetBulbTemperature
35     (OutdoorHumidityRatio, OutdoorWetbulbTemperature, ATMOSPHERIC_PRESSURE);
36   GetMakeupWater(HeatRejection, hOA, OutdoorHumidityRatio, WaterFlowRate,
37     AirFlowRate, MaxAirFlowRate, DriftWaterRate, ConcentrationRatio, out ew, out dw, out bw);
38   EvaporationWater = ew;
39   DriftWater = dw;
40   BlowDownWater = bw;
41 }
42
43 /// <summary>状態を更新する</summary>
44 /// <param name="inletWaterTemperature">入口水温[C]</param>
45 /// <param name="airFlowRate">風量[kg/s]</param>
46 public void Update(double inletWaterTemperature, double airFlowRate)
47 {
48   IsOverLoad = false;
49   update(inletWaterTemperature, airFlowRate);
50 }
51
52 /// <summary>状態を更新する</summary>
53 /// <param name="inletWaterTemperature">入口水温[C]</param>
54 /// <param name="controlOutletWaterTemperature">出口水温を制御するか</param>
55 public void Update(double inletWaterTemperature, bool controlOutletWaterTemperature)
56 {
57   //入力条件設定
58   InletWaterTemperature = inletWaterTemperature;
59
60   //出口温度を制御する場合には風量を調整
61   if (controlOutletWaterTemperature)
62   {
63     if (InletWaterTemperature <= OutletWaterSetPointTemperature) ShutOff();
64     else
65     {
66       bool oload;
67       double af = GetAirFlowRate
68         (inletWaterTemperature, OutletWaterSetPointTemperature, OutdoorWetbulbTemperature,
69         WaterFlowRate, MaxAirFlowRate, coefC, AirFlowType, out oload);
70       IsOverLoad = oload;
71       update(inletWaterTemperature, af);
72     }
73   }
74   else Update(inletWaterTemperature, MaxAirFlowRate);
75 }
76
77 /// <summary>停止させる</summary>
78 public void ShutOff()
79 {
80   OutletWaterTemperature = InletWaterTemperature;
81   HeatRejection = ElectricConsumption = 0;
82   EvaporationWater = DriftWater = BlowDownWater = 0;
83 }
```

【例題 12.3】

例題 12.1 で示した冷却塔について、冷却水入口温度、空気入口湿球温度、冷却水流量、空気風量の変化に対する感度を確認するプログラムを作成せよ。

【解】

プログラム 12.10 に示す。4 行で冷却塔クラスのインスタンスを作成する。9~19 行, 21~32 行, 34~44 行, 46~55 行でそれぞれの入力条件を変化させて出口状態を更新する。計算結果をグラフ化すると図 12.2 が得られる。いずれの要素に対してもほぼ線形に変化することがわかる。水量比と風量比では、相対的に水量比の方が影響が大きく、風量を変化させてもあまり能力に変化は無い。

プログラム **12.10**　冷却塔の感度解析

```
private static void CoolingTowerTest1()
{
  double twb = MoistAir.GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio(35, 0.0195, 101.325);
  CoolingTower ct=new CoolingTower(37, 32, twb, 43.3, 35.7, CoolingTower.AirFlowDirection.CrossFlow,7.5,false);

  using (StreamWriter sWriter = new StreamWriter
    ("CoolingTowerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
  {
    //入口冷却水温度に対する感度
    sWriter.WriteLine("冷却水温度[C], 除去熱量[kW], 出口水温[C]");
    double wbt = MoistAir.GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio(35, 0.0195, 101.325);
    ct.SetOutdoorAirState(wbt, 0.0195);
    for (int i = 0; i < 10; i++)
    {
      double wit = 37 - i * 0.5;
      ct.Update(wit, false);
      sWriter.WriteLine(wit.ToString("F1") + ", " + ct.HeatRejection.ToString("F1")
        + ", " + ct.OutletWaterTemperature.ToString("F2"));
    }

    //入口空気相対湿度に対する感度
    sWriter.WriteLine("相対湿度[%], 除去熱量[kW], 出口水温[C]");
    for (int i = 0; i < 10; i++)
    {
      double rhd = 55 - 2 * i;
      double hrt = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity(35, rhd, 101.325);
      double wbt2 = MoistAir.GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio(35, hrt, 101.325);
      ct.SetOutdoorAirState(wbt2, hrt);
      ct.Update(37, false);
      sWriter.WriteLine(rhd.ToString("F0") + ", " + ct.HeatRejection.ToString("F1")
        + ", " + ct.OutletWaterTemperature.ToString("F2"));
    }

    //冷却水流量に対する感度
    sWriter.WriteLine("冷却水流量比[-], 除去熱量[kW], 出口水温[C]");
    ct.SetOutdoorAirState(wbt, 0.0195);
    for (int i = 0; i < 10; i++)
    {
      double pl = (1 - 0.05 * i);
      ct.WaterFlowRate = 43.3 * pl;
      ct.Update(37, false);
      sWriter.WriteLine(pl.ToString("F2") + ", " + ct.HeatRejection.ToString("F1")
        + ", " + ct.OutletWaterTemperature.ToString("F2"));
    }

    //風量に対する感度
    sWriter.WriteLine("風量比[-], 除去熱量[kW], 出口水温[C]");
    ct.WaterFlowRate = 43.3;
    for (int i = 0; i < 10; i++)
    {
      double pl = (1 - 0.05 * i);
      ct.Update(37, ct.MaxAirFlowRate * pl);
      sWriter.WriteLine(pl.ToString("F2") + ", " + ct.HeatRejection.ToString("F1")
        + ", " + ct.OutletWaterTemperature.ToString("F2"));
    }
  }
}
```

【例題 12.4】

例題 12.1 で示した冷却塔について、負荷率とファン消費電力および水消費量の関係を計算せよ。

【解】

現実の設備システムにおいては、冷却水の変流量制御を導入しない場合には、熱源機の負荷率減少に伴って、冷却水の出入口温度差が小さくなる。この変化が負荷率に比例的と仮定して 19 行で冷却水入口水温の低下を計算する。計算結果を図 12.3 に示す。INV 制御の場合にはファンの消費電力が負荷率の 3 乗に比例するため、負荷の低下に伴って大きく消費電力が低下することがわかる。水の消費量は蒸発量が支配的であり、飛散量とブロー量はほぼ同等である。

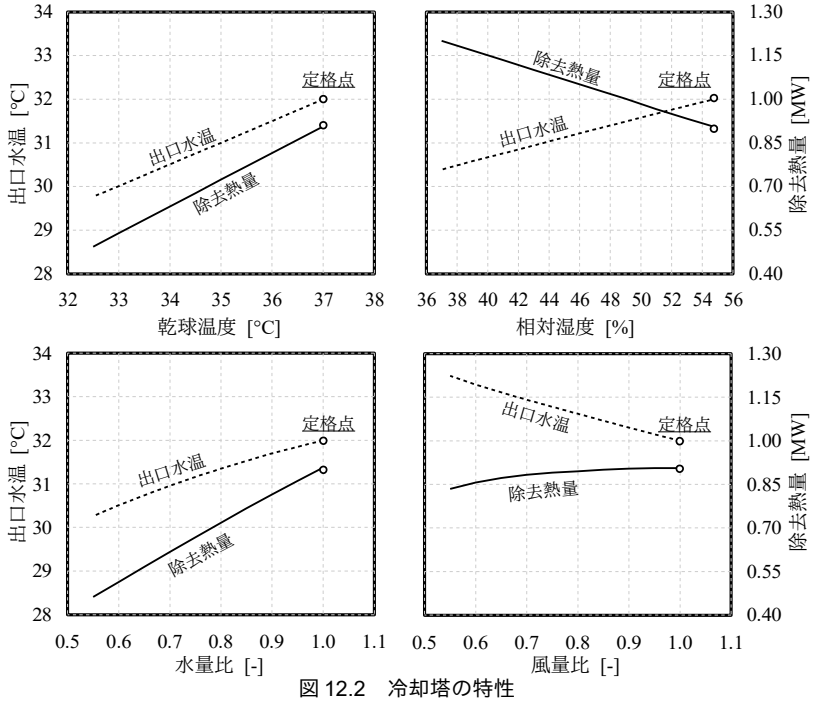


図 12.2　冷却塔の特性

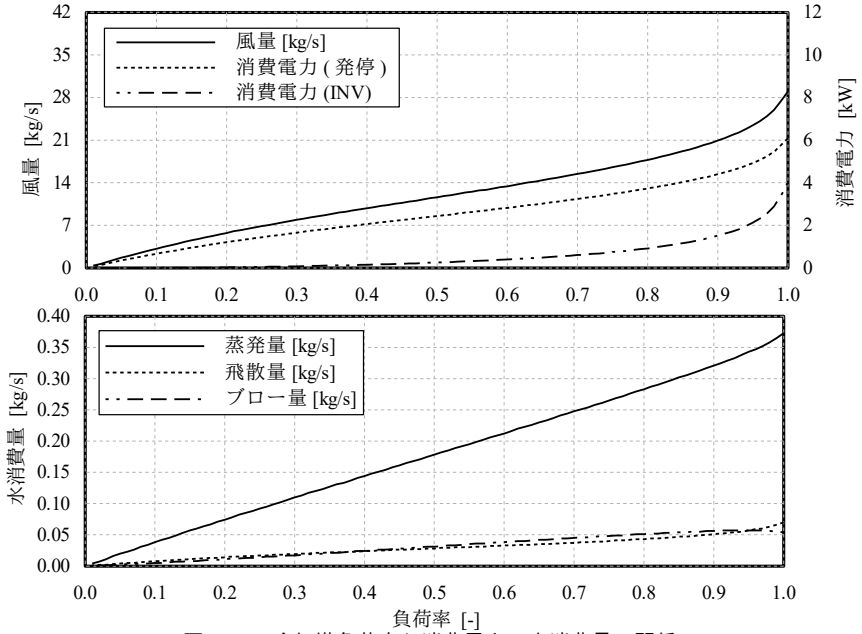


図 12.3　冷却塔負荷率と消費電力・水消費量の関係

プログラム **12.11**　冷却塔負荷率とファン動力および水消費量の関係

```
private static void CoolingTowerTest2()
{
  double twb = MoistAir.GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio(35, 0.0195, 101.325);
  CoolingTower ct_INV = new CoolingTower
    (37, 32, twb, 43.3, 35.7, CoolingTower.AirFlowDirection.CrossFlow, 7.5, true);
  CoolingTower ct_NON = new CoolingTower
    (37, 32, twb, 43.3, 35.7, CoolingTower.AirFlowDirection.CrossFlow, 7.5, false);

  using (StreamWriter sWriter = new StreamWriter
    ("CoolingTowerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
  {
    //負荷とファン消費電力・水消費量との関係
```

```
13     sWriter.WriteLine
14       ("負荷率[-], 風量[kg/s], 消費電力(発停), 消費電力(INV), 蒸発量[kg/s], 飛散量[kg/s], ブロー量[kg/s]");
15     for (int i = 0; i < 100; i++)
16     {
17       //負荷率によって冷却水入口温度を調整
18       double pLoad = 1 - i * 0.01;
19       double wInletTemp = 32 + 5 * pLoad;
20
21       //状態更新処理
22       ct_INV.Update(wInletTemp, true); //INV 制御
23       ct_NON.Update(wInletTemp, true); //発停制御
24
25       //結果出力
26       sWriter.WriteLine(pLoad.ToString("F2") + ", " + ct_INV.AirFlowRate.ToString("F2") + ", "
27         + ct_NON.ElectricConsumption.ToString("F2") + ", " + ct_INV.ElectricConsumption.ToString("F2") + ", "
28         + ct_INV.EvaporationWater.ToString("F3") + ", " + ct_INV.DriftWater.ToString("F3") + ", "
29         + ct_INV.BlowDownWater.ToString("F3"));
30     }
31   }
32 }
```

【例題 12.5】

冷却塔は水の蒸発潜熱を用いて放熱を行うため、パッケージ空調機や空気熱源ヒートポンプなど、顕熱交換のみで放熱を行う機器に比較するとヒートアイランド現象を抑制する働きがあるとされている [12.7]。冷却塔の放熱量に占める潜熱交換量の比率を乾球温度と相対湿度の組み合わせ別に計算せよ。ただし機器仕様は例題 12.1 の通りとし、冷却水入口温度は 37℃ で固定の上、成り行き状態を計算する。

【解】

計算処理をプログラム 12.12 に示す。15, 18 行で相対湿度と乾球温度のループを行う。21~25 行で外気絶対湿度を求め、成り行きでの出口状態を計算する。28, 29 行で交換熱量から出口空気の比エンタルピーを求め、湿り空気の物性計算により出口空気の絶対湿度を計算する。33, 34 行で出入口空気の絶対湿度差と水の蒸発潜熱にもとづき潜熱交換量を計算し、全放熱量との比率を計算する。

計算結果を等高線図で表現すると図 12.4 が得られる。第 7 章で計算した東京都のクリモグラフも記載した。通常の建物では 6 月~9 月頃が冷房期間であるため、80~90%程度の熱が潜熱として放熱されることになる†1)。また、乾球温度の低い冬季においても潜熱交換比率が 65%を超えており、冷却塔は潜熱交換に頼った放熱システムであるということがわかる。なお、図の右下には潜熱交換量が 100%を超える領域があるが、これは蒸発によって出口空気温度が入口空気温度よりも小さくなる運転点である。

プログラム 12.12　冷却塔の潜熱交換量比率の計算

```
 1 private static void CoolingTowerTest3()
 2 {
 3   double twb = MoistAir.GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio(35, 0.0195, 101.325);
 4   CoolingTower ct = new CoolingTower
 5     (37, 32, twb, 43.3, 35.7, CoolingTower.AirFlowDirection.CrossFlow, 7.5, false);
 6
 7   using (StreamWriter sWriter = new StreamWriter
 8     ("CoolingTowerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
 9   {
10     //タイトル行
11     sWriter.Write(" ");
12     for (int j = -5; j <= 35; j++) sWriter.Write("," + j.ToString("F0"));
13     sWriter.WriteLine(" ");
14
15     for (int i = 40; i <= 100; i++)
16     {
17       sWriter.Write(i.ToString("F0"));
18       for (int j = -5; j <= 35; j++)
19       {
20         //外気条件を更新して成り行き出口状態を計算
21         double hrti =
22           MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity(j, i, 101.325);
23         double wbti = MoistAir.GetWetBulbTemperatureFromDryBulbTemperatureAndHumidityRatio(j, hrti, 101.325);
24         ct.SetOutdoorAirState(wbti, hrti);
25         ct.Update(37, false);
26
27         //熱交換量から出口空気状態を計算
28         double enth = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(j, hrti)
29           + ct.HeatRejection / ct.AirFlowRate;
30         double hrto = MoistAir.GetHumidityRatioFromEnthalpyAndRelativeHumidity(enth, 100, 101.325);
```

†1　このクリモグラフは月平均乾球温度と相対湿度にもとづいて作成しているため、昼間の運転が主である冷却塔の運転点の評価として妥当なのか、気になるかもしれない。冷却塔性能に影響を与える湿球温度の昼夜間の変動は、乾球温度や絶対湿度の変動に比較して小さいため、大きな問題は無いはずである。筆者はこの傾向を知らず、夜間冷気を用いて冷却水放熱を夜間移行する設計を試みて失敗したことがある。

```
31
32          //蒸発潜熱と絶対湿度から潜熱交換量を計算
33          double lh = (hrto - hrti) * MoistAir.LatentHeatOfVaporization * ct.AirFlowRate;
34          double lhr = lh / ct.HeatRejection;
35
36          sWriter.Write("," + lhr.ToString("F3"));
37        }
38        sWriter.WriteLine();
39      }
40    }
41 }
```

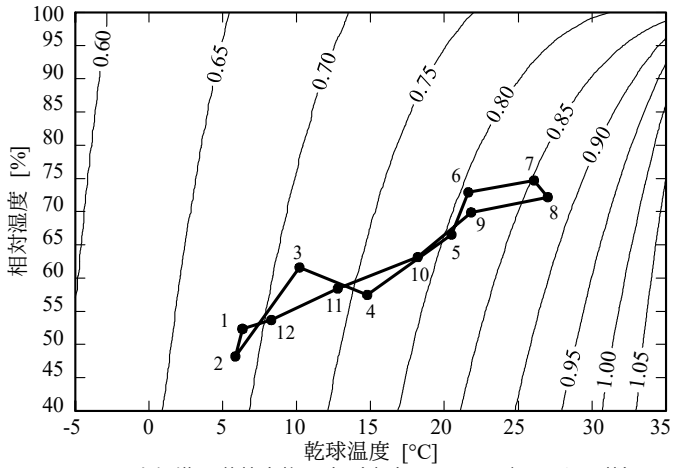


図 12.4　冷却塔の潜熱交換比率（東京のクリモグラフを記載）

【第 12 章 記号表】

a	：体積あたりの水の表面積 [m^2/m^3]	M	：補給水量 [kg/s]
B	：冷却水のブロー量 [kg/s]	N	：濃縮倍数 [-]
c	：冷却塔の特性係数 [-]	NTU	：移動単位数 [-]
c_{pa}	：湿り空気の定圧比熱 [kJ/(kg·K)]	P_{ct}	：冷却塔ファンの消費電力 [kW]
c_{pw}	：水の定圧比熱 [kJ/(kg·K)]	q	：熱交換量 [kW]
E	：冷却水の蒸発量 [kg/s]	t	：温度 [°C]
h	：比エンタルピー [kJ/kg]	V	：体積 [m^3]
K_G	：対流熱伝達率 [$kW/(m^2·K)$]	W	：冷却水の飛散量 [kg/s]
K_x	：物質伝達率 [$kg/(s·m^2·kg/kg)$]	x	：絶対湿度 [kg/kg]
m	：質量流量 [kg/s]	ε	：熱通過有効度 [-]
sub scripts			
a	：空気	sw	：飽和状態
w	：水	$rated$	：定格点

【第 12 章 参考文献】

12.1） Baker, Donald R., and Shryock, Howard A., A Comprehensive Approach to the Analysis of Cooling Tower Performance, 1961, ASME J. Heat Transfer, 83, pp. 339–349

12.2） Lu, L. and Cai, W., "A Universal Engineering Model For Cooling Towers" (2002). International Refrigeration and Air Conditioning Conference. Paper 625.

12.3） 高田秋一, 川原孝七: クーリングタワー 省エネルギー技術実践シリーズ, 省エネルギーセンター, 2003

12.4） Braun, J.E.: Methodologies for Design and Control of Central Cooling Plants, Ph.D. Thesis, University of Wisconsin-Madison

12.5） Simpson, W.M. and Sherwood, T.K., : Performance of Small Mechanical Draft Cooling Towers, Refrigerating Engineering, Vol.52, No.6, pp.535~543, 574~576, 1946

12.6） R.L. Webb, Ph.D., P.E. A. Villacres : Algorithms for Performance Simulation of Cooling Towers, Evaporative Condensers, and Fluid Coolers, ASHRAE Transactions, vol.90, 1984, pp416-458

12.7） 田中俊彦, 亀谷茂樹, 水野稔, 下田吉之 ,葛原浩美 ,西隆良: 空調システムをもつ建物からの熱環境負荷に関する研究 ：第 1 報-空調システムの相違による建物からの廃熱特性の検討 , 空気調和・衛生工学会論文集, No.64, pp.49-59, 1997

12.8） 中村孟: TAKENAKA BOOKS 3 近代建築設備の系譜, 株式会社 竹中工務店, p.220, 1987

第13章　ボイラ
(Boiller)

13.1　概要

冷水、温水、蒸気などを製造する設備機器を熱源と呼ぶ。空調熱源設備のエネルギー消費量の大部分は熱源によるものであり、エネルギー消費量の計算にあたって熱源の挙動を正しくつかむことは重要である。温水や蒸気を供給する熱源を温熱源、冷水を供給する熱源を冷熱源と呼ぶ。この他、温熱と冷熱を同時供給可能な機器もある。本章で解説を行うボイラは温熱源の代表格であり、温水ボイラと蒸気ボイラに大別できる。

ボイラを含め、熱源はこれまでの章で解説した各種の機器と比較すると、外部から投入される電気やガスなどのエネルギーが大きい。従って、計算を行う際にはこれらの投入エネルギーを含めて熱（エネルギー）収支をとることが大切である。ボイラは、電気ではなく都市ガスや LNG などの燃料を投入するため、燃料が持つエネルギーを正しく計算する必要がある。この計算は第 15 章で解説する吸収式冷凍機の計算においても必要となる。また、機器やシステムの効率を一次エネルギーで評価するための基礎ともなる。本章では上記の 2 種類のボイラを題材に、燃料、空気、水（蒸気）のエネルギー収支をとるモデルの作成法を学ぶ。

写真 13.1　虎ノ門の工学寮（後の工部大学校）
明治 6 年に我が国初の蒸気暖房設備が取り付けられた
（*Wikipedia, public domain*）

13.2 理論

13.2.1 基礎式

1）熱収支

ボイラには温水を製造する温水ボイラと蒸気を製造する蒸気ボイラがあるが、熱の源を外部に頼らず、燃料を直接燃焼させてエネルギーを得るという点においては同じである[†1]。従って、モデルの作成にあたっては両機器の熱収支を同様に捉えても大きな誤りは生じない。ボイラの熱収支を図 13.1 に示す。式 13.1 に示す熱収支式が成立する。

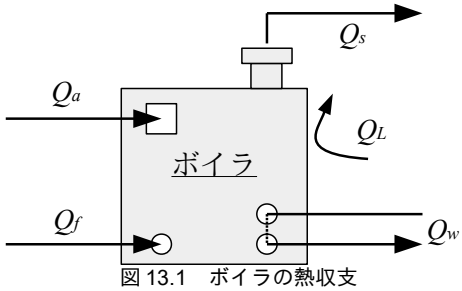


図 13.1　ボイラの熱収支

$$Q_a + Q_f = Q_s + Q_L + Q_w \tag{13.1}$$

Q_w [kW]は熱媒（蒸気または温水）の流出入熱の差分であり、ボイラによって熱媒に与えられた熱である。この他にボイラに流入する熱としては、燃焼空気自体が持つ熱 Q_a [kW]と燃料が持つ熱 Q_f [kW]がある。流出する熱としては、煙突から排気ガスとして排出される熱 Q_s [kW]とボイラの缶体から漏洩する熱損失 Q_L [kW]がある。以下、各々について計算法を記す。

まず Q_w に関しては、温水と蒸気とでそれぞれ式 13.2 と 13.3 で計算ができる。温水の場合には、出入口温度差に水の定圧比熱 c_{pw} [kJ/(kg·K)]および質量流量 m_w [kg/s]を乗じる。蒸気の場合には入口温水（還水と呼ばれる）の比エンタルピー $h_{w,i}$ [kJ/kg]と飽和蒸気の比エンタルピー h_{Wsv} [kJ/kg]との差に対して質量流量 m_w を乗じる。水および水蒸気の比エンタルピーの計算法に関しては第 4 章で記した。なお蒸気の状態は、熱量計算上は比エンタルピーで捉えるが、熱源の設計や制御上は飽和圧力 P_{Wsv} [kPa]で表現することが通常である。

$$Q_w = c_{pw} m_w \left(t_{w,o} - t_{w,i}\right) = m_w \left(h_{w,o} - h_{w,i}\right) \quad \text{（温水の場合）} \tag{13.2}$$

$$Q_w = m_w \left(h_{Wsv} - h_{w,i}\right) \quad \text{（蒸気の場合）} \tag{13.3}$$

燃焼空気自体が持つ熱 Q_a は、空気の質量流量 m_a [kg/s]に乾き空気の比熱 c_{pa} [kJ/(kg·K)]と乾球温度 $t_{a,i}$ [°C]を乗じ、式 13.4 で計算する。水蒸気が考慮されていないが、後述するように排気ガスの比熱においても空気中の水分の影響は軽微として考慮しておらず、整合している。空気予熱器付ボイラ[†2]でない場合には入口空気温度 $t_{a,i}$ は周囲温度 t_a [°C]と等しいとしてもよい。

$$Q_a = c_{pa} m_a t_{a,i} \tag{13.4}$$

空気の質量流量 m_a は、燃料の投入量 m_f、燃料を完全燃焼させるために必要な理論空気量 k、空気比 λ [-]によって式 13.5 で計算できる。ここで、ρ_a [kg/m^3]は空気の比重量である。

†1　これに対してヒートポンプと呼ばれる機器は、投入するエネルギーを駆動力にして外部から熱を奪うことで加熱を行う。即ち、投入するエネルギー自体が加熱に用いられるわけではない。なお後章の「吸収式冷凍機」で触れるが、吸収冷温水機と呼ばれる機械も温水製造時には燃料の直接燃焼という方法をとるため、熱収支としてはボイラと同様に捉えることができる。

†2　熱損失の抑制を目的に排気ガスから燃焼空気に熱を移動させる機構を持ったボイラである。

$$m_a = \rho_a m_f k \lambda \tag{13.5}$$

代表的な燃料の理論空気量 k、高位発熱量 HV_h、低位発熱量 HV_l を表 13.1 に示す。基準単位が、気体燃料に関しては Nm^3、固体および液体燃料に関しては kg となることに注意する。

表 13.1　燃料の理論空気量[13.3]、高位発熱量[13.4]、低位発熱量[†1]

燃料	13A	LNG	LPG	石炭
理論空気量 k	10.949 Nm^3/Nm^3	13.093 Nm^3/kg	12.045 Nm^3/kg	7.800 Nm^3/kg
高位発熱量 HV_h	45.6 MJ/Nm^3	54.6 MJ/kg	50.8 MJ/kg	29.0 MJ/kg
低位発熱量 HV_l	41.0 MJ/Nm^3	49.1 MJ/kg	45.7 MJ/kg	27.6 MJ/kg

空気比に関してはエネルギーの使用の合理化に関する法律[13.2]（以下「省エネ法」）によって運用上の管理標準値が定められており 1.1~1.3 程度の数値となる[†2]。

燃料が持つ熱 Q_fは、燃料の顕熱と燃料の燃焼によって得られる高位発熱量 HV_h [kJ/kg]の合計であるが、燃焼によるエネルギーに比較して顕熱分は微小であるため[†3]、顕熱分を無視して式 13.6 で計算する。なお、燃料別の高位発熱量は表 13.1 に記した。

$$Q_f = 1000\, m_f HV_h \tag{13.6}$$

燃料の種類によって排気ガスの組成は異なるため、比熱 c_{ps} [kJ/(m^3·K)]の値も厳密には異なるが、空気比を 1.05~1.30、排ガス温度を 120~250°C としてケーススタディを行った結果からは、固体、液体、気体燃料共に平均値として 1.38 kJ/(m^3·K)を採用しても問題ないという結果が得られている[13.5]。また、空気中の湿分の影響による誤差も 0.6%を上回ることは稀であることから、比熱の設定にあたっては考慮外としている。従って、煙突から排出される排気ガスによる熱流出量 Q_s は、排ガスの比熱 c_{ps} を用いて、式 13.7 で計算する。ただし、式 13.7 は 0°C を基準としており、燃料および燃焼用空気の比エンタルピーの基準温度も 0°C に合わせる必要がある。r_{sf}は単位燃料の燃焼によって生まれる排ガスの体積であり、燃料の種別、空気比、低位発熱量により式 13.8 で計算できる[13.5] [13.6]。HV_l は低位発熱量であり、燃料別の値は表 13.1 に記した。r_{sf}と HV_lの単位は、燃料が気体の場合は「MJ/Nm^3」、燃料が固体あるいは液体の場合は「MJ/kg」である。また、係数 A_{sf}, B_{sf}, C_{sf}, D_{sf}は燃料が液体か固体か気体かによって異なり、表 13.2 に値を示す。

$$Q_s = c_{ps} m_f r_{sf} t_s \tag{13.7}$$

$$r_{sf} = \left[A_{sf} + B_{sf}(\lambda - 1)\right] HV_l + C_{sf} + D_{sf}(\lambda - 1) \tag{13.8}$$

表 13.2　排ガス量推定式の係数

係数	A_{sf}	B_{sf}	C_{sf}	D_{sf}
固体	0.0216	0.0241	1.67	0.56
液体	0.376	0.296	-3.91	-1.36
気体	0.293	0.268	0	0

ボイラの缶体は高温な部分、低温な部分が混在しており、漏洩する熱損失量 Q_L を積み上げにより計算することは困難である。そこで、簡易化の為に、缶体からの漏洩熱は製造する温水または蒸気の温度と周囲温度との差に熱損失係数 U_L [W/K]を乗じて式 13.9 または式 13.10 で計算することとし、熱

†1　石炭に関しては高位発熱量×0.95、その他に関しては高位発熱量×0.9 として計算を行った。

†2　完全燃焼に必要な空気量を 1 とした場合の、実際に投入した空気量の比を空気比と呼ぶ。空気比が 1 を下回ると不完全燃焼が生じるため、実運用においては 1 を上回る値で管理する。一方で、式 13.4 から明らかなように、過剰な空気は排ガスとして熱損失を増やすため、出来る限り小さな値に抑える方がボイラ効率は高くなる。

†3　都市ガス 13A を例に取ると、主成分であるメタンの比熱は常温で約 2.2kJ/(kg·K)であるため、25°C までの温度変化による熱量は 2.2kJ/(kg·K)×25°C=55kJ/kg である。一方で、高位発熱量は 45.6MJ/kg であるため、比重 0.638 kg/m^3 を乗じて、燃焼による熱量は 29 MJ/kg となる。従って顕熱を無視することによる誤差は 55÷29,000=0.2%程度となる。

損失係数 U_L は定格性能値から逆算する。ここで t_{Wsv} [°C]は飽和蒸気の飽和温度である。

$$Q_L = U_L\left(t_{w,o} - t_a\right) \quad \text{（温水の場合）} \tag{13.9}$$

$$Q_L = U_L\left(t_{Wsv} - t_a\right) \quad \text{（蒸気の場合）} \tag{13.10}$$

設備システムの性能評価にあたっては、現地で計測可能な熱である Q_f と Q_w の比を用いて式 13.11 でボイラ効率 COP_B [-]を評価することが多い。一方で、機器カタログ上のボイラ効率は JIS 規格にもとづく値であり、定義が異なることに注意する必要がある。

$$COP_B = Q_w / Q_f \tag{13.11}$$

2）ブロー水量

蒸気ボイラの場合、蒸発と凝縮が繰り返されることで水中のカルシウム濃度などが高くなり、ボイラや配管などに悪影響をおよぼす。これを回避するため、新しい水を補給して還水の一部を排水する。この処理をブローと呼ぶ。通常は補給される水の温度は 20~30 °C 程度の常温であり、排水する還水に比較して温度が低いため、ブロー処理による熱損失が発生する。この熱損失 $Q_{L,b}$ [kW]は式 13.12 で計算する。t_s [°C]は給水温度であるが周囲温度 t_a と等しいとしても良いだろう。m_b [kg/s]はブロー水量であり、設計段階では循環水量の 5 %程度とし、運用段階で水質を確認しながら確定させる。ただし、ブロー処理はボイラ自体から直接に行うのではなく、通常は還水槽への給水および排水などの手段で行うため、シミュレーションにおいてもボイラモデルではなくシステムモデルの中で考慮する方がわかりやすいだろう。

$$Q_{L,b} = m_b\left(t_{w,i} - t_s\right) \tag{13.12}$$

13.2.2　燃料消費量・出口温度・蒸気流量の関係

建築設備のエネルギーシミュレーションにおいては、達成したい温水の出口温度や蒸気の圧力（ないしは製造熱量）が入力条件として与えられ、これに対して必要な燃料消費を計算することが多い。式 13.2~式 13.10 を式 13.1 に代入して燃料消費 m_f について整理すると、式 13.13 が得られる。ただし、Q_w は温水または蒸気の別に応じて式 13.2 または式 13.3 で計算する。

$$m_f = \frac{U_L\left(t_{w,o} - t_a\right) + Q_w}{\rho_a k \lambda c_{pa} t_{a,i} + 1000 H V_h - c_{ps} t_s r_{sf}} \tag{13.13}$$

式 13.13 で入力条件として得ることができない変数は排ガスの温度 t_s [°C]であるが、これに関しては、製造する温水の温度および負荷率の関数として特性式で表現することもできる。製造者へのヒアリングにより作成した、排ガス温度特性を図 13.2 に示す。

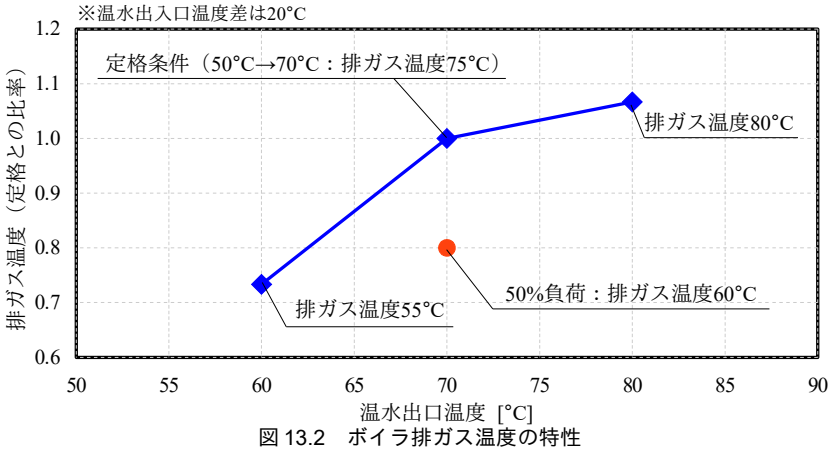


図 13.2　ボイラ排ガス温度の特性

温水出口温度の低下、負荷率の低下に伴い、排ガス温度が低下する傾向が確認できる。式 13.14 に示す通り、温水出口温度を排ガス温度で表現することにする。$t_{s,N}$ [°C]と $t_{w,o,N}$ [°C]はそれぞれ定格の排ガス温度と温水出口温度である。A_{st} と B_{st} は回帰係数で、データ数は少ないが図 13.2 にもとづいて求めると A_{st} = 1.17、B_{st} = -0.17 となる。

$$(t_s/t_{s,N}) = A_{ts}(t_{w,o}/t_{w,o,N}) + B_{ts} \tag{13.14}$$

省エネ法では排ガスの基準温度が定めれており、200 °C 前後の値であるため、カタログ等に定格の排ガス温度の情報が無い場合にはこの程度の値を設定する。なお、図 13.2 は潜熱回収ボイラの例であるため、やや温度が低い[†1]。

投入した燃料に比較して負荷が大きすぎる場合には、温水ボイラは出口温度を設定値に維持することができない。式 13.2~式 13.9 を式 13.1 に代入し、出口水温 $t_{w,o}$ について解くと式 13.15 が得られる。一方、蒸気ボイラが過負荷の場合には製造蒸気量が低下する。このときの蒸気量は式 13.3 を式 13.1 に代入して式 13.16 で計算する。

$$t_{w,o} = \frac{m_f\left(\rho_a k \lambda c_{pa} t_{a,i} + 1000\, HV_h + c_{ps} t_{s,N} r_{sf} B_{st}\right) + U_L t_a + c_{pw} m_W t_{w,i}}{c_{pw} m_w + U_L + A_{st} c_{ps} r_{sf} m_f t_{s,N}/t_{w,o,N}} \tag{13.15}$$

$$m_w = \frac{Q_a + Q_f - Q_s - Q_L}{h_{Wsv} - h_{w,i}} \tag{13.16}$$

【例題 13.1】

以下の定格性能を持つ温水ボイラについて熱損失係数 U_L を推定せよ。

温水入口温度：50 °C　　温水出口温度：70 °C　　温水流量：710L/min

燃料種別：都市ガス 13A　　燃料消費量：83.6 Nm³/h　　排気ガス温度：80 °C

ただし、空気比 λ は 1.15、缶体を囲む空気の乾球温度は 25 °C とする。

【解】

定格加熱量は、温水出入口温度および水量を用いて、式 13.2 により、

Q_w = 4.186 × 710 / 60 × (70 – 50) = 990 kW

である。また、燃料は都市ガス 13A であるため理論空気量は 10.949 Nm³/Nm³ であり、燃焼空気の量は式 13.5 により、

m_a = 1.2 × 83.6 / 3,600 × 1.15 × 10.949 = 0.351 kg/s

と求められる。燃焼空気の熱量は、式 13.4 により、

Q_a = 1.005 × 0.351 × 25 = 8.8 kW

となる。都市ガス 13A の高位発熱量は 45.6 MJ/Nm³ であるため、燃料燃焼によって得られる熱量は、式 13.6 により、

Q_f = 83.6 / 3,600 × 45.6 × 1,000 = 1,059 kW

となる。単位燃料あたりの排ガス量は、式 13.8 により、

r_{sf} = 0.293 + 0.268 × (1.15 – 1) × 41.0 = 13.7 kg/Nm³

となる。従って、排ガスによる熱損失量は、式 13.7 により、

Q_s = 1.38 × (83.6 / 3,600 × 13.7) × 80 = 35.1 kW

となる。以上により、Q_w, Q_a, Q_f, Q_s が求められたため、式 13.1 により、

Q_L = 8.8 + 1,059 – 990 – 35.1 = 42.7 kW

となる。式 13.9 を用いて、熱損失係数は

U_L = 42.7 ÷ (70 – 25) = 0.95 kW/K

と求められる。

†1　排ガスとボイラ入口温水との間に伝熱性能の高い熱交換器を設けることで、排ガス内の水分が凝縮するレベルまで熱回収を行う機器を潜熱回収ボイラと呼ぶ。

13.3 計算法

温水ボイラおよび蒸気ボイラに共通する処理について静的クラス（Boiler class）を定義した後、これらを用いて温水ボイラクラス（HotWaterBoiler class）および蒸気ボイラクラス（SteamBoiler class）を作成する方法を示す。

13.3.1 「ボイラクラス」の作成

ボイラクラスの定数宣言および列挙型の定義をプログラム 13.1 に示す。空気の比熱は乾き空気の比熱とする点に注意する。

プログラム 13.1 ボイラクラスの定数宣言および列挙型の定義

Popolo.HVAC.HeatSource.Boiler class

```
/// <summary>空気の比熱[kJ/(kgK)]</summary>
private const double AIR_SPECIFIC_HEAT = 1.006;

/// <summary>排ガスの比熱[kJ/(kgK)]</summary>
private const double SMOKE_SPECIFIC_HEAT = 1.38;

/// <summary>排ガス温度近似係数 1</summary>
private const double A_ST = 1.17;

/// <summary>排ガス温度近似係数 2</summary>
private const double B_ST = -0.17;

/// <summary>燃料種別</summary>
public enum Fuel
{
  /// <summary>都市ガス 13A</summary>
  Gas13A,
  /// <summary>LNG</summary>
  LNG,
  /// <summary>LPG</summary>
  LPG,
  /// <summary>石炭</summary>
  Coal
}
```

排ガス熱量・発熱量・理論空気量の計算処理をプログラム 13.2 に示す。表 13.1 および表 13.2 に示したとおり、いずれも燃料の種類によって値が異なるため、プログラム 13.1 に定義した列挙型を引数にして条件分岐を行う。

プログラム 13.2 排ガス熱量・発熱量・理論空気量の計算

Popolo.HVAC.HeatSource.Boiler class

```
/// <summary>排ガス熱量計算のための係数を計算する</summary>
/// <param name="airRatio">空気比[-]</param>
/// <param name="fuel">燃料の種類</param>
/// <returns>排ガス熱量計算のための係数</returns>
private static double getSmokeCoefficient(double airRatio, Fuel fuel)
{
  double am1 = airRatio - 1d;
  switch (fuel)
  {
    case Fuel.Gas13A: return (0.293 + 0.268 * am1) * 41d;
    case Fuel.LNG: return (0.376 + 0.296 * am1) * 49.1d - 3.91 - 1.36 * am1;
    case Fuel.LPG: return (0.376 + 0.296 * am1) * 45.7d - 3.91 - 1.36 * am1;
    default: return (0.0216 + 0.0241 * am1) * 27.6d + 1.67 + 0.56 * am1;  //石炭
  }
}

/// <summary>燃料の発熱量[MJ/Nm3, MJ/kg]を取得する</summary>
/// <param name="fuel">燃料の種類</param>
/// <param name="isHighValue">高位発熱量か否か</param>
/// <returns>燃料の発熱量[MJ/Nm3, MJ/kg]</returns>
public static double GetHeatingValue(Fuel fuel, bool isHighValue)
{
  switch (fuel)
  {
    case Fuel.Gas13A:
      if (isHighValue) return 45.6;
      else return 41.0;
```

```
    case Fuel.LNG:
      if (isHighValue) return 54.6;
      else return 49.1;
    case Fuel.LPG:
      if (isHighValue) return 50.8;
      else return 45.7;
    default:  //石炭
      if (isHighValue) return 29.0;
      else return 27.6;
  }
}

/// <summary>理論空気量[Nm3/Nm3,Nm3/kg]を取得する</summary>
/// <param name="fuel">燃料の種類</param>
/// <returns>理論空気量[Nm3/Nm3,Nm3/kg]</returns>
public static double GetTheoreticalAir(Fuel fuel)
{
  switch (fuel)
  {
    case Fuel.Gas13A: return 10.949;
    case Fuel.LNG: return 13.093;
    case Fuel.LPG: return 12.045;
    default: return 7.8;  //石炭
  }
}
```

熱損失係数の計算処理をプログラム 13.3 に示す。例題 13.1 の実装である。

プログラム 13.3　熱損失係数の計算処理

Popolo.HVAC.HeatSource.Boiler class

```
/// <summary>熱損失係数[kW/K]を計算する</summary>
/// <param name="heatLoad">加熱量[kW]</param>
/// <param name="outletWaterTemperature">水（蒸気）の出口温度[C]</param>
/// <param name="inletAirTemperature">入口空気温度[C]</param>
/// <param name="fuel">燃料の種類</param>
/// <param name="smokeTemperature">排ガス温度[C]</param>
/// <param name="airRatio">空気比[-]</param>
/// <param name="fuelConsumption">料消費量[kg/s, Nm3/s]</param>
/// <param name="ambientTemperature">周囲の温度[C]</param>
/// <returns>熱損失係数[kW/K]</returns>
public static double GetHeatLossCoefficient
  (double heatLoad, double outletWaterTemperature, double inletAirTemperature, Fuel fuel,
  double smokeTemperature, double airRatio, double fuelConsumption, double ambientTemperature)
{
  double airHeat = AIR_DENSITY * fuelConsumption * GetTheoreticalAir(fuel) * airRatio
    * AIR_SPECIFIC_HEAT * inletAirTemperature;
  double fuelHeat = fuelConsumption * GetHeatingValue(fuel, true) * 1000;
  double smokeHeat = SMOKE_SPECIFIC_HEAT * fuelConsumption
    * getSmokeCoefficient(airRatio, fuel) * smokeTemperature;
  double heatLoss = airHeat + fuelHeat - heatLoad - smokeHeat;

  return heatLoss / (outletWaterTemperature - ambientTemperature);
}
```

プログラム 13.4 に燃料消費量の計算処理を示す。式 13.13 の実装であり、引数として加熱負荷 Q_w を受け取る仕様とする。

プログラム 13.4　燃料消費量の計算

Popolo.HVAC.HeatSource.Boiler class

```
/// <summary>燃料消費量[kg/s, Nm3/s]を計算する</summary>
/// <param name="heatLoad">加熱負荷[kW]</param>
/// <param name="outletWaterTemperature">水（蒸気）の出口温度[C]</param>
/// <param name="inletAirTemperature">入口空気温度[C]</param>
/// <param name="fuel">燃料の種類</param>
/// <param name="nominalSmokeTemperature">定格排ガス温度[C]</param>
/// <param name="airRatio">空気比[-]</param>
/// <param name="heatLossCoefficient">熱損失係数[kW/K]</param>
/// <param name="ambientTemperature">周囲の温度[C]</param>
/// <param name="nominalOutletWaterTemperature">定格出口水温[C]</param>
/// <returns>燃料消費量[kg/s, Nm3/s]</returns>
public static double GetFuelConsumption
  (double heatLoad, double outletWaterTemperature, double inletAirTemperature, Fuel fuel,
  double nominalSmokeTemperature, double airRatio, double heatLossCoefficient,
  double ambientTemperature, double nominalOutletWaterTemperature)
{
  double smokeTemp = nominalSmokeTemperature
    * (A_ST * outletWaterTemperature / nominalOutletWaterTemperature + B_ST);
  double m1 = heatLoad + heatLossCoefficient * (outletWaterTemperature - ambientTemperature);
```

```
20   double m2 = AIR_DENSITY * GetTheoreticalAir(fuel) * airRatio * AIR_SPECIFIC_HEAT * inletAirTemperature
21     + 1000d * GetHeatingValue(fuel, true);
22   m2 -= SMOKE_SPECIFIC_HEAT * smokeTemp * getSmokeCoefficient(airRatio, fuel);
23
24   return m1 / m2;
25 }
```

燃料消費量が与えられた場合の計算処理をプログラム 13.5 に示す。1~30 行は温水ボイラの出口冷水温度を求める処理であり、式 13.15 の実装である。32~61 行は蒸気ボイラの蒸気流量を求める処理であり、式 13.16 の実装である。

プログラム 13.5　燃料消費量が与えられた場合の計算

Popolo.HVAC.HeatSource.Boiler class

```
 1 /// <summary>出口状態を計算する</summary>
 2 /// <param name="inletWaterTemperature">入口水温[C]</param>
 3 /// <param name="waterFlowRate">水量[kg/s]</param>
 4 /// <param name="fuelConsumption">燃料消費量[Nm3/s, kg/s]</param>
 5 /// <param name="inletAirTemperature">入口空気温度[C]</param>
 6 /// <param name="fuel">燃料の種類</param>
 7 /// <param name="nominalSmokeTemperature">定格排ガス温度[C]</param>
 8 /// <param name="airRatio">空気比[-]</param>
 9 /// <param name="heatLossCoefficient">熱損失係数[kW/K]</param>
10 /// <param name="ambientTemperature">周囲の温度[C]</param>
11 /// <param name="nominalOutletWaterTemperature">定格出口水温[C]</param>
12 /// <param name="outletWaterTemperature">出口水温[C]</param>
13 /// <param name="heatLoad">加熱量[kW]</param>
14 public static void GetOutletWaterTemperature
15   (double inletWaterTemperature, double waterFlowRate, double fuelConsumption, double inletAirTemperature,
16   Fuel fuel, double nominalSmokeTemperature, double airRatio, double heatLossCoefficient,
17   double ambientTemperature, double nominalOutletWaterTemperature,
18   out double outletWaterTemperature, out double heatLoad)
19 {
20   double rf = getSmokeCoefficient(airRatio, fuel);
21   double bf1 = AIR_DENSITY * GetTheoreticalAir(fuel) * airRatio * AIR_SPECIFIC_HEAT * inletAirTemperature;
22   bf1 += 1000d * GetHeatingValue(fuel, true);
23   bf1 -= SMOKE_SPECIFIC_HEAT * nominalSmokeTemperature * rf * B_ST;
24   bf1 *= fuelConsumption;
25   bf1 += heatLossCoefficient *ambientTemperature + waterFlowRate* WATER_SPECIFIC_HEAT* inletWaterTemperature;
26   double bf2 = (waterFlowRate * WATER_SPECIFIC_HEAT + heatLossCoefficient);
27   bf2 +=A_ST* SMOKE_SPECIFIC_HEAT* rf* fuelConsumption*nominalSmokeTemperature/nominalOutletWaterTemperature;
28   outletWaterTemperature = bf1 / bf2;
29   heatLoad = WATER_SPECIFIC_HEAT * waterFlowRate * (outletWaterTemperature - inletWaterTemperature);
30 }
31
32 /// <summary>蒸気流量[kg/s]を計算する</summary>
33 /// <param name="inletWaterTemperature">入口水温[C]</param>
34 /// <param name="steamPressure">蒸気圧力[kPa]</param>
35 /// <param name="fuelConsumption">燃料消費量[Nm3/s, kg/s]</param>
36 /// <param name="inletAirTemperature">入口空気温度[C]</param>
37 /// <param name="fuel">燃料の種類</param>
38 /// <param name="nominalSmokeTemperature">定格排ガス温度[C]</param>
39 /// <param name="airRatio">空気比[-]</param>
40 /// <param name="heatLossCoefficient">熱損失係数[kW/K]</param>
41 /// <param name="ambientTemperature">周囲の温度[C]</param>
42 /// <param name="nominalSteamTemperature">定格蒸気温度[C]</param>
43 /// <param name="steamFlowRate">出力：蒸気流量[kg/s]</param>
44 /// <param name="heatLoad">出力：加熱量[kW]</param>
45 public static void GetSteamFlowRate
46   (double inletWaterTemperature, double steamPressure, double fuelConsumption, double inletAirTemperature,
47   Fuel fuel, double nominalSmokeTemperature, double airRatio, double heatLossCoefficient,
48   double ambientTemperature, double nominalSteamTemperature, out double steamFlowRate, out double heatLoad)
49 {
50   double rf = getSmokeCoefficient(airRatio, fuel);
51   double twsv = Water.GetSaturationTemperature(steamPressure);
52   double hwsv = Water.GetSaturatedVaporEnthalpy(twsv);
53   double hwi = inletWaterTemperature * WATER_SPECIFIC_HEAT;
54   double qa = AIR_DENSITY * GetTheoreticalAir(fuel) * airRatio * AIR_SPECIFIC_HEAT * inletAirTemperature;
55   double qf = 1000d * GetHeatingValue(fuel, true);
56   double qs = SMOKE_SPECIFIC_HEAT* rf* nominalSmokeTemperature* (A_ST* twsv/ nominalSteamTemperature + B_ST);
57   double ql = heatLossCoefficient * (twsv - ambientTemperature);
58
59   heatLoad = (qa + qf - qs) * fuelConsumption - ql;
60   steamFlowRate = heatLoad / (hwsv - hwi);
61 }
```

13.3.2　「温水ボイラクラス」の作成

プログラム 13.6 にインスタンス変数およびプロパティの定義を示す。14 行は電力の一次エネルギー換算係数であり、ボイラの補機類の電力消費量を含めたエネルギー消費量にもとづいて COP（62~72 行）を計算するためのものである。

プログラム 13.6　インスタンス変数およびプロパティの定義

Popolo.HVAC.HeatSource.HotWaterBoiler class

```
 1 /// <summary>空気比[-]</summary>
 2 private double airRatio = 1.1;
 3
 4 /// <summary>熱損失係数[kW/K]</summary>
 5 private double heatLossCoefficient = 0.0;
 6
 7 /// <summary>定格出口水温[C]</summary>
 8 private double nomOutletWaterTemperature;
 9
10 /// <summary>定格排ガス温度[C]</summary>
11 private double nominalSmokeTemperature;
12
13 /// <summary>電力の一次エネルギー換算係数[MJ/kWh]を設定・取得する</summary>
14 public double PrimaryEnergyFactor { get; set; } = 9.76
15
16 /// <summary>燃料種別を取得する</summary>
17 public Boiler.Fuel Fuel { get; private set;}
18
19 /// <summary>温水出口温度[C]を取得する</summary>
20 public double OutletWaterTemperature { get; private set; }
21
22 /// <summary>温水出口温度設定値[C]を設定・取得する</summary>
23 public double OutletWaterSetPointTemperature { get; set; }
24
25 /// <summary>温水入口温度[C]を取得する</summary>
26 public double InletWaterTemperature { get; private set; }
27
28 /// <summary>温水流量[kg/s]を取得する</summary>
29 public double WaterFlowRate { get; private set; }
30
31 /// <summary>温水上限流量[kg/s]を取得する</summary>
32 public double MaxWaterFlowRate { get; private set; }
33
34 /// <summary>温水下限流量比[-]を設定・取得する</summary>
35 public double MinWaterFlowRatio { get; set; } = 0.2;
36
37 /// <summary>定格加熱能力[kW]を取得する</summary>
38 public double NominalCapacity { get; private set; }
39
40 /// <summary>定格燃料消費量[kg/s, Nm3/s]を取得する</summary>
41 public double NominalFuelConsumption { get; private set; }
42
43 /// <summary>消費電力[kW]を取得する</summary>
44 public double ElectricConsumption { get; private set; }
45
46 /// <summary>燃料消費量[kg/s, Nm3/s]を取得する</summary>
47 public double FuelConsumption { get; private set; }
48
49 /// <summary>暖房能力[kW]を取得する</summary>
50 public double HeatLoad { get; private set; }
51
52 /// <summary>周囲の温度[C]を設定・取得する</summary>
53 public double AmbientTemperature { get; set; }
54
55 /// <summary>空気比を設定・取得する</summary>
56 public double AirRatio
57 {
58   get { return airRatio; }
59   set { airRatio = Math.Max(1.0, Math.Min(1.5, value)); }
60 }
61
62 /// <summary>COP[-]を取得する</summary>
63 public double COP
64 {
65   get
66   {
67     if (HeatLoad == 0) return 0;
68     else
69       return 0.001 * HeatLoad / (ElectricConsumption * PrimaryEnergyFactor
```

```
70         + FuelConsumption * Boiler.GetHeatingValue(Fuel, true));
71   }
72 }
73
74 /// <summary>過負荷か否か</summary>
75 public bool IsOverLoad { get; private set; }
```

プログラム 13.7 にコンストラクタを示す。定格性能値を引数とし、16~25 行で定格性能をプロパティに保存する。27~30 行でプログラム 13.3 を用いて定格性能から熱損失係数を推定し、インスタンス変数に保存する。33 行は運転停止処理であり、36~41 行に示すように、出口温度=入口温度とた上でエネルギー消費量や流量を 0 に設定する。

プログラム 13.7　コンストラクタ

Popolo.HVAC.HeatSource.HotWaterBoiler class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="inletWaterTemperature">入口温水温度[C]</param>
 3 /// <param name="outletWaterTemperature">出口温水温度[C]</param>
 4 /// <param name="waterFlowRate">温水流量[kg/s]</param>
 5 /// <param name="fuelConsumption">燃料消費量[kg/s, Nm3/s]</param>
 6 /// <param name="electricConsumption">消費電力[kW]</param>
 7 /// <param name="ambientTemperature">周囲の温度[C]</param>
 8 /// <param name="airRatio">空気比[-]</param>
 9 /// <param name="fuel">燃料種別</param>
10 /// <param name="smokeTemperature">排ガス温度[C]</param>
11 public HotWaterBoiler
12   (double inletWaterTemperature, double outletWaterTemperature, double waterFlowRate,
13   double fuelConsumption, double electricConsumption, double ambientTemperature,
14   double airRatio, Boiler.Fuel fuel, double smokeTemperature)
15 {
16   //プロパティ保存
17   Fuel = fuel;
18   AirRatio = airRatio;
19   nominalSmokeTemperature = smokeTemperature;
20   NominalFuelConsumption = fuelConsumption;
21   ElectricConsumption = electricConsumption;
22   AmbientTemperature = ambientTemperature;
23   OutletWaterSetPointTemperature = nomOutletWaterTemperature = outletWaterTemperature;
24   MaxWaterFlowRate  = WaterFlowRate = waterFlowRate;
25   NominalCapacity = (outletWaterTemperature - inletWaterTemperature) * WaterFlowRate * WATER_SPECIFIC_HEAT;
26
27   //熱損失係数の計算
28   heatLossCoefficient = Boiler.GetHeatLossCoefficient
29     (NominalCapacity, outletWaterTemperature, AmbientTemperature, fuel,
30     nominalSmokeTemperature, AirRatio, NominalFuelConsumption, AmbientTemperature);
31
32   //停止させる
33   ShutOff();
34 }
35
36 /// <summary>停止する</summary>
37 public void ShutOff()
38 {
39   OutletWaterTemperature = InletWaterTemperature;
40   ElectricConsumption = WaterFlowRate = HeatLoad = FuelConsumption = 0;
41 }
```

プログラム 13.8 に状態更新処理を示す。引数は入口水温と水量のみであり、その他の情報はプロパティやインスタンス変数を参照する。まず、14~16 行でプログラム 13.4 を用いて必要燃料消費量を計算する。最大（定格）燃料消費量よりも大きい場合には 20 行で条件分岐を行い、最大燃料消費量で成り行き計算を行う（22~27 行）。この場合には出口温度が設定温度よりも低下することになる。

プログラム 13.8　状態更新処理

Popolo.HVAC.HeatSource.HotWaterBoiler class

```
 1 /// <summary>状態を更新する</summary>
 2 /// <param name="inletWaterTemperature">入口水温[C]</param>
 3 /// <param name="waterFlowRate">水量[kg/s]</param>
 4 public void Update(double inletWaterTemperature, double waterFlowRate)
 5 {
 6   //入力条件保存
 7   InletWaterTemperature = inletWaterTemperature;
 8   WaterFlowRate = waterFlowRate;
```

```
 9
10   //流量 0、設定温度<入口温度で停止
11   if (WaterFlowRate <= 0 || OutletWaterSetPointTemperature < InletWaterTemperature) ShutOff();
12
13   //燃料消費量を計算
14   double hl = WATER_SPECIFIC_HEAT * waterFlowRate * (OutletWaterSetPointTemperature - inletWaterTemperature);
15   FuelConsumption = Boiler.GetFuelConsumption(hl, OutletWaterSetPointTemperature, AmbientTemperature, Fuel,
16     nominalSmokeTemperature, airRatio, heatLossCoefficient, AmbientTemperature, nomOutletWaterTemperature);
17
18   //過負荷の場合には成り行き計算
19   IsOverLoad = NominalFuelConsumption < FuelConsumption;
20   if (IsOverLoad)
21   {
22     FuelConsumption = NominalFuelConsumption;
23     double to;
24     Boiler.GetOutletWaterTemperature(InletWaterTemperature, WaterFlowRate,
25       NominalFuelConsumption, AmbientTemperature, Fuel, nominalSmokeTemperature,
26       AirRatio, heatLossCoefficient, AmbientTemperature, nomOutletWaterTemperature, out to, out hl);
27     OutletWaterTemperature = to;
28   }
29   else OutletWaterTemperature = OutletWaterSetPointTemperature;
30   HeatLoad = hl;
31 }
```

【例題 13.2】

例題 13.1 の温水ボイラについて、負荷率を変化させた場合のボイラ効率を計算せよ。ただし、空気比は 1.1, 1.2, 1.3 の 3 通り、温水出入口温度は 40 → 60°C, 50 → 70°C, 60 → 80°C の 3 通りとする。

【解】

プログラム 13.9 に計算方法を示す。3,4 行で温水ボイラのインスタンスを作成する。11~33 行が負荷率のループである。17~23 行で空気比を変化せた場合の COP を、25~31 行で温水出入口温度を変化させた場合の COP を計算する。計算結果をグラフにすると図 13.3 が得られる。COP に与える影響の大きさは空気比より出口温度設定値の方が大きい。

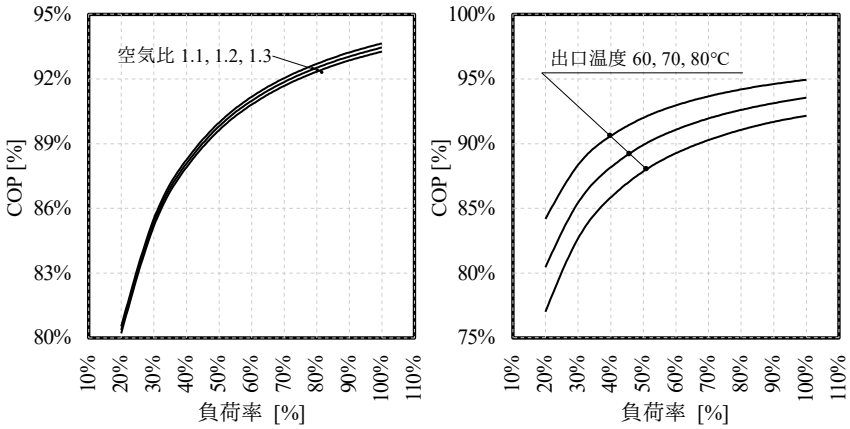


図 13.3　ボイラ負荷率とボイラ効率の関係（空気比・温水出口温度別）

プログラム 13.9　ボイラ負荷率とボイラ効率の関係の計算（空気比・温水出口温度別）

```
 1 private static void HotWaterBoilerTest()
 2 {
 3   HotWaterBoiler hBoiler = new HotWaterBoiler
 4     (50, 70, 710d / 60, 83.6 / 3600, 0, 25, 1.15, Boiler.Fuel.Gas13A, 80);
 5
 6   using (StreamWriter sWriter = new StreamWriter
 7     ("HotWaterBoilerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
 8   {
 9     sWriter.WriteLine
10       ("負荷率, 空気比 1.1, 空気比 1.2, 空気比 1.3, 40->60C, 50->70C, 60->80C");
11     for (int i = 0; i < 10; i++)
12     {
13       double pl = 1 - 0.1 * i;
14       sWriter.Write(pl);
15       double wFlow = (710d / 60) * pl;
16
17       hBoiler.OutletWaterSetPointTemperature = 70;
18       for (int j = 0; j < 3; j++)
19       {
20         hBoiler.AirRatio = 1.1 + 0.1 * j;
```

```
21          hBoiler.Update(50, wFlow);
22          sWriter.Write("," + hBoiler.COP.ToString("F4"));
23        }
24
25        hBoiler.AirRatio = 1.15;
26        for (int j = 0; j < 3; j++)
27        {
28          hBoiler.OutletWaterSetPointTemperature = 60 + j * 10;
29          hBoiler.Update(40 + j * 10, wFlow);
30          sWriter.Write("," + hBoiler.COP.ToString("F4"));
31        }
32        sWriter.WriteLine();
33      }
34    }
35 }
```

13.3.3 「蒸気ボイラクラス」の作成

プログラム 13.10 にインスタンス変数およびプロパティの定義を示す。温水ボイラクラスとほぼ同様であるが、出口水温のかわりに蒸気の状態が必要になるため、圧力と温度の両方をプロパティとして持つ。

プログラム 13.10　インスタンス変数およびプロパティの定義

Popolo.HVAC.HeatSource.SteamBoiler class

```
 1 /// <summary>空気比[-]</summary>
 2 private double airRatio = 1.1;
 3
 4 /// <summary>熱損失係数[kW/K]</summary>
 5 private double heatLossCoefficient = 0.0;
 6
 7 /// <summary>定格蒸気温度[C]</summary>
 8 private double nominalSteamTemperature;
 9
10 /// <summary>定格排ガス温度[C]</summary>
11 private double nominalSmokeTemperature;
12
13 /// <summary>電力の一次エネルギー換算係数[MJ/kWh]を設定・取得する</summary>
14 public double PrimaryEnergyFactor { get; set; } = 9.76;
15
16 /// <summary>燃料種別を取得する</summary>
17 public Boiler.Fuel Fuel { get; private set;}
18
19 /// <summary>蒸気圧力[kPa]を設定・取得する</summary>
20 public double SteamPressure { get; set; }
21
22 /// <summary>蒸気温度[C]を取得する</summary>
23 public double SteamTemperature
24 { get { return Water.GetSaturationTemperature(SteamPressure); } }
25
26 /// <summary>温水入口温度[C]を取得する</summary>
27 public double InletWaterTemperature { get; private set; }
28
29 /// <summary>蒸気流量[kg/s]を取得する</summary>
30 public double SteamFlowRate { get; private set; }
31
32 /// <summary>定格加熱能力[kW]を取得する</summary>
33 public double NominalCapacity { get; private set; }
34
35 /// <summary>定格燃料消費量[kg/s, Nm3/s]を取得する</summary>
36 public double NominalFuelConsumption { get; private set; }
37
38 /// <summary>消費電力[kW]を取得する</summary>
39 public double ElectricConsumption { get; private set; }
40
41 /// <summary>燃料消費量[kg/s, Nm3/s]を取得する</summary>
42 public double FuelConsumption { get; private set; }
43
44 /// <summary>暖房能力[kW]を取得する</summary>
45 public double HeatLoad { get; private set; }
46
47 /// <summary>周囲の温度[C]を設定・取得する</summary>
48 public double AmbientTemperature { get; set; }
49
50 /// <summary>空気比を設定・取得する</summary>
51 public double AirRatio
52 {
53   get { return airRatio; }
54   set { airRatio = Math.Max(1.0, Math.Min(1.5, value)); }
```

```
55 }
56
57 /// <summary>COP[-]を取得する</summary>
58 public double COP
59 {
60   get
61   {
62     if (HeatLoad == 0) return 0;
63     else return 0.001 * HeatLoad / (ElectricConsumption * PrimaryEnergyFactor
64     + FuelConsumption * Boiler.GetHeatingValue(Fuel, true));
65   }
66 }
```

プログラム 13.11 にコンストラクタを示す。これも構成は温水ボイラクラスとほぼ同様であり、16~24 行で必要な情報をプロパティに保存した後、26~34 行で熱損失係数の計算を行う。

プログラム 13.11　コンストラクタ

Popolo.HVAC.HeatSource.SteamBoiler class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="inletWaterTemperature">入口温水温度[C]</param>
 3 /// <param name="steamPressure">蒸気圧力[kPa]</param>
 4 /// <param name="steamFlowRate">蒸気流量[kg/s]</param>
 5 /// <param name="fuelConsumption">燃料消費量[kg/s, Nm3/s]</param>
 6 /// <param name="electricConsumption">消費電力[kW]</param>
 7 /// <param name="ambientTemperature">周囲の温度[C]</param>
 8 /// <param name="airRatio">空気比[-]</param>
 9 /// <param name="fuel">燃料種別</param>
10 /// <param name="smokeTemperature">排ガス温度[C]</param>
11 public SteamBoiler(double inletWaterTemperature, double steamPressure, double steamFlowRate,
12   double fuelConsumption, double electricConsumption, double ambientTemperature, double airRatio,
13   Boiler.Fuel fuel, double smokeTemperature)
14 {
15   //プロパティ保存
16   Fuel = fuel;
17   AirRatio = airRatio;
18   nominalSmokeTemperature = smokeTemperature;
19   NominalFuelConsumption = fuelConsumption;
20   ElectricConsumption = electricConsumption;
21   AmbientTemperature = ambientTemperature;
22   SteamPressure = steamPressure;
23   SteamFlowRate = steamFlowRate;
24   nominalSteamTemperature = Water.GetSaturationTemperature(steamPressure);
25
26   double ts = Water.GetSaturationTemperature(steamPressure);
27   double hs = Water.GetSaturatedVaporEnthalpy(ts);
28   double hw = Water.GetSaturatedLiquidEnthalpy(inletWaterTemperature);
29   NominalCapacity = steamFlowRate * (hs - hw);
30
31   //熱損失係数の計算
32   heatLossCoefficient = Boiler.GetHeatLossCoefficient
33     (NominalCapacity, ts, AmbientTemperature, fuel, nominalSmokeTemperature, AirRatio,
34     NominalFuelConsumption, AmbientTemperature);
35
36   //停止させる
37   ShutOff();
38 }
39
40 /// <summary>停止する</summary>
41 public void ShutOff()
42 {
43   SteamPressure = 101.325;
44   ElectricConsumption = SteamFlowRate = HeatLoad = FuelConsumption = 0;
45 }
```

プログラム 13.12 に状態更新処理を示す。14~20 行で必要な燃料消費量を求め、最大（定格）燃料消費量よりも大きい場合には 23~31 行で成り行きの計算処理に移る。この場合には出力が不足するため、蒸気流量が入力値よりも小さい値となる。

プログラム 13.12　状態更新処理

Popolo.HVAC.HeatSource.SteamBoiler class

```
1 /// <summary>状態を更新する</summary>
2 /// <param name="inletWaterTemperature">入口水温[C]</param>
3 /// <param name="steamFlowRate">蒸気流量[kg/s]</param>
4 public void Update(double inletWaterTemperature, double steamFlowRate)
5 {
```

```
 6   //入力条件保存
 7   InletWaterTemperature = inletWaterTemperature;
 8   SteamFlowRate = steamFlowRate;
 9
10   //流量0、設定圧力 < 入口水温の飽和圧力で停止
11   double pwi = Water.GetSaturationPressure(InletWaterTemperature);
12   if (SteamFlowRate <= 0) ShutOff();
13
14   //燃料消費量を計算
15   double ts = Water.GetSaturationTemperature(SteamPressure);
16   double hs = Water.GetSaturatedVaporEnthalpy(ts);
17   double hw = Water.GetSaturatedLiquidEnthalpy(inletWaterTemperature);
18   double hl = steamFlowRate * (hs - hw);
19   FuelConsumption = Boiler.GetFuelConsumption(hl, ts, AmbientTemperature, Fuel,
20     nominalSmokeTemperature, airRatio, heatLossCoefficient, AmbientTemperature, nominalSteamTemperature);
21
22   //過負荷の場合には上記流量を調整
23   if (NominalFuelConsumption < FuelConsumption)
24   {
25     FuelConsumption = NominalFuelConsumption;
26     double sf;
27     Boiler.GetSteamFlowRate(InletWaterTemperature, SteamPressure,
28       NominalFuelConsumption, AmbientTemperature, Fuel, nominalSmokeTemperature,
29       AirRatio, heatLossCoefficient, AmbientTemperature, nominalSteamTemperature, out sf, out hl);
30     SteamFlowRate = sf;
31   }
32   HeatLoad = hl;
33 }
```

【例題 13.3】

下記の仕様の蒸気ボイラについて、負荷率を変化させた場合のボイラ効率を計算せよ。ただし、空気比は 1.1, 1.2, 1.3 の 3 通り、蒸気出口圧力は 490, 690, 890 kPa の 3 通りとする。

温水入口温度[†1]：100 °C　　蒸気圧力：490 kPa　　蒸気量：1500 kg/h

燃料種別：都市ガス 13A　　燃料消費量：81.7 Nm^3/h　　排気ガス温度：80 °C

【解】

プログラム 13.13 に計算方法を示す。3,4 行で蒸気ボイラのインスタンスを作成する。6~9 行で各蒸気圧力で負荷率 100 %となる蒸気流量を計算する。16~38 行が負荷率のループである。23~28 行で空気比を変化せた場合の COP を、31~36 行で蒸気圧力を変化させた場合の COP を計算する。計算結果をグラフにすると図 13.4 が得られる。COP に与える影響の大きさは空気比より蒸気圧力の方が大きい。

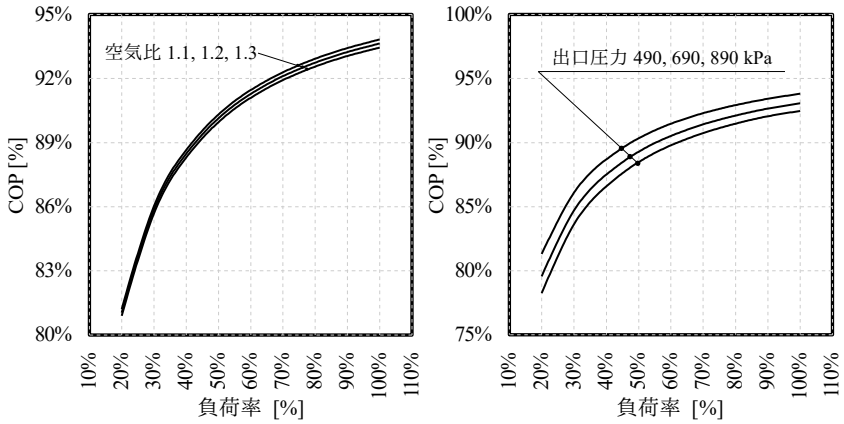


図 13.4　ボイラ負荷率とボイラ効率の関係（空気比・蒸気圧力別）

プログラム **13.13**　ボイラ負荷率とボイラ効率の関係の計算（空気比・蒸気圧力別）

```
 1 private static void SteamBoilerTest()
 2 {
 3   SteamBoiler sBoiler = new SteamBoiler
 4     (100, 490, 1500d / 3600, 81.7 / 3600, 0, 25, 1.15, Boiler.Fuel.Gas13A, 80);
 5
 6   double[] sf = new double[3];
 7   for(int i=0;i<sf.Length;i++)
 8     sf[i] = sBoiler.NominalCapacity / (Water.GetSaturatedVaporEnthalpy
 9         (Water.GetSaturationTemperature(490 + 200 * i)) - 418.6);
10
```

†1　ボイラの定格性能は 100 °C の温水を入口条件として記載されることが多い。このような条件における蒸気量を「換算蒸発量」と呼ぶ。一方、実条件における蒸発量を「実際蒸発量」と呼ぶ。

```
11   using (StreamWriter sWriter = new StreamWriter
12     ("SteamBoilerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
13   {
14     sWriter.WriteLine
15       ("負荷率, 空気比 1.1, 空気比 1.2, 空気比 1.3, 490kPa, 690kPa, 890kPa");
16     for (int i = 0; i < 10; i++)
17     {
18       double pl = 1.0 - 0.1 * i;
19       sWriter.Write(pl);
20
21       sBoiler.SteamPressure = 490;
22       double sFlow = (1500d / 3600) * pl;
23       for (int j = 0; j < 3; j++)
24       {
25         sBoiler.AirRatio = 1.1 + 0.1 * j;
26         sBoiler.Update(100, sFlow);
27         sWriter.Write("," + sBoiler.COP);
28       }
29
30       sBoiler.AirRatio = 1.15;
31       for (int j = 0; j < 3; j++)
32       {
33         sBoiler.SteamPressure = 470 + j * 200;
34         sBoiler.Update(100, sf[j] * pl);
35         sWriter.Write("," + sBoiler.COP);
36       }
37       sWriter.WriteLine();
38     }
39   }
40 }
```

【第 13 章 記号表】

COP_b	：ボイラ効率 **[-]**	Q_L	：ボイラ缶体からの熱損失量 **[kW]**
c_p	：定圧比熱 **[kJ/(kg·K)]**	$Q_{L,b}$	：ブロー熱損失 **[kW]**
HV_h	：高位発熱量 **[MJ/Nm^3]**	r_{sf}	：排ガス生成量 **[MJ/Nm^3] or [MJ/kg]**
HV_l	：低位発熱量 **[MJ/Nm^3]**	t	：温度 **[°C]**
k	：理論空気量 **[Nm^3/Nm^3] or [Nm^3/kg]**	U_L	：熱損失係数 **[kW/K]**
m	：質量流量 **[kg/s]**	λ	：空気比 **[-]**
Q	：熱量 **[kW]**	ρ	：比重量 **[kg/m^3]**
sub scripts			
a	：乾き空気	o	：出口
f	：燃料	s	：排気ガス
i	：入口	sv	：飽和蒸気
N	：定格条件	w	：水

【第 13 章 参考文献】

13.1） G. Claus and W. Stephan: A General Computer Simulation Model for Furnaces and Boilers, ASHRAE Transaction, vol.91(1), pp.47-59, 1985

13.2） エネルギーの使用の合理化に関する法律 第五条第一項に基づく経済産業省告示第六十五号

13.3） 温室効果ガス排出量算定に関する検討結果 エネルギー・工業プロセス分科会報告書 , 環境省 温室効果ガス排出量算定方法検討会, H18.8

13.4） 経済産業省 資源エネルギー庁 総合エネルギー統計検討会事務局: 005 年度以降適用する標準発熱量の検討結果と改訂値について, H19.5

13.5） 日本規格協会: JIS B 8222 陸用ボイラ 熱勘定方式, 2011

13.6） Werner Boie, Von Brennstoff zum Rauchgas, Terbner Verlag, 1957

第14章　圧縮式冷凍機
(Compression Refrigerator)

14.1　概要

圧縮式冷凍機は代表的な冷熱源であり、家庭用のルームエアコン、中小規模ビルの個別分散型マルチパッケージ、大規模ビルの遠心冷凍機（ターボ冷凍機）、空気熱源ヒートポンプチラーなどはすべて圧縮式冷凍機に分類される。

空調分野で用いられる冷熱源としては本章の圧縮式冷凍機と次章の吸収式冷凍機が主であり、いずれも 19 世紀以降に発明された比較的新しい技術である。圧縮式冷凍機は低温低圧の冷媒を得る過程において高温高圧の冷媒を生成する必要があり、この高温冷媒から熱を受けることで温熱源としても機能させることができる。この点は吸収式冷凍機と比較したときの圧縮式冷凍機の大きな特徴である。低温媒体から熱を汲み上げるという機能に注目し、このような機種を特に「ヒートポンプ」と呼ぶ。ルームエアコンやマルチパッケージでは標準でこのような機能があるため、冷房と暖房の両方に対応が可能である。

本章ではまず圧縮式冷凍サイクルの理論について解説し、代表的な圧縮式冷凍機であるターボ冷凍機と空気熱源ヒートポンプの計算法を示す。

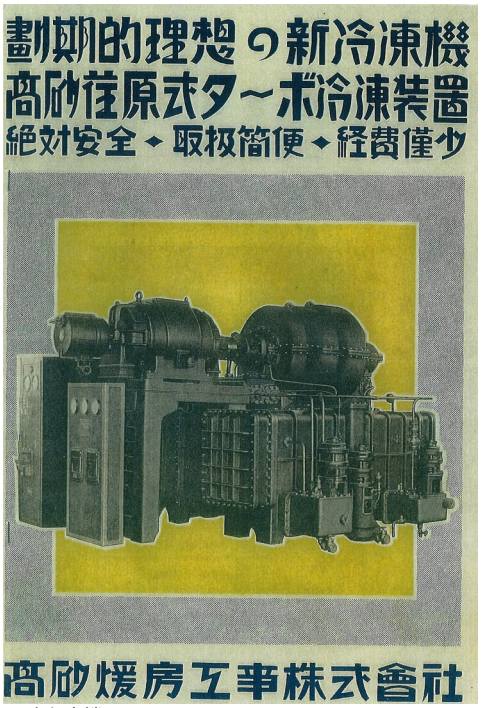


写真 14.1　高砂荏原式ターボ冷凍機

（日本初のターボ冷凍機であり、柳町政之助らが 1930 年に開発した。右は当時のカタログ）

14.2　理論

14.2.1　熱機関と逆カルノーサイクル

高温の熱源から受けた熱を低温の熱源に捨てる過程で外部にエネルギーを与える（仕事をする）機械を熱機関と呼ぶ。熱機関の内、最も効率が良いシステムはカルノーによって提案されたカルノーサイクルである。一方で、冷凍機は熱機関とは逆に、外部からエネルギーを加えることで低温熱源から熱を汲み出し、高温熱源に熱を捨てる。このようなサイクルを逆カルノーサイクルと呼ぶ。低温熱源から汲み出した熱（冷却）Q_{evp} [kW]を、投入したエネルギー E_{cmp} [kW]で除した値を成績係数[†1]（*COP* :Coefficient of Performance）と呼ぶ（式 14.1）。

$$COP = Q_{evp} / E_{cmp} \tag{14.1}$$

逆カルノーサイクルは以下の 4 つのステップから成り立つ。1）流体を断熱圧縮し、高温高圧高エンタルピーの流体にする。2）温度を保ったまま熱を放出させ、高温高圧低エンタルピーの流体にする。3）断熱膨張させ、低温低圧低エンタルピーの流体にする。4）温度を保ったまま熱を加え、低温低圧高エンタルピーの流体にして 1）に戻る。逆カルノーサイクルの成績係数 COP_{cT}は高温熱源と低温熱源の絶対温度（T_h[K] と T_c[K]）によって表現でき、式 14.2 で表現される。逆カルノーサイクルの成績係数は理想的な最大効率のサイクルであり 理論成績係数と呼ばれる。式 14.2 は冷却能力に着目した式であるが、加熱能力に着目すると理論成績係数は式 14.3 で表現される。

$$COP_{cT} = \frac{T_c}{T_h - T_c} \tag{14.2}$$

$$COP_{hT} = \frac{T_h}{T_h - T_c} \tag{14.3}$$

【例題 14.1】

冷却水温度を 32°C→37°C、冷水温度を 12°C→7°C と設計した冷凍機の理論成績係数を求めよ。

【解】

冷却水出口温度 = 高温熱源温度 T_h、冷水出口温度 = 低温熱源温度 T_cである。従って理論成績係数は、

$COP_T = (7 + 273.15) / (37 - 7) = 9.3$

となる。現在の遠心冷凍機の最大効率機種の COP は 6.5 程度のため、理想値と比較して 70 %程度の効率が達成できていることになる。

14.2.2　冷凍サイクルとモリエル線図

実際の冷凍機では厳密に逆カルノーサイクルに従った運転はできず、少し異なったサイクル（以下、冷凍サイクルと呼ぶ）が成立している。冷凍サイクルを表現・理解するためには冷凍機の中を流れる冷媒の圧力 P [kPa]を縦軸（対数軸とすることが多い）、比エンタルピー h [kJ/kg]を横軸にとった PH 線図（モリエル線図と呼ばれる）を用いるとわかりやすい。図 14.1 にモリエル線図と冷凍サイクルの例を示す。比エンタルピーと圧力に加え、等温度線、等比エントロピー線、飽和蒸気線、飽和液線が描画される。飽和液線の左方は液体、飽和蒸気線の右方は蒸気を表している。また、両飽和線に囲まれた領域は液体と蒸気が混ざって存在する領域であり、二相域、遷移域などと呼ばれる。

図中の E 点を起点に冷凍サイクルを説明する。E 点での冷媒は低温低圧低エンタルピー状態にある。温度は 5°C 程度であり、12°C の冷水と熱交換を行うと沸騰が生じて F 点まで比エンタルピーが上昇する。この過程で冷水は 7°C まで冷却される。遠心冷凍機では F 点が圧縮機入口状態となるが、

†1　単純に「効率」と呼ぶことも多い。

往復動圧縮機やスクロール圧縮機（パッケージに使用される）を用いた冷凍機では、効率向上を目的に過冷却器（図中 CD）との熱交換が行われる。この結果、冷媒はさらに A 点まで加熱される。FA 間を過熱度と呼ぶ。A 点で圧縮機に吸い込まれた低温低圧高エンタルピー冷媒は断熱圧縮され、等比エントロピー線上を通って B 点まで移動する[†1]。B 点の高圧高温高エンタルピー状態の冷媒は、32°C の冷却水と熱交換を行うことで凝縮され、C 点まで移動する。C 点において冷媒は飽和液となるが、効率向上を目的にさらに D 点まで冷却される。D 点の高温高圧低エンタルピー状態の冷媒は、膨張弁により圧力が解放されて E 点に戻る。以上のサイクルが連続的に繰り返されることで、低温熱源から高温熱源へと熱の移動が継続する。

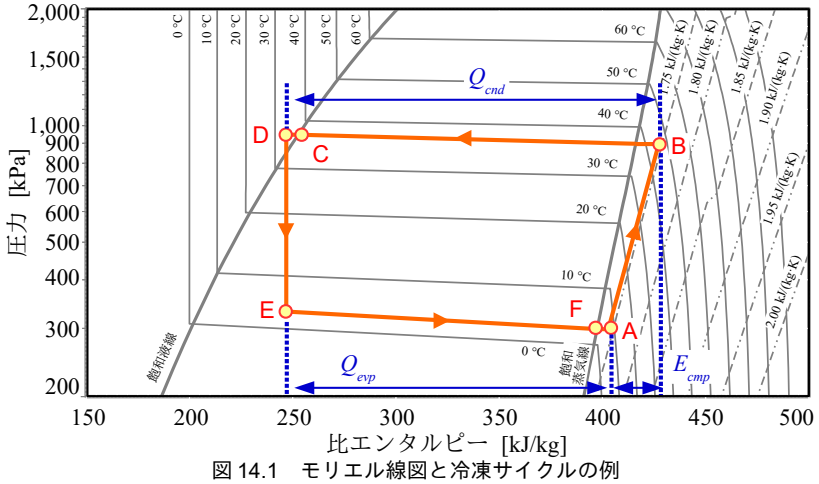


図 14.1　モリエル線図と冷凍サイクルの例

以下、具体的な冷凍機の機構に沿って冷凍サイクルを解説する。図 14.2 に圧縮式冷凍機の構造と熱収支を示す。図中の *A*, *B*, *D*, *E* はそれぞれ図 14.1 の冷媒の状態点と対応している。

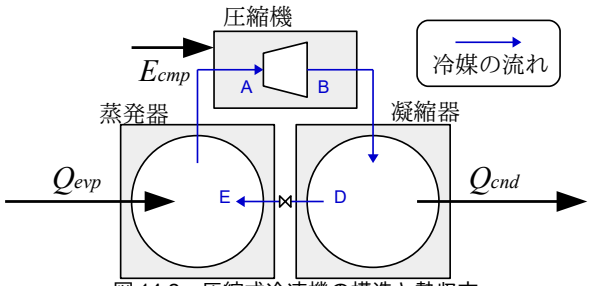


図 14.2　圧縮式冷凍機の構造と熱収支

1）蒸発器（E 点~A 点）

蒸発器は低温低圧冷媒を加熱する機器である。遠心冷凍機やスクリュー冷凍機などのように冷水で冷媒を加熱する（冷水を冷却する）機器の場合には、シェルアンドチューブ型熱交換器が用いられる。円筒胴（シェル）内は冷媒液で満たされ、この中に多数の伝熱管（チューブ）が配置される。伝熱管の中には冷水が流れ、冷媒に熱を放出する。冷水から熱を受けた冷媒は蒸発して圧縮機へと向かう。一方、家庭用エアコンや個別分散型マルチパッケージでは、冷水ではなく直接に室内空気と熱交換が行われる。この場合にはシェルアンドチューブではなく、プレートフィン付管熱交換器が用いられる[†2]。モリエル線図上の動きは両熱交換器ともに同じであり、E 点から水または空気によって加熱

†1　現実には圧縮機で生じる損失により冷媒は不可逆断熱圧縮されるため、比エントロピーがやや増加する方向に移動する。
†2　この場合のプレートフィン付管熱交換器を特に「直膨コイル」と呼ぶ（第 10 章）。

されて A 点に至る。

2）圧縮機（A 点~B 点）

圧縮機は冷媒を圧縮する機器である。容積式と遠心式に大別され、容積式はさらに往復式・ロータリー式・スクロール式・スクリュー式に細分化される。容積式は、ある空間に冷媒を閉じ込め、容積を減少させることで圧縮する方式である。遠心式は高速回転する羽根車によって冷媒に遠心力を加えることで圧縮する方式である。空調用としてはスクロール式、スクリュー式、遠心式の圧縮機が用いられることが多い。スクロール式は小容量の機器に向いているため、家庭用エアコンや個別分散型マルチパッケージで用いられる。スクリュー式は高圧力比の運転に適しているため、地中熱や河川水熱などの中温熱を用いたヒートポンプ機で採用される。遠心式は大容量に適しており、遠心式圧縮機を搭載した冷凍機をターボ冷凍機と呼ぶ。

3）凝縮器（B 点~D 点）

凝縮器は高温高圧冷媒を冷却する機器である。蒸発器と同様に、水で冷却する場合にはシェルアンドチューブ、空気で冷却する場合にはプレートフィン付管熱交換器が用いられる。冷媒は B 点を起点に水あるいは空気で冷却され、液化して D 点に至る。

14.2.3　基礎式

1）全体の熱収支

圧縮式冷凍機において、冷水および冷却水以外に冷媒とエネルギーの授受を行う要素は圧縮機であり、冷媒の圧力上昇に相当するエネルギーが与えられ、最終的には熱となる。即ち圧縮式冷凍機の熱収支は式 14.4 の通りとなる。E_{cmp} [kW]は圧縮機に投入した電気エネルギー、Q_{evp} [kW]は蒸発器において冷水または空気が冷媒に与えた熱、Q_{cnd} [kW]は凝縮器において冷媒が冷却水または空気に与えた熱である。それぞれの熱は図 14.1 に図示したとおり、モリエル線図上は横軸方向の幅で表現される。式 14.1 に示したように冷凍機の COP は蒸発器での冷却熱量 Q_{evp} を圧縮機への投入エネルギー量 E_{cmp} で除した値である。従って、蒸発器の圧力を上げる（蒸発温度を上げる）ないしは凝縮器の圧力を下げる（凝縮温度を下げる）ことにより、モリエル線図上の E_{cmp} の横幅を縮めることができ、COP が上昇することがわかる。特にインバータが搭載された機種に関しては、圧縮機の仕事量を柔軟に変化させることができるため、このような蒸発温度・凝縮温度の変化による効率向上の効果の恩恵を大きく受けることができる。

$$E_{cmp}+Q_{evp}=Q_{cnd} \tag{14.4}$$

2）圧縮機

圧縮機の断熱圧縮仕事 W_{cmp} [kW]は式 14.5 で計算する。v_R [m³/s]は圧縮機入口での冷媒吸込量、κ [-]は冷媒の比熱比、P_i [kPa]は吸込圧力、P_o [kPa]は吐出圧力である。断熱圧縮仕事 W_{cmp} は出入口における冷媒の比エンタルピーから式 14.6 で計算することもできる。$h_{cmp,i}$ と $h_{cmp,o}$ はそれぞれ圧縮機の入口と出口での比エンタルピー[kJ/kg]、m_R [kg/s]は冷媒の質量流量である。

$$W_{cmp}=v_R\frac{\kappa}{\kappa-1}P_i\left\{\left(\frac{P_o}{P_i}\right)^{(\kappa-1)/\kappa}-1\right\} \tag{14.5}$$

$$W_{cmp}=m_R\left(h_{cmp,o}-h_{cmp,i}\right) \tag{14.6}$$

現実には圧縮機への入力の全てが圧縮仕事とはならず、モーター損失や機械損失など、様々な損失

が生じる。これらの損失を考慮した効率を圧縮機の効率 η_{cmp} [-]とし、入力 E_{cmp} [kW]と圧縮仕事の関係を式 14.7 で表現する。

$$E_{cmp}=W_{cmp}/\eta_{cmp} \tag{14.7}$$

3）凝縮器

凝縮器において冷却水と交換される熱 Q_{cnd} を式 14.8~14.10に示す。式 14.8 は冷却水の出入口温度差にもとづく基礎式であり、温度差に式 14.11 で示される熱容量流量 mc_{cd} [kW/K]を乗じることで計算する。ここで m_{cd} [kg/s]は冷却水の質量流量、c_{pw} [kJ/(kg·K)]は冷却水の定圧比熱である。式 14.9 は冷媒と冷却水の温度差にもとづく基礎式であり、凝縮器の熱通過率 K_{cd} [kW/(m^2·K)]と伝熱面積 A_{cd} [m^2]に冷媒と冷却水の平均的な温度差である ΔT [K]を乗じて計算する。平均的な温度差とは第 8 章で解説した対数平均温度差であり、式 14.12~14.14で計算する。厳密には熱交換の過程で冷媒温度は変化するが、熱交換の大部分は冷媒温度が一定となる二相状態で行われるため凝縮温度 T_{cnd} [K]で一定としている。式 14.10 は冷媒側からみた熱交換量の式であり、$m_{R,cnd}$ [kg/m^3]は凝縮器を通過する冷媒質量流量、$h_{cnd,i}$ と $h_{cnd,o}$ はそれぞれ凝縮器の出入口比エンタルピー[kJ/kg]である。

$$Q_{cnd}=mc_{cd}\left(T_{cdo}-T_{cdi}\right) \tag{14.8}$$

$$Q_{cnd}=K_{cd}A_{cd}\Delta T_{LM,cd} \tag{14.9}$$

$$Q_{cnd}=m_{R,cnd}\left(h_{cnd,i}-h_{cnd,o}\right) \tag{14.10}$$

$$mc_{cd}=m_{cd}c_{pw} \tag{14.11}$$

$$\Delta T_{LM,cd}=\frac{\Delta T_{cdi}-\Delta T_{cdo}}{\ln\left(\Delta T_{cdi}/\Delta T_{cdo}\right)}=\frac{T_{cdo}-T_{cdi}}{\ln\left(\Delta T_{cdi}/\Delta T_{cdo}\right)} \tag{14.12}$$

$$\Delta T_{cdi}=T_{cnd}-T_{cdi} \tag{14.13}$$

$$\Delta T_{cdo}=T_{cnd}-T_{cdo} \tag{14.14}$$

式 14.8 と式 14.9 を凝縮温度 T_{cnd} [K]について解くと式 14.15 が得られる。

$$T_{cnd}=T_{cd,i}+\frac{Q_{cnd}}{\left\{1-\exp\left(-K_{cd}A_{cd}/mc_{cd}\right)\right\}mc_{cd}} \tag{14.15}$$

以上は蒸発器で冷却を行う際に凝縮器で冷却水に排熱することを想定した記述であるが、逆カルノーサイクルを用いた加熱運転（ヒートポンプ）の場合には凝縮器で温水を加熱することになる。この場合には単純に冷却水を温水に読み替えればよい。以降、本章では冷却水の場合に添字の cd、温水の場合に添字の h を用いて表現する。また、空気熱源ヒートポンプやパッケージ空調機で空気に排熱する場合には、式 14.8 の代わりに空気の比エンタルピー差や温度差を用いて式 14.16 で計算する。

$$Q_{cnd}=m_{ma}\left(h_{ao}-h_{ai}\right)\approx mc_{ma}\left(t_{ao}-t_{ai}\right) \tag{14.16}$$

4）蒸発器

蒸発器の基礎式は凝縮器と同様であり、式 14.17~式 14.25 で計算する。

$$Q_{evp}=mc_{ch}\left(T_{chi}-T_{cho}\right) \tag{14.17}$$

$$Q_{evp}=K_{ch}A_{ch}\Delta T_{LM,ch} \tag{14.18}$$

$$Q_{evp}=m_{R,evp}\left(h_{evp,o}-h_{evp,i}\right) \tag{14.19}$$

$$mc_{ch}=m_{ch}c_{pw} \tag{14.20}$$

$$Q_{evp} = m_{ch}\left(h_{chi} - h_{cho}\right) \tag{14.21}$$

$$\Delta T_{LM,ch} = \frac{\Delta T_{chi} - \Delta T_{cho}}{\ln\left(\Delta T_{chi} / \Delta T_{cho}\right)} = \frac{T_{chi} - T_{cho}}{\ln\left(\Delta T_{chi} / \Delta T_{cho}\right)} \tag{14.22}$$

$$\Delta T_{chi} = T_{chi} - T_{evp} \tag{14.23}$$

$$\Delta T_{cdo} = T_{cho} - T_{evp} \tag{14.24}$$

$$T_{evp} = T_{ch,i} - \frac{Q_{evp}}{\left\{1 - \exp\left(-K_{ch} A_{ch} mc_{ch}\right)\right\} mc_{ch}} \tag{14.25}$$

【例題 14.2】

500 kW の冷却能力を持つターボ冷凍機の COP が 6.0 であったとする。この場合に冷却水温度を 32°C→37°C と設計した際の冷却水流量を求めよ。

【解】

蒸発器で冷水が放熱する熱量は 500 kW であるため、式 14.4 の Q_{evp} は 500 kW である。COP は圧縮機動力に対する冷熱製造量の大きさであるため、Q_{cmp} は 500 kW÷6=83 kW である。従って、式 14.4 により凝縮器で冷媒が放熱する熱量は 500+83=583 kW となる。温度差は 37 °C-32 °C=5 °C であるため、冷却水流量は 583 kW÷5 °C÷4.186 kJ/(kg·K) = 27.9 kg/s = 1,670 L/min となる。

14.3　計算法

特性式による簡易計算法と理論式による計算法をそれぞれ解説する。計算が簡便であり速度も早いため、設備のエネルギーシミュレーションにおいては前者が用いられることが多い。一方で、特性式は外挿性能に関しては保証が無いため、未知の入力に対する感度を検討したい場合などには後者を検討する必要がある。両者の違いを意識せずに使えるようにするため、共通する処理をインターフェースとしてまとめておく。プログラム 14.1 にターボ冷凍機インターフェースの定義を示す。

プログラム 14.1　ターボ冷凍機インターフェースの定義

Popolo.HVAC.HeatSource.ICentrifugalChiller class

```
/// <summary>ターボ冷凍機</summary>
public interface ICentrifugalChiller: ImmutableCentrifugalChiller
{

  /// <summary>運転中か否か</summary>
  new bool IsOperating { get; set; }

  /// <summary>冷水出口温度設定値[C]を設定・取得する</summary>
  new double ChilledWaterOutletSetPointTemperature { get; set; }

  /// <summary>機器を停止させる</summary>
  void ShutOff();

  /// <summary>状態を更新する</summary>
  /// <param name="coolingWaterInletTemperature">冷却水入口温度[C]</param>
  /// <param name="chilledWaterInletTemperature">冷水入口温度[C]</param>
  /// <param name="coolingWaterFlowRate">冷水質量流量[kg/s]</param>
  /// <param name="chilledWaterFlowRate">冷却水質量流量[kg/s]</param>
  void Update
  (double coolingWaterInletTemperature, double chilledWaterInletTemperature,
  double coolingWaterFlowRate, double chilledWaterFlowRate);

}

/// <summary>読み取り専用ターボ冷凍機</summary>
public interface ImmutableCentrifugalChiller
{

  /// <summary>運転中か否か</summary>
  bool IsOperating { get; }

  /// <summary>冷水出口温度[C]を取得する</summary>
  double ChilledWaterOutletTemperature { get; }

  /// <summary>冷水出口温度設定値[C]を取得する</summary>
  double ChilledWaterOutletSetPointTemperature { get; }
```

```
37
38  /// <summary>冷水入口温度[C]を取得する</summary>
39  double ChilledWaterInletTemperature { get; }
40
41  /// <summary>冷却水出口温度[C]を取得する</summary>
42  double CoolingWaterOutletTemperature { get; }
43
44  /// <summary>冷却水入口温度[C]を取得する</summary>
45  double CoolingWaterInletTemperature { get; }
46
47  /// <summary>冷水流量[kg/s]を取得する</summary>
48  double ChilledWaterFlowRate { get; }
49
50  /// <summary>冷却水流量[kg/s]を取得する</summary>
51  double CoolingWaterFlowRate { get; }
52
53  /// <summary>定格冷凍能力[kW]を取得する</summary>
54  double NominalCapacity { get; }
55
56  /// <summary>定格入力[kW]を取得する</summary>
57  double NominalInput { get; }
58
59  /// <summary>定格 COP[-]を取得する</summary>
60  double NominalCOP { get; }
61
62  /// <summary>容量制御下限値[-]を取得する</summary>
63  double MinimumPartialLoadRatio { get; }
64
65  /// <summary>消費電力[kW]を取得する</summary>
66  double ElectricConsumption { get; }
67
68  /// <summary>冷却負荷[kW]を取得する</summary>
69  double CoolingLoad { get; }
70
71  /// <summary>COP[-]を取得する</summary>
72  double COP { get; }
73
74  /// <summary>冷水上限流量[kg/s]を取得する</summary>
75  double MaxChilledWaterFlowRate { get; }
76
77  /// <summary>冷水下限流量比[-]を取得する</summary>
78  double MinChilledWaterFlowRatio { get; }
79
80  /// <summary>過負荷か否か</summary>
81  bool IsOverLoad { get; }
82
83 }
```

14.3.1　特性式にもとづくターボ冷凍機の計算

1）特性式の形式と解き方[14.20)]

圧縮式冷凍機の効率に主に影響を与える要素は冷凍機の負荷率、冷水温度、冷却水温度であり、通常はこれらを入力とした特性式を用意する。図 14.1 から明らかなように、蒸発器（あるいは凝縮器）において冷媒は液相から気相（気相から液相）へ状態変化するため、温度は一定である。熱交換の結果、冷水（冷却水）はこの一定の蒸発温度（凝縮温度）に漸近していくため、蒸発温度（凝縮温度）と関係性が強い温度は冷水（冷却水）の入口温度ではなく、出口温度である。式 14.2 で示したとおり、冷凍機の成績係数は蒸発温度および凝縮温度に大きく影響を受けるため、特性式も冷水および冷却水の入口温度ではなく、出口温度についての式とすることが望ましい。そこで式 14.26 で冷凍機の特性式を定義する。R_E は入力比であり、定格消費電力 E_N [kW]と圧縮機入力 E_{cmp} [kW]を用いて式 14.27 で表される。R_{COP} [-]は理論成績係数 COP_T [-]を定格条件における理論成績係数 $COP_{T,N}$ [-]で除した値である。理論成績係数 COP_T は式 14.2 で計算できるが、低温熱源温度 T_c=蒸発温度 T_{evp}≈冷水出口温度 T_{cho}、高温熱源温度 T_h=凝縮温度 T_{cnd}≈冷却水出口温度 T_{cdo} とみなして式 14.29 で計算する。Pl は負荷率[-]であり処理熱量 Q_{evp} [kW]と定格能力 $Q_{evp,N}$ [kW]の比として式 14.30 で定義する。

$$R_E=\alpha_{0,0}+\alpha_{0,1}\,Pl+\alpha_{0,2}\,Pl^2+\alpha_{1,0}\,R_{COP}^{-1}+\alpha_{2,0}\,R_{COP}^{-2}+\alpha_{1,1}\,Pl\cdot R_{COP}^{-1} \tag{14.26}$$

$$R_E = E_{cmp} / E_N \tag{14.27}$$

$$R_{COP} = COP_T / COP_{T,N} \tag{14.28}$$

$$COP_T = \frac{T_{cho}}{T_{cdo} - T_{cho}} \tag{14.29}$$

$$Pl = Q_{evp} / Q_{evp,N} \tag{14.30}$$

通常、設備のエネルギーシミュレーションにおいては冷凍機の冷水出口温度設定値 T_{chos} [K]、冷水流量 m_{ch} [kg/s]、冷却水流量 m_{cd} [kg/s]が与えられる。また冷水入口温度 T_{chi} [K]や冷却水入口温度 T_{cdi} [K]は二次側負荷や冷却塔を含めたシステムの中では収束計算で解くことがあるが、冷凍機単体の計算においては入力条件として与えられる。冷凍機が過負荷状態にないとすれば冷水出口温度 T_{cho} = 冷水出口温度設定値 T_{chos} であるため、T_{cho} に T_{chos} を代入し、式 14.4、式 14.8、式 14.26、式 14.27 を整理すると R_E を消去することができ、式 14.31 に示すように冷却水出口温度 T_{do} に関しての二次式が得られる。ただし各々の係数は式 14.32~14.36である。二次式であるため、式 14.37 の解の公式によって反復計算を行うこと無く冷却水出口温度 T_{cdo} を求めることができる。冷却水出口温度さえ求まれば、式 14.26 と式 14.27 を整理した式 14.38 を用いて圧縮機入力 E_{cmp} を計算することができる。現実の機器では圧縮機入力の他に補機類の消費電力が発生するが、特性モデルにおいては、これらは特性式の中に織り込まれているとみなし、機器全体の消費電力 $E = E_{cmp}$ とする。

$$0 = a_{cd} T_{cdo}^2 + b_{cd} T_{cdo} + c_{cd} \tag{14.31}$$

$$a_{cd} = \alpha_{2,0} COP_{T,N}^2 \tag{14.32}$$

$$b_{cd} = T_{chos} \left(-2 a_{cd} + d_{cd} - \frac{T_{chos} \cdot mc_{cd}}{E_N} \right) \tag{14.33}$$

$$c_{cd} = T_{chos}^2 \left(a_{cd} - d_{cd} + e_{cd} + \frac{Q_{evp} + mc_{cd} T_{cdi}}{E_N} \right) \tag{14.34}$$

$$d_{cd} = (\alpha_{1,0} + \alpha_{1,1} Pl) \cdot COP_{T,N} \tag{14.35}$$

$$e_{cd} = \alpha_{0,0} + \alpha_{0,1} Pl + \alpha_{0,2} Pl^2 \tag{14.36}$$

$$T_{cdo} = \frac{-b_{cd} \pm \sqrt{b_{cd}^2 - 4 a_{cd} \cdot c_{cd}}}{2 a_{cd}} \tag{14.37}$$

$$E_{cmp} = E_N \left(a_{cd} COP_T^{-2} + d_{cd} COP_T^{-1} + e_{cd} \right) \tag{14.38}$$

特性式の係数 α はメーカーから特性線図を入手して重回帰分析を行うことで推定する。国内の代表的メーカー 4 社から入手した特性線図をもとに平均的な特性を求めた結果を表 14.1 に示す。インバータ搭載機と定速機では大きく特性が異なるため、それぞれに係数を推定した。

表 14.1　圧縮式冷凍機の特性係数の例

種別	$\alpha_{0,0}$	$\alpha_{0,1}$	$\alpha_{1,0}$	$\alpha_{1,1}$	$\alpha_{2,0}$	$\alpha_{0,2}$
インバータ機	-4.964×10^{-4}	4.865×10^{-1}	5.053×10^{-1}	1.781×10^{-1}	-1.923×10^{-1}	2.292×10^{-2}
固定速機	4.425×10^{-4}	-2.532×10^{-1}	4.256×10^{-1}	3.098×10^{-1}	6.910×10^{-2}	4.482×10^{-1}

2）「特性式にもとづくターボ冷凍機クラス」の作成

特性式を用いたターボ冷凍機クラスを作成する。インスタンス変数およびプロパティの定義をプログラム 14.2 に示す。10~62 行はプログラム 14.1 に示したインターフェースの実装である。

プログラム **14.2**　インスタンス変数およびプロパティの定義

Popolo.HVAC.HeatSource.SimpleCentrifugalChiller class

```
/// <summary>機器特性係数</summary>
private readonly double[] coef;

/// <summary>理論 COP[-]</summary>
private readonly double theoreticalCOP;

/// <summary>インバータ制御か否か</summary>
public bool HasInverter { get; private set; }

/// <summary>運転中か否か</summary>
public bool IsOperating { get; set; }

/// <summary>冷水出口温度[C]を取得する</summary>
public double ChilledWaterOutletTemperature { get; private set; }

/// <summary>冷水出口温度設定値[C]を設定・取得する</summary>
public double ChilledWaterOutletSetPointTemperature { get; set; }

/// <summary>冷水入口温度[C]を取得する</summary>
public double ChilledWaterInletTemperature { get; private set; }

/// <summary>冷却水出口温度[C]を取得する</summary>
public double CoolingWaterOutletTemperature { get; private set; }

/// <summary>冷却水入口温度[C]を取得する</summary>
public double CoolingWaterInletTemperature { get; private set; }

/// <summary>冷水流量[kg/s]を取得する</summary>
public double ChilledWaterFlowRate { get; private set; }

/// <summary>冷却水流量[kg/s]を取得する</summary>
public double CoolingWaterFlowRate { get; private set; }

/// <summary>定格冷凍能力[kW]を取得する</summary>
public double NominalCapacity { get; private set; }

/// <summary>定格入力[kW]を取得する</summary>
public double NominalInput { get; private set; }

/// <summary>定格 COP[-]を取得する</summary>
public double NominalCOP { get; private set; }

/// <summary>容量制御下限値[-]を取得する</summary>
public double MinimumPartialLoadRatio { get; private set; }

/// <summary>消費電力[kW]を取得する</summary>
public double ElectricConsumption { get; private set; }

/// <summary>冷却負荷[kW]を取得する</summary>
public double CoolingLoad { get; private set; }

/// <summary>COP[-]を取得する</summary>
public double COP { get { return (ElectricConsumption == 0) ? 0 : CoolingLoad / ElectricConsumption; } }

/// <summary>冷水上限流量[kg/s]を取得する</summary>
public double MaxChilledWaterFlowRate { get; private set; }

/// <summary>冷水下限流量比[-]を設定・取得する</summary>
public double MinChilledWaterFlowRatio { get; set; } = 0.4;

/// <summary>過負荷か否か</summary>
public bool IsOverLoad { get; private set; }
```

コンストラクタをプログラム 14.3 に示す。定格入力 E_N [kW]、定格冷水・冷却水出口温度の他、容量制御の下限値とインバータ搭載機か否かの別を引数とする。

現実の機械は 0 kW に至るまで連続的に能力を下げられず、ある負荷を下回った場合には運転と停止を繰り返す発停状態となる。このような状態におけるエネルギー消費量を正しく計算することは難しく、容量制御下限値以下の負荷においては、下限値におけるエネルギー消費が必要になるという安全側（エネルギー多消費の結果となる）の仮定を置くモデルが多い。容量制御下限値は計算上はこのような閾値を設定する意味を持っており、具体的な数値は通常はメーカーのカタログに記載されてい

る。ターボ冷凍機の場合には 20 %程度まで容量制御可能な機種が多い。

14~19 行でプロパティに設定値を保存する。20 行は定格条件における理論 COP の計算であり、式 14.2 を用いる。25~28 行において特性係数の初期化を行い、インバータ搭載機か否かに応じて係数を切り替える。

プログラム 14.3　コンストラクタ

Popolo.HVAC.HeatSource.SimpleCentrifugalChiller class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="nominalInput">定格入力[kW]</param>
 3 /// <param name="minimumPartialLoadRatio">容量制御下限値[-]</param>
 4 /// <param name="chilledWaterInletTemperature">冷水入口温度[C]</param>
 5 /// <param name="chilledWaterOutletTemperature">冷水出口水温[C]</param>
 6 /// <param name="coolingWaterOutletTemperature">冷却水出口水温[C]</param>
 7 /// <param name="chilledWaterFlowRate">冷水質量流量[kg/s]</param>
 8 /// <param name="hasInverter">インバータ付か否か</param>
 9 public SimpleCentrifugalChiller
10   (double nominalInput, double minimumPartialLoadRatio, double chilledWaterInletTemperature,
11   double chilledWaterOutletTemperature, double coolingWaterOutletTemperature,
12   double chilledWaterFlowRate, bool hasInverter)
13 {
14   this.NominalCapacity = chilledWaterFlowRate * WATER_SPECIFIC_HEAT
15     * (chilledWaterInletTemperature - chilledWaterOutletTemperature);
16   this.MaxChilledWaterFlowRate = chilledWaterFlowRate;
17   this.NominalInput = nominalInput;
18   this.MinimumPartialLoadRatio = minimumPartialLoadRatio;
19   this.HasInverter = hasInverter;
20   this.theoreticalCOP = (chilledWaterOutletTemperature + 273.15)
21     / (coolingWaterOutletTemperature - chilledWaterOutletTemperature);
22   this.NominalCOP = NominalCapacity / NominalInput;
23   this.ChilledWaterOutletSetPointTemperature = chilledWaterOutletTemperature;
24
25   //INV 機の特性係数
26   if (HasInverter) coef = new double[] { 4.425e-4, -2.532e-1, 4.256e-1, 3.098e-1, 6.910e-2, 4.482e-1 };
27   //定速機の特性係数
28   else coef = new double[] { -4.964e-4, 4.865e-1, 5.053e-1, 1.781e-1, -1.923e-1, 2.292e-2 };
29 }
```

出口状態の更新処理をプログラム 14.4 に示す。引数は冷水および冷却水の入口温度と流量である。10~14 行で引数をプロパティに保存する。非稼働時は 60~67 行のメソッドを呼び出して停止させる。24, 25 行は式 14.11 の熱容量流量の計算処理である。28 行で、冷水出口温度設定値と入口水温との差に熱容量流量を乗じて負荷を計算する。32~36 行は過負荷の場合の処理であり、負荷が定格能力を上回る場合には冷凍機の出口温度を上げる（35 行）。冷却可能な場合には冷水出口温度 T_{cho} = 冷水出口温度設定値 T_{chos} とする（37 行）。40 行は容量制御範囲に関しての判定であり、制御下限値よりも負荷が小さい場合には、エネルギー計算上は負荷を下限値に設定する。42~53 行は冷却水出口温度の計算処理である。式 14.31~14.37を用いる。55~57 行は消費電力の計算であり、式 14.29 に冷水および冷却水の出口温度を代入して理論成績係数 COP_T を求め、これを式 14.38 に代入することで消費電力 E_{cmp} を求める。

プログラム 14.4　出口状態更新処理

Popolo.HVAC.HeatSource.SimpleCentrifugalChiller class

```
 1 /// <summary>状態を更新する</summary>
 2 /// <param name="coolingWaterInletTemperature">冷却水入口温度[C]</param>
 3 /// <param name="chilledWaterInletTemperature">冷水入口温度[C]</param>
 4 /// <param name="coolingWaterFlowRate">冷却水質量流量[kg/s]</param>
 5 /// <param name="chilledWaterFlowRate">冷水質量流量[kg/s]</param>
 6 public void Update
 7   (double coolingWaterInletTemperature, double chilledWaterInletTemperature,
 8   double coolingWaterFlowRate, double chilledWaterFlowRate)
 9 {
10   //状態値を保存
11   this.ChilledWaterInletTemperature = chilledWaterInletTemperature;
12   this.CoolingWaterInletTemperature = coolingWaterInletTemperature;
13   this.ChilledWaterFlowRate = chilledWaterFlowRate;
```

```
14   this.CoolingWaterFlowRate = coolingWaterFlowRate;
15
16   //非稼働ならば停止処理
17   if (!IsOperating)
18   {
19     ShutOff();
20     return;
21   }
22
23   //熱容量流量を計算
24   double mcch = WATER_SPECIFIC_HEAT * ChilledWaterFlowRate;
25   double mccd = WATER_SPECIFIC_HEAT * CoolingWaterFlowRate;
26
27   //負荷[kW]を計算
28   CoolingLoad = mcch * (ChilledWaterInletTemperature - ChilledWaterOutletSetPointTemperature);
29
30   //過負荷の場合には出口温度を補正
31   IsOverLoad = NominalCapacity < CoolingLoad;
32   if (IsOverLoad)
33   {
34     CoolingLoad = NominalCapacity;
35     ChilledWaterOutletTemperature = ChilledWaterInletTemperature - CoolingLoad / mcch;
36   }
37   else ChilledWaterOutletTemperature = ChilledWaterOutletSetPointTemperature;
38
39   //容量制御下限値未満の場合には下限値として消費電力を計算
40   double load = Math.Max(MinimumPartialLoadRatio * NominalCapacity, CoolingLoad);
41
42   //冷却水出口温度の計算
43   double tcho = ChilledWaterOutletTemperature + 273.15;
44   double tcdi = coolingWaterInletTemperature + 273.15;
45   double pL = load / NominalCapacity;
46   double dcd = (coef[2] + coef[3] * pL) * theoreticalCOP;
47   double ecd = coef[0] + pL * (coef[1] + coef[5] * pL);
48   double acd = coef[4] * theoreticalCOP * theoreticalCOP;
49   double bcd = tcho * (-2 * acd + dcd - (tcho * mccd) / NominalInput);
50   double ccd = tcho * tcho * (acd - dcd + ecd + (load + tcdi * mccd) / NominalInput);
51
52   //2次方程式の解の公式
53   CoolingWaterOutletTemperature = (-bcd - Math.Sqrt(bcd * bcd - 4 * acd * ccd)) / (2d * acd) - 273.15;
54
55   //消費電力の計算
56   double tCop = tcho / (CoolingWaterOutletTemperature - ChilledWaterOutletTemperature);
57   ElectricConsumption = NominalInput * ((acd / tCop + dcd) / tCop + ecd);
58 }
59
60 /// <summary>機器を停止させる</summary>
61 public void ShutOff()
62 {
63   ChilledWaterOutletTemperature = ChilledWaterInletTemperature;
64   CoolingWaterOutletTemperature = CoolingWaterInletTemperature;
65   CoolingLoad = ChilledWaterFlowRate = CoolingWaterFlowRate = 0;
66   ElectricConsumption = 0;
67 }
```

【例題 14.3】

例題 14.2 の冷凍機について、横軸に負荷率、縦軸に COP を取り、冷却水入口温度別に特性線図を作成せよ。冷却水入口温度は 12°C, 16°C, 20°C, 24°C, 28°C, 32°C とする。冷水温度は 12°C→7°C で一定とし、負荷は冷水流量で調整するものとする。冷却水流量は、1）定格値で固定した場合、2）負荷と比例的に減少させた場合、の 2 種類を検討せよ。

【解】

プログラム 14.5 にターボ冷凍機の部分負荷特性の計算を示す。12, 13 行で固定速機と可変速機の冷凍機インスタンスを生成する。16~20 行は外部書き出しのヘッダ行であり、固定速機と可変速機の 2 機種について冷却水量固定と変動の 2 種類の運用を検討する。24~52 行が COP 計算処理である。計算結果をグラフ化すると図 14.3 が得られる。上段は固定速機、下段は可変速機の結果、左は冷却水量固定の運用、右は冷却水量変動の運用結果である。固定速機の場合、負荷率の低下にともなって COP が減少するが、可変速機の場合には、冷却水温度によってはやや山なりのカーブとなり、極大値が表れることがわかる。入口冷却水温度が低いほど凝縮温度を低く抑えられるため COP は高い（左列の図）。また、冷却水入口温度が同じでも、流量が大きいと出口温度が低くなるため、定格条件で冷却水量を固定したほうが COP は高い。この効果は特に可変速機で大きく表れることがわかる。一方で、冷却水量の減少はポンプの搬送動力の削減になるため、厳密には両者のトレードオフを考慮して最適な冷却水量を求めるべきであるが、ポンプ搬送動力

に比較して熱源の消費電力の方が遥かに大きいため、冷却水流量を定格値で固定した方が有利な結果となることが多い。熱源メーカーのカタログに記載されている特性線図は、通常は冷却水量を定格値で固定したものであり、図 14.3 の左列である。

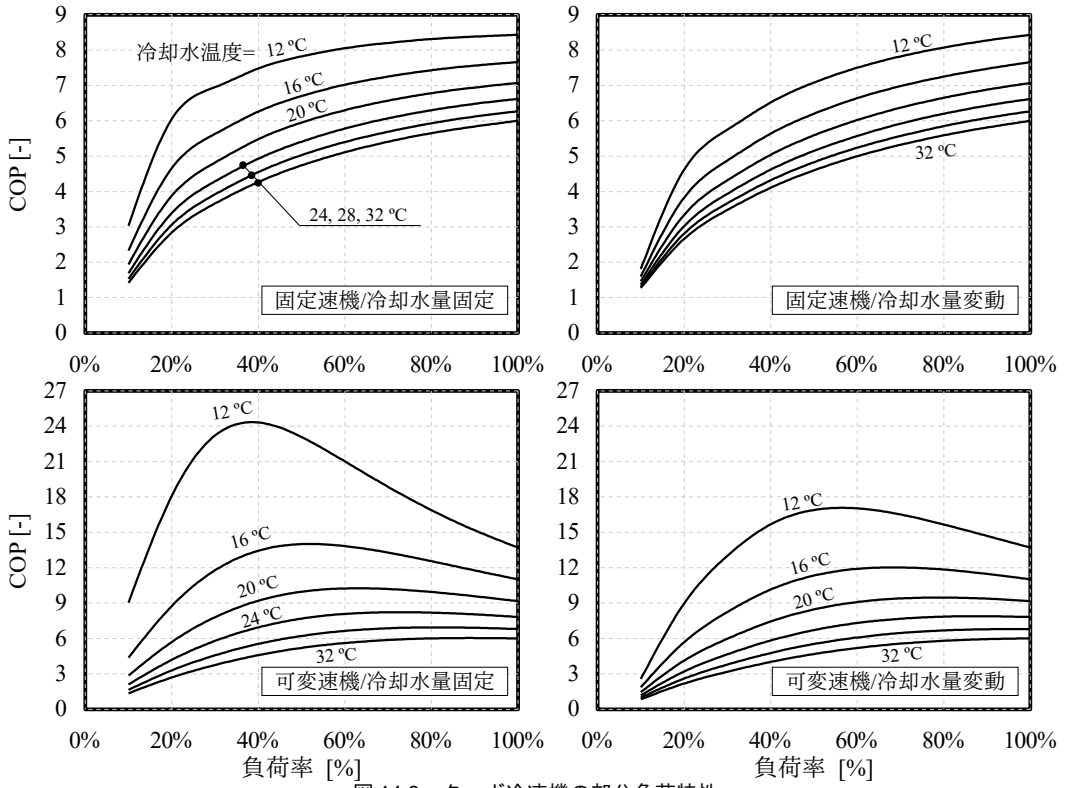


図 14.3　ターボ冷凍機の部分負荷特性

プログラム **14.5**　ターボ冷凍機の部分負荷特性の計算

```
 1 private static void SimpleCentrifugalChillerTest()
 2 {
 3   using (StreamWriter sWriter = new StreamWriter
 4     ("SimpleCentrifugalChillerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
 5   {
 6     //定格冷水量[kg/s]
 7     const double NCH_FLOW = 500d / (12 - 7) / 4.186;
 8     //定格冷却水量[kg/s]
 9     const double NCD_FLOW = 1670d / 60;
10
11     //固定速機と INV 機のインスタンスを生成
12     SimpleCentrifugalChiller cR = new SimpleCentrifugalChiller(500d / 6d, 0.2, 12, 7, 37, NCH_FLOW, false);
13     SimpleCentrifugalChiller iR = new SimpleCentrifugalChiller(500d / 6d, 0.2, 12, 7, 37, NCH_FLOW, true);
14     cR.IsOperating = iR.IsOperating = true;
15
16     sWriter.Write("負荷率");
17     for (int i = 0; i < 6; i++) sWriter.Write(",固+固" + (32 - 4 * i));
18     for (int i = 0; i < 6; i++) sWriter.Write(",INV+固" + (32 - 4 * i));
19     for (int i = 0; i < 6; i++) sWriter.Write(",固+変" + (32 - 4 * i));
20     for (int i = 0; i < 6; i++) sWriter.Write(",INV+変" + (32 - 4 * i));
21     sWriter.WriteLine();
22     for (int i = 0; i < 10; i++)
23     {
24       //負荷率[-]
25       double pl = 1.0 - 0.1 * i;
26       sWriter.Write(pl);
27
28       //固定速機・冷却水量固定
29       for (int j = 0; j < 6; j++)
30       {
31          cR.Update(32 - 4 * j, 12, NCD_FLOW, NCH_FLOW * pl);
32          sWriter.Write(", " + cR.COP.ToString("F2"));
33       }
34       //INV 機・冷却水量固定
```

```
35        for (int j = 0; j < 6; j++)
36        {
37          iR.Update(32 - 4 * j, 12, NCD_FLOW, NCH_FLOW * pl);
38          sWriter.Write(", " + iR.COP.ToString("F2"));
39        }
40        //固定速機・冷却水量変動
41        for (int j = 0; j < 6; j++)
42        {
43          cR.Update(32 - 4 * j, 12, NCD_FLOW * pl, NCH_FLOW * pl);
44          sWriter.Write(", " + cR.COP.ToString("F2"));
45        }
46        //INV機・冷却水量変動
47        for (int j = 0; j < 6; j++)
48        {
49          iR.Update(32 - 4 * j, 12, NCD_FLOW * pl, NCH_FLOW * pl);
50          sWriter.Write(", " + iR.COP.ToString("F2"));
51        }
52        sWriter.WriteLine();
53      }
54    }
55 }
```

14.3.2　理論式にもとづくターボ冷凍機の計算

1）モデルの構造[14.19)]

完全に物理式のみによるモデルでは柔軟性が小さいため、蒸発器・凝縮器・圧縮機の物理モデルから冷凍サイクルを解き、断熱圧縮仕事からエネルギー消費量を推定するモデルとする。具体的には式14.39で示される特性式にもとづいて消費電力を計算する。右辺第一項は式14.7に示した圧縮機入力E_{cmp}である。第二項は定格消費電力E_Nに依存する項であり、負荷率に比例する部分である$a_{pl}\cdot pl$と固定的な部分であるb_{pl}に分けられるという想定をしている。

$$E = W_{cmp}/\eta_{cmp} + (a_{pl}\cdot pl + b_{pl})E_N \tag{14.39}$$

式14.39を適用するためには断熱圧縮仕事と効率の計算が必要であり、以下にその方法を示す。

2）冷凍サイクルの計算

まず、モデルを初期化するために凝縮器および蒸発器における伝熱性能を把握する必要がある。冷凍機の定格性能値を式14.15と式14.25に代入し、$K_{cd}A_{cd}$ [kW/K]および$K_{ch}A_{ch}$ [kW/K]を計算する。熱交換量、冷水および冷却水の流量に関しては定格性能値としてカタログ等から情報を得ることができるが、通常は凝縮温度および蒸発温度の値は明記されていない。一般的には、凝縮温度は空冷式の場合には外気温度+12~20°C、水冷式の場合には冷却水出口温度+1~5°Cである[14.2)]。また、蒸発温度は冷水出口温度-2°C程度である。本書のモデルではメーカー技術資料などを参考に、凝縮温度を冷却水出口温度+1°C、蒸発温度を冷水出口温度-2 °Cとする。

【例題14.4】

例題14.2の冷凍機について、凝縮器の$K_{cd}A_{cd}$を計算せよ。

【解】

冷却水出口温度は37°Cであるため、凝縮温度T_{cd}は37+1=38°Cとする。従って、凝縮器出入口での流体温度差ΔT_{cdi}とΔT_{cdo}は式14.13と式14.14によりそれぞれ38-32=6°Cと38-37=1°Cである。これらを式14.12に代入し、対数平均温度差$\Delta T_{LM,cd}$は(37-32)÷ln(6÷1)=2.8°Cとなる。例題14.2により凝縮器の放熱量Q_{cd}は583kWであるため、これと対数平均温度差$\Delta T_{LM,cd}$を式14.9に代入して、

$K_{cd}A_{cd} = Q_{cd} \div \Delta T_{LM,cd} = 583 \div 2.8 = 208$ kW/K

となる。

冷凍サイクルは図14.4に示すように、近年、空調用ターボ冷凍機で採用が多い2段圧縮1段エコノマイザサイクルを前提とする。

以下に示すように、凝縮器と蒸発器における熱交換量が与えられれば、図14.4の冷凍サイクルの

各運転点を求めることができる。まず、冷媒物性計算により、蒸発温度 T_{evp} と凝縮温度 T_{cnd} から蒸発圧力 P_{evp} と P_{cnd} を計算する。圧縮機の 1 段目羽根車と 2 段目羽根車の圧縮比が等しいと仮定すれば、中間圧力 P_{mid} は式 14.40 で計算できる。

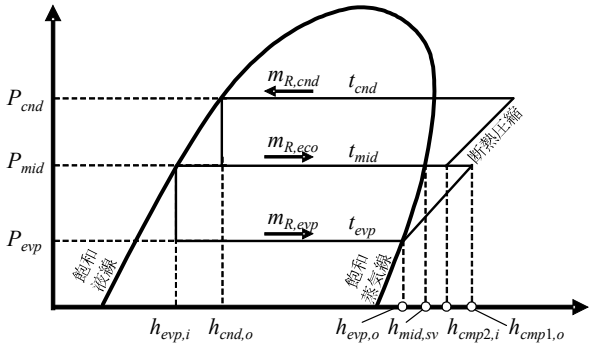


図 14.4　2 段圧縮 1 段エコノマイザサイクル

$$P_{mid}=\sqrt{(P_{evp}\cdot P_{cnd})} \tag{14.40}$$

中間圧力では一部の冷媒を蒸発させることで残余の冷媒を飽和液線上まで冷却する。このときの冷媒比エンタルピーが蒸発器における入口比エンタルピー $h_{evp,i}$ である。また、低圧 P_{evp} における飽和蒸気線上の比エンタルピー $h_{evp,o}$ が蒸発器の出口比エンタルピーであるから、これらを式 14.19 に代入すれば蒸発器を通過する冷媒質量流量 $m_{R,evp}$ が計算できる。また、これは圧縮機の 1 段目羽根車の吸込冷媒であり、冷媒物性計算によりこの点の冷媒密度 $\rho_{evp,o}$ を求めて体積流量 $v_{R,evp,o}$ に変換すれば、式 14.5 から断熱圧縮仕事 W_{cmp1} を計算できる。また、断熱圧縮仕事を式 14.6 に代入すれば 1 段目羽根車出口比エンタルピー $h_{cmp1,o}$ が求められる。

中間圧力で蒸発させる冷媒の質量流量 $m_{R,eco}$ は式 14.41 で計算する。これと蒸発器通過冷媒とが合流して 2 段目羽根車に吸い込まれるため、凝縮器の冷媒質量流量は式 14.42 となる。また、2 段目羽根車の入口冷媒比エンタルピーは式 14.43 で計算できる。これを式 14.5 に代入して 2 段目羽根車の断熱圧縮仕事 W_{cmp2} を計算する。式 14.44 で 1 段目羽根車と 2 段目羽根車の断熱圧縮仕事を合算し、冷凍サイクル全体の断熱圧縮仕事 W_{cmp} とする。

$$m_{R,eco}=m_{R,evp}\frac{h_{cnd,o}-h_{evp,i}}{h_{mid,sv}-h_{cnd,o}} \tag{14.41}$$

$$m_{R,cnd}=m_{R,evp}+m_{R,eco} \tag{14.42}$$

$$h_{cmp2,i}=\frac{h_{cmp1,o}\cdot m_{R,evp}+h_{mid,sv}\cdot m_{R,eco}}{m_{R,cnd}} \tag{14.43}$$

$$W_{cmp}=W_{cmp,1}+W_{cmp,2} \tag{14.44}$$

3）効率 η_{cmp} の推定

一般にターボ圧縮機の断熱効率特性は図 14.5 左に示すように極大点を持つ。また、図 14.5 右に示すように、この極大点は回転数に応じて移動する。一般化した断熱効率特性式とするためには、無次元化した流量係数と圧力係数に変換した上で両者の関係式を作成する方法がある [14.11)]。しかし、このためには代表長さや回転数を知る必要があり、パラメータ推定が非常に煩雑になる。そこで式 14.45 と式 14.46 に示すように、各回転数において最大効率を示す冷媒体積流量 v_{RM}[m^3/s]と実際の冷媒体積

流量 v_R[m³/s]との比率である冷媒流量比 R_{vR}[-]を用いて効率特性を表現することとする（a_{Rv}、b_{Rv}、c_{Rv}は係数）。ただし、最大効率を示す冷媒体積流量 v_{RM}は断熱ヘッドを用いて式 14.47 で推定する。ここで、$v_{R,N}$ [m³/s]と $W_{cmp,N}$ [kW]はそれぞれ定格条件における冷媒体積流量と断熱圧縮仕事である（a_{Wcmp}と b_{Wcmp}は係数）。式 14.39、式 14.45、式 14.47 の係数は冷凍機の機種ごとに推定することが望ましいが、機器ごとの細かな特性が不要な初期検討などの場面では、表 14.2 に示すインバータターボ冷凍機汎用モデルパラメータを用いても良い†1)。

$$\eta_{cmp}=1/(a_{Rv}\cdot R_{vR}^2+b_{Rv}\cdot R_{vR}+c_{Rv}) \tag{14.45}$$

$$R_{vR}=v_R/v_{RM} \tag{14.46}$$

$$v_{RM}/v_{R,N}=a_{Wcmp}\sqrt{(W_{cmp}/W_{cmp,N})}+b_{Wcmp} \tag{14.47}$$

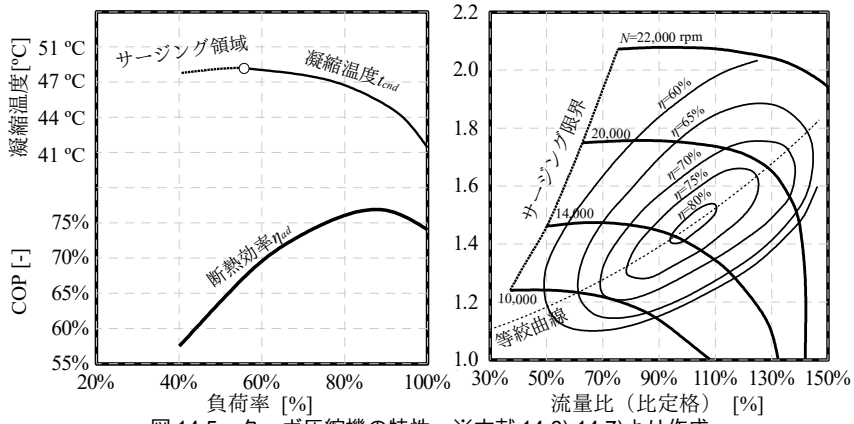


図 14.5　ターボ圧縮機の特性　※文献 14.6) 14.7)より作成

表 14.2　汎用インバータターボ冷凍機用パラメータ

a_{Wcmp}	b_{Wcmp}	a_{Rv}	b_{Rv}	c_{Rv}	a_{pl}	b_{pl}
0.8553	0.0590	1.8515	3.9452	3.4178	0.0945	0.0438

4）計算フロー

実際には凝縮器および蒸発器で処理する熱量は未知であるため、収束計算が必要となる。計算フローを図 14.6 に示す。まず、過負荷の判定を行うために消費電力は定格消費電力 E_N、冷凍能力は必要冷凍能力 Q_{chs}とし、凝縮器と蒸発器で処理する熱量 Q_{cnd}と Q_{evp}を設定する。必要冷却能力 Q_{chs}は式 14.17 で冷水出口温度 T_{cho}に冷水出口温度設定値 t_{chos}を代入して求める。前記の方法により冷凍サイクルを計算して断熱圧縮仕事を求め、式 14.39 により消費電力 E を計算する。得られた消費電力 E が定格消費電力 E_Nよりも小さければ最大能力範囲内、大きければ過負荷状態である。最大能力範囲内であれば必要冷却能力 Q_{chs}を固定して消費電力 E を収束計算で求める。過負荷状態であれば逆に消費電力 E を固定（$=E_N$）して冷却能力 Q_{ch}を収束計算で求める。いずれも解の存在範囲が明らかな一変数関数の求根問題であり、二分法や Brent 法などの囲い込み法で確実に解くことができる。また、誤差関数は滑らかであるため、ニュートン・ラフソン法であっても発散する可能性は高くない。

†1　パラメータ推定法の詳細に関しては文献 14.19 を参照。

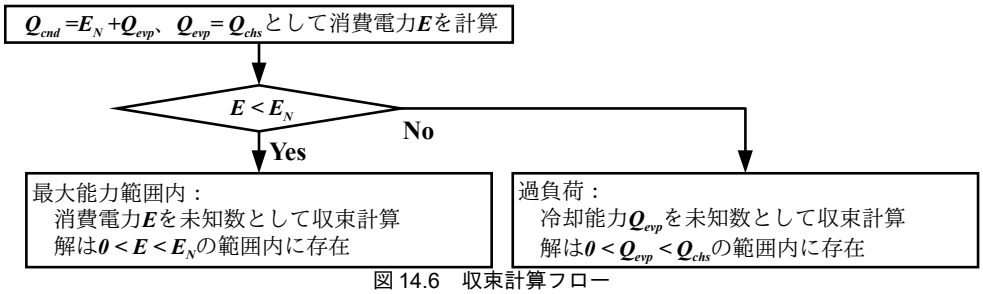


図 14.6　収束計算フロー

5）「理論式によるインバータターボ冷凍機クラス」の作成

理論式を用いたインバータターボ冷凍機クラスを作成する。インスタンス変数およびプロパティは簡易圧縮式冷凍機クラスは同じため、省略する。プログラム 14.6 にコンストラクタを示す。コンストラクタでは蒸発器と凝縮器の伝熱係数 KA、定格条件での冷媒体積流量 v_{RN}、断熱圧縮仕事 W_{cmpN} の計算を行う。28 行で表 14.2 に示した汎用パラメータで初期化を行う。無次元にしているため、本パラメータは冷凍機の容量によらず適用できる。30~44 行で蒸発器および凝縮器の伝熱係数 KA を求める。手順は例題 14.4 に示した通りである。47 行は断熱圧縮仕事および冷媒体積流量を計算するメソッドであり、後述する。

プログラム 14.6　コンストラクタ

Popolo.HVAC.HeatSource.DetailedCentrifugalInverterChiller class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="nominalInput">定格入力[kW]</param>
 3 /// <param name="minimumPartialLoadRatio">容量制御下限値[-]</param>
 4 /// <param name="chilledWaterInletTemperature">冷水入口温度[C]</param>
 5 /// <param name="chilledWaterOutletTemperature">冷水出口温度[C]</param>
 6 /// <param name="coolingWaterInletTemperature">冷却水入口温度[C]</param>
 7 /// <param name="chilledWaterFlowRate">冷水質量流量[kg/s]</param>
 8 /// <param name="coolingWaterFlowRate">冷却水質量流量[kg/s]</param>
 9 public DetailedCentrifugalInverterChiller
10   (double nominalInput, double minimumPartialLoadRatio,
11   double chilledWaterInletTemperature, double chilledWaterOutletTemperature,
12   double coolingWaterInletTemperature, double chilledWaterFlowRate, double coolingWaterFlowRate)
13 {
14   //定格能力などを保存
15   this.CoolingWaterFlowRate = coolingWaterFlowRate;
16   this.ChilledWaterFlowRate = chilledWaterFlowRate;
17   this.MaxChilledWaterFlowRate = chilledWaterFlowRate;
18   double mccd = CoolingWaterFlowRate * WATER_SPECIFIC_HEAT;
19   double mcch = ChilledWaterFlowRate * WATER_SPECIFIC_HEAT;
20   this.ChilledWaterOutletSetPointTemperature = chilledWaterOutletTemperature;
21   this.CoolingWaterInletTemperature = coolingWaterInletTemperature;
22   this.ChilledWaterInletTemperature = chilledWaterInletTemperature;
23   this.NominalCapacity = mcch * (chilledWaterInletTemperature - chilledWaterOutletTemperature);
24   this.NominalInput = nominalInput;
25   this.MinimumPartialLoadRatio = minimumPartialLoadRatio;
26
27   //特性係数初期化//汎用モデルで固定
28   coef = new double[] { 0.8553, 0.0590, 1.8515, -3.9452, 3.4178, -0.0945, 0.0438 };
29
30   //蒸発器の熱伝達率[kW/K]を計算
31   double tEvp = ChilledWaterOutletSetPointTemperature - 2;
32   double dt1 = ChilledWaterInletTemperature- tEvp;
33   double dt2 = ChilledWaterOutletSetPointTemperature - tEvp;
34   double lmtd = (dt1 - dt2) / Math.Log(dt1 / dt2);
35   evaporatorHeatTransferCoefficient = NominalCapacity / lmtd;
36
37   //凝縮器の熱伝達率[kW/K]を計算
38   double qcd = NominalCapacity + NominalInput;
39   CoolingWaterOutletTemperature = CoolingWaterInletTemperature + qcd / mccd;
40   double tCnd = CoolingWaterOutletTemperature + 1;
41   dt1 = tCnd - CoolingWaterInletTemperature;
42   dt2 = tCnd - CoolingWaterOutletTemperature;
43   lmtd = (dt1 - dt2) / Math.Log(dt1 / dt2);
44   condenserHeatTransferCoefficient = qcd / lmtd;
45
```

```
46   //定格断熱圧縮仕事[kW]と冷媒流量[m3/s]を計算
47   getHeadAndFlowVolume(
48     qcd, NominalCapacity, CoolingWaterInletTemperature, ChilledWaterInletTemperature,
49     condenserHeatTransferCoefficient, evaporatorHeatTransferCoefficient, mccd, mcch, coef,
50     out nominalHead, out nominalFlowVolume);
51 }
```

プログラム 14.7 に断熱圧縮仕事および冷媒体積流量の計算処理を示す。2）で示した冷凍サイクルの計算処理に従う。17~30 行で式 14.15 と式 14.25 を用いて蒸発温度と凝縮温度を求め、冷媒物性計算により蒸発圧力および凝縮圧力を求める。32~36 行は中間圧力の計算であり、式 14.40 である。39~41 行で、式 14.19 を用いて蒸発器を通過する冷媒質量流量を求める。冷媒物性計算により密度を求め、質量流量を体積流量に換算した後、47 行で 1 段目羽根車の断熱圧縮仕事を計算する（式 14.5）。50, 51 行で式 14.41 と式 14.43 にもとづいてエコノマイザーへ向かう冷媒流量と、圧縮機の 2 段目羽根車入口比エンタルピーを計算し、53~57 行で 2 段目羽根車の断熱圧縮仕事を計算する。

プログラム 14.7　断熱圧縮仕事および冷媒体積流量の計算処理

Popolo.HVAC.HeatSource. DetailedCentrifugalInverterChiller class

```
 1 /// <summary>断熱圧縮仕事[kW]と体積流量[m3/s]を計算する</summary>
 2 /// <param name="qcd">凝縮器熱交換量[kW]</param>
 3 /// <param name="qch">蒸発器熱交換量[kW]</param>
 4 /// <param name="tcdi">冷却水入口温度[C]</param>
 5 /// <param name="tchi">冷水入口温度[C]</param>
 6 /// <param name="kacd">凝縮器熱通過率[kW/K]</param>
 7 /// <param name="kach">蒸発器熱通過率[kW/K]</param>
 8 /// <param name="mccd">凝縮器熱容量流量[kW/K]</param>
 9 /// <param name="mcch">蒸発器熱容量流量[kW/K]</param>
10 /// <param name="coefs">特性係数</param>
11 /// <param name="head">出力:断熱圧縮仕事[kW]</param>
12 /// <param name="volume">出力:体積流量[m3/s]</param>
13 private static void getHeadAndFlowVolume
14   (double qcd, double qch, double tcdi, double tchi, double kacd, double kach,
15   double mccd, double mcch, double[] coefs, out double head, out double volume)
16 {
17   //冷媒は R134a
18   Refrigerant r134a = new Refrigerant(Refrigerant.Fluid.R134a);
19
20   //凝縮温度・圧力を計算
21   double tCnd = tcdi + qcd / ((1 - Math.Exp(-kacd / mccd)) * mccd);
22   double dlCnd, dvCnd, pCnd;
23   r134a.GetSaturatedPropertyFromTemperature(tCnd + 273.15, out dlCnd, out dvCnd, out pCnd);
24   double hoCnd = r134a.GetEnthalpyFromTemperatureAndDensity(tCnd + 273.15, dlCnd);
25
26   //蒸発温度・圧力を計算
27   double tEvp = tchi - qch / ((1 - Math.Exp(-kach / mcch)) * mcch);
28   double dlEvp, dvEvp, pEvp;
29   r134a.GetSaturatedPropertyFromTemperature(tEvp + 273.15, out dlEvp, out dvEvp, out pEvp);
30   double hiCmp1 = r134a.GetEnthalpyFromTemperatureAndDensity(tEvp + 273.15, dvEvp);
31
32   //中間圧力を計算
33   double pMid = Math.Sqrt(pEvp * pCnd);
34   double dlMid, dvMid, tMid;
35   r134a.GetSaturatedPropertyFromPressure(pMid, out dlMid, out dvMid, out tMid);
36   double hsvMid = r134a.GetEnthalpyFromTemperatureAndDensity(tMid, dvMid);
37
38   //蒸発器冷媒流量を計算
39   double hiEvp = r134a.GetEnthalpyFromTemperatureAndDensity(tMid, dlMid);
40   double mRevp = qch / (hiCmp1 - hiEvp);
41   volume = mRevp / dvEvp;
42
43   //1 段目羽根車の断熱ヘッド[kW]と出口比エンタルピー[kJ/kg]を計算
44   double kappa = r134a.GetSpecificHeatRatioFromTemperatureAndDensity(tEvp + 273.15, dvEvp);
45   kappa = (kappa - 1) / kappa;
46   double head1 = volume / kappa * pEvp * (Math.Pow(pMid / pEvp, kappa) - 1);
47   double hoCmp1 = hiCmp1 + head1 / mRevp;
48
49   //2 段目羽根車断熱ヘッド[kW]を計算
50   double mRe = (hoCnd - hiEvp) / (hsvMid - hoCnd) * mRevp;
51   double hiCmp2 = (hoCmp1 * mRevp + hsvMid * mRe) / (mRevp + mRe);
52   double tiCmp2, diCmp2, siCmp2, uiCmp2;
53   r134a.GetStateFromPressureAndEnthalpy(pMid, hiCmp2, out tiCmp2, out diCmp2, out siCmp2, out uiCmp2);
54   double vRCmp2 = (mRevp + mRe) / diCmp2;
55   kappa = r134a.GetSpecificHeatRatioFromTemperatureAndDensity(tiCmp2 + 273.15, diCmp2);
```

```
56   kappa = (kappa - 1) / kappa;
57   double head2 = vRCmp2 / kappa * pMid * (Math.Pow(pCnd / pMid, kappa) - 1);
58
59   //圧縮機入力[kW]の計算
60   head = head1 + head2;
61 }
```

プログラム 14.8 に状態更新処理を示す。図 14.6 に示した計算フローの実装である。27~29 行で必要冷却能力 Q_{chs} を求め、32~37 行で消費電力 E を求める。37 行は断熱圧縮仕事などから消費電力を計算するメソッドであり、79~88 行に示す通り、式 14.39 の実装である。図 14.6 に示したように、E が定格消費電力 E_N と比較して大きいか否かで収束計算の対象を切り分け、最大能力の範囲内の場合には 42~52 行、過負荷の場合には 57~66 行を実行する。収束計算にはニュートン・ラフソン法を用いる。

プログラム 14.8　状態更新処理

Popolo.HVAC.HeatSource. DetailedCentrifugalInverterChiller class

```
 1 /// <summary>状態を更新する</summary>
 2 /// <param name="coolingWaterInletTemperature">冷却水入口温度[C]</param>
 3 /// <param name="chilledWaterInletTemperature">冷水入口温度[C]</param>
 4 /// <param name="coolingWaterFlowRate">冷却水質量流量[kg/s]</param>
 5 /// <param name="chilledWaterFlowRate">冷水質量流量[kg/s]</param>
 6 public void Update
 7   (double coolingWaterInletTemperature, double chilledWaterInletTemperature,
 8   double coolingWaterFlowRate, double chilledWaterFlowRate)
 9 {
10   //状態値を保存
11   this.ChilledWaterInletTemperature = chilledWaterInletTemperature;
12   this.CoolingWaterInletTemperature = coolingWaterInletTemperature;
13   this.ChilledWaterFlowRate = chilledWaterFlowRate;
14   this.CoolingWaterFlowRate = coolingWaterFlowRate;
15
16   //非稼働ならば停止処理
17   if (!IsOperating)
18   {
19     ShutOff();
20     return;
21   }
22
23   //熱容量流量[kW/K]を計算
24   double mccd = CoolingWaterFlowRate * WATER_SPECIFIC_HEAT;
25   double mcch = ChilledWaterFlowRate * WATER_SPECIFIC_HEAT;
26
27   //必要能力[kW]を計算
28   CoolingLoad = mcch * (ChilledWaterInletTemperature - ChilledWaterOutletSetPointTemperature);
29   double partialLoad = Math.Max(MinimumPartialLoadRatio, CoolingLoad / NominalCapacity);
30
31   //過負荷判定
32   double head, volume;
33   getHeadAndFlowVolume(
34     CoolingLoad + NominalInput, CoolingLoad, CoolingWaterInletTemperature, ChilledWaterInletTemperature,
35     condenserHeatTransferCoefficient, evaporatorHeatTransferCoefficient, mccd, mcch, coef,
36     out head, out volume);
37   ElectricConsumption = getElectricity(head, volume, partialLoad);
38
39   //最大能力範囲内:消費電力を収束計算
40   if (ElectricConsumption < NominalInput)
41   {
42     IsOverLoad = false;
43     Roots.ErrorFunction eFnc = delegate (double econs)
44     {
45       getHeadAndFlowVolume(CoolingLoad + econs, CoolingLoad, CoolingWaterInletTemperature,
46         ChilledWaterInletTemperature, condenserHeatTransferCoefficient, evaporatorHeatTransferCoefficient,
47         mccd, mcch, coef, out head, out volume);
48       return getElectricity(head, volume, partialLoad) - econs;
49     };
50     double eTol = NominalInput * 1e-4;  //定格消費電力の 0.01%までの誤差を許容
51     ElectricConsumption = Roots.Newton(eFnc, ElectricConsumption, 1e-4, eTol, eTol, 10);
52     ChilledWaterOutletTemperature = ChilledWaterOutletSetPointTemperature;
53   }
54   //過負荷//冷却能力を収束計算
55   else
56   {
57     IsOverLoad = true;
58     Roots.ErrorFunction eFnc = delegate (double cLoad)
59     {
```

```
60        getHeadAndFlowVolume(cLoad + NominalInput, cLoad, CoolingWaterInletTemperature,
61          ChilledWaterInletTemperature, condenserHeatTransferCoefficient, evaporatorHeatTransferCoefficient,
62          mccd, mcch, coef, out head, out volume);
63        return getElectricity(head, volume, partialLoad) - NominalInput;
64      };
65      CoolingLoad = Roots.Newton(eFnc, NominalCapacity, 1e-4, NominalInput * 1e-4, NominalCapacity * 1e-4, 10);
66      ChilledWaterOutletTemperature = ChilledWaterInletTemperature - CoolingLoad / mcch;
67    }
68 }
69
70 /// <summary>機器を停止させる</summary>
71 public void ShutOff()
72 {
73   ChilledWaterOutletTemperature = ChilledWaterInletTemperature;
74   CoolingWaterOutletTemperature = CoolingWaterInletTemperature;
75   CoolingLoad = ChilledWaterFlowRate = CoolingWaterFlowRate = 0;
76   ElectricConsumption = 0;
77 }
78
79 /// <summary>消費電力[kW]を計算する</summary>
80 /// <param name="head">断熱圧縮仕事[kW]</param>
81 /// <param name="volume">体積流量[m3/s]</param>
82 /// <param name="partialLoad">負荷率[-]</param>
83 /// <returns>消費電力[kW]</returns>
84 private double getElectricity(double head, double volume, double partialLoad)
85 {
86   double rvR = volume / ((Math.Sqrt(head / nominalHead) * coef[0] + coef[1]) * nominalFlowVolume);
87   return head * (coef[4] + rvR * (coef[3] + rvR * coef[2])) + NominalInput * (partialLoad * coef[5] + coef[6]);
88 }
```

例題 14.3 と同様に、冷却水流量と負荷率を変動させた場合の特性を確認する。計算処理をプログラム 14.9 に、計算結果を図 14.7 に示す。傾向は特性モデルと同様である。

プログラム 14.9　インバーターターボ冷凍機の部分負荷特性の計算

```
 1 private static void DetailedCentrifugalInverterChillerTest()
 2 {
 3   using (StreamWriter sWriter = new StreamWriter
 4     ("DetailedCentrifugalInverterChillerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
 5   {
 6     //インスタンスを生成
 7     DetailedCentrifugalInverterChiller iR =
 8       new DetailedCentrifugalInverterChiller(178.8, 0.2, 12, 7, 32, 3023d / 60d, 3567d / 60d);
 9
10     const double NCH_FLOW = 3023d / 60d;  //定格冷水量[kg/s]
11     const double NCD_FLOW = 3567d / 60d;  //定格冷却水量[kg/s]
12
13     sWriter.Write("負荷率");
14     for (int i = 0; i < 6; i++) sWriter.Write(",INV+固" + (32 - 4 * i));
15     for (int i = 0; i < 6; i++) sWriter.Write(",INV+変" + (32 - 4 * i));
16     sWriter.WriteLine();
17     for (int i = 0; i < 10; i++)
18     {
19       //負荷率[-]
20       double pl = 1.0 - 0.1 * i;
21       sWriter.Write(pl);
22
23       //冷却水量固定
24       for (int j = 0; j < 6; j++)
25       {
26         iR.Update(32 - 4 * j, 12, NCD_FLOW, NCH_FLOW * pl);
27         sWriter.Write(", " + iR.COP.ToString("F2"));
28       }
29       //冷却水量変動
30       for (int j = 0; j < 6; j++)
31       {
32         iR.Update(32 - 4 * j, 12, NCD_FLOW * pl, NCH_FLOW * pl);
33         sWriter.Write(", " + iR.COP.ToString("F2"));
34       }
35       sWriter.WriteLine();
36     }
37   }
38 }
```

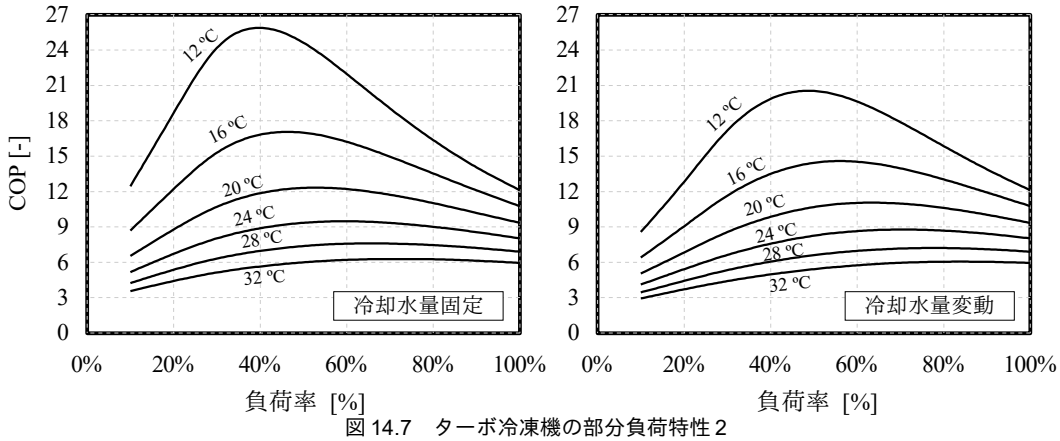


図 14.7　ターボ冷凍機の部分負荷特性 2

14.3.3　特性式にもとづく空気熱源ヒートポンプの計算

1）モジュール単体の計算

ターボ冷凍機の特性式モデルと同様に、空気と水の出口温度にもとづく理論 COP で機器特性を記述する。式 14.48 に特性式を示す。$R_{COP,FL}$ [-]と $R_{COP,FLT}$ [-]はそれぞれ式 14.49 と 14.50 に示すように、実際の COP と理論 COP を定格性能値で除して無次元化した値である。理論 COP は出口水温 T_{wo} [K]と出口空気温度 T_{ao} [K]を用いて式 14.51 で計算する。

$$R_{COP,FI}=\alpha_{COP,0}R_{COP,FIT}^{-2}+\alpha_{COP,1}R_{COP,FIT}^{-1}+\alpha_{COP,2} \tag{14.48}$$

$$R_{COP,FI}=COP_{FI}/COP_{FI,N} \tag{14.49}$$

$$R_{COP,FIT}=COP_{FIT}/COP_{FIT,N} \tag{14.50}$$

$$COP_{FIT}=\frac{T_{wo}}{T_{ao}-T_{wo}} \tag{14.51}$$

空気熱源ヒートポンプは冷却運転時には蒸発器側流体が水、凝縮器側流体が空気となり、加熱運転時には逆に蒸発器側流体が空気、凝縮器側流体が水となる。加熱運転と冷却運転の両方に汎用的に適用できる式にするため、冷却運転時に-1、加熱運転時に+1 をとる係数 s_g を導入する。加熱または冷却負荷を Q_{LD} [kW]とすると、空気の出入口温度は式 14.52 で表現される。

$$mc_{ma}(T_{ao}-T_{ai})=E-s_gQ_{LD} \tag{14.52}$$

消費電力 E は負荷を COP で除して式 14.53 で計算する。ただし、COP_{Pl} [-]は部分負荷を考慮した実際の COP であり、全負荷での COP_{FI} [-]に負荷率補正係数 R_{Pl} [-]を乗じた値である。R_{Pl} の計算法は後述する。

$$E=\frac{Q_{LD}}{COP_{Pl}}=\frac{Q_{LD}}{R_{Pl}COP_{FI}} \tag{14.53}$$

式 14.48~式 14.53 を整理すると空気出口温度 T_{ao} の多項式である式 14.54 が得られる。ただし係数は式 14.55~式 14.62 に示すとおりである。式 14.54 は 3 次式であるため、カルダーノの方法を用いて代数的に解を得ることができる。

$$\alpha_{Tao,0}T_{ao}^3+\alpha_{Tao,1}T_{ao}^2+\alpha_{Tao,2}T_{ao}+\alpha_{Tao,3}=0 \tag{14.54}$$

$$\alpha_{Tao,0}=\alpha_{Tao,bf0}^2\alpha_{COP,0}mc_{ma} \tag{14.55}$$

$$\alpha_{Tao,1}=-\alpha_{Tao,bf0}\left(mc_{ma}\alpha_{Tao,bf1}+\alpha_{Tao,bf0}\alpha_{COP,0}\alpha_{Tao,bf3}\right) \tag{14.56}$$

$$\alpha_{Tao,2}=\alpha_{Tao,bf0}\cdot\alpha_{Tao,bf1}\cdot\alpha_{Tao,bf3}+mc_{ma}\cdot\alpha_{Tao,bf2} \tag{14.57}$$

$$\alpha_{Tao,3}=-\alpha_{Tao,bf2}\cdot\alpha_{Tao,bf3}-\frac{Q_{LD}}{R_{Pl}COP_{Fl}} \tag{14.58}$$

$$\alpha_{Tao,bf0}=COP_{FITR}/T_{wo} \tag{14.59}$$

$$\alpha_{Tao,bf1}=2\alpha_{COP,0}COP_{FIT,N}+s_g\alpha_{COP,1} \tag{14.60}$$

$$\alpha_{Tao,bf2}=\alpha_{COP,0}COP_{FIT,N}^2+s_g\alpha_{COP,1}COP_{FIT,N}+\alpha_{COP,2} \tag{14.61}$$

$$\alpha_{Tao,bf3}=mc_{ma}T_{ai}-s_gQ_{LD} \tag{14.62}$$

水熱源に比較すると空気熱源の場合には、熱媒である空気の温度の変動幅が大きく、これに伴って機器の能力が大きく変化する。そこで最大能力 Q_{max} [kW]は固定値とせず、空気と水の温度にもとづいて式 14.63 で計算する。負荷率 Pl は式 14.64 で計算でき、これを用いて部分負荷による補正係数 R_{Pl} を式 14.65 で計算する。

$$Q_{max}=\alpha_{max,0}T_{ai}T_{wo}+\alpha_{max,1}T_{ai}+\alpha_{max,2}T_{wo}+\alpha_{max,3} \tag{14.63}$$

$$Pl=Q_{LD}/Q_{max} \tag{14.64}$$

$$R_{Pl}=\alpha_{Pl,0}Pl^2+\alpha_{Pl,1}Pl+\alpha_{Pl,2} \tag{14.65}$$

国内の空気熱源ヒートポンプ製造業者 4 社のカタログから回帰して作成した特性係数を表 14.3 に示す。部分負荷特性に関しては通常はカタログには記載が無い。そこで空気熱源ヒートポンプで通常用いられる密閉型スクロール圧縮機の効率特性[14.21)]を用いて $\alpha_{pl,0}$ = -1.4118、$\alpha_{pl,1}$ = 1.3611、$\alpha_{pl,2}$ = 1.0507 と推定した。ただし近似式の適用範囲は Pl=30 %までとし、これ未満の範囲は原点に向けて直線補間を行う。30 %未満は急速に効率が低下するという仮定である。

表 14.3　汎用空気熱源ヒートポンプの COP と最大能力の特性係数

	$\alpha_{COP,0}$	$\alpha_{COP,1}$	$\alpha_{COP,2}$	$\alpha_{max,0}$	$\alpha_{max,1}$	$\alpha_{max,2}$	$\alpha_{max,3}$
冷却運転	0.60824	-2.4486	2.8067	-4.1557e-04	1.0518e-01	1.5448e-01	-3.8823e01
加熱運転	0.088163	-1.3885	2.2981	9.2525e-06	1.7117e-02	-6.7223e-03	-2.4744

2）台数制御

空気熱源ヒートポンプの中には複数のユニットを連結させて総合的に制御する機種がある。このような機種をモジュールヒートポンプあるいはモジュールチラーと呼び、1 つ 1 つのユニットをモジュールと呼ぶ。運転台数の決定方法は大きく 2 つあり、1 つは運転効率を最大化しようとするもの、もう 1 つは運転時間を最小化しようとするものである。

近年のモジュールチラーは通常はインバータが組み込まれており、低負荷の場合に最大効率となる。従って効率を最大化するためには、最小限の運転台数として負荷率が 100 %に近い点で運転させるよりも、あえて必要以上の台数にすることで負荷率を低下させることが有効になる。このような観点にもとづく運転台数決定の概念を図 14.8 に示す。例えば全負荷 120 %から 200 %の範囲を見ると、2 台運転も可能ではあるが 3 台運転とすることで 1 台あたりの負荷率を減少させ、全体としての効率を向上させられることがわかる。式 14.65 の R_{Pl} を最大化する運転点を知るため、式 14.65 右辺の微分値が 0 となるように式を整理すると式 14.66 が得られる。軽負荷時には運転台数はこの負荷率に近づくように決定すればよい。参考に、前記の特性係数を使う場合には Pl=48 %程度で最大値をとる。

$$Pl = -\frac{\alpha_{Pl,1}}{2\alpha_{Pl,0}} \tag{14.66}$$

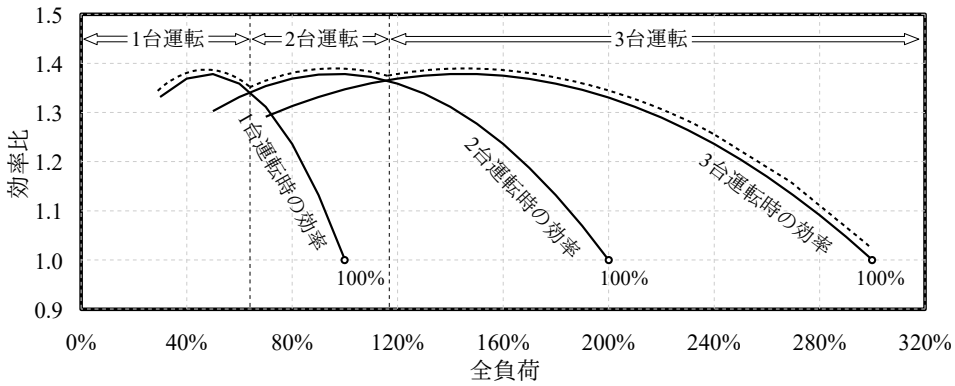


図 14.8　モジュールチラーの運転台数（機器効率最大化）

一方で、上記のような低負荷高効率での運転を狙うと機器の延運転時間は増大する。機器の寿命は運転時間に影響を受けるため、機器を長く使い続けるという観点からは必要不可欠な台数に絞ることで、できるだけ 100 %に近い負荷率で運転を行うという作戦も考えられる。この場合には、外気条件や冷温水温度から最大能力を算出し、必要な熱を最大能力で除することで台数を算出すればよい。

3）「空気熱源ヒートポンプクラス」の作成

空気熱源ヒートポンプクラスの列挙型定義と定数宣言をプログラム 14.10 に示す。運転状態として冷暖に加えて停止を定義する。12~25 行は各種の特性係数である。定格理論 COP はコンストラクタで初期化した後は変更しないため、28 行に示すように readonly による定数とする。

プログラム 14.10　列挙型の定義と定数宣言

Popolo.HVAC.HeatSource.AirHeatSourceModularChillers class

```
 1 /// <summary>運転モード</summary>
 2 public enum OperatingMode
 3 {
 4   /// <summary>加熱運転</summary>
 5   Heating,
 6   /// <summary>冷却運転</summary>
 7   Cooling,
 8   /// <summary>停止</summary>
 9   ShutOff
10 }
11
12 /// <summary>最大能力比近似係数（冷房運転）</summary>
13 private readonly double[] aMax_C = new double[] { -4.1557e-04, 1.0518e-01, 1.5448e-01, -3.8823e01 };
14
15 /// <summary>最大能力比近似係数（暖房運転）</summary>
16 private readonly double[] aMax_H = new double[] { 9.2525e-06, 1.7117e-02, -6.7223e-03, -2.4744 };
17
18 /// <summary>COP 比近似係数（冷房運転）</summary>
19 private readonly double[] aCop_C = new double[] { 0.60824, -2.4486, 2.8067 };
20
21 /// <summary>COP 比近似係数（暖房運転）</summary>
22 private readonly double[] aCop_H = new double[] { 0.088163, -1.3885, 2.2981 };
23
24 /// <summary>部分負荷特性係数</summary>
25 private readonly double[] a_PL = new double[] { -1.4118, 1.3611, 1.0507 };
26
27 /// <summary>定格理論 COP[-]</summary>
28 private readonly double copFLRT_C, copFLRT_H;
```

インスタンス変数およびプロパティをプログラム 14.11 に示す。

プログラム **14.11**　空気熱源ヒートポンプクラスのインスタンス変数およびプロパティの定義

Popolo.HVAC.HeatSource.AirHeatSourceModularChillers class

```
/// <summary>定格理論 COP[-]</summary>
private readonly double copFLRT_C, copFLRT_H;

/// <summary>風量[(kg/s)/台]</summary>
private double coolingAirFlowRate, heatingAirFlowRate;

/// <summary>補機消費電力[kW]</summary>
private double auxElec;

/// <summary>運転モードを設定・取得する</summary>
public OperatingMode Mode { get; set; }

/// <summary>運転効率[-]を最大化するか否か</summary>
public bool MaximizeEfficiency { get; set; } = true;

/// <summary>モジュール数[台]を取得する</summary>
public int NumberOfUnits { get; private set; }

/// <summary>稼働台数[台]を取得する s</summary>
public int OperatingNumber { get; private set; }

/// <summary>ヒートポンプ機か否かを取得する</summary>
public bool IsHeatPumpModel { get; private set; }

/// <summary>定格冷却能力[kW/台]を取得する</summary>
public double NominalCoolingCapacity { get; private set; }

/// <summary>定格冷却 COP[-]を取得する</summary>
public double NominalCoolingCOP { get; private set; }

/// <summary>定格加熱能力[kW/台]を取得する</summary>
public double NominalHeatingCapacity { get; private set; }

/// <summary>定格加熱 COP[-]を取得する</summary>
public double NominalHeatingCOP { get; private set; }

/// <summary>出口水温[C]を取得する</summary>
public double WaterOutletTemperature { get; private set; }

/// <summary>出口水温設定値[C]を設定・取得する</summary>
public double WaterOutletSetPointTemperature { get; set; }

/// <summary>入口水温[C]を取得する</summary>
public double WaterInletTemperature { get; private set; }

/// <summary>水流量[(kg/s)/台]を取得する</summary>
public double WaterFlowRate { get; private set; }

/// <summary>風量[(kg/s)/台]を取得する</summary>
public double AirFlowRate
{
  get
  {
    if (Mode == OperatingMode.Cooling) return coolingAirFlowRate;
    else if (Mode == OperatingMode.Heating) return heatingAirFlowRate;
    else return 0;
  }
}

/// <summary>消費電力[kW/台]を取得する</summary>
public double ElectricConsumption { get; private set; }

/// <summary>補機消費電力[kW/台]を取得する</summary>
public double AuxiliaryElectricConsumption { get; private set; }

/// <summary>周囲の空気乾球温度[C]を取得する</summary>
public double AmbientTemperature { get; private set; }

/// <summary>COP[-]を取得する</summary>
public double COP
{
  get
  {
    if (ElectricConsumption != 0)
      return Math.Abs(WaterOutletTemperature - WaterInletTemperature)
        * WATER_SPECIFIC_HEAT * WaterFlowRate / (ElectricConsumption * OperatingNumber);
    else return 0;
  }
```

```
 89 }
 90
 91 /// <summary>冷却量[kW]を取得する</summary>
 92 public double CoolingLoad
 93 {
 94   get
 95   {
 96     if (Mode == OperatingMode.Cooling)
 97       return (WaterInletTemperature - WaterOutletTemperature) * WATER_SPECIFIC_HEAT * WaterFlowRate;
 98     else return 0;
 99   }
100 }
101
102 /// <summary>加熱量[kW]を取得する</summary>
103 public double HeatingLoad
104 {
105   get
106   {
107     if (Mode == OperatingMode.Heating)
108       return (WaterOutletTemperature - WaterInletTemperature) * WATER_SPECIFIC_HEAT * WaterFlowRate;
109     else return 0;
110   }
111 }
112
113 /// <summary>冷水上限流量[kg/s]を取得する</summary>
114 public double MaxChilledWaterFlowRate { get; private set; }
115
116 /// <summary>温水上限流量[kg/s]を取得する</summary>
117 public double MaxHotWaterFlowRate { get; private set; }
118
119 /// <summary>冷水下限流量[kg/s]を取得する</summary>
120 public double MinChilledWaterFlowRate { get; private set; }
121
122 /// <summary>温水下限流量[kg/s]を取得する</summary>
123 public double MinHotWaterFlowRate { get; private set; }
124
125 /// <summary>過負荷か否か</summary>
126 public bool IsOverLoad { get; private set; }
```

プログラム 14.12 にコンストラクタを示す。1~32 行は冷房専用機であり冷房時の水空気条件などを引数とする。34~95 行はヒートポンプ可能な機種の初期化処理であり、暖房時の水空気条件も引数とする。15 行と 66 行に示すようにヒートポンプ可能か否かをプロパティに保存する。いずれの機種も定格の冷水あるいは温水条件から理論 COP を計算し、インスタンス変数に保存する。

プログラム 14.12　コンストラクタ

Popolo.HVAC.HeatSource.AirHeatSourceModularChillers class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="coolingCapacity">冷却能力[kW/台]</param>
 3 /// <param name="chilledWaterOutletTemperature">冷水出口温度[C]</param>
 4 /// <param name="chilledWaterFlowRate">冷水流量[(kg/s)/台]</param>
 5 /// <param name="coolingAirTemperature">冷却時空気温度[C]</param>
 6 /// <param name="coolingAirFlowRate">冷却時風量[(kg/s)/台]</param>
 7 /// <param name="coolingElectricity">冷却時消費電力[kW/台]</param>
 8 /// <param name="numberOfUnits">モジュール数[台]</param>
 9 /// <param name="auxiliaryElectricConsumption">補機消費電力[kW/台]</param>
10 public AirHeatSourceModularChillers(
11   double coolingCapacity, double chilledWaterOutletTemperature, double chilledWaterFlowRate,
12   double coolingAirTemperature, double coolingAirFlowRate, double coolingElectricity,
13   int numberOfUnits, double auxiliaryElectricConsumption)
14 {
15   IsHeatPumpModel = false;
16
17   this.coolingAirFlowRate = coolingAirFlowRate;
18   this.NominalCoolingCapacity = coolingCapacity;
19   this.NumberOfUnits = numberOfUnits;
20   this.auxElec = auxiliaryElectricConsumption;
21   this.heatingAirFlowRate = this.NominalHeatingCapacity = 0;
22
23   //冷房運転 COP の計算
24   double mcw = WATER_SPECIFIC_HEAT * chilledWaterFlowRate;
25   double mcma = (1.005 + 1.846 * 0.020) * coolingAirFlowRate;
26   double tao = coolingAirTemperature + (coolingElectricity + coolingCapacity) / mcma;
27   copFLRT_C = (chilledWaterOutletTemperature + 273.15) / (tao - chilledWaterOutletTemperature);
28   NominalCoolingCOP = coolingCapacity / coolingElectricity;
29
```

```
30   //停止させる
31   shutOff();
32 }
33
34 /// <summary>インスタンスを初期化する</summary>
35 /// <param name="coolingCapacity">冷却能力[kW/台]</param>
36 /// <param name="chilledWaterOutletTemperature">冷水出口温度[C]</param>
37 /// <param name="chilledWaterFlowRate">冷水流量[(kg/s)/台]</param>
38 /// <param name="coolingAirTemperature">冷却時空気温度[C]</param>
39 /// <param name="coolingAirFlowRate">冷却時風量[(kg/s)/台]</param>
40 /// <param name="coolingElectricity">冷却時消費電力[kW/台]</param>
41 /// <param name="heatingCapacity">加熱能力[kW/台]</param>
42 /// <param name="hotWaterOutletTemperature">温水出口温度[C]</param>
43 /// <param name="hotWaterFlowRate">温水流量[(kg/s)/台]</param>
44 /// <param name="heatingAirTemperature">加熱時空気温度[C]</param>
45 /// <param name="heatingAirFlowRate">加熱時風量[(kg/s)/台]</param>
46 /// <param name="heatingElectricity">加熱時消費電力[kW/台]</param>
47 /// <param name="numberOfUnits">モジュール数[台]</param>
48 /// <param name="auxiliaryElectricConsumption">補機消費電力[kW/台]</param>
49 public AirHeatSourceModularChillers(
50   double coolingCapacity, double chilledWaterOutletTemperature, double chilledWaterFlowRate,
51   double coolingAirTemperature, double coolingAirFlowRate, double coolingElectricity,
52   double heatingCapacity, double hotWaterOutletTemperature, double hotWaterFlowRate,
53   double heatingAirTemperature, double heatingAirFlowRate, double heatingElectricity,
54   int numberOfUnits, double auxiliaryElectricConsumption)
55 {
56   IsHeatPumpModel = true;
57
58   this.coolingAirFlowRate = coolingAirFlowRate;
59   this.heatingAirFlowRate = heatingAirFlowRate;
60   this.NominalCoolingCapacity = coolingCapacity;
61   this.NominalHeatingCapacity = heatingCapacity;
62   this.NumberOfUnits = numberOfUnits;
63   this.auxElec = auxiliaryElectricConsumption;
64   this.MaxChilledWaterFlowRate = chilledWaterFlowRate * numberOfUnits;
65   this.MaxHotWaterFlowRate = hotWaterFlowRate * numberOfUnits;
66   this.MinChilledWaterFlowRate = chilledWaterFlowRate * 0.4;
67   this.MinHotWaterFlowRate = hotWaterFlowRate * 0.4;
68
69   //冷房運転 COP の計算
70   double mcw = WATER_SPECIFIC_HEAT * chilledWaterFlowRate;
71   double mcma = (1.005 + 1.846 * 0.020) * coolingAirFlowRate;
72   double tao = coolingAirTemperature + (coolingElectricity + coolingCapacity) / mcma;
73   copFLRT_C = (chilledWaterOutletTemperature + 273.15) / (tao - chilledWaterOutletTemperature);
74   NominalCoolingCOP = coolingCapacity / coolingElectricity;
75
76   //暖房運転 COP の計算
77   mcw = WATER_SPECIFIC_HEAT * hotWaterFlowRate;
78   mcma = (1.005 + 1.846 * 0.002) * heatingAirFlowRate;
79   tao = heatingAirTemperature + (heatingElectricity - heatingCapacity) / mcma;
80   copFLRT_H = (hotWaterOutletTemperature + 273.15) / (hotWaterOutletTemperature - tao);
81   NominalHeatingCOP = heatingCapacity / heatingElectricity;
82
83   //停止させる
84   ShutOff();
85 }
```

プログラム 14.13 に機器の停止処理を示す。1~6 行がクラスの外部から呼び出す public メソッドであり、プロパティで定義した運転モードを ShutOff にした後、8~15 行の機器停止処理を呼び出す。

プログラム 14.13　機器の停止処理

Popolo.HVAC.HeatSource.AirHeatSourceModularChillers class

```
1 /// <summary>機器を停止させる</summary>
2 public void ShutOff()
3 {
4   Mode = OperatingMode.ShutOff;
5   shutOff();
6 }
7
8 /// <summary>機器を停止させる</summary>
9 private void shutOff()
10 {
11   WaterOutletTemperature = WaterInletTemperature;
12   ElectricConsumption = 0;
13   AuxiliaryElectricConsumption = 0;
14   OperatingNumber = 0;
15 }
```

プログラム 14.14 に最大能力の計算処理を示す。式 14.63 の実装であるが、10, 11 行と 16, 17 行に示

すように、出口水温と入口空気温度を機器特性の有効範囲内に納めてから式を適用する。

プログラム 14.14　最大能力の計算処理

Popolo.HVAC.HeatSource.AirHeatSourceModularChillers class

```
/// <summary>最大能力[kW/台]を計算する</summary>
/// <param name="waterOutletTemperature">出口水温[C]</param>
/// <param name="ambientTemperature">空気温度[C]</param>
/// <returns>最大能力[kW/台]</returns>
public double GetMaxCapacity(double waterOutletTemperature, double ambientTemperature)
{
  if (Mode == OperatingMode.ShutOff) return 0;
  else if (Mode == OperatingMode.Cooling)
  {
    double two = Math.Min(15, Math.Max(3, waterOutletTemperature)) + 273.15;
    double tai = Math.Min(43, Math.Max(20, ambientTemperature)) + 273.15;
    return NominalCoolingCapacity * (tai * (two * aMax_C[0] + aMax_C[1]) + two * aMax_C[2] + aMax_C[3]);
  }
  else
  {
    double two = Math.Min(55, Math.Max(35, waterOutletTemperature)) + 273.15;
    double tai = Math.Min(21, Math.Max(-15, ambientTemperature)) + 273.15;
    return NominalHeatingCapacity * (tai * (two * aMax_H[0] + aMax_H[1]) + two * aMax_H[2] + aMax_H[3]);
  }
}
```

プログラム 14.15 に運転台数の計算処理を示す。部分負荷を考慮した機器効率の最大化を狙わない場合には 22 行に示すように単純に負荷を最大能力で除して台数を決定する。10~19 行は機器効率を最大化する場合の処理であり、10 行（式 14.66）で効率が最大化する負荷率を求める。この負荷率をぎりぎりで上回るか下回る運転台数で機器効率が最大化するが、いずれが有利かを判定するために 15, 16 行で両方の効率を計算し、高い方の台数を出力する。部分負荷効率は 25~34 行（式 14.65）で計算する。

プログラム 14.15　運転台数の計算処理

Popolo.HVAC.HeatSource.AirHeatSourceModularChillers class

```
/// <summary>運転台数[台]を計算する</summary>
/// <param name="load">要求負荷[kW]</param>
/// <param name="mCap">最大能力[kW/台]</param>
/// <returns>運転台数[台]</returns>
private int getOperatingNumber(double load, double mCap)
{
  //部分負荷運転により機器効率を最大化させる場合
  if (MaximizeEfficiency)
  {
    double optCap = mCap * (-0.5 * a_PL[1] / a_PL[0]);
    int optNum = (int)Math.Floor(load / optCap);
    if (NumberOfUnits <= optNum) return NumberOfUnits;
    else
    {
      double plf1 = getPartialLoadFactor(load / optNum);
      double plf2 = getPartialLoadFactor(load / (optNum + 1));
      if (plf1 < plf2) return (optNum + 1);
      else return optNum;
    }
  }
  //最大負荷で運転時間を最小化させる場合（機器寿命重視）
  else return (int)Math.Min(Math.Ceiling(load / mCap), NumberOfUnits);
}

/// <summary>部分負荷効率[-]を計算する</summary>
/// <param name="partialLoad">部分負荷率[-]</param>
/// <returns>部分負荷効率[-]</returns>
private double getPartialLoadFactor(double partialLoad)
{
  double pl = Math.Max(0.2, partialLoad);
  double plf = pl * (a_PL[0] * pl + a_PL[1]) + a_PL[2];
  if (partialLoad < 0.2) return plf * (partialLoad / 0.2);
  else return plf;
}
```

プログラム 14.16 に状態更新処理を示す。12~18 行で冷暖モードに応じて必要な能力を計算する。ヒートポンプ機能が無く暖房需要がある場合には 0 とする。負荷がなければ 20~25 行で停止処理を行

い終了する。27~29 行はプログラム 14.15 に示した運転台数の計算である。30 行で全負荷を運転台数で除することで 1 台あたりの負荷 Q_{LD} を計算する。34~62 行で、冷暖運転モードに応じた係数を計算する。64~78 行は出口空気温度の計算処理であり、式 14.55~14.62で式 14.54 の係数を求め、78 行でカルダーノの公式により 3 次方程式を解く。複数の実数解が得られた場合には適切な解を選択する必要がある。81~90 行で定格の COP からおおよその空気出口温度を推定し、最も近い解を選択する。

プログラム 14.16　状態更新処理

Popolo.HVAC.HeatSource.AirHeatSourceModularChillers class

```
/// <summary>状態を更新する</summary>
/// <param name="waterInletTemperature">入口水温[C]</param>
/// <param name="waterFlowRate">水量[kg/s]</param>
/// <param name="ambientTemperature">周囲の温度[C]</param>
public void Update(double waterInletTemperature, double waterFlowRate, double ambientTemperature)
{
  //状態値を保存
  this.WaterInletTemperature = waterInletTemperature;
  this.WaterFlowRate = waterFlowRate;
  this.AmbientTemperature = ambientTemperature;

  //必要能力を計算
  double load = 0;  //ShutOffの場合には0になる
  double mcw = WaterFlowRate * WATER_SPECIFIC_HEAT;
  if (Mode == OperatingMode.Cooling)
    load = mcw * (WaterInletTemperature - WaterOutletSetPointTemperature);
  else if (Mode == OperatingMode.Heating && IsHeatPumpModel)
    load = mcw * (WaterOutletSetPointTemperature - WaterInletTemperature);

  //負荷0ならば停止処理
  if (load <= 0)
  {
    shutOff();
    return;
  }

  //過負荷判定//最適運転台数を計算する
  double cap = GetMaxCapacity(WaterOutletSetPointTemperature, AmbientTemperature);
  OperatingNumber = getOperatingNumber(load, cap);
  double qLD = load / OperatingNumber;
  IsOverLoad = cap < qLD;
  if (IsOverLoad) qLD = cap;

  //冷暖切替係数
  double sg, copFLRT, copFLR, mcma;
  double[] aCOP;
  if (Mode == OperatingMode.Cooling)
  {
    sg = -1;
    copFLRT = copFLRT_C;
    copFLR = NominalCoolingCOP;
    aCOP = aCop_C;
    mcma = AirFlowRate * (1.005 + 1.846 * 0.020); //夏季絶対湿度は20g/kgとする
  }
  else
  {
    sg = 1;
    copFLRT = copFLRT_H;
    copFLR = NominalHeatingCOP;
    aCOP = aCop_H;
    mcma = AirFlowRate * (1.005 + 1.846 * 0.002); //冬季絶対湿度は2g/kgとする
  }

  //3次方程式の係数計算
  WaterOutletTemperature = WaterInletTemperature + sg * qLD * OperatingNumber / mcw;
  double abf0 = copFLRT / (WaterOutletTemperature + 273.15);
  double abf1 = 2 * aCOP[0] * copFLRT + sg * aCOP[1];
  double abf2 = copFLRT * (aCOP[0] * copFLRT + sg * aCOP[1]) + aCOP[2];
  double abf3 = mcma * (AmbientTemperature + 273.15) - sg * qLD;
  double[] aTau = new double[4];
  aTau[0] = abf0 * abf0 * aCOP[0] * mcma;
  aTau[1] = -abf0 * (mcma * abf1 + abf0 * aCOP[0] * abf3);
  aTau[2] = abf0 * abf1 * abf3 + mcma * abf2;
  aTau[3] = - abf2 * abf3 - qLD / (getPartialLoadFactor(qLD / cap) * NominalCoolingCOP);

  double x1, x2, x3, tao;
```

```
77   bool hasMS;
78   CubicEquation.Solve(aTau, out x1, out x2, out x3, out hasMS);
79   if (hasMS)
80   {
81     double dt;
82     if (Mode == OperatingMode.Cooling) dt = qLD / NominalCoolingCOP;
83     else dt = qLD / NominalHeatingCOP;
84     double tao2 = AmbientTemperature + (dt - sg * qLD) / mcma + 273.15;
85     double x1er = Math.Abs(tao2 - x1);
86     double x2er = Math.Abs(tao2 - x2);
87     double x3er = Math.Abs(tao2 - x3);
88     if (x1er < x2er && x1er < x3er) tao = x1 - 273.15;
89     else if (x2er < x3er) tao = x2 - 273.15;
90     else tao = x3 - 273.15;
91   }
92   else tao = x1 - 273.15;
93
94   //出力設定
95   ElectricConsumption = mcma * (tao - AmbientTemperature) + sg * qLD;
96   AuxiliaryElectricConsumption = auxElec;
97 }
```

【例題 14.5】

下記の定格仕様を持つモジュールヒートポンプを 3 台結合した場合に、外気温度と負荷率に応じて消費電力がどのように変化するかを検討せよ。また、機器効率最大化を狙った台数制御と運転時間最小化を狙った台数制御の両方について計算を行い比較せよ。補機の消費電力は 0 とする。

表 14.4　モジュールヒートポンプの定格仕様

	能力 [kW/台]	消費電力 [kW/台]	水量 [L/min]	水温 [ºC]	風量 [m³/min]	外気温度 [ºC]
冷房	150	49.8	430	12→7	850	35
暖房	150	50.0	430	40→45	850	7

【解】

プログラム 14.17 に計算処理を示す。3~5 行で空気熱源ヒートポンプのインスタンスを作成する。6 行で機器効率最大化と運転時間最小化のいずれで台数制御を行うかを設定する。

計算結果をグラフ化すると図 14.9 が得られる。運転台数を最小化する場合には負荷率に応じて効率が大きく上下することがわかる。運転台数を切り替える負荷率が外気温度によって異なるのは、外気温度により最大能力が変化するためである。機器効率を最大化する場合には全負荷の低下にともなって効率が概ね上昇することがわかる。

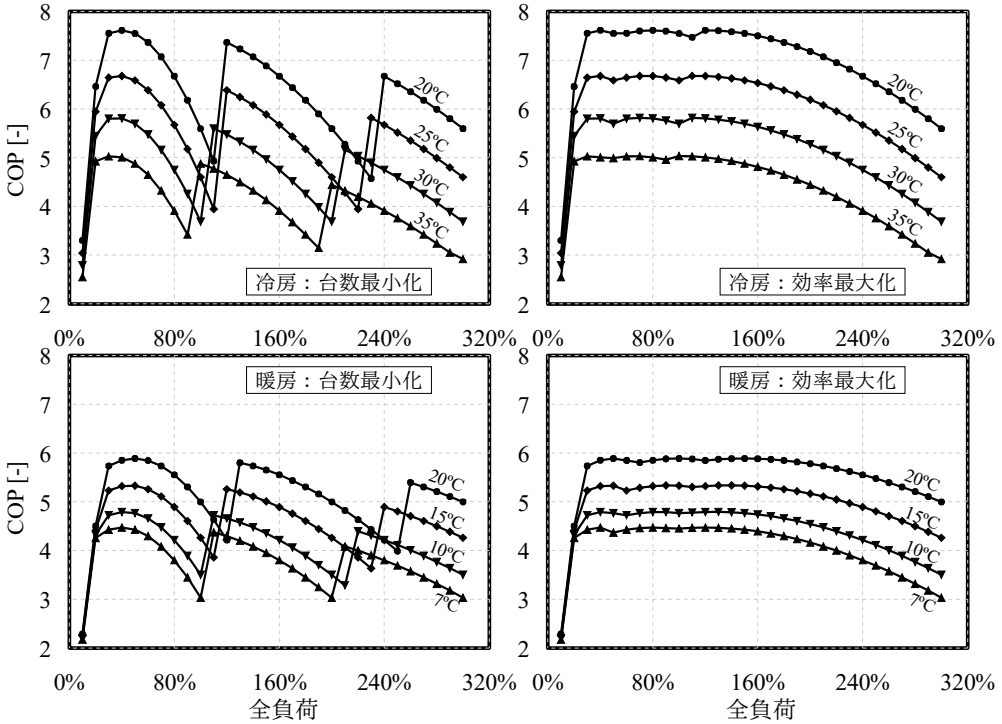


図 14.9　空気熱源ヒートポンプの負荷率と効率の関係（外気温度別）

プログラム **14.17**　空気熱源ヒートポンプの感度解析

```
 1 private static void AirSourceHeatPumpTest()
 2 {
 3   //空気熱源 HP のインスタンスを生成
 4   AirHeatSourceModularChillers ahp = new AirHeatSourceModularChillers
 5     (150, 7, 430d / 60, 35, 850d / 60 * 1.2, 49.8, 150, 45, 430d / 60, 7, 850d / 60 * 1.2, 50.0, 3, 0.0);
 6   ahp.MaximizeEfficiency = true;
 7
 8   //タイトル行
 9   for (int i = 0; i <= 30; i++) Console.Write(", " + (10 * i).ToString("F0"));
10   Console.WriteLine();
11
12   //冷房運転
13   Console.WriteLine("Cooling");
14   ahp.Mode = AirHeatSourceModularChillers.OperatingMode.Cooling;
15   ahp.WaterOutletSetPointTemperature = 7;
16   double[] tai = new double[] { 20, 25, 30, 35 };
17   for (int i = 0; i < tai.Length; i++)
18   {
19     Console.Write(tai[i]);
20     for (int j = 0; j <= 30; j++)
21     {
22       double pl = 0.1 * j;
23       double mw = 430d * 3 / 60;
24       double twi = 7 + (150d * pl) / (4.186 * mw);
25       ahp.Update(twi, mw, tai[i]);
26       Console.Write(", " + ahp.COP.ToString("F3"));
27     }
28     Console.WriteLine();
29   }
30
31   //暖房運転
32   Console.WriteLine("Heating");
33   ahp.Mode = AirHeatSourceModularChillers.OperatingMode.Heating;
34   ahp.WaterOutletSetPointTemperature = 45;
35   tai = new double[] { 9, 7, 5, 3 };
36   for (int i = 0; i < tai.Length; i++)
37   {
38     Console.Write(tai[i]);
39     for (int j = 0; j <= 30; j++)
40     {
51       double pl = 0.1 * j;
52       double mw = 430d * 3 / 60;
53       double twi = 45 - (150d * pl) / (4.186 * mw);
54       ahp.Update(twi, mw, tai[i]);
55       Console.Write(", " + ahp.COP.ToString("F3"));
56     }
57     Console.WriteLine();
58   }
59 }
```

【第 14 章 記号表】

A	：伝熱面積 **[m²]**	Pl	：負荷率 **[-]**
a,b,c	：特性係数 **[-]**	Q	：熱交換量 **[kW]**
COP	：成績係数 **[-]**	R_E	：入力比 **[-]**
c_{pw}	：水の定圧比熱 **[kJ/(kg·K)]**	R_{vR}	：冷媒流量比 **[-]**
E	：消費電力 **[kW]**	s_g	：冷暖モード切替係数（冷:**-1**, 暖:**+1**）
E_{cmp}	：圧縮機入力 **[kW]**	T	：絶対温度 **[K]**
h	：比エンタルピー **[kJ/kg]**	v	：体積流量 **[m³/s]**
K	：熱通過率 **[kW/(m²·K)]**	W_{cmp}	：断熱圧縮仕事 **[kW]**
m	：質量流量 **[kg/s]**	ε	：熱通過有効度 **[-]**
mc	：熱容量流量 **[kW/K]**	κ	：比熱比 **[-]**
P	：圧力 **[kPa]**	η	：断熱圧縮仕事に対する効率 **[-]**

sub scripts

a	：空気	*mid*	：中間圧力
cd	：冷却水	*N*	：定格条件
ch	：冷水	*o*	：出口
*cmp***1**	：圧縮機（**1** 段目羽根車）	*Pl*	：部分負荷
*cmp***2**	：圧縮機（**2** 段目羽根車）	*R*	：冷媒
cnd	：凝縮器	*s*	：セットポイント
evp	：蒸発器	*sv*	：飽和蒸気
Fl	：全負荷	*T*	：理論
i	：入口	*w*	：水
M	：最大効率点		

【第 14 章 参考文献】

14.1） 冷媒圧縮機 日本冷凍空調学会専門書シリーズ, （社）日本冷凍空調学会, 2013.3
14.2） SI による上級冷凍受験テキスト, （社）日本冷凍空調学会, 2011.12
14.3） 南在成, 渡辺俊行, 龍有二, 赤司泰義: ヒートポンプを用いた床暖冷房システムの数値解析, 日本建築学会計画系論文報告集 (451), (September, 1993), pp.75-83
14.4） 宇田川光弘, 村田太市: 空調システムシミュレーションのための圧縮式冷凍機の計算モデル, 空気調和・衛生工学会論文集, No.64, (January, 1997), pp.73-82
14.5） JISB8621 遠心冷凍機 (2011), 日本規格協会
14.6） 生井武文: 遠心軸流 送風機と圧縮機 第 5 章, p.177, 図 5.67, 朝倉書店, 1960
14.7） 高田秋一: ターボ冷凍機 第 11 章, p.213, 図 11.1, 日本冷凍協会, 1976
14.8） 上田憲治, 長谷川泰士, 下田吉之: 民生業務用熱源システムにおける高効率ターボ冷凍機の使用法に関する研究, 空気調和・衛生工学会論文集, 136, pp.17-25, 2008
14.9） 三菱重工ターボ冷凍機 -AART・NART シリーズ-, 取扱説明書, 三菱重工株式会社, 2008.11
14.10） 高効率二段ターボ冷凍機 設備設計資料, ダイキン工業 アプライド・ソリューション事業本部, 2011.03
14.11） 上田憲治, 栂野良枝, 下田吉之: ターボ冷凍機部分負荷性能推定手法の開発（第一報）性能推定手法の概要と実用化, 空気調和・衛生工学会大会学術講演論文集, pp.93-96, 2010
14.12） 姜信愛 ,赤司泰義 ,小塩真奈美 ,浦山真一: 建物空調システムの最大負荷計算と設計法に関する研究 : (第 2 報)インバータターボ冷凍機を用いた空調システム設計が運用時のエネルギー消費量に与える影響, 空気調和・衛生工学会 学術講演会論文集, pp.1607-1610, 2010
14.13） 高田修 ,田中良彦 ,坂本雄三 ,松尾陽: 空調システムの最適化を目的とした統合的設計と運転に関する研究 : (第 14 報)インバータターボ冷凍機システムの設計と制御の研究, 空気調和・衛生工学会 学術講演会論文集, pp.1555-1558, 2010
14.14） 藤田尚志: フリークーリングとインバータターボ冷凍機の効率的運転のための検討, 空気調和・衛生工学会 学術講演会論文集, pp.1739-1742, 2009
14.15） 青竹紀子, 相良和伸, 山口弘雅, 三浦光城, 小林陽一: インバータ駆動式冷凍機を用いた蓄熱システムの最適運転に関する研究, 空気調和・衛生工学会 学術講演会論文集, pp.207-210, 2006
14.16） 高山紗輝, 赤司泰義, 宮田征門, 福崎達也: オフィスビルにおける高効率化技術の導入効果 その 1 インバータターボ冷凍機導入による省エネルギー効果, 日本建築学会九州支部研究報告, 第 49 号, pp.309-312, 2010
14.17） 下田吉之, 宇野義隆, 砂川良, 栂野良枝, 渡辺健一郎, 荘司豊: 地域冷暖房リニューアルに伴う高効率化シミュレーション（その２）, 空気調和・衛生工学会 学術講演会論文集, pp.831-834, 2009
14.18） ターボ機械 -入門編-, ターボ機械協会, 日本工業出版, 2005
14.19） 富樫英介: 空調設備の年間シミュレーションのための外挿性能を持ったインバータターボ冷凍機モデルの開発, 空気調和・衛生工学会論文集, No.223, pp.19-26, 2015.10
14.20） 富樫英介: 特性式による圧縮式冷凍機のモデルの提案, 空気調和・衛生工学会 学術講演論文集, 2015
14.21） 冷凍空調便覧 2 機器編: p.22 図 1.4.17 より作成, 第 6 版, 2006

第15章　吸収式冷凍機
(Absorption Chiller)

15.1　概要

空調設備分野で冷熱源として用いられる機器は、主には前章で解説した圧縮式冷凍機と本章で取り上げる吸収式冷凍機である。吸収式冷凍機も圧縮式冷凍機と同様に液体の気化作用を利用して冷却を行うものであり、濃い水溶液に水を吸収させることで水の蒸発を促すためにこのような名称がついている。圧縮式冷凍機の主たるエネルギー源は電気[†1]だが、吸収式冷凍機のエネルギー源は温水や蒸気など、直接的な熱エネルギーである。熱エネルギーをもとに冷却作用を生み出すことができるため、コージェネレーションの廃熱や太陽熱温水など、熱エネルギーが入手可能な場合には有利である。また、直焚吸収冷温水機と呼ばれる、機器内で直接にガスなどを燃焼させて熱エネルギーを得る機種もある。直焚吸収冷温水機の冷房 COP は単効用で 1.0 未満、二重効用で 1.2~1.4 程度、三重効用でも 2.0 弱であるため、一次エネルギー効率という観点からは圧縮冷凍機にやや劣る[†2]。しかし、1 台で冷水と温水の製造が可能であるため、省コスト・省スペース効果は高い。また、主なエネルギー源がガスなどであるため、被災時に電力供給が不安定になった場合などにも運転継続が比較的容易であるという長所もある。

本章では、吸収冷凍サイクルの計算をするにあたって必要となる臭化リチウムの物性値計算法、温水を熱源として動作する温水吸収冷凍機の計算法、直接にガスを燃焼する直焚吸収冷温水機の計算法について説明する。

写真 15.1　矢崎ソーラーハウス 1 号（写真撮影：宇田川光弘）

（85 度の太陽熱温水によって吸収式冷凍機（自然循環型）を駆動した日本初のソーラーハウス（1974 年））

†1　ガスで蒸気により圧縮を行う機器もある。

†2　遠心冷凍機の高効率機は二次 COP で 6.4、一次 COP で 2.3 程度である。

15.2　理論

15.2.1　吸収冷凍サイクル

1）単効用吸収冷凍サイクル

不揮発性の溶質を溶媒に溶かした場合、純粋な溶媒に比較して溶媒の沸点が上昇し、蒸気圧が低下する。吸収式冷凍機はこの作用を利用して、水を連続的に蒸発させて冷却を行う機械である。図 15.1 に吸収冷凍サイクルを示す。

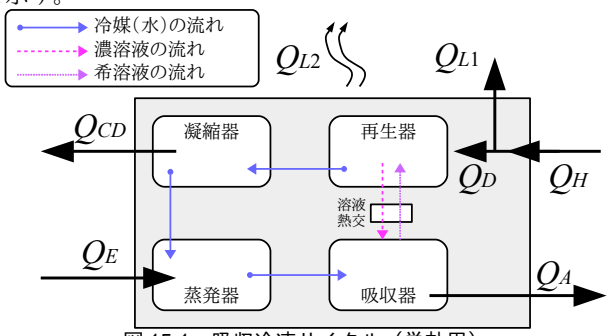


図 15.1　吸収冷凍サイクル（単効用）

図 15.1 は最も単純な単効用吸収冷凍サイクルと呼ばれるサイクルである。式 15.1 に吸収冷凍サイクルの熱収支を示す。吸収冷凍機を構成する主要機器としては再生器、吸収器、凝縮器、蒸発器、溶液熱交換器がある。Q_H [kW]は吸収冷凍サイクルへの投入エネルギーであり、一部がエネルギー損失 Q_{L1} [kW]となり、残余の Q_D [kW]が再生器に投入される。また、蒸発器においても外部から Q_E [kW]の熱を受ける。再生器および蒸発器に与えられた熱の一部は直接に外部に漏洩し（Q_{L2} [kW]）、残余の熱は冷却水を介して吸収器および凝縮器から外部へ放出される（$Q_A + Q_{CD}$ [kW]）。

$$(Q_H - Q_{L1}) + Q_E - Q_{L2} = Q_D + Q_E - Q_{L2} = Q_{CD} + Q_A \tag{15.1}$$

吸収式冷凍機の場合、溶質として臭化リチウム（LiBr: Lithium Bromide）、溶媒として水（冷媒と呼ぶ）を用いることが多い。蒸発器では前記の蒸気圧降下が生じており、蒸発器内に通した冷水配管に対して純粋な水を散布すると、冷水配管から熱を奪いながら水が蒸発する。蒸発した水は LiBr 水溶液に溶け込み（吸収）、濃度を低下させる。この LiBr 水溶液への吸収が生じる空間が「吸収器」である。LiBr 水溶液の濃度の低下は蒸気圧の上昇をもたらし、水の蒸発量（=水の吸熱量）を低下させるため、冷却能力を保つためには濃度を維持する必要がある。このため低濃度の LiBr 水溶液（稀溶液）を加熱して溶液を沸騰させ、水分を気化させて取り除く（再生）。この空間が「再生器」である。単効用吸収冷凍サイクルの場合には 90 °C 程度の温水を供給することで加熱を行うことが通常である。従って、発電機排熱や太陽熱により温水を安価に製造できる場合などには温水吸収冷凍機を用いることが効果的である。再生された高濃度の LiBr 水溶液（濃溶液）は吸収器に戻される。この時に吸収器から出てくる稀溶液と吸収式に入る濃溶液との間で熱交換を行う。この熱交換器を溶液熱交換器と呼ぶ。溶液熱交換器から出てきた濃溶液は冷却水でさらに冷却される。一方、再生器で加熱気化して分離された水蒸気は「凝縮器」において冷却凝縮される。液化した高温の水は冷却水で熱を除去され、再び蒸発器において散布される。このサイクルを繰り返すことで冷水を連続的に製造する。

2）二重効用吸収冷凍サイクル

単効用吸収冷凍サイクルでは温水により稀溶液の再生を行ったが、ガスの直接燃焼や蒸気の利用な

どで再生温度を高くすることができる場合には、より効率の高い「二重効用吸収冷凍サイクル」を組むことができる。

図 15.2 に二重効用吸収冷凍サイクルを示す。再生器が高温再生器と低温再生器の二つに分けられている点が特徴である。高温再生器で発生する蒸気の温度が高いため、この蒸気を低温再生器に送って低圧の水溶液を加熱気化させることで熱を二段階に使うものである。現在のところ、空調分野ではガスを用いる冷熱源としては二重効用が主流である。この他にも製品としては、高温、中温、低温の再生器を持つ三重効用吸収冷凍機もあるが、高価であり、導入事例はまだ多くは無い。

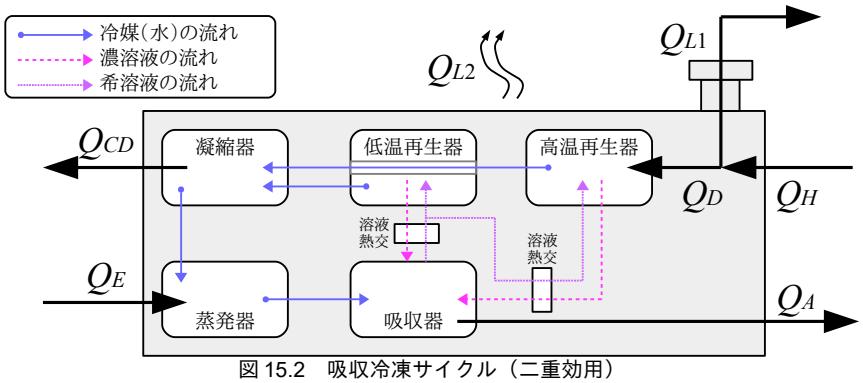


図 15.2　吸収冷凍サイクル（二重効用）

【例題 15.1】

下記仕様の二重効用の直焚吸収冷温水機についてエネルギー損失 Q_{L1} を求めよ。ただし再生器以降の熱ロス Q_{L2} は再生器投入熱量 Q_D [kW]に対して 5 %とする。

冷水出入口条件：7 °C → 15 °C、冷却水出入口条件：32 °C → 37.2 °C

冷水流量：189 m^3/h、冷却水流量：500 m^3/h、ガス消費量：1,288 kW

【解】

冷却能力は 189÷3,600×1,000×(15-7) × 4.186 = 1,758 kW である。冷却水による熱除去量は 500÷3,600×1,000×(37.2-32) × 4.186 = 2,907 kW である。従って、再生器に供給される熱量は、Q_D = 2,907 – 1,758 + Q_D × 5%であるため、Q_D = 1,209 kW である。一方、ガス消費量は 1,288 kW であるため、エネルギー損失 Q_{L1} は、1,288 – 1,209 = 79 kW となる。

15.2.2　臭化リチウム水溶液の物性

前述のとおり、吸収冷凍機の溶質としては臭化リチウム水溶液を用いることが通常である。従って、吸収冷凍機の特性を知るためには、臭化リチウム水溶液の熱力学性質の計算が必要である。本節では水溶液の温度と濃度に対して平衡する蒸気の飽和温度・比エンタルピー・比熱を計算する方法を示す。

1）水溶液と平衡する蒸気の飽和温度

佐藤によれば、臭化リチウム水溶液の温度 T_{lb} [K]と蒸気の飽和温度 T_{lbs} [K]はほぼ直線の関係とみなせ、式 15.2 で計算できる[15.2)]。

$$T_{lbs} = A_{lb} + B_{lb} \cdot T_{lb} \tag{15.2}$$

式 15.2 の A_{lb} と B_{lb} は近似係数であるが、水溶液の質量分率 w_{lb} [-]の関数として式 15.3 と式 15.4 で計算できる。a_{lbn} と b_{lbn} は近似係数であり、その値は表 15.1 に示すとおりである。式 15.2~15.4を用いて計算を行い、横軸に臭化リチウム水溶液の温度 T_{lb}、縦軸に蒸気の飽和温度 T_{lbs} をとると図 15.3 が得られる。このような図を Dühring（デューリング）線図と呼ぶ[†1)]。

†1　飽和温度のかわりに飽和圧力を縦軸とすることも多い。

$$A_{lb}=a_{lb0}+a_{lb1}\cdot w_{lb}+a_{lb2}\cdot w_{lb}^{2}+a_{lb3}\cdot w_{lb}^{3}+a_{lb4}\cdot w_{lb}^{7} \tag{15.3}$$

$$B_{lb}=b_{lb0}+b_{lb1}\cdot w_{lb} \tag{15.4}$$

表 15.1　臭化リチウムの物性値近似係数 1

n	0	1	2	3	4
a_{lbn}	-22.8937	152.554	-254.786	152.949	-171.599
b_{lbn}	1.09851	-0.394508	-	-	-

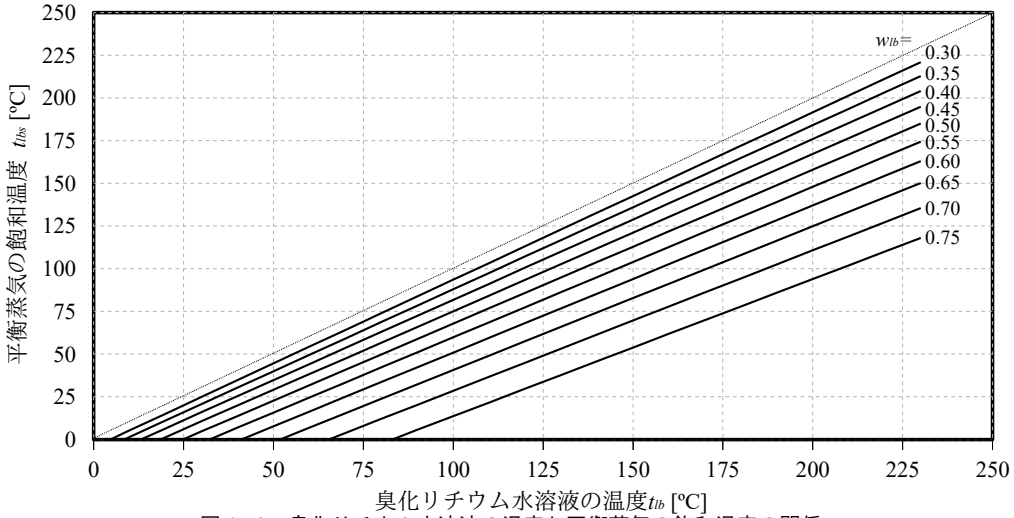


図 15.3　臭化リチウム水溶液の温度と平衡蒸気の飽和温度の関係

2）水溶液の比エンタルピー

式 15.5 は McNeely による比エンタルピーの近似式である [15.3) 15.4)]。近似式の適用可能範囲は濃度 40~70%、温度 15~165℃ であり、二重効用吸収冷凍機の運転範囲を包含する。本式は 0 ℃ の飽和水を 0 kJ/kg としており、3 章で解説した水の比エンタルピーと絶対値で加減計算が可能[†1)]である。c_{lb}、d_{lb}、e_{lb} は近似係数であり、値は表 15.2 に示す通りである。式 15.5 にもとづいて濃度、比エンタルピー、温度の関係を計算した結果を図 15.4 に示す。

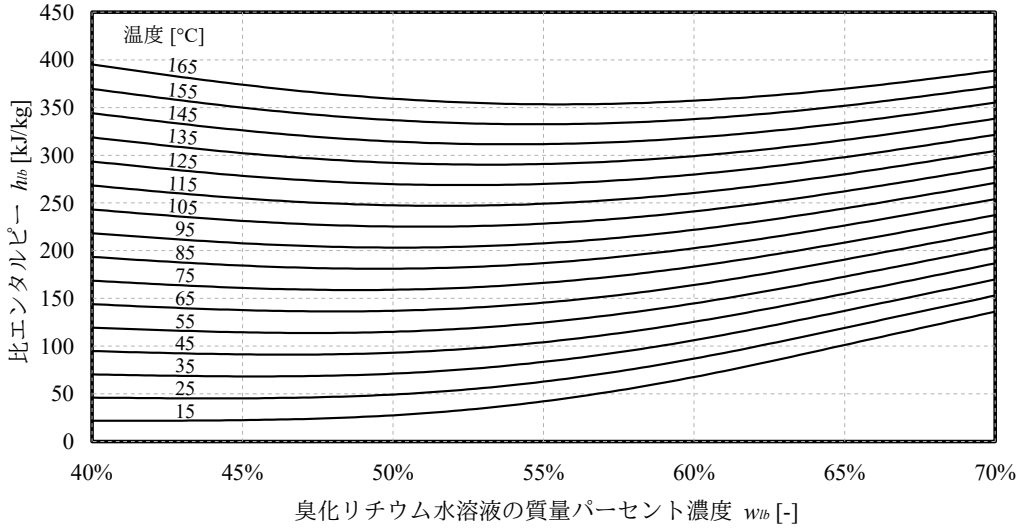


図 15.4　臭化リチウム水溶液の濃度、比エンタルピー、温度の関係

†1　資料によっては例えば 0 ℃ の飽和水を 100 kJ/kg とした図表としていることもあるため、比エンタルピーの絶対値を加減するような計算にあたっては基準の置き方を確認する必要がある。

$$h_{lb}=\sum_{n=0}^{4}\left\{c_{lb,n}w_{lb}^{n}+d_{lb,n}w_{lb}^{n}(T_{lb}-273.15)+e_{lb,n}w_{lb}^{n}(T_{lb}-273.15)^{2}\right\} \tag{15.5}$$

表 15.2　臭化リチウムの物性値近似係数 2

n	0	1	2	3	4
$c_{lb,n}$	-2.02433e3	1.63309e4	-4.88161e4	6.302948e4	-2.913705e4
$d_{lb,n}$	1.82829e1	-1.1691757e2	3.248041e2	-4.034184e2	1.8520569e2
$e_{lb,n}$	-3.7008214e-2	2.8877666e-1	-8.1313015e-1	9.9116628e-1	-4.4441207e-1

3）水溶液の比熱

式 15.5 を微分して式 15.6 で計算する。

$$c_{plb}=\frac{dh}{dT}=\sum_{n=0}^{4}\left\{d_{lb,n}w_{lb}^{n}+2e_{lb,n}w_{lb}^{n}(T_{lb}-273.15)\right\} \tag{15.6}$$

15.2.3　デューリング線図上のサイクル

デューリング線図上で吸収冷凍サイクルを再確認する。以下、水を *RL*（Refrigerant Liquid）、蒸気を *RV*（Refrigerant Vapor）、臭化リチウム水溶液を *S*（Solution）、再生器を *D*（Desobor）、吸収器を *A*（Absorbor）、凝縮器を *CD*（Condensor）、蒸発器を *E*（Evaporator）、高温側を *H*（High）、低温側を *L*（Low）で表現する。また入口および出口を添字の *i*（inlet）と *o*（outlet）で表す。

1）単効用吸収冷凍サイクル

図 15.5 にデューリング線図上の単効用冷凍サイクルのフローを示す。

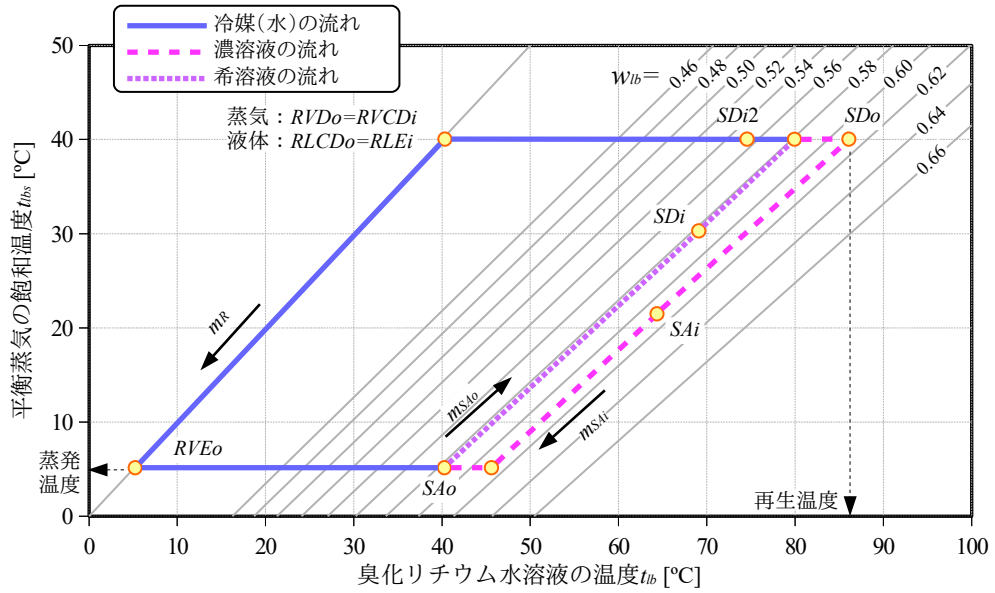


図 15.5　吸収冷凍サイクル（単効用）

蒸発器では、環境温度にある凝縮水が蒸発する（*RLEi*→*RVEo*）。この際に外部の冷水から熱を奪う。蒸発器での冷媒の温度を蒸発温度と呼ぶ。蒸発温度は通常は 4~5°C 程度で設計される[†1)]。吸収器では蒸発器で発生した蒸気を吸収することで濃溶液の濃度が低下し稀溶液が発生する（*RVEo*→*SAo* と *SAi*→*SAo*）。稀溶液を濃溶液に戻すためには再生器で加熱・沸騰させる必要があるが、その前に溶液熱交換器を用いて再生器出口の濃溶液との間で熱交換を行う（*SAo*→*SDi* と *SDo*→*SAi*）。この熱交換は稀溶液の加熱に必要な温熱と濃溶液の冷却に必要な冷熱を削減するため、冷凍サイクルの効率

†1　これよりも蒸発温度を下げるためには蒸発器の真空度を高める必要があり、冷媒の凍結の可能性も出てくる。従って、吸収冷凍機の場合には、定格能力での冷水温度を下げるためには蒸発温度の変更ではなく、蒸発器の伝熱面積の増大による温度アプローチの低減という手法が取られる。ただし水以外の冷媒を用いることで氷点下の蒸発温度を可能とした事例はある[15.5)]。

を向上させる。再生器側では外部から熱が与えられることで稀溶液が蒸気と濃溶液に分離する（*SDi→RVDo, SDo*）。発生した濃溶液は稀溶液との間の熱交換で冷却された後、吸収器に送られる（*SDo→SAi*）。蒸気は凝縮器で冷却されて蒸発器に送られる（*RVDo=RVCDi→RLCDo=RLEi*）。

2）二重効用吸収冷凍サイクル

既に述べたとおり、二重効用吸収冷凍サイクルの特徴は、二つの再生器によって二段階に熱を活用して効率向上を図っている点にある。溶液の循環方法にはいくつかの種類があり名称が異なる。図 15.6 に溶液の流れと名称の対応を示す。

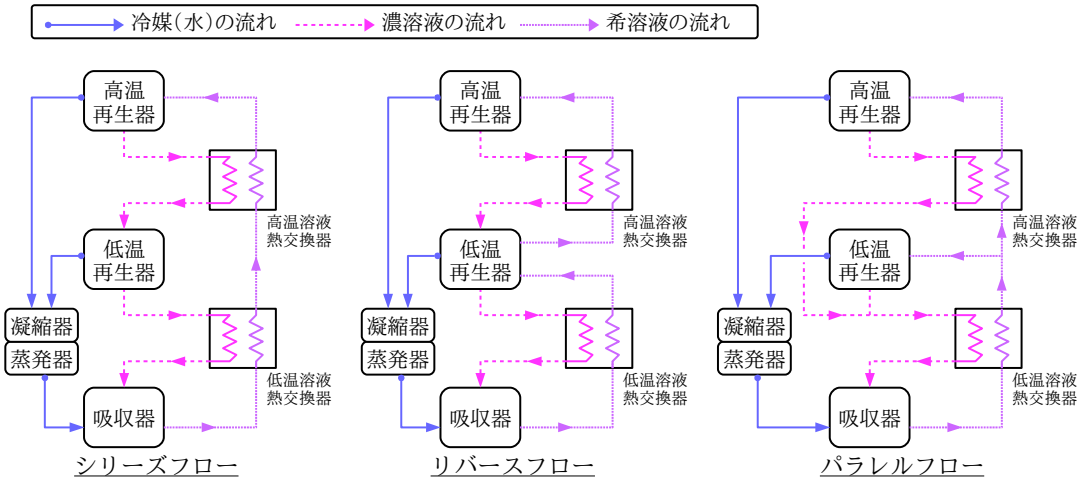


図 15.6　二重効用吸収冷凍サイクルの溶液循環方式

図 15.6 左は、吸収器で発生した稀溶液の全量を高温再生器に投入し、高温再生器で発生した中間濃度の溶液をさらに低温再生器に投入して濃度を高める方式であり、シリーズフローと呼ばれる。図 15.6 中央は、シリーズフローとは逆に、稀溶液の全量をまず低温再生器に投入し、低温再生器で発生した中間濃度の溶液をさらに高温再生器に投入する方式であり、リバースフローと呼ばれる。図 15.6 右は、低温溶液熱交換器を通過した後の稀溶液を高温再生器と低温再生器に分配して投入する方式であり、パラレルフローと呼ばれる。効率としてはシリーズフローに比較してパラレルフローおよびリバースフローが高く、製造業者もこの二種類のいずれかの方式を採用していることが多い。

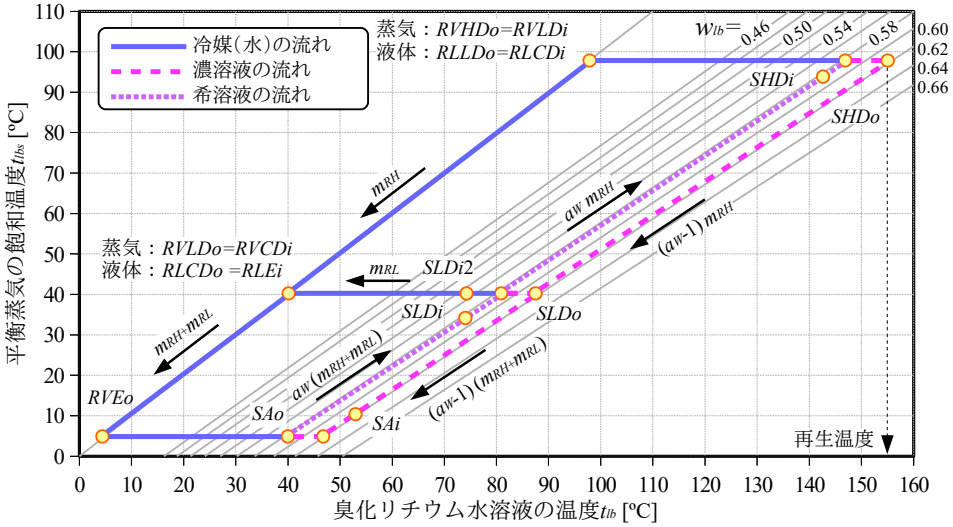


図 15.7　吸収冷凍サイクル（二重効用）

図 15.7 にデューリング線図上の二重効用吸収冷凍サイクルのフローを示す。蒸発器・凝縮器・吸収器における冷媒・水溶液の流れは単効用と同様のため省略する。再生器が二段にわかれているため、溶液熱交換器も高温側と低温側の二つがある。吸収器出口稀溶液は低温溶液熱交換器によって加熱された後、一部が低温再生器に入り、残余が高温再生器に向かう。高温再生器に向かった稀溶液は高温溶液熱交換器によってさらに加熱される（*SLDi→SHDi*）。高温再生器で外部から熱を与えることで稀溶液は蒸気と濃溶液に分離する（*SHDi→RVHDo, SHDo*）。高温再生器での濃溶液の温度を再生温度と呼び、通常は 160°C 程度の値で設計される†1)。また、容器内の圧力が大気圧を超えると圧力容器となり、取扱作業主任者が必要となるため、通常は二重効用吸収冷凍機の圧力は 100 kPa を超えないように設計される（三重効用では大気圧を超える）。結果として再生器で発生する蒸気の温度は 100°C 弱となる。高温再生器で発生した高温蒸気は低温再生器に供給されて稀溶液を加熱するが、この過程で冷却されるため凝縮水となる（*RVHDo→RLLDo*）。凝縮水は、凝縮器でさらに冷却された後、蒸発器に投入される（*RLLDo→RLCDo=RLEi*）。低温再生器では、稀溶液が高温再生器で発生した高温蒸気により加熱され、蒸気と濃溶液に分離される（*SLDi→RVLDo, SLDo*）。蒸気は凝縮器で冷却・凝縮され、蒸発器に投入される（*RVLDo→RLCDo=RLEi*）。濃溶液は高温再生器側からの濃溶液と合流した後、低温溶液熱交換器を通って冷却され、吸収器に入る（*SLDo→SAi*）。

15.2.4　基礎式

各部の計算を行う前提として第 3 章で解説した飽和蒸気の計算が必要となる。

1）単効用吸収冷凍サイクル

・蒸発器

蒸発器における冷媒加熱量は冷媒の気化熱に相当するため、冷媒の循環量を m_R [kg/s]とすると、式 15.7 で計算できる。h_{RVEi} [kJ/kg]は蒸発器出口冷媒（蒸気）の比エンタルピーである。h_{RLEi} [kJ/kg]は蒸発器入口冷媒（液体）の比エンタルピーであり、凝縮器出口冷媒の比エンタルピー h_{RLCDo} に等しい。また、蒸発器処理熱量は冷水の出入口温度差と冷水流量 m_{ch} [kg/s]を用いて式 15.8 でも表される。c_{pw} は水の定圧比熱 (4.186 kJ/(kg·K))である。蒸発器での冷媒温度を一定とすれば、片側温度一定の熱交換器であるため、第 8 章で解説した熱通過有効度 ε を用いて熱交換量を式 15.9~15.11で計算できる。

$$Q_E = m_R\left(h_{RVEo} - h_{RLEi}\right) \tag{15.7}$$

$$Q_E = m_{ch} c_{pw}\left(t_{ch,i} - t_{ch,o}\right) \tag{15.8}$$

$$Q_E = \varepsilon_E m_{ch} c_{pw}\left(t_{ch,i} - t_E\right) \tag{15.9}$$

$$\varepsilon_E = 1 - \exp\left(-NTU_E\right) \tag{15.10}$$

$$NTU_E = KA_E / \left(m_{ch} c_{pw}\right) \tag{15.11}$$

・凝縮器

凝縮器における冷媒冷却量は式 15.12 で計算できる。h_{RVCDi} [kJ/kg]は凝縮器入口比エンタルピーであり、再生器出口比エンタルピーに等しい。Q_{L2} は凝縮器を介さない放熱である。

$$Q_{CD} = m_R\left(h_{RVCDi} - h_{RLCDo}\right) - Q_{L2} \tag{15.12}$$

・吸収器

吸収器における水溶液出入口流量 m_{SAo} [kg/s]と m_{SAi} [kg/s]は、濃溶液濃度 w_{thk} [-]、稀溶液濃度

†1　もともと臭化リチウムは腐食性を持つが、温度が 160 度を超えると腐食性が更に高まるため。

w_{thn}[-]、冷媒流量 m_R [kg/s]を用いて式 15.13 で計算できる。a_w[-]は、1 単位の冷媒を運ぶために必要な稀溶液の流量を示しており、溶液循環比と呼ばれる概念である。吸収器では冷却水を用いて SAi から SAo まで水溶液を冷却するとともに蒸発器の蒸気が濃溶液に溶け込む際の凝縮熱を除去するため、その熱量は式 15.15 で計算できる。

$$m_{SAo}=m_R a_w \quad , \quad m_{SAi}=m_R(a_w-1) \tag{15.13}$$

$$a_w=w_{thk}/(w_{thk}-w_{thn}) \tag{15.14}$$

$$Q_A=m_{SAi}(h_{SAi}-h_{SAo})+m_R(h_{RVEo}-h_{SAo}) \tag{15.15}$$

ここで、計算を簡易にするため、吸収器の出口水溶液温度 t_{SAo} [°C]と凝縮器の出口水溶液温度 t_{SCDo}[°C]が等しい（$=t_{CDA}$）とし、かつ、熱交換の際に水溶液の温度変化が無いとみなせば†1)、凝縮器および吸収器における熱交換量は式 15.16~15.18で計算できる。

$$Q_{CD}+Q_A=\varepsilon_{CDA} m_{cd} c_{pw}(t_{CDA}-t_{cd,i}) \tag{15.16}$$

$$\varepsilon_{CDA}=1-\exp(-NTU_{CDA}) \tag{15.17}$$

$$NTU_{CDA}=KA_{CDA}/(m_{cd} c_{pw}) \tag{15.18}$$

・再生器

再生器への投入熱量は発生した蒸気と濃溶液の比エンタルピー（h_{RVDo}と h_{SDo}）から入口稀溶液の比エンタルピー h_{SDi}を減じて式 15.19 で計算する。また、温水による加熱量は式 15.20 で計算する。ここで、m_{hw} [kg/s]と t_{hw} [°C]は温水の流量と温度である。

$$Q_D=m_R h_{RVDo}+m_{SAi} h_{SDo}-m_{SAo} h_{SDi} \tag{15.19}$$

$$Q_D=m_{hw} c_{pw}(t_{hw,i}-t_{hw,o}) \tag{15.20}$$

図 15.5 に示すように溶液熱交換器を出た稀溶液 SDi は再生器内の圧力よりも低い状態（平衡蒸気の飽和温度が低い）にあるため、再生器に流入すると同時に蒸気を吸収し、再生器内の圧力と平衡して SDi2 に変化する。温水と水溶液との間の熱交換量は式 15.21 でも表される。ここで c_{pSD} [kJ/(kg·K)]は SDi2 を基準とした冷媒蒸発量も考慮した容積比熱であり、式 15.22 と式 15.23 で計算する。式 15.20 が温水側からみた熱交換量、式 15.21 が水溶液側からみた熱交換量であり、向流型の熱交換器とみなし、第 8 章の式 8.24 を用いて出口状態を推定する。

$$Q_D=m_{SAo} c_{pSD}(t_{SDo}-t_{SDi2}) \tag{15.21}$$

$$c_{pSD}=\frac{((1-r_{sl})\cdot h_{RVDo}+r_{sl}\cdot h_{SDo})-h_{SDi2}}{t_{SDo}-t_{SDi}} \tag{15.22}$$

$$r_{sl}=w_{SDi2}/w_{thk} \tag{15.23}$$

・溶液熱交換器

溶液熱交換器での交換熱量 Q_X [kW]は式 15.24 および式 15.25 で計算する。向流型の熱交換器とみなし、第 8 章の式 8.24 を用いて出口状態を推定する。

$$Q_X=m_{SAo} c_{pSAo}(t_{SDi}-t_{SAo})=m_{SAo}(h_{SDi}-h_{SAo}) \tag{15.24}$$

$$Q_X=m_{SAi} c_{pSDo}(t_{SDo}-t_{SAi})=m_{SAi}(h_{SDo}-h_{SAi}) \tag{15.25}$$

†1　図 15.5 および図 15.7 から明らかなように実際には温度変化（顕熱移動）があり、また、通常は吸収器と凝縮器を直列に接続して冷却水を流すため、やや現実とは離れた仮定をおいていることになる。しかし吸収器および凝縮器での熱交換の大部分は潜熱移動（式 15.12 と式 15.15 の左辺の合計：例えば図 15.5 では 85%が潜熱分となる）であるため、誤差は大きくないと推測できることと、また、それぞれの熱交換量を別個に解くと凝縮温度を特定することが非常に難しく、実用上、収束計算に支障が生じることから、このような仮定をおいている。

【例題 15.2】

下記仕様の温水吸収冷凍機についてデューリング線図上の運転点を特定せよ。ただし、熱損失 Q_{L1} および Q_{L2} は 0 とし、蒸発温度は 5°C、凝縮温度は 40°C、温水入口温度と再生温度とのアプローチは 2°C とする。

冷水出入口条件：7 °C → 12.5 °C　　冷水流量：275 L/min
冷却水出入口条件：31 °C → 35 °C　　冷却水流量：918 L/min
温水出入口条件：88 °C → 83 °C　　温水流量：432 L/min

【解】

まずそれぞれの熱媒の交換熱量は、冷水：275 / 60 × (12.5 – 7) × 4.186 = 105 kW、冷却水：918 / 60 × (35 – 31) × 4.186 = 256 kW 、温水：432 / 60 × (88 – 83) × 4.186 = 151 kW、である。

吸収器出口水溶液状態（SAo）は凝縮温度 40°C と蒸発温度 5°C の交点であり、臭化リチウムの物性計算を行うと比エンタルピーは 106 kJ/kg、質量分率は 58.1 %、定圧比熱が 1.98 kJ/(kg·K)であることがわかる。

再生器の再生温度は温水入口温度である 88°C から 2°C を減じて 86°C である。従って、再生器出口水溶液状態は再生温度 86°C と凝縮温度 40°C の交点であり、比エンタルピーは 208 kJ/kg、質量分率は 60.9 %、定圧比熱は 1.91 kJ/(kg·K)となる。

稀溶液の質量分率が 58.1 %、稀溶液の質量分率が 60.9 %であることがわかったため、これらを式 15.14 に代入すると溶液循環比は、a_W=60.9÷(60.9-58.1)=22.3 となる。

凝縮温度は 40°C であり飽和水と飽和蒸気の比エンタルピーは 168 kJ/kg と 2,575 kJ/kg となる。同様に蒸発温度 5°C における飽和蒸気の比エンタルピーは 2,511 kJ/kg となる。従って式 15.7 を用いると、冷媒循環量 m_R は m_R = 105 ÷ (2,511 – 168) = 0.045 kg/s となる。式 15.13 を用いて稀溶液と濃溶液の循環量を計算すると、m_{SAo} = 0.045×22.3=1.004 kg/s、m_{SAi} = 0.045×(22.3 – 1)=0.959 kg/s となる。

式 15.19 を用いると再生器入口水溶液の比エンタルピーは、h_{SDi} = (0.0045×2,575 – 0.0959×208 – 151)÷0.1004 = 164 kJ/kg となる。従って、溶液熱交換器の交換熱量は式 15.24 により、Q_X = 1.004×(164 – 106) = 58kW である。これを式 15.25 に代入し、吸収器入口比エンタルピーは、h_{SAi} = 208 – 58÷0.959 = 148 kJ/kg となる。以上によりデューリング線図上の運転点が特定された。これをプロットすると図 15.5 となる。

【例題 15.3】

例題 15.2 の吸収冷凍機の蒸発器、凝縮器・吸収器、溶液熱交換器、再生器の伝熱係数 KA を求めよ。

【解】

第 8 章の式 8.30 を用いて蒸発器の温度効率は、ε_E = (12.5 – 7.0) / (12.5 – 5) = 0.73 となる。片側流体温度一定のため、式 8.23 により移動単位数は NTU_E = –log(1 – 0.73) = 1.32 となる。冷水の熱容量流量は 4.186×275 / 60=19.18 kW/K であり、これを式 8.21 に代入して伝熱係数は KA_E = 1.32 × 19.18 = 25.3 kW/K となる。

凝縮器・吸収器の場合も同様であり、式 8.31 により温度効率は、ε_{CDA} = (35 – 31) / (40 – 31) = 0.44。伝熱係数は KA_{CDA} = –log(1 – 0.44) × (4.186×918 / 60) = 37.6 kW/K である。

溶液熱交換器は両流体の温度が変化するため、蒸発器や凝縮器とは異なった計算が必要となる。熱容量流量は稀溶液側が 1.98×1.004 = 1.99 kW/K、濃溶液側が 1.91×0.959 = 1.83 kW/K となる。式 8.19 により熱通過有効度は、ε_X = 58 / (1.83 × (86 – 40)) = 0.69 である。向流型の場合の移動単位数計算式（式 8.24）により NTU_X が 2.04 となり、これに熱容量流量 1.83 kW/K を乗じて伝熱係数は KA_X = 2.04×1.83 = 3.73 kW/K となる。

再生器に流入した水溶液 SDi は、再生器内の蒸気を吸収して SDi2 になる。両者の比エンタルピーは 164 kJ/kg で等しく、水溶液の物性計算を行うと、質量分率が 54.8 %、温度が 74.1°C、容積比熱が 2.08 kJ/(kg·K)となる。式 15.22 を用いて冷媒蒸発も考慮した容積比熱を計算すると、((208 × (54.8 / 60.1) + 2575 × (1 – 54.8 / 60.1)) – 164) / (86 – 74.1) = 23.2 kJ/(kg·K)となる。従って、水溶液側の熱容量流量は 23.2×1.004 = 23.3 kW/K、温水側の熱容量流量は 4.186×7.2 = 30.1 kW/K となる。式 8.19 により熱通過有効度は、ε_D = 151 / (23.3 × (88 – 74.1)) = 0.46 である。向流型の場合の移動単位数計算式（式 8.24）により NTU_D が 0.79 となり、これに熱容量流量 23.3 kW/K を乗じて伝熱係数は KA_D = 0.79×23.3 = 18.5 kW/K となる。

2）二重効用吸収冷凍サイクル

二重効用吸収冷凍サイクルの蒸発器、吸収器、溶液熱交換器の基礎式は、単効用の場合の基礎式と同様であるため、低温再生器、高温再生器、凝縮器について説明する。

・冷媒および水溶液循環量

高温側再生器に送る水溶液の配分比を r_H [-]とすると、高温再生器と低温再生器の冷媒循環量（m_{RH}

と m_{RL} [kg/s]）は式 15.26 で計算できる。また、それぞれの再生器の出入口水溶液流量（m_{SHDi}, m_{SHDo}, m_{SLDi}, m_{SLDo} [kg/s]）は式 15.27 と 15.28 で計算できる。ところで、近年、冷凍機の期間的な効率を高めるために溶液循環ポンプにインバータを搭載した機種が登場している。これは、低負荷時に溶液循環量を抑えて溶液循環比を高めることで効率向上を狙ったものである。このような機種では負荷に応じて効率が最大化するように溶液循環量が調整されるため、定格値で一定とすることはできない。この計算法については後述する。

$$m_{RH}=m_R r_H \quad , \quad m_{RH}=m_R(1-r_H) \tag{15.26}$$

$$m_{SHDi}=m_{RH} a_w \quad , \quad m_{SHDo}=m_{RH}(a_w-1) \tag{15.27}$$

$$m_{SLDi}=m_{RL} a_w \quad , \quad m_{SLDo}=m_{RL}(a_w-1) \tag{15.28}$$

・低温再生器

高温再生器で発生した蒸気によって加熱を行うため、加熱量は蒸気の凝縮熱であり式 15.29 で計算する。水溶液側からの熱交換量は単効用の場合と同様に式 15.30 で計算を行う。ただし、片側温度一定の熱交換のため、第 8 章の式 8.24 を用いて出口状態を推定する。

$$Q_{LD}=m_{RH}(h_{RVHDo}-h_{RLLDo}) \tag{15.29}$$

$$Q_{LD}=m_{SLDi} c_{pSLD}(t_{SLDo}-t_{SLDi2}) \tag{15.30}$$

・高温再生器

低温再生器と同様に水溶液側の交換熱量は式 15.31 で表現する。蒸気またはガスによる直接加熱のため、片側温度一定の熱交換器として扱う。

$$Q_{HD}=m_{SHDi} c_{pSHD}(t_{SHDo}-t_{SHDi2}) \tag{15.31}$$

・凝縮器

単効用の場合と同様に低温再生器で発生した蒸気を凝縮させるための熱が存在する他、低温再生器に投入した高温再生器廃熱蒸気の凝縮水を冷却するための熱が必要となる。従って、凝縮器の交換熱量は式 15.32 となる。

$$Q_{CD}=m_R(h_{RVCDi}-h_{RLCDo})+m_{RH}(h_{RLLo}-h_{RLCDo})-Q_{L2} \tag{15.32}$$

【例題 15.4】

例題 15.1 の二重効用の直焚吸収冷温水機についてデューリング線図上の運転点を計算せよ。ただし、高温再生器側への水溶液配分比 r_H は 55 %、蒸発温度は 5 °C、凝縮温度は 40 °C、再生温度は 155 °C（飽和蒸気 98 °C）とする。

【解】

例題 15.1 の結果により、冷却能力は 1,758kW、冷却水による熱除去量は 2,907kW、再生器に供給される熱量は 1,209 kW である。

吸収器出口水溶液状態（SAo）は凝縮温度 40°C と蒸発温度 5°C の交点であり、臭化リチウムの物性計算を行うと比エンタルピーは 106 kJ/kg、質量分率は 58.1 %、定圧比熱が 1.98 kJ/(kg·K)であることがわかる。

再生器出口水溶液状態（SHDo）は再生温度 155°C と飽和蒸気温度 98°C の交点であり、比エンタルピーは 341 kJ/kg、質量分率は 61.3 %、定圧比熱は 1.89 kJ/(kg·K)となる。

低温再生器の出口水溶液（SLDo）は質量分率 61.3 %、飽和蒸気温度 40°C の交点であり、温度 87°C、比エンタルピーは 213 kJ/kg、定圧比熱は 1.89 kJ/(kg·K)となる。これは高温溶液熱交換器の濃溶液出口状態とも等しい。

稀溶液の質量分率である 58.1 %、稀溶液の質量分率である 61.3 %を式 15.14 に代入すると溶液循環比は、a_W=61.3÷(61.3–58.1)=18.6 となる。

高温再生器飽和蒸気温度である 98 °C の飽和水と飽和蒸気の比エンタルピーは 411 kJ/kg と 2,673 kJ/kg となる。同様に低温再生器飽和蒸気温度である 40°C における飽和水と飽和蒸気の比エンタルピーは 168 kJ/kg

と 2,575 kJ/kg となる。また、蒸発温度 5°C における飽和蒸気の比エンタルピーは 2,511 kJ/kg である。

式 15.7 を用いると、冷媒循環量 m_R は m_R = 1758 ÷ (2,511 – 168) = 0.75 kg/s となる。式 15.26 により、高温再生器側の冷媒循環量は 0.75×0.55=0.41 kg/s、低温再生器側は 0.75×(1–0.55)=0.34 kg/s となる。さらに式 15.27 と 15.28 を適用し、各々の稀溶液および濃溶液の循環量は、m_{SHDi}= 0.41×18.6=7.6kg/s、m_{SHDo}= 0.41×(18.6-1) = 7.2kg/s、m_{SLDi} = 0.34×18.6 = 6.3kg/s、m_{SLDo}= 0.34×(18.6-1) = 6.0kg/s である。また、式 15.13 を用いて吸収器の出入口流量を計算すると、m_{SAo} = 0.75×18.6=13.95 kg/s、m_{SAi} = 0.75×(18.6 – 1)=13.20 kg/s となる。

式 15.32 により凝縮器の処理熱量は、Q_{CD} = 0.34×(2,575–168) + 0.41×(411–168) – 1,209×0.05= 845 kW となる。さらに冷却水による総除去熱量である 2,907kW から凝縮器処理熱量を差し引いて、吸収器の処理熱量は 2,907–845=2,062 kW となる。

式 15.15 により吸収器入口水溶液（SAi）の比エンタルピーは、h_{SAi} = 106 + (2,062 – 0.75×(2,511–106)) / 13.2=126 kJ/kg である。低温溶液熱交換器の交換熱量は式 15.25 により、Q_X = 13.2×(213–126)=1,148kW となり、これを式 15.24 に代入することで濃溶液側の出口比エンタルピーは、h_{LDi} = 106+1,148÷13.95=188 kJ/kg となる。以上によりデューリング線図上の運転点が特定された。これをプロットすると図 15.7 となる。

15.3 計算法

まず臭化リチウム水溶液の物性計算法について示した後、これを用いて吸収冷凍サイクルを計算する方法を示す。さらに温水吸収冷凍機と直焚吸収冷温水機クラスの作成法について説明する。

15.3.1 臭化リチウム水溶液の計算

1）デューリング線図の計算

プログラム 15.1 に水溶液温度、質量分率、飽和蒸気温度の計算処理を示す。式 15.2 の実装であり、デューリング線図描画に必要な処理である。25~37 行は水溶液温度と飽和温度から質量分率を計算するメソッドであるが、この場合には式 15.2 が解析的に解けないため、ニュートン・ラフソン法で質量分率を未知数にして収束計算を行う。39~53 行は式 15.2 と式 15.3 の係数計算である。

プログラム 15.1　水溶液温度、質量分率、飽和蒸気温度

Popolo.ThermophysicalProperty.LithiumBromide class

```
 1 /// <summary>溶液温度と質量分率から飽和温度を計算する</summary>
 2 /// <param name="liquidTemperature">水溶液の温度[K]</param>
 3 /// <param name="massFraction">水溶液の質量分率[-]</param>
 4 /// <returns>平衡蒸気の飽和温度[K]</returns>
 5 public static double GetVaporTemperatureFromLiquidTemperatureAndMassFraction
 6   (double liquidTemperature, double massFraction)
 7 {
 8   double alb, blb;
 9   getCoefficient(massFraction, out alb, out blb);
10   return alb + blb * liquidTemperature;
11 }
12
13 /// <summary>飽和温度と質量分率から溶液温度を計算する</summary>
14 /// <param name="vaporTemperature">水蒸気の飽和温度[K]</param>
15 /// <param name="massFraction">水溶液の質量分率[-]</param>
16 /// <returns>水溶液の温度[K]</returns>
17 public static double GetLiquidTemperatureFromVaporTemperatureAndMassFraction
18   (double vaporTemperature, double massFraction)
19 {
20   double alb, blb;
21   getCoefficient(massFraction, out alb, out blb);
22   return (vaporTemperature - alb) / blb;
23 }
24
25 /// <summary>溶液温度と飽和温度から質量分率を計算する</summary>
26 /// <param name="liquidTemperature">水溶液の温度[K]</param>
27 /// <param name="vaporTemperature">平衡蒸気の飽和温度[K]</param>
28 /// <returns>水溶液の質量分率[-]</returns>
29 public static double GetMassFractionFromLiquidTemperatureAndVaporTemperature
30   (double liquidTemperature, double vaporTemperature)
31 {
32   Roots.ErrorFunction eFnc = delegate (double mf)
33   {
```

```
34     return vaporTemperature - GetVaporTemperatureFromLiquidTemperatureAndMassFraction(liquidTemperature, mf);
35   };
36   return Roots.Newton(eFnc, 0.5, 0.001, 0.00001, 0.00001, 20);
37 }
38
39 /// <summary>物性値近似係数を計算する</summary>
40 /// <param name="massFraction">水溶液の質量分率[-]</param>
41 /// <param name="ca">出力：係数 a</param>
42 /// <param name="cb">出力：係数 b</param>
43 private static void getCoefficient(double massFraction, out double ca, out double cb)
44 {
45   const double a0 = -22.8937; const double a1 = 152.554;
46   const double a2 = -254.786; const double a3 = 152.949;
47   const double a4 = -171.599;
48   const double b0 = 1.09851;  const double b1 = -0.394508;
49
50   double mf = Math.Min(1, Math.Max(0, massFraction));
51   ca = a0 + mf * (a1 + mf * (a2 + mf * (a3 + mf * mf * mf * mf * a4)));
52   cb = b0 + mf * b1;
53 }
```

2）比エンタルピー関連の計算

プログラム 15.2 に比エンタルピー関連の計算処理を示す。9~20 行は式 15.5 の実装であり、近似係数の計算は 64~81 行のメソッドで行う。質量分率のかわりに平衡蒸気の温度が与えられた場合には、プログラム 15.1 に定義したメソッドを使って質量分率を求めた後に比エンタルピーの計算を行う（22~31 行）。比エンタルピーと質量分率あるいは飽和温度が与えられた場合には、水溶液温度は解析的に求められないため、33~48 行あるいは 50~62 行に示すように収束計算によって解を求める。

プログラム 15.2　比エンタルピー関連の処理

Popolo.ThermophysicalProperty.LithiumBromide class

```
 1 /// <summary>比エンタルピー近似係数</summary>
 2 readonly static double[] C_LBN =
 3   new double[] { -2024.33, 16330.9, -48816.1, 6.302948e4, -2.913705e4 };
 4 readonly static double[] D_LBN =
 5   new double[] { 18.2829, -116.91757, 3.248041e2, -4.034184e2, 1.8520569e2 };
 6 readonly static double[] E_LBN =
 7   new double[] {-3.7008214e-2,2.8877666e-1,-8.1313015e-1,9.9116628e-1,-4.4441207e-1};
 8
 9 /// <summary>溶液温度と質量分率から比エンタルピーを計算する</summary>
10 /// <param name="liquidTemperature">水溶液の温度[K]</param>
11 /// <param name="massFraction">水溶液の質量分率[-]</param>
12 /// <returns>水溶液の比エンタルピー[kJ/kg]</returns>
13 public static double GetEnthalpyFromLiquidTemperatureAndMassFraction
14   (double liquidTemperature, double massFraction)
15 {
16   double clb, dlb, elb;
17   getCoefficient(massFraction, out clb, out dlb, out elb);
18   double ltk = liquidTemperature - 273.15;
19   return clb + ltk * (dlb + ltk * elb);
20 }
21
22 /// <summary>溶液温度と飽和温度から比エンタルピーを計算する</summary>
23 /// <param name="liquidTemperature">水溶液の温度[K]</param>
24 /// <param name="vaporTemperature">平衡蒸気の温度[K]</param>
25 /// <returns>水溶液の比エンタルピー[kJ/kg]</returns>
26 public static double GetEnthalpyFromLiquidTemperatureAndVaporTemperature
27   (double liquidTemperature, double vaporTemperature)
28 {
29   double mf = GetMassFractionFromLiquidTemperatureAndVaporTemperature(liquidTemperature, vaporTemperature);
30   return GetEnthalpyFromLiquidTemperatureAndMassFraction(liquidTemperature, mf);
31 }
32
33 /// <summary>比エンタルピーと質量分率から溶液温度を計算する</summary>
34 /// <param name="enthalpy">水溶液の比エンタルピー[kJ/kg]</param>
35 /// <param name="massFraction">水溶液の質量分率[-]</param>
36 /// <returns>水溶液の温度[K]</returns>
37 public static double GetLiquidTemperatureFromEnthalpyAndMassFraction(double enthalpy, double massFraction)
38 {
39   double clb, dlb, elb;
40   getCoefficient(massFraction, out clb, out dlb, out elb);
41
42   //ニュートン法で収束計算
43   Roots.ErrorFunction eFnc = delegate (double lTemp)
```

```
 44   {
 45     return enthalpy - (clb + lTemp * (dlb + lTemp * elb));
 46   };
 47   return Roots.Newton(eFnc, 40, 0.001, 0.00001, 0.00001) + 273.15;
 48 }
 49
 50 /// <summary>比エンタルピーと飽和温度から溶液温度を計算する</summary>
 51 /// <param name="enthalpy">水溶液の比エンタルピー[kJ/kg]</param>
 52 /// <param name="vaporTemperature">平衡蒸気の飽和温度[K]</param>
 53 /// <returns>水溶液の温度[K]</returns>
 54 public static double GetMassFractionFromEnthalpyAndVaporTemperature(double enthalpy, double vaporTemperature)
 55 {
 56   Roots.ErrorFunction eFnc = delegate (double mf)
 57   {
 58     return GetLiquidTemperatureFromEnthalpyAndMassFraction(enthalpy, mf)
 59     - GetLiquidTemperatureFromVaporTemperatureAndMassFraction(vaporTemperature, mf);
 60   };
 61   return Roots.Newton(eFnc, 0.5, 0.001, 0.00001, 0.00001);
 62 }
 63
 64 /// <summary>物性値近似係数を計算する</summary>
 65 /// <param name="massFraction">水溶液の質量分率[-]</param>
 66 /// <param name="cc">出力：係数a</param>
 67 /// <param name="cd">出力：係数b</param>
 68 /// <param name="ce">出力：係数b</param>
 69 private static void getCoefficient(double massFraction, out double cc, out double cd, out double ce)
 70 {
 71   massFraction = Math.Min(1.0, massFraction);
 72   cc = C_LBN[4];
 73   cd = D_LBN[4];
 74   ce = E_LBN[4];
 75   for (int i = 3; 0 <= i; i--)
 76   {
 77     cc = cc * massFraction + C_LBN[i];
 78     cd = cd * massFraction + D_LBN[i];
 79     ce = ce * massFraction + E_LBN[i];
 80   }
 81 }
```

3）その他の物性

プログラム 15.3 にその他の物性計算の処理を示す。1~10 行は密度の計算であり、水密度と質量比で合算した値を出力している。12~26 行は比エンタルピーの計算処理であり、式 15.6 の実装である。

プログラム 15.3　その他の物性の計算

Popolo.ThermophysicalProperty.LithiumBromide class

```
  1 /// <summary>密度[kg/m3]を計算する</summary>
  2 /// <param name="liquidTemperature">臭化リチウム水溶液の温度[K]</param>
  3 /// <param name="massFraction">臭化リチウム水溶液の質量分率[-]</param>
  4 /// <returns>密度[kg/m3]</returns>
  5 public static double GetDensity(double liquidTemperature, double massFraction)
  6 {
  7   const double LBD = 3460;  //臭化リチウムの密度[kg/m3]
  8   double wd = Water.GetLiquidDensity(liquidTemperature);
  9   return LBD * massFraction + wd * (1 - massFraction);
 10 }
 11
 12 /// <summary>溶液温度と質量分率から比熱を計算する</summary>
 13 /// <param name="liquidTemperature">水溶液の温度[K]</param>
 14 /// <param name="massFraction">水溶液の質量分率[-]</param>
 15 /// <returns>水溶液の比熱[kJ/(kgK)]</returns>
 16 public static double GetSpecificHeat(double liquidTemperature, double massFraction)
 17 {
 18   double dlb = D_LBN[4];
 19   double elb = E_LBN[4];
 20   for (int i = 3; 0 <= i; i--)
 21   {
 22     dlb = dlb * massFraction + D_LBN[i];
 23     elb = elb * massFraction + E_LBN[i];
 24   }
 25   return dlb + 2 * elb * (liquidTemperature - 273.15);
 26 }
```

4）臭化リチウム水溶液クラスの作成

物性の管理を容易にするため、臭化リチウム水溶液のクラスを作成する。プログラム 15.4 に LithiumBromide クラスのインスタンスを示す。

プログラム **15.4**　**LithiumBromide** クラスのインスタンス

Popolo.ThermophysicalProperty.LithiumBromide class

```
/// <summary>比エンタルピーを取得する</summary>
public double Enthalpy { get; private set; }

/// <summary>水溶液の温度[K]を取得する</summary>
public double LiquidTemperature { get; private set; }

/// <summary>飽和蒸気の温度[K]を取得する</summary>
public double VaporTemperature { get; private set; }

/// <summary>質量分率[-]を取得する</summary>
public double MassFraction { get; private set; }

/// <summary>比熱[kJ/(kgK)]を取得する</summary>
public double SpecificHeat { get; private set; }
```

インスタンスの生成はクラスメソッドにより行う。プログラム 15.5 にインスタンスの生成処理を示す。1~16 行と 18~33 行で定義した 2 つが主なメソッドであり、入力条件として与えられた 2 つの物性値から他のすべての物性値を計算し、プロパティに設定した後にインスタンスを出力する。35~64 行は 2 つの物性から必要最低限の反復計算処理を行った上で上記の 2 つのメソッドを呼び出す。

プログラム **15.5**　インスタンスの生成処理

Popolo.ThermophysicalProperty.LithiumBromide class

```
/// <summary>溶液温度と質量分率からインスタンスを生成する</summary>
/// <param name="liquidTemperature">水溶液の温度[K]</param>
/// <param name="massFraction">水溶液の質量分率[-]</param>
/// <returns>臭化リチウム水溶液のインスタンス</returns>
public static LithiumBromide MakeFromLiquidTemperatureAndMassFraction
  (double liquidTemperature, double massFraction)
{
  LithiumBromide lb = new LithiumBromide();
  lb.LiquidTemperature = liquidTemperature;
  lb.MassFraction = massFraction;
  lb.VaporTemperature = GetVaporTemperatureFromLiquidTemperatureAndMassFraction
    (liquidTemperature, massFraction);
  lb.Enthalpy = GetEnthalpyFromLiquidTemperatureAndMassFraction(liquidTemperature, massFraction);
  lb.SpecificHeat = GetSpecificHeat(liquidTemperature, massFraction);
  return lb;
}

/// <summary>飽和温度と質量分率からインスタンスを生成する</summary>
/// <param name="vaporTemperature">平衡蒸気の飽和温度[K]</param>
/// <param name="massFraction">水溶液の質量分率[-]</param>
/// <returns>臭化リチウム水溶液のインスタンス</returns>
public static LithiumBromide MakeFromVaporTemperatureAndMassFraction
  (double vaporTemperature, double massFraction)
{
  LithiumBromide lb = new LithiumBromide();
  lb.VaporTemperature = vaporTemperature;
  lb.MassFraction = massFraction;
  lb.LiquidTemperature = GetLiquidTemperatureFromVaporTemperatureAndMassFraction
    (vaporTemperature, massFraction);
  lb.Enthalpy = GetEnthalpyFromLiquidTemperatureAndMassFraction(lb.LiquidTemperature, massFraction);
  lb.SpecificHeat = GetSpecificHeat(lb.LiquidTemperature, massFraction);
  return lb;
}

/// <summary>溶液温度と飽和温度からインスタンスを生成する</summary>
/// <param name="liquidTemperature">水溶液の温度[K]</param>
/// <param name="vaporTemperature">平衡蒸気の飽和温度[K]</param>
/// <returns>臭化リチウム水溶液のインスタンス</returns>
public static LithiumBromide MakeFromLiquidTemperatureAndVaporTemperature
  (double liquidTemperature, double vaporTemperature)
{
  double mf = GetMassFractionFromLiquidTemperatureAndVaporTemperature(liquidTemperature, vaporTemperature);
  return MakeFromLiquidTemperatureAndMassFraction(liquidTemperature, mf);
}

/// <summary>比エンタルピーと質量分率からインスタンスを生成する</summary>
/// <param name="enthalpy">水溶液の比エンタルピー[kJ/kg]</param>
/// <param name="massFraction">水溶液の質量分率[-]</param>
/// <returns>臭化リチウム水溶液のインスタンス</returns>
public static LithiumBromide MakeFromEnthalpyAndMassFraction(double enthalpy, double massFraction)
{
```

```
52   double lt = GetLiquidTemperatureFromEnthalpyAndMassFraction(enthalpy, massFraction);
53   return MakeFromLiquidTemperatureAndMassFraction(lt, massFraction);
54 }
55
56 /// <summary>水溶液温度と飽和温度からインスタンスを生成する</summary>
57 /// <param name="enthalpy">水溶液の比エンタルピー[kJ/kg]</param>
58 /// <param name="vaporTemperature">平衡蒸気の飽和温度[K]</param>
59 /// <returns>臭化リチウム水溶液のインスタンス</returns>
60 public static LithiumBromide MakeFromEnthalpyAndVaporTemperature(double enthalpy, double vaporTemperature)
61 {
62   double mf = GetMassFractionFromEnthalpyAndVaporTemperature(enthalpy, vaporTemperature);
63   return MakeFromVaporTemperatureAndMassFraction(vaporTemperature, mf);
64 }
```

15.3.2　単効用吸収冷凍サイクルの計算

定数宣言をプログラム 15.6 に示す。後述する二重効用吸収冷凍サイクルも同じ値を用いる。

プログラム 15.6　定数宣言

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
1 /// <summary>定格蒸発温度[C]</summary>
2 /// <remarks>真空度保持の観点から 5C 程度が下限</remarks>
3 public const double NOM_EVP_TEMP = 5;
4
5 /// <summary>定格凝縮温度[C]</summary>
6 public const double NOM_CND_TEMP = 40;
```

1）伝熱係数の推定

プログラム 15.7 に定格性能から伝熱係数を推定する処理を示す。1~68 行が主たるメソッドであり、処理内容は例題 15.2 および例題 15.3 と同じである。70~81 行のメソッドを用いて 24, 25 行で蒸発器および凝縮器の伝熱係数を推定する。片側温度一定の熱交換器として熱通過有効度 ε を求めた後、伝熱係数 KA を出力するメソッドである。27~51 行で吸収冷凍サイクルの運転点および冷媒・水溶液循環量を確定し（例題 15.2 の内容）、53~67 行で溶液熱交換器および再生器の伝熱係数を計算する（例題 15.3 の内容）。ただし Q_{L2}=0 としている。53~57 行は溶液熱交換器の伝熱係数計算であり、向流型の熱交換器として熱容量流量と入口温度から推定する。59~67 行は再生器の伝熱係数計算である。式 15.22 で説明した比熱が必要となり、その計算処理は 83~93 行である。

プログラム 15.7　伝熱係数の推定（単効用吸収冷凍サイクル）

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
1 /// <summary>定格条件から機器の伝熱係数[kW/K]を計算する</summary>
2 /// <param name="chWaterITemperature">冷水入口温度[C]</param>
3 /// <param name="chWaterOTemperature">冷水出口温度[C]</param>
4 /// <param name="chWaterFlowRate">冷水質量流量[kg/s]</param>
5 /// <param name="cdWaterITemperature">冷却水入口温度[C]</param>
6 /// <param name="cdWaterOTemperature">冷却水出口温度[C]</param>
7 /// <param name="cdWaterFlowRate">冷却水質量流量[kg/s]</param>
8 /// <param name="htWaterITemperature">温水入口温度[C]</param>
9 /// <param name="hotWaterFlowRate">温水質量流量[kg/s]</param>
10 /// <param name="dsbTemperatureApploach">再生温度アプローチ[C]</param>
11 /// <param name="evaporatorKA">蒸発器伝熱係数[kW/K]</param>
12 /// <param name="condensorKA">凝縮器（吸収器）伝熱係数[kW/K]</param>
13 /// <param name="desorborKA">再生器伝熱係数[kW/K]</param>
14 /// <param name="hexKA">溶液熱交換器の伝熱係数[kW/K]</param>
15 /// <param name="solFlowRate">稀溶液循環量[kg/s]</param>
16 /// <param name="desorbHeat">高温再生器投入熱量[kW]</param>
17 public static void GetHeatTransferCoefficients
18   (double chWaterITemperature, double chWaterOTemperature, double chWaterFlowRate, double cdWaterITemperature,
19   double cdWaterOTemperature, double cdWaterFlowRate, double htWaterITemperature, double hotWaterFlowRate,
20   double dsbTemperatureApploach, out double evaporatorKA, out double condensorKA, out double desorborKA,
21   out double hexKA, out double solFlowRate, out double desorbHeat)
22 {
23   //凝縮器（吸収器）と蒸発器の伝熱係数 KA[kW/K]
24   evaporatorKA = getRefrigerantHexKA(chWaterITemperature, chWaterOTemperature, chWaterFlowRate, NOM_EVP_TEMP);
25   condensorKA = getRefrigerantHexKA(cdWaterITemperature, cdWaterOTemperature, cdWaterFlowRate, NOM_CND_TEMP);
26
27   //再生器投入熱量[kW]
28   double qE = chWaterFlowRate * WATER_SPECIFIC_HEAT * (chWaterITemperature - chWaterOTemperature);
29   double qCDAB = cdWaterFlowRate * WATER_SPECIFIC_HEAT * (cdWaterOTemperature - cdWaterITemperature);
```

```
  desorbHeat = qCDAB - qE;
  double hotWaterOutletTemperature =
    htWaterITemperature - desorbHeat / (WATER_SPECIFIC_HEAT * hotWaterFlowRate);

  //再生器と吸収器出口の水溶液状態
  LithiumBromide lbDo = LithiumBromide.MakeFromLiquidTemperatureAndVaporTemperature
    (htWaterITemperature - dsbTemperatureApploach + 273.15, NOM_CND_TEMP + 273.15);
  LithiumBromide lbAo = LithiumBromide.MakeFromLiquidTemperatureAndVaporTemperature
    (NOM_CND_TEMP + 273.15, NOM_EVP_TEMP + 273.15);

  //溶液循環比[-]
  double aW = lbDo.MassFraction / (lbDo.MassFraction - lbAo.MassFraction);

  //冷媒の比エンタルピー[kJ/kg]
  double hRVDo = Water.GetSaturatedVaporEnthalpy(NOM_CND_TEMP);
  double hRLEi = Water.GetSaturatedLiquidEnthalpy(NOM_CND_TEMP);
  double hRVEo = Water.GetSaturatedVaporEnthalpy(NOM_EVP_TEMP);

  //冷媒および水溶液の循環量[kg/s]
  double mR = qE / (hRVEo - hRLEi);
  solFlowRate = mR * aW;
  double mAi = solFlowRate - mR;

  //溶液熱交換器の伝熱係数KA[kW/K]
  double hSDi = (mR * hRVDo + lbDo.Enthalpy * mAi - desorbHeat) / solFlowRate;
  double qX = (hSDi - lbAo.Enthalpy) * solFlowRate;
  hexKA = HeatExchange.GetHeatTransferCoefficient(lbDo.LiquidTemperature, lbAo.LiquidTemperature,
    lbDo.SpecificHeat * mAi, lbAo.SpecificHeat * solFlowRate, qX, HeatExchange.FlowType.CounterFlow);

  //再生器の伝熱係数KA[kW/K]
  LithiumBromide lbDi2 = LithiumBromide.MakeFromEnthalpyAndVaporTemperature(hSDi, NOM_CND_TEMP + 273.15);
  double cp = getSolutionAverageSpecificHeat(lbDi2, lbDo);
  double mcHW = hotWaterFlowRate * WATER_SPECIFIC_HEAT;
  double mcSL = solFlowRate * cp;
  double mcMin = Math.Min(mcHW, mcSL);
  double mcMax = Math.Max(mcHW, mcSL);
  double effectiveness = desorbHeat / (mcMin * (htWaterITemperature - (lbDi2.LiquidTemperature - 273.15)));
  desorborKA = HeatExchange.GetNTU(effectiveness, mcMin / mcMax, HeatExchange.FlowType.CounterFlow) * mcMin;
}

/// <summary>蒸発・凝縮器の伝熱係数[kW/K]を計算する</summary>
/// <param name="iwTemperature">入口水温[C]</param>
/// <param name="owTemperature">出口水温[C]</param>
/// <param name="wFlowRate">水の流量[kg/s]</param>
/// <param name="rTemperature">蒸発・凝縮温度[C]</param>
/// <returns>蒸発・凝縮器の伝熱係数[kW/K]</returns>
private static double getRefrigerantHexKA
  (double iwTemperature, double owTemperature, double wFlowRate, double rTemperature)
{
  double effectiveness = (iwTemperature - owTemperature) / (iwTemperature - rTemperature);
  return -Math.Log(1 - effectiveness) * wFlowRate * WATER_SPECIFIC_HEAT;
}

/// <summary>水溶液１基準の比熱[kJ/kgK]を計算する</summary>
/// <param name="sol1">水溶液 1</param>
/// <param name="sol2">水溶液２（一部蒸発）</param>
/// <returns>水溶液１基準の比熱[kJ/kgK]</returns>
private static double getSolutionAverageSpecificHeat(LithiumBromide sol1, LithiumBromide sol2)
{
  double hw = Water.GetSaturatedVaporEnthalpy(sol1.VaporTemperature - 273.15);
  double slRate = sol1.MassFraction / sol2.MassFraction;
  double hco = sol2.Enthalpy * slRate + hw * (1 - slRate);
  return (hco - sol1.Enthalpy) / (sol2.LiquidTemperature - sol1.LiquidTemperature);
}
```

2）出口状態の計算（成り行き）

期間シミュレーションを行うにあたっては、主に 2 つの計算処理が求められる。1 つは、温水流量を固定した場合に成り行きでどのような冷水出口温度になるかという計算であり、もう 1 つは、ある冷水出口温度を実現するために必要となる温水流量はいくらかという計算である。ここではまず前者の冷水出口温度計算について説明する。

計算のフローを図 15.8 に示す。2 段階の収束計算が必要であり、内側の計算ループでは溶液熱交換器の処理熱量 Q_X を 2 通りの計算法で算出し、これらが一致するように再生器投入熱量 Q_D を修正す

る。収束後、必要となる温水入口温度が実際の温水入口温度よりも高い場合には、その差を誤差として出力する。逆に低い場合には温水流量を減らせるので、余剰流量を誤差として出力する。第 2 章で解説した一変数関数の最適化手法を用いて誤差が最小化するように冷水出口温度 $t_{ch,o}$ を調整する。

プログラム 15.8 に内側の計算ループを示す。1~93 が主な計算処理である。36~68 行で定義した誤差関数が 0 になるように 72 行でニュートン・ラフソン法の収束計算をかける。再生器投入熱量の初期値は単効用吸収冷凍サイクルの一般的な COP である 0.75 から割り戻して設定する。38~55 行で、仮定した再生器投入熱量から凝縮温度を計算し、冷媒・水溶液流量や冷凍サイクルの運転点を求める。57~60 行は式 15.12 と式 15.15 による冷却水側処理熱量にもとづく溶液熱交換器の交換熱量である。62~65 行は式 15.24 と式 15.25 による溶液熱交換器伝熱係数にもとづく交換熱量である。これらが一致するように収束計算を行う。74 行以降は外部ループ用の計算処理である。78~84 行で再生器が必要とする温水入口温度を計算する。必要な温水入口温度と実際の入口温度との大小に応じて 86~92 行で出力する誤差を切り替える。

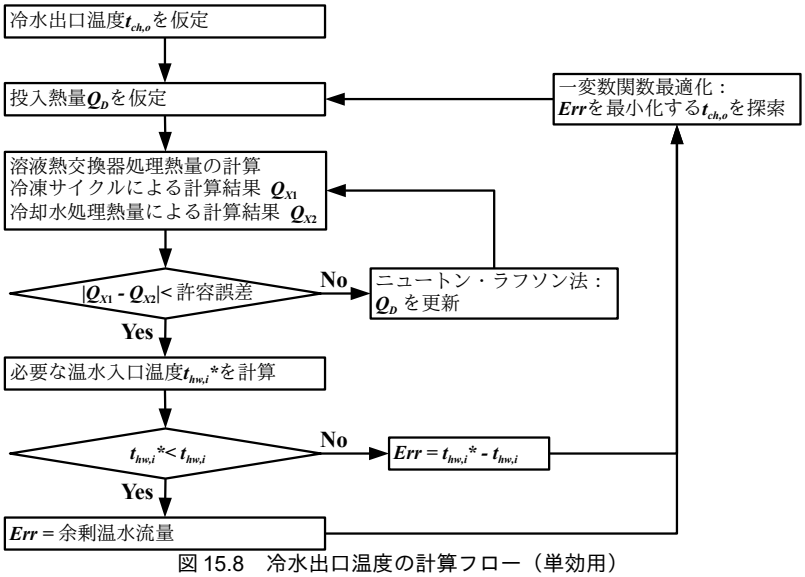


図 15.8　冷水出口温度の計算フロー（単効用）

プログラム **15.8**　単効用吸収冷凍サイクル内部ループ

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
 1 /// <summary>単効用吸収冷凍サイクルの誤差評価</summary>
 2 /// <param name="chWaterITemperature">冷水入口温度[C]</param>
 3 /// <param name="chWaterFlowRate">冷水質量流量[kg/s]</param>
 4 /// <param name="cdWaterITemperature">冷却水入口温度[C]</param>
 5 /// <param name="cdWaterFlowRate">冷却水質量流量[kg/s]</param>
 6 /// <param name="htWaterITemperature">温水入口温度[C]</param>
 7 /// <param name="htWaterFlowRate">温水質量流量[kg/s]</param>
 8 /// <param name="evaporatorKA">蒸発器伝熱係数[kW/K]</param>
 9 /// <param name="condensorKA">凝縮器伝熱係数[kW/K]</param>
10 /// <param name="desorborKA">再生器伝熱係数[kW/K]</param>
11 /// <param name="hexKA">溶液熱交換器伝熱係数[kW/K]</param>
12 /// <param name="solFlowRate">水溶液流量[kg/s]</param>
13 /// <param name="chWaterOTemperature">冷水出口温度[C]</param>
14 /// <param name="cdWaterOTemperature">出力:冷却水出口温度[C]</param>
15 /// <param name="htWaterOTemperature">出力:温水出口温度[C]</param>
16 /// <returns>単効用吸収冷凍サイクル誤差</returns>
17 private static double getError(double chWaterITemperature, double chWaterFlowRate,
18   double cdWaterITemperature, double cdWaterFlowRate, double htWaterITemperature,
19   double htWaterFlowRate, double evaporatorKA, double condensorKA, double desorborKA, double hexKA,
20   double solFlowRate, double chWaterOTemperature,
21   out double cdWaterOTemperature, out double htWaterOTemperature)
22 {
23   //蒸発温度の計算
```

```
  double qE = chWaterFlowRate * WATER_SPECIFIC_HEAT * (chWaterITemperature - chWaterOTemperature);
  double evaporatingTemperature = getRefrigerantTemperature
    (chWaterITemperature, chWaterOTemperature, chWaterFlowRate, evaporatorKA);
  double hRVEo = Water.GetSaturatedVaporEnthalpy(evaporatingTemperature);

  LithiumBromide lbAo = null;
  LithiumBromide lbDo = null;
  double qX = 0;
  double condensingTemperature = 0;
  double tcdo = 0;
  Roots.ErrorFunction eFnc = delegate (double dsvH)
  {
    //凝縮温度と比エンタルピー
    double qCDAB = qE + dsvH;
    tcdo = cdWaterITemperature + qCDAB / (cdWaterFlowRate * WATER_SPECIFIC_HEAT);
    condensingTemperature = getRefrigerantTemperature
    (cdWaterITemperature, tcdo, cdWaterFlowRate, condensorKA);
    double hRVDo = Water.GetSaturatedVaporEnthalpy(condensingTemperature);
    double hRLCDo = Water.GetSaturatedLiquidEnthalpy(condensingTemperature);

    //冷媒流量と溶液循環比[-]
    double mR = qE / (hRVEo - hRLCDo);
    double aW = solFlowRate / mR;
    double mSAi = solFlowRate - mR;

    //吸収器・再生器の出口水溶液状態
    lbAo = LithiumBromide.MakeFromLiquidTemperatureAndVaporTemperature
    (condensingTemperature + 273.15, evaporatingTemperature + 273.15);
    lbDo = LithiumBromide.MakeFromVaporTemperatureAndMassFraction
    (condensingTemperature + 273.15, aW / (aW - 1) * lbAo.MassFraction);

    //冷却水熱量にもとづく溶液熱交換器の処理熱量
    double qAB = qCDAB - mR * (hRVDo - hRLCDo);
    double hSAi = lbAo.Enthalpy + (qAB - mR * (hRVEo - lbAo.Enthalpy)) / mSAi;
    qX = (lbDo.Enthalpy - hSAi) * mSAi;

    //溶液熱交換器伝熱係数にもとづく処理熱量
    double qX2 = HeatExchange.GetHeatTransfer
    (lbDo.LiquidTemperature, lbAo.LiquidTemperature, lbDo.SpecificHeat * mSAi,
    lbAo.SpecificHeat * solFlowRate, hexKA, HeatExchange.FlowType.CounterFlow);

    return qX - qX2;
  };

  //投入熱量を収束計算
  double desorbHeat = qE / 0.75;
  desorbHeat = Roots.Newton(eFnc, desorbHeat, 0.001, 0.0001, desorbHeat * 0.001, 20);

  //冷却水と温水の出口温度を計算
  cdWaterOTemperature = tcdo;
  htWaterOTemperature = htWaterITemperature - desorbHeat / (htWaterFlowRate * WATER_SPECIFIC_HEAT);

  //再生器が必要とする再生温度を計算
  LithiumBromide lbDi = LithiumBromide.MakeFromEnthalpyAndMassFraction
    (lbAo.Enthalpy + qX / solFlowRate, lbAo.MassFraction);
  LithiumBromide lbDi2 = LithiumBromide.MakeFromEnthalpyAndVaporTemperature
    (lbDi.Enthalpy, condensingTemperature + 273.15);
  double desorbTemp = getDesorbTemperature
    (desorborKA, desorbHeat, htWaterFlowRate, lbDi2, lbDo, solFlowRate);

  //温水入口温度が必要温度未満の場合
  if (0 < desorbTemp - (htWaterITemperature + 273.15))
    return desorbTemp - (htWaterITemperature + 273.15);
  //必要温度以上の場合には余剰温水流量を計算
  else
    return htWaterFlowRate - getHotWaterFlowRate
      (desorborKA, desorbHeat, htWaterITemperature + 273.15, lbDi2, lbDo, solFlowRate);
}

/// <summary>蒸発・凝縮温度[C]を計算する</summary>
/// <param name="iwTemperature">入口水温[C]</param>
/// <param name="owTemperature">出口水温[C]</param>
/// <param name="wFlowRate">水の流量[kg/s]</param>
/// <param name="heatTransferCoefficient">蒸発・凝縮器の伝熱係数[kW/K]</param>
/// <returns>蒸発・凝縮温度[C]</returns>
private static double getRefrigerantTemperature
  (double iwTemperature, double owTemperature, double wFlowRate, double heatTransferCoefficient)
{
  double ntu = heatTransferCoefficient / (wFlowRate * WATER_SPECIFIC_HEAT);
  double effectiveness = 1 - Math.Exp(-ntu);
```

```
104   return iwTemperature - (iwTemperature - owTemperature) / effectiveness;
105 }
106
107 /// <summary>再生温度[K]を計算する</summary>
108 /// <param name="desorborKA">再生器の伝熱係数[kW/K]</param>
109 /// <param name="desorbHeat">再生熱量[kW]</param>
110 /// <param name="hwFlowRate">温水流量[kg/s]</param>
111 /// <param name="iSolution">入口水溶液</param>
112 /// <param name="oSolution">出口水溶液</param>
113 /// <param name="slFlowRate">水溶液流量[kg/s]</param>
114 /// <returns>再生温度[K]</returns>
115 private static double getDesorbTemperature
116   (double desorborKA, double desorbHeat, double hwFlowRate,
117   LithiumBromide iSolution, LithiumBromide oSolution, double slFlowRate)
118 {
119   double cp = getSolutionAverageSpecificHeat(iSolution, oSolution);
120   double mcHW = hwFlowRate * WATER_SPECIFIC_HEAT;
121   double mcSL = slFlowRate * cp;
122   double mcMin = Math.Min(mcHW, mcSL);
123   double mcMax = Math.Max(mcHW, mcSL);
124   double effectiveness = HeatExchange.GetEffectiveness
125     (desorborKA / mcMin, mcMin / mcMax, HeatExchange.FlowType.CounterFlow);
126   return desorbHeat / (mcMin * effectiveness) + iSolution.LiquidTemperature;
127 }
128
129 /// <summary>温水流量[kg/s]を計算する</summary>
130 /// <param name="desorborKA">再生器の伝熱係数[kW/K]</param>
131 /// <param name="desorbHeat">再生熱量[kW]</param>
132 /// <param name="hwITemperature">再生温度[K]</param>
133 /// <param name="iSolution">入口水溶液</param>
134 /// <param name="oSolution">出口水溶液</param>
135 /// <param name="slFlowRate">水溶液流量[kg/s]</param>
136 /// <returns>温水流量[kg/s]</returns>
137 private static double getHotWaterFlowRate
138   (double desorborKA, double desorbHeat, double hwITemperature,
139   LithiumBromide iSolution, LithiumBromide oSolution, double slFlowRate)
140 {
141   Roots.ErrorFunction eFnc = delegate (double hwf)
142   {
143     return getDesorbTemperature(desorborKA, desorbHeat, hwf, iSolution, oSolution, slFlowRate)
144       - hwITemperature;
145   };
146   return Roots.Newton(eFnc, 0.0001, 0.001, 0.001, 0.0001, 20);
147 }
148
149
```

外部ループをプログラム 15.9 に示す。プログラム 15.8 で作成した関数を用いて 24~29 行で誤差関数を定義する。32 行で黄金分割法を用い、誤差関数を最小化する冷水出口温度を探索する。探索の最低温度は蒸発温度、最高温度は冷水入口温度である。

プログラム 15.9　単効用吸収冷凍サイクルの計算（成り行き）

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
 1 /// <summary>出口温度を計算する（成り行き）</summary>
 2 /// <param name="chWaterITemperature">冷水入口温度[C]</param>
 3 /// <param name="chWaterFlowRate">冷水質量流量[kg/s]</param>
 4 /// <param name="cdWaterITemperature">冷却水入口温度[C]</param>
 5 /// <param name="cdWaterFlowRate">冷却水質量流量[kg/s]</param>
 6 /// <param name="htWaterITemperature">温水入口温度[C]</param>
 7 /// <param name="htWaterFlowRate">温水質量流量[kg/s]</param>
 8 /// <param name="evaporatorKA">蒸発器伝熱係数[kW/K]</param>
 9 /// <param name="condensorKA">凝縮器伝熱係数[kW/K]</param>
10 /// <param name="desorborKA">再生器伝熱係数[kW/K]</param>
11 /// <param name="hexKA">溶液熱交換器伝熱係数[kW/K]</param>
12 /// <param name="solFlowRate">水溶液流量[kg/s]</param>
13 /// <param name="chWaterOTemperature">出力:冷水出口温度[C]</param>
14 /// <param name="cdWaterOTemperature">出力:冷却水出口温度[C]</param>
15 /// <param name="htWaterOTemperature">出力:温水出口温度[C]</param>
16 public static void GetOutletTemperatures
17   (double chWaterITemperature, double chWaterFlowRate, double cdWaterITemperature,
18   double cdWaterFlowRate, double htWaterITemperature, double htWaterFlowRate, double evaporatorKA,
19   double condensorKA, double desorborKA, double hexKA, double solFlowRate,
20   out double chWaterOTemperature, out double cdWaterOTemperature, out double htWaterOTemperature)
21 {
22   double tcdo = 0;
23   double tho = 0;
24   Minimization.MinimizeFunction mFnc = delegate (double tcho)
```

```
25   {
26     return getError(chWaterITemperature, chWaterFlowRate, cdWaterITemperature, cdWaterFlowRate,
27       htWaterITemperature, htWaterFlowRate, evaporatorKA, condensorKA, desorborKA, hexKA, solFlowRate, tcho,
28       out tcdo, out tho);
29   };
30
31   chWaterOTemperature = NOM_EVP_TEMP + 0.001;
32   Minimization.GoldenSection(ref chWaterOTemperature, chWaterITemperature - 0.001, mFnc);
33   cdWaterOTemperature = tcdo;
34   htWaterOTemperature = tho;
35 }
```

3）出口状態の計算（冷水出口温度指定）

冷水出口温度を指定した場合の計算も成り行き計算と同様に 2 重の収束計算が必要である。内部ループは成り行き計算の場合と同じであるが、外部ループでは冷水出口温度を固定して温水流量を調整する。プログラム 15.10 に計算処理を示す。プログラム 15.8 で作成した関数を用いて 23~28 行で誤差関数を定義する。31 行で黄金分割法を用い、誤差関数を最小化する温水流量比（定格流量に対する流量比）を探索する。

プログラム 15.10　単効用吸収冷凍サイクルの計算（冷水出口温度指定）

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
 1 /// <summary>出口温度を計算する（冷水出口温度指定）</summary>
 2 /// <param name="chWaterITemperature">冷水入口温度[C]</param>
 3 /// <param name="chWaterFlowRate">冷水質量流量[kg/s]</param>
 4 /// <param name="cdWaterITemperature">冷却水入口温度[C]</param>
 5 /// <param name="cdWaterFlowRate">冷却水質量流量[kg/s]</param>
 6 /// <param name="htWaterITemperature">温水入口温度[C]</param>
 7 /// <param name="htWaterFlowRate">温水質量流量[kg/s]</param>
 8 /// <param name="evaporatorKA">蒸発器伝熱係数[kW/K]</param>
 9 /// <param name="condensorKA">凝縮器伝熱係数[kW/K]</param>
10 /// <param name="desorborKA">再生器伝熱係数[kW/K]</param>
11 /// <param name="hexKA">溶液熱交換器伝熱係数[kW/K]</param>
12 /// <param name="solFlowRate">水溶液流量[kg/s]</param>
13 /// <param name="chWaterOTemperatureSP">冷水出口温度設定値[C]</param>
14 /// <param name="cdWaterOTemperature">出力:冷却水出口温度[C]</param>
15 /// <param name="htWaterOTemperature">出力:温水出口温度[C]</param>
16 public static void GetOutletTemperatures(double chWaterITemperature, double chWaterFlowRate,
17   double cdWaterITemperature, double cdWaterFlowRate, double htWaterITemperature, double htWaterFlowRate,
18   double evaporatorKA, double condensorKA, double desorborKA, double hexKA, double solFlowRate,
19   double chWaterOTemperatureSP, out double cdWaterOTemperature, out double htWaterOTemperature)
20 {
21   double tcdo = 0;
22   double tho = 0;
23   Minimization.MinimizeFunction mFnc = delegate (double hwr)
24   {
25     return getError(chWaterITemperature, chWaterFlowRate, cdWaterITemperature, cdWaterFlowRate,
26       htWaterITemperature, htWaterFlowRate * hwr, evaporatorKA, condensorKA, desorborKA, hexKA,
27       solFlowRate, chWaterOTemperatureSP, out tcdo, out tho);
28   };
29
30   double hwRatio = 0.01;
31   Minimization.GoldenSection(ref hwRatio, 1.0, mFnc);
32   cdWaterOTemperature = tcdo;
33   htWaterOTemperature = tho * hwRatio + htWaterITemperature * (1 - hwRatio);
34 }
```

15.3.3　二重効用吸収冷凍サイクルの計算

定数宣言をプログラム 15.11 に示す。

プログラム 15.11　定数宣言

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
 1 /// <summary>定格再生温度（水溶液）[C]</summary>
 2 /// <remarks>軟鋼腐食の問題から 160C 程度が限界</remarks>
 3 public const double NOM_DSB_LIQ_TEMP = 155;
 4
 5 /// <summary>定格再生温度（飽和蒸気）[C]</summary>
 6 /// <remarks>圧力容器とさせないために大気圧未満にすると 98C 程度となる</remarks>
 7 public const double NOM_DSB_VAP_TEMP = 98;
 8
 9 /// <summary>高温再生器投入熱量に対する熱ロスの比率</summary>
10 private const double HEATLOSS_RATE = 0.05;
```

1）伝熱係数の推定

プログラム 15.12 に定格性能から伝熱係数を推定する処理を示す。単効用の場合とほぼ同様な処理であるが、大きな違いは高温再生器と低温再生器に配分する水溶液の比率が不明な点である。例題 15.4 では 55 %という前提で計算を行ったが現実にはこのような情報はカタログ等の資料には記載が無いため、推定する必要がある。そこで本モデルでは高温再生器で発生する蒸気凝縮熱を全て低温再生器側で使い切るという条件を満たすように、86 行で溶液配分比を収束計算で求める。誤差関数は 57~83 行である。その他の計算は例題 15.4 に記した通りである。

プログラム 15.12　伝熱係数の推定（二重効用吸収冷凍サイクル）

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
/// <summary>定格条件から機器の伝熱係数[kW/K]を計算する</summary>
/// <param name="chWaterITemperature">冷水入口温度[C]</param>
/// <param name="chWaterOTemperature">冷水出口温度[C]</param>
/// <param name="cdWaterITemperature">冷却水入口温度[C]</param>
/// <param name="cdWaterOTemperature">冷却水出口温度[C]</param>
/// <param name="chWaterFlowRate">冷水流量[kg/s]</param>
/// <param name="cdWaterFlowRate">冷却水流量[kg/s]</param>
/// <param name="evaporatorKA">蒸発器伝熱係数[kW/K]</param>
/// <param name="condensorKA">凝縮器（吸収器）伝熱係数[kW/K]</param>
/// <param name="lowDesorborKA">低温再生器伝熱係数[kW/K]</param>
/// <param name="lHexKA">低温溶液熱交換器の伝熱係数[kW/K]</param>
/// <param name="solFlowRate">稀溶液循環量[kg/s]</param>
/// <param name="desorbHeat">高温再生器投入熱量[kW]</param>
public static void GetHeatTransferCoefficients
  (double chWaterITemperature, double chWaterOTemperature, double chWaterFlowRate,
  double cdWaterITemperature, double cdWaterOTemperature, double cdWaterFlowRate,
  out double evaporatorKA, out double condensorKA, out double lowDesorborKA,
  out double lHexKA, out double solFlowRate, out double desorbHeat)
{
  //凝縮器（吸収器）と蒸発器の伝熱係数 KA[kW/K]
  evaporatorKA = getRefrigerantHexKA(chWaterITemperature, chWaterOTemperature, chWaterFlowRate, NOM_EVP_TEMP);
  condensorKA = getRefrigerantHexKA(cdWaterITemperature, cdWaterOTemperature, cdWaterFlowRate, NOM_CND_TEMP);

  //再生器投入熱量[kW]の計算
  double qE = chWaterFlowRate * WATER_SPECIFIC_HEAT * (chWaterITemperature - chWaterOTemperature);
  double qCDAB = cdWaterFlowRate * WATER_SPECIFIC_HEAT * (cdWaterOTemperature - cdWaterITemperature);
  double qD = desorbHeat = (qCDAB - qE) / (1 - HEATLOSS_RATE);

  //水溶液状態の計算
  LithiumBromide lbHDo = LithiumBromide.MakeFromLiquidTemperatureAndVaporTemperature
    (NOM_DSB_LIQ_TEMP + 273.15, NOM_DSB_VAP_TEMP + 273.15);
  LithiumBromide lbAo = LithiumBromide.MakeFromLiquidTemperatureAndVaporTemperature
    (NOM_CND_TEMP + 273.15, NOM_EVP_TEMP + 273.15);
  LithiumBromide lbLDo = LithiumBromide.MakeFromVaporTemperatureAndMassFraction
    (NOM_CND_TEMP + 273.15, lbHDo.MassFraction);

  //溶液循環比[-]
  double aW = lbHDo.MassFraction / (lbHDo.MassFraction - lbAo.MassFraction);

  //冷媒比エンタルピー[kJ/kg]の計算
  double hRVHDo = Water.GetSaturatedVaporEnthalpy(NOM_DSB_VAP_TEMP);
  double hRLHDo = Water.GetSaturatedLiquidEnthalpy(NOM_DSB_VAP_TEMP);
  double hRVLDo = Water.GetSaturatedVaporEnthalpy(NOM_CND_TEMP);
  double hRLEi = Water.GetSaturatedLiquidEnthalpy(NOM_CND_TEMP);
  double hRVEo = Water.GetSaturatedVaporEnthalpy(NOM_EVP_TEMP);

  //冷媒循環量[kg/s]
  double mR = qE / (hRVEo - hRLEi);
  solFlowRate = mR * aW;

  //溶液状態保持変数
  LithiumBromide lbLDi = null;  //低温再生器入口水溶液
  LithiumBromide lbAi = null;  //低温熱交換器出口水溶液
  double mRH = 0;  //高温側冷媒流量[kg/s]
  double mRL = 0;  //低温側冷媒流量[kg/s]

  //誤差関数の定義
  Roots.ErrorFunction eFnc = delegate (double rhgRate)
  {
    //冷媒流量[kg/s]の計算
    mRH = mR * rhgRate;
    mRL = mR - mRH;
```

```
63     double mSAo = mR * aW;
64     double mSAi = mSAo - mR;
65
66     //凝縮器・吸収器の処理熱量[kW]
67     double qCD = (hRLHDo - hRLEi) * mRH + (hRVLDo - hRLEi) * mRL - qD * HEATLOSS_RATE;
68     double qAB = qCDAB - qCD;
69
70     //低温溶液熱交換器出口水溶液
71     double hSAi = lbAo.Enthalpy + (qAB - mR * (hRVEo - lbAo.Enthalpy)) / mSAi;
72     lbAi = LithiumBromide.MakeFromEnthalpyAndMassFraction(hSAi, lbHDo.MassFraction);
73
74     //低温再生器入口水溶液
75     double hLDi = lbAo.Enthalpy + (lbLDo.Enthalpy - lbAi.Enthalpy) * mSAi / mSAo;
76     lbLDi = LithiumBromide.MakeFromEnthalpyAndMassFraction(hLDi, lbAo.MassFraction);
77
78     //低温再生器投入熱量
79     double qLD1 = (hRVHDo - hRLHDo) * mRH;
80     double qLD2 = hRVLDo * mRL + lbLDo.Enthalpy * (mRL * (aW - 1)) - hLDi * (mRL * aW);
81
82     return qLD1 - qLD2;
83   };
84
85   //溶液配分比[-]を収束計算
86   double rRatio = Roots.Newton(eFnc, 0.5, 0.001, 0.0001, 0.0001, 20);
87
88   //低温再生器の伝熱係数 KA[kW/K]の計算
89   LithiumBromide lbLDi2 = LithiumBromide.MakeFromEnthalpyAndVaporTemperature
90     (lbLDi.Enthalpy, NOM_CND_TEMP + 273.15);
91   double cp = getSolutionAverageSpecificHeat(lbLDi2, lbLDo);
92   double effectiveness = (lbLDo.LiquidTemperature - lbLDi2.LiquidTemperature)
93     / (NOM_DSB_VAP_TEMP + 273.15 - lbLDi2.LiquidTemperature);
94   lowDesorborKA = -Math.Log(1 - effectiveness) * (cp * (mRL * aW));
95
96   //溶液熱交換器の伝熱係数 KA[kW/K]の計算
97   double qLX = (lbLDo.Enthalpy - lbAi.Enthalpy) * mR * (aW - 1);
98   double mcH = lbLDo.SpecificHeat * mR * (aW - 1);
99   double mcC = lbAo.SpecificHeat * mR * aW;
100  lHexKA = HeatExchange.GetHeatTransferCoefficient
101    (lbLDo.LiquidTemperature, lbAo.LiquidTemperature, mcC, mcH, qLX, HeatExchange.FlowType.CounterFlow);
102 }
```

2）出口状態の計算（成り行き）

投入熱量を固定して成り行きでの冷水温度を計算する処理と、冷水温度を固定して投入熱量を計算する処理の 2 つが必要となる。単効用と同様にいずれの処理も内外 2 重の収束計算が必要となる。投入熱量を固定して冷水出口温度を計算するためのフローを図 15.9 に示す。外側のループで冷水出口温度を、内側のループで溶液配分比を収束計算する。プログラム 15.13 に内側ループの処理を示す。55~84 行で定義した誤差関数を用いて、高温再生器の廃熱蒸気の凝縮熱量と低温再生器の必要再生熱が一致するように収束計算をする。

プログラム 15.13　二重効用吸収冷凍サイクル内部ループ

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
/// <summary>二重効用吸収冷凍サイクル誤差関数</summary>
/// <param name="chWaterITemperature">冷水入口温度[C]</param>
/// <param name="chWaterFlowRate">冷水流量[kg/s]</param>
/// <param name="cdWaterITemperature">冷却水入口温度[C]</param>
/// <param name="cdWaterFlowRate">冷却水流量[kg/s]</param>
/// <param name="evaporatorKA">蒸発器の伝熱係数[kW/K]</param>
/// <param name="condensorKA">凝縮器器の伝熱係数[kW/K</param>
/// <param name="lowDesorborKA">低温再生器の伝熱係数[kW/K]</param>
/// <param name="lHexKA">低温溶液熱交換器の伝熱係数[kW/K]</param>
/// <param name="desorbHeat">高温再生器投入熱量[kW]</param>
/// <param name="solFlowRate">水溶液流量[kg/s]</param>
/// <param name="chWaterOTemperature">冷水出口温度[C]</param>
/// <param name="cdWaterOTemperature">出力：冷却水出口温度[C]</param>
/// <param name="dsbTemperature">出力:高温再生器再生温度[C]</param>
/// <param name="evpTemperature">出力：蒸発温度[C]</param>
/// <param name="cndTemperature">出力：凝縮温度[C]</param>
/// <param name="thinMFraction">出力:稀溶液質量分率[-]</param>
/// <param name="thickMFraction">出力:濃溶液質量分率[-]</param>
/// <returns>二重効用吸収冷凍サイクル誤差</returns>
private static double getError
  (double chWaterITemperature, double chWaterFlowRate, double cdWaterITemperature, double cdWaterFlowRate,
```

```
 22   double evaporatorKA, double condensorKA, double lowDesorborKA, double lHexKA, double desorbHeat,
 23   double solFlowRate, double chWaterOTemperature, out double cdWaterOTemperature, out double dsbTemperature,
 24   out double evpTemperature, out double cndTemperature, out double thinMFraction, out double thickMFraction)
 25 {
 26   //冷却水出口温度
 27   double qE = chWaterFlowRate * WATER_SPECIFIC_HEAT * (chWaterITemperature - chWaterOTemperature);
 28   double qCDAB = qE + desorbHeat / (1 + HEATLOSS_RATE);
 29   cdWaterOTemperature = cdWaterITemperature + qCDAB / (cdWaterFlowRate * WATER_SPECIFIC_HEAT);
 30
 31   //蒸発温度と凝縮温度
 32   evpTemperature = getRefrigerantTemperature
 33     (chWaterITemperature, chWaterOTemperature, chWaterFlowRate, evaporatorKA);
 34   cndTemperature = getRefrigerantTemperature
 35     (cdWaterITemperature, cdWaterOTemperature, cdWaterFlowRate, condensorKA);
 36
 37   //凝縮・蒸発温度における冷媒エンタルピー
 38   double hRVLDo = Water.GetSaturatedVaporEnthalpy(cndTemperature);
 39   double hRLLDo = Water.GetSaturatedLiquidEnthalpy(cndTemperature);
 40   double hRVEo = Water.GetSaturatedVaporEnthalpy(evpTemperature);
 41
 42   //冷媒流量と溶液循環比[-]
 43   double mR = qE / (hRVEo - hRLLDo);
 44   double aW = solFlowRate / mR;
 45
 46   LithiumBromide lbAo = LithiumBromide.MakeFromLiquidTemperatureAndVaporTemperature
 47     (cndTemperature + 273.15, evpTemperature + 273.15);
 48   LithiumBromide lbLDo = LithiumBromide.MakeFromVaporTemperatureAndMassFraction
 49     (cndTemperature + 273.15, aW / (aW - 1) * lbAo.MassFraction);
 50
 51   double mSAo = solFlowRate;
 52   double mSAi = solFlowRate - mR;
 53   double tDesorb = 0;
 54   double tcndK = cndTemperature + 273.15;
 55   Roots.ErrorFunction eFnc = delegate (double rRatio)
 56   {
 57     //冷媒・水溶液流量
 58     double mRH = mR * rRatio;
 59     double mRL = mR - mRH;
 60     double mSLDi = mSAo * (1 - rRatio);
 61     double mSLDo = mSLDi - mRL;
 62
 63     //低温再生器の再生温度
 64     double qLX = HeatExchange.GetHeatTransfer
 65     (lbLDo.LiquidTemperature, lbAo.LiquidTemperature, lbLDo.SpecificHeat * mSAi,
 66     lbAo.SpecificHeat * mSAo, lHexKA, HeatExchange.FlowType.CounterFlow);
 67     LithiumBromide lbLDi = LithiumBromide.MakeFromEnthalpyAndMassFraction
 68     (lbAo.Enthalpy + qLX / mSAo, lbAo.MassFraction);
 69     LithiumBromide lbLDi2 = LithiumBromide.MakeFromEnthalpyAndVaporTemperature(lbLDi.Enthalpy, tcndK);
 70     tDesorb = getDesorbTemperature(lowDesorborKA, mSLDi, lbLDi2, lbLDo);
 71
 72     //低温再生器の処理熱量1[kW]
 73     double hRVHD = Water.GetSaturatedVaporEnthalpy(tDesorb - 273.15);
 74     double hRLHD = Water.GetSaturatedLiquidEnthalpy(tDesorb - 273.15);
 75     double qLD1 = (hRVHD - hRLHD) * mRH;
 76     //低温再生器の処理熱量2[kW]
 77     double qLD2 = hRVLDo * mRL + lbLDo.Enthalpy * mSLDo - lbLDi.Enthalpy * mSLDi;
 78
 79     //冷却水除去熱量
 80     qCDAB = mRH * (hRLHD - hRLLDo) + mRL * (hRVLDo - hRLLDo)
 81     + mR * (hRVEo - lbAo.Enthalpy) + mSAi * (lbLDo.Enthalpy - lbAo.Enthalpy) - qLX;
 82
 83     return qLD1 - qLD2;
 84   };
 85
 86   //溶液配分比を収束計算
 87   Roots.Newton(eFnc, 0.5, 0.001, 0.0001, 0.0001, 20);
 88
 89   //再生温度と溶液質量分率を出力
 90   dsbTemperature = LithiumBromide.GetLiquidTemperatureFromVaporTemperatureAndMassFraction
 91     (tDesorb, lbLDo.MassFraction) - 273.15;
 92   thinMFraction = lbAo.MassFraction;
 93   thickMFraction = lbLDo.MassFraction;
 94
 95   return (qCDAB - qE) - desorbHeat;
 96 }
 97
 98 /// <summary>再生温度[K]を計算する</summary>
 99 /// <param name="desorborKA">再生器の伝熱係数[kW/K]</param>
100 /// <param name="slFlowRate">水溶液流量[kg/s]</param>
101 /// <param name="iSolution">入口水溶液</param>
```

```
102 /// <param name="oSolution">出口水溶液</param>
103 /// <returns>再生温度[K]</returns>
104 private static double getDesorbTemperature
105   (double desorborKA, double slFlowRate, LithiumBromide iSolution, LithiumBromide oSolution)
106 {
107   double cp = getSolutionAverageSpecificHeat(iSolution, oSolution);
108   double effectiveness = 1 - Math.Exp(-desorborKA / (cp * slFlowRate));
109   return (oSolution.LiquidTemperature-iSolution.LiquidTemperature) /effectiveness+ iSolution.LiquidTemperature;
110 }
```

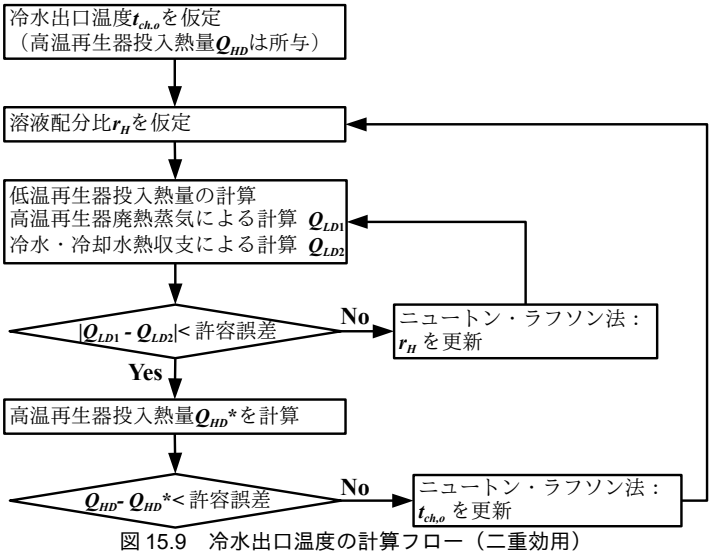


図 15.9　冷水出口温度の計算フロー（二重効用）

プログラム 15.14 に二重効用冷凍サイクルの成り行き計算処理を示す。プログラム 15.13 を用いて 30~35 行で誤差関数を定義し、38 行でニュートン・ラフソン法を用いて冷水出口温度を収束計算する。蒸発温度を初期値とする。冷水および冷却水に異常値が入力されると吸収冷凍サイクルが成立しないため、26 行で上下限値内に入るように調整を行う（実装は 49~77 行）。調整が発生した場合には 42~46 行に示すように計算結果をシフトさせる。

プログラム 15.14　二重効用冷凍サイクルの計算（成り行き）

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
  1 /// <summary>冷水出口温度[C]を計算する</summary>
  2 /// <param name="chWaterITemperature">冷水入口温度[C]</param>
  3 /// <param name="chWaterFlowRate">冷水流量[kg/s]</param>
  4 /// <param name="cdWaterITemperature">冷却水入口温度[C]</param>
  5 /// <param name="cdWaterFlowRate">冷却水流量[kg/s]</param>
  6 /// <param name="evaporatorKA">蒸発器の伝熱係数[kW/K]</param>
  7 /// <param name="condensorKA">凝縮器器の伝熱係数[kW/K</param>
  8 /// <param name="lowDesorborKA">低温再生器の伝熱係数[kW/K]</param>
  9 /// <param name="lHexKA">低温溶液熱交換器の伝熱係数[kW/K]</param>
 10 /// <param name="solFlowRate">水溶液流量[kg/s]</param>
 11 /// <param name="desorbHeat">高温再生器投入熱量[kW]</param>
 12 /// <param name="cdWaterOTemperature">出力:冷却水出口温度[C]</param>
 13 /// <param name="dsbTemperature">出力:高温再生器再生温度[C]</param>
 14 /// <param name="evpTemperature">出力：蒸発温度[C]</param>
 15 /// <param name="cndTemperature">出力：凝縮温度[C]</param>
 16 /// <param name="thinMFraction">出力:稀溶液質量分率[-]</param>
 17 /// <param name="thickMFraction">出力:濃溶液質量分率[-]</param>
 18 /// <returns>冷水出口温度[C]</returns>
 19 public static double GetChilledWaterOutletTemperature
 20   (double chWaterITemperature, double chWaterFlowRate, double cdWaterITemperature, double cdWaterFlowRate,
 21   double evaporatorKA, double condensorKA, double lowDesorborKA, double lHexKA, double solFlowRate,
 22   double desorbHeat, out double cdWaterOTemperature, out double dsbTemperature, out double evpTemperature,
 23   out double cndTemperature, out double thinMFraction, out double thickMFraction)
 24 {
 25   double dtCH, dtCD;
 26   adjustRange(ref chWaterITemperature, ref cdWaterITemperature, out dtCH, out dtCD);
 27
 28   double tdsv, tcnd, tevp, tcdo, wth, wtk;
```

```
29   tdsv = tcnd = tevp = tcdo = wth = wtk = 0;
30   Roots.ErrorFunction eFnc = delegate (double tcho)
31   {
32     return getError(chWaterITemperature, chWaterFlowRate, cdWaterITemperature, cdWaterFlowRate,
33   evaporatorKA, condensorKA, lowDesorborKA, lHexKA, desorbHeat, solFlowRate, tcho,
34   out tcdo, out tdsv, out tevp, out tcnd, out wth, out wtk);
35   };
36
37   //冷水出口温度を収束計算
38   double chilledWaterOutletTemperature = Roots.Newton(eFnc, NOM_EVP_TEMP + 0.01, 0.001, 0.0001, 0.01, 20);
39   dsbTemperature = tdsv;
40   thinMFraction = wth;
41   thickMFraction = wtk;
42   cdWaterOTemperature = tcdo + dtCD;
43   evpTemperature = tevp + dtCH;
44   cndTemperature = tcnd + dtCD;
45
46   return chilledWaterOutletTemperature + dtCH;
47 }
48
49 /// <summary>冷水と冷却水温度を調整する</summary>
50 /// <param name="tCHi">冷水入口温度</param>
51 /// <param name="tCDi">冷却水入口温度</param>
52 /// <param name="dtCH">出力:冷水調整幅</param>
53 /// <param name="dtCD">出力:冷却水調整幅</param>
54 private static void adjustRange(ref double tCHi, ref double tCDi, out double dtCH, out double dtCD)
55 {
56   dtCH = dtCD = 0;
57   if (tCHi < 3)
58   {
59     dtCH = tCHi - 5;
60     tCHi = 5;
61   }
62   else if (18 < tCHi)
63   {
64     dtCH = tCHi - 18;
65     tCHi = 18;
66   }
67   if (tCDi < 20)
68   {
69     dtCD = tCDi - 20;
70     tCDi = 20;
71   }
72   else if (37 < tCDi)
73   {
74     dtCD = tCDi - 37;
75     tCDi = 37;
76   }
77 }
```

3）出口状態の計算（冷水出口温度指定）

冷水出口温度を固定して高温再生器投入熱量を計算する処理をプログラム 15.15 に示す。プログラム 15.13 を用いて 31~37 行で誤差関数を定義し、42 行でニュートン・ラフソン法を用いて冷水出口温度を収束計算する。初期値は、二重効用冷凍サイクルの標準的な COP である 1.4 で冷水処理熱量を割り戻すことで得る。冷水および冷却水の上下限値の調整については成行計算の場合と同様である。

プログラム 15.15　二重効用冷凍サイクルの計算（冷水出口温度指定）

Popolo.HVAC.HeatSource.AbsorptionRefrigerationCycle class

```
1 /// <summary>高温再生器投入熱量[kW]を計算する</summary>
2 /// <param name="chWaterITemperature">冷水入口温度[C]</param>
3 /// <param name="chWaterFlowRate">冷水流量[kg/s]</param>
4 /// <param name="cdWaterITemperature">冷却水入口温度[C]</param>
5 /// <param name="cdWaterFlowRate">冷却水流量[kg/s]</param>
6 /// <param name="evaporatorKA">蒸発器の伝熱係数[kW/K]</param>
7 /// <param name="condensorKA">凝縮器器の伝熱係数[kW/K</param>
8 /// <param name="lowDesorborKA">低温再生器の伝熱係数[kW/K]</param>
9 /// <param name="lHexKA">低温溶液熱交換器の伝熱係数[kW/K]</param>
10 /// <param name="solFlowRate">水溶液流量[kg/s]</param>
11 /// <param name="chWaterOTemperature">冷水出口温度設定値[C]</param>
12 /// <param name="cdWaterOTemperature">出力:冷却水出口温度[C]</param>
13 /// <param name="dsbTemperature">出力:高温再生器再生温度[C]</param>
14 /// <param name="evpTemperature">出力：蒸発温度[C]</param>
15 /// <param name="cndTemperature">出力：凝縮温度[C]</param>
16 /// <param name="thinMFraction">出力:稀溶液質量分率[-]</param>
17 /// <param name="thickMFraction">出力:濃溶液質量分率[-]</param>
```

```
/// <returns>高温再生器投入熱量[kW]</returns>
public static double GetDesorbHeat
  (double chWaterITemperature, double chWaterFlowRate, double cdWaterITemperature, double cdWaterFlowRate,
  double evaporatorKA, double condensorKA, double lowDesorborKA, double lHexKA, double solFlowRate,
  double chWaterOTemperature, out double cdWaterOTemperature, out double dsbTemperature,
  out double evpTemperature, out double cndTemperature, out double thinMFraction, out double thickMFraction)
{
  double dtCH, dtCD;
  adjustRange(ref chWaterITemperature, ref cdWaterITemperature, out dtCH, out dtCD);
  chWaterOTemperature += dtCH;

  double tdsv, tcnd, tevp, tcdo, wth, wtk;
  tdsv = tcnd = tevp = tcdo = wth = wtk = 0;
  Roots.ErrorFunction eFnc = delegate (double dsvH)
  {
    return getError
    (chWaterITemperature, chWaterFlowRate, cdWaterITemperature, cdWaterFlowRate, evaporatorKA, condensorKA,
    lowDesorborKA, lHexKA, dsvH, solFlowRate, chWaterOTemperature,
    out tcdo, out tdsv, out tevp, out tcnd, out wth, out wtk);
  };

  //再生器投入熱量を収束計算
  double qE = chWaterFlowRate * WATER_SPECIFIC_HEAT * (chWaterITemperature - chWaterOTemperature);
  double desorbHeat = qE / 1.4;
  desorbHeat = Roots.Newton(eFnc, desorbHeat, 0.001, 0.0001, desorbHeat * 0.001, 20);
  dsbTemperature = tdsv;
  thinMFraction = wth;
  thickMFraction = wtk;
  cdWaterOTemperature = tcdo + dtCD;
  evpTemperature = tevp + dtCH;
  cndTemperature = tcnd + dtCD;

  return desorbHeat;
}
```

15.3.4　「温水吸収冷凍機クラス」の作成

15.3.2 節のプログラムを用いて温水吸収冷凍機のクラスを作成する。定数宣言とインスタンス変数・プロパティの定義をプログラム 15.16 に示す。

プログラム **15.16**　定数宣言とインスタンス変数・プロパティの定義

Popolo.HVAC.HeatSource.HotWaterAbsorptionChiller class

```
/// <summary>再生温度と水溶液温度のアプローチ[C]</summary>
private const double DESORB_TEMPERATURE_APPROACH = 2;

/// <summary>水の定圧比熱[kJ/(kgK)]</summary>
private const double WATER_SPECIFIC_HEAT = 4.186;

/// <summary>蒸発器の伝熱性能[kW/K]</summary>
private double evaporatorKA;

/// <summary>凝縮器の伝熱性能[kW/K]</summary>
private double condensorKA;

/// <summary>再生器の伝熱性能[kW/K]</summary>
private double desorborKA;

/// <summary>溶液熱交換器の伝熱性能[kW/K]</summary>
private double solutionHexKA;

/// <summary>冷水最小流量比[kg/s]</summary>
private double chilledWaterMinimumFLowRatio = 0.4;

/// <summary>冷却水最小流量比[kg/s]</summary>
private double coolingWaterMinimumFLowRatio = 0.4;

/// <summary>温水最小流量比[kg/s]</summary>
private double hotWaterMinimumFLowRatio = 0.4;

/// <summary>定格冷媒流量[m3/s]</summary>
private double nominalSolutionFlowRate;

/// <summary>熱損失率[-]</summary>
private double heatLossRate = 0.0;

/// <summary>冷水出口温度[C]を取得する</summary>
public double ChilledWaterOutletTemperature { get; private set; }

```

```
 37 /// <summary>冷水出口温度設定値[C]を設定・取得する</summary>
 38 public double ChilledWaterOutletSetPointTemperature { get; set; }
 39
 40 /// <summary>冷水入口温度[C]を取得する</summary>
 41 public double ChilledWaterInletTemperature { get; private set; }
 42
 43 /// <summary>冷却水出口温度[C]を取得する</summary>
 44 public double CoolingWaterOutletTemperature { get; private set; }
 45
 46 /// <summary>冷却水入口温度[C]を取得する</summary>
 47 public double CoolingWaterInletTemperature { get; private set; }
 48
 49 /// <summary>温水出口温度[C]を取得する</summary>
 50 public double HotWaterOutletTemperature { get; private set; }
 51
 52 /// <summary>温水入口温度[C]を取得する</summary>
 53 public double HotWaterInletTemperature { get; private set; }
 54
 55 /// <summary>冷水流量[kg/s]を取得する</summary>
 56 public double ChilledWaterFlowRate { get; private set; }
 57
 58 /// <summary>冷却水流量[kg/s]を取得する</summary>
 59 public double CoolingWaterFlowRate { get; private set; }
 60
 61 /// <summary>温水流量[kg/s]を取得する</summary>
 62 public double HotWaterFlowRate { get; private set; }
 63
 64 /// <summary>冷水定格流量[kg/s]を取得する</summary>
 65 public double NominalChilledWaterFlowRate { get; private set; }
 66
 67 /// <summary>冷却水定格流量[kg/s]を取得する</summary>
 68 public double NominalCoolingWaterFlowRate { get; private set; }
 69
 70 /// <summary>温水定格流量[kg/s]を取得する</summary>
 71 public double NominalHotWaterFlowRate { get; private set; }
 72
 73 /// <summary>定格冷凍能力[kW]を取得する</summary>
 74 public double NominalCapacity { get; private set; }
 75
 76 /// <summary>冷水最小流量比[kg/s]を設定・取得する</summary>
 77 public double ChilledWaterMinimumFLowRatio
 78 {
 79   get { return chilledWaterMinimumFLowRatio; }
 80   private set { chilledWaterMinimumFLowRatio = Math.Min(1, Math.Max(0.4, value)); }
 81 }
 82
 83 /// <summary>冷却水最小流量比[kg/s]を設定・取得する</summary>
 84 public double CoolingWaterMinimumFLowRatio
 85 {
 86   get { return coolingWaterMinimumFLowRatio; }
 87   private set { coolingWaterMinimumFLowRatio = Math.Min(1, Math.Max(0.4, value)); }
 88 }
 89
 90 /// <summary>温水最小流量比[kg/s]を設定・取得する</summary>
 91 public double HotWaterMinimumFLowRatio
 92 {
 93   get { return hotWaterMinimumFLowRatio; }
 94   private set { hotWaterMinimumFLowRatio = Math.Min(1, Math.Max(0.4, value)); }
 95 }
 96
 97 /// <summary>冷凍能力[kW]を取得する</summary>
 98 public double CoolingLoad { get; private set; }
 99
100 /// <summary>COP[-]を取得する</summary>
101 public double COP
102 {
103   get
104   {
105     if (HotWaterFlowRate == 0) return 0;
106     else return CoolingLoad / (HotWaterFlowRate * WATER_SPECIFIC_HEAT
107     * (HotWaterInletTemperature - HotWaterOutletTemperature));
108   }
109 }
```

コンストラクタをプログラム 15.12 に示す 16~22 行でプログラム 15.7 を呼び出し、蒸発器・凝縮器・再生器・溶液熱交換器の伝熱係数、定格の水溶液流量、再生器投入熱量を計算する。24~26 行は熱損失の計算であり、再生器投入熱量と温水熱量の差にもとづいて、図 15.1 および式 15.1 の Q_{L1} を計

算する。28~34 行で各種のパラメータをプロパティに保存し、37 行で機器を停止させる。

プログラム 15.17　コンストラクタ

Popolo.HVAC.HeatSource.HotWaterAbsorptionChiller class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="chilledWaterInletTemperature">冷水入口温度[C]</param>
 3 /// <param name="chilledWaterOutletTemperature">冷水出口温度[C]</param>
 4 /// <param name="chilledWaterFlowRate">冷水質量流量[kg/s]</param>
 5 /// <param name="coolingWaterInletTemperature">冷却水入口温度[C]</param>
 6 /// <param name="coolingWaterOutletTemperature">冷却水出口温度[C]</param>
 7 /// <param name="coolingWaterFlowRate">冷却水質量流量[kg/s]</param>
 8 /// <param name="hotWaterInletTemperature">温水入口温度[C]</param>
 9 /// <param name="hotWaterOutletTemperature">温水出口温度[C]</param>
10 /// <param name="hotWaterFlowRate">温水質量流量[kg/s]</param>
11 public HotWaterAbsorptionChiller
12   (double chilledWaterInletTemperature, double chilledWaterOutletTemperature, double chilledWaterFlowRate,
13   double coolingWaterInletTemperature, double coolingWaterOutletTemperature, double coolingWaterFlowRate,
14   double hotWaterInletTemperature, double hotWaterOutletTemperature, double hotWaterFlowRate)
15 {
16   //伝熱係数を計算
17   double dsvHL;
18   AbsorptionRefrigerationCycle.GetHeatTransferCoefficients
19     (chilledWaterInletTemperature, chilledWaterOutletTemperature, chilledWaterFlowRate,
20     coolingWaterInletTemperature, coolingWaterOutletTemperature, coolingWaterFlowRate,
21     hotWaterInletTemperature, hotWaterFlowRate, DESORB_TEMPERATURE_APPROACH, out evaporatorKA,
22     out condensorKA, out desorborKA, out solutionHexKA, out nominalSolutionFlowRate, out dsvHL);
23
24   //熱損失率[-]を計算
25   double qHW = (hotWaterInletTemperature - hotWaterOutletTemperature) * hotWaterFlowRate * WATER_SPECIFIC_HEAT;
26   heatLossRate = (qHW - dsvHL) / qHW;
27
28   this.ChilledWaterOutletSetPointTemperature = chilledWaterOutletTemperature;
29   this.CoolingWaterInletTemperature = coolingWaterInletTemperature;
30   this.ChilledWaterInletTemperature = chilledWaterInletTemperature;
31   this.HotWaterInletTemperature = hotWaterInletTemperature;
32   this.NominalCoolingWaterFlowRate = coolingWaterFlowRate;
33   this.NominalChilledWaterFlowRate = chilledWaterFlowRate;
34   this.NominalHotWaterFlowRate = hotWaterFlowRate;
35
36   //機器を停止
37   ShutOff();
38 }
39
40 /// <summary>運転を停止させる</summary>
41 public void ShutOff()
42 {
43   ChilledWaterOutletTemperature = ChilledWaterInletTemperature;
44   CoolingWaterOutletTemperature = CoolingWaterInletTemperature;
45   HotWaterOutletTemperature = HotWaterInletTemperature;
46   ChilledWaterFlowRate = CoolingWaterFlowRate = HotWaterFlowRate = 0;
47   CoolingLoad = 0;
48 }
```

状態更新処理をプログラム 15.18 に示す。12~23 行で入力情報をプロパティに設定する。ただし冷水・冷却水・温水の流量に関しては下限値（標準では定格の 40 %とした）以上となるように値を調整する。異常に小さな値では成立する冷凍サイクルが存在せず、収束計算が破綻するからである。なお、現実の温水吸収冷凍機の運転可能範囲としては、冷水が 100~120 %、冷却水が 100~110 %、温水が 120 %以下、という例がある。

26 行で条件分岐を行い、入口水温が既に出口水温設定値よりも低い場合には 54 行で運転を停止させる。運転を行う場合には、まず 30 行で成り行きの出口水温を求め、出口冷水温度が設定値よりも低く、過負荷でないことを確認した後、39 行で設定水温になるように状態を更新する。45~51 行で各種の出口状態をプロパティに保存する。

プログラム 15.18　状態更新処理

Popolo.HVAC.HeatSource.HotWaterAbsorptionChiller class

```
 1 /// <summary>状態を更新する</summary>
 2 /// <param name="chilledWaterInletTemperature">冷水入口温度[C]</param>
 3 /// <param name="chilledWaterFlowRate">冷水質量流量[kg/s]</param>
```

```
/// <param name="coolingWaterInletTemperature">冷却水入口温度[C]</param>
/// <param name="coolingWaterFlowRate">冷却水質量流量[kg/s]</param>
/// <param name="hotWaterInletTemperature">温水入口温度[C]</param>
/// <param name="hotWaterFlowRate">温水質量流量[kg/s]</param>
public void Update
  (double chilledWaterInletTemperature, double chilledWaterFlowRate, double coolingWaterInletTemperature,
  double coolingWaterFlowRate, double hotWaterInletTemperature, double hotWaterFlowRate)
{
  //状態値を保存
  this.ChilledWaterInletTemperature = chilledWaterInletTemperature;
  this.CoolingWaterInletTemperature = coolingWaterInletTemperature;
  this.HotWaterInletTemperature = hotWaterInletTemperature;
  double rch = chilledWaterFlowRate / NominalChilledWaterFlowRate;
  this.ChilledWaterFlowRate =
    Math.Max(ChilledWaterMinimumFLowRatio, Math.Min(1, rch)) * NominalChilledWaterFlowRate;
  double rcd = coolingWaterFlowRate / NominalCoolingWaterFlowRate;
  this.CoolingWaterFlowRate =
    Math.Max(CoolingWaterMinimumFLowRatio, Math.Min(1, rcd)) * NominalCoolingWaterFlowRate;
  double rht = hotWaterFlowRate / NominalHotWaterFlowRate;
  this.HotWaterFlowRate = Math.Max(HotWaterMinimumFLowRatio, Math.Min(1, rht)) * NominalHotWaterFlowRate;

  //冷却運転
  if (ChilledWaterOutletSetPointTemperature < chilledWaterInletTemperature)
  {
    double cho, cdo, ho;
    //成り行きの出口状態を計算
    AbsorptionRefrigerationCycle.GetOutletTemperatures
      (ChilledWaterInletTemperature, ChilledWaterFlowRate, CoolingWaterInletTemperature,
      CoolingWaterFlowRate, HotWaterInletTemperature, HotWaterFlowRate, evaporatorKA, condensorKA,
      desorborKA, solutionHexKA, nominalSolutionFlowRate, out cho, out cdo, out ho);

    //処理可能の場合
    if (cho < ChilledWaterOutletSetPointTemperature)
    {
      cho = ChilledWaterOutletSetPointTemperature;
      AbsorptionRefrigerationCycle.GetOutletTemperatures
        (ChilledWaterInletTemperature, ChilledWaterFlowRate, CoolingWaterInletTemperature,
        CoolingWaterFlowRate, HotWaterInletTemperature, HotWaterFlowRate, evaporatorKA, condensorKA,
        desorborKA, solutionHexKA, nominalSolutionFlowRate, cho, out cdo, out ho);
    }

    //出口状態設定
    ChilledWaterOutletTemperature = cho;
    CoolingWaterOutletTemperature = cdo;
    HotWaterOutletTemperature = (ho - heatLossRate * HotWaterInletTemperature) / (1 - heatLossRate);
    //処理熱量計算
    CoolingLoad = (ChilledWaterInletTemperature - ChilledWaterOutletTemperature)
      * WATER_SPECIFIC_HEAT * chilledWaterFlowRate;
  }
  //運転停止
  else ShutOff();
}
```

【例題 15.5】

例題 15.2 の温水吸収冷凍機について、負荷率と COP の関係を冷却水温度別に、温水温度と冷凍能力の関係を温水流量別に求めて特性線図を作成せよ。

【解】

プログラム 15.19 に計算処理を示す。3,4 行で温水吸収冷凍機クラスのインスタンスを作成し、この入力値を変えて特性を確認する。9~23 行は負荷率と COP の関係に関する処理である。冷水出入口温度差は一定とし、14 行で負荷率に比例的に流量を変化させる。26~39 行は温水温度と冷凍能力の関係に関する処理である。計算結果をグラフ化すると図 15.10 が得られる。負荷率が低下するにつれて効率も低下する関係にある。また、冷却水温度が低いほど効率が高くなるが、これは凝縮温度を低くすることが可能となり、溶液循環比を高くすることができるからである。冷凍能力は温水温度が高いほど、また、温水流量が高いほど、大きな値となる。

プログラム 15.19　温水吸収冷凍機の機器特性の計算

```
1 private static void HotWaterAbsorptionChillerTest()
2 {
3   HotWaterAbsorptionChiller ar = new HotWaterAbsorptionChiller
4     (12.5, 7, 274.9 / 60, 31, 35, 918d / 60, 88, 83, 432d / 60);
5
6   using (StreamWriter sWriter = new StreamWriter
7     ("HotWaterAbsorptionChillerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
8   {
9     sWriter.WriteLine("負荷率・冷却水温度・COP の関係");
```

```
10      sWriter.WriteLine("負荷率, 31C, 28C, 25C, 22C");
11      for (int i = 0; i < 13; i++)
12      {
13        double pl = (1 - 0.05 * i);
14        double fl = (274.9 / 60) * pl;
15
16        sWriter.Write((pl * 100).ToString("F0"));
17        for (int j = 0; j < 4; j++)
18        {
19          ar.Update(12.5, fl, 31 - j * 3, 918d / 60, 88, 432d / 60);
20          sWriter.Write("," + ar.COP.ToString("F3"));
21        }
22        sWriter.WriteLine();
23      }
24
25      sWriter.WriteLine();
26      sWriter.WriteLine("温水温度・温水流量・処理熱量の関係");
27      sWriter.WriteLine("温水温度, 100%, 75%, 50%");
28      ar.ChilledWaterOutletSetPointTemperature = 0;
29      for (int i = 0; i < 26; i++)
30      {
31        double thw = 70 + i;
32        sWriter.Write(thw.ToString("F0"));
33        for (int j = 0; j < 3; j++)
34        {
35          ar.Update(12.5, 274.9 / 60, 31, 918d / 60, thw, 432d / 60 * (1 - 0.25 * j));
36          sWriter.Write("," + ar.CoolingLoad.ToString("F1"));
37        }
38        sWriter.WriteLine();
39      }
40    }
41  }
```

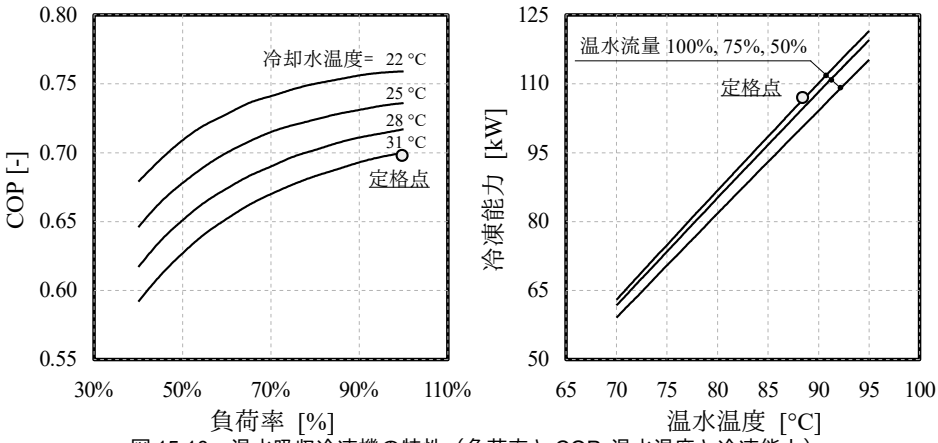


図 15.10　温水吸収冷凍機の特性（負荷率と COP, 温水温度と冷凍能力）

15.3.5　「二重効用吸収冷凍機クラス」の作成

15.3.3 節のプログラムを用いて二重効用吸収冷凍機のクラスを作成する。定数宣言をプログラム 15.20 に示す。再生器投入熱量と燃料消費量との関係を計算するために第 13 章で作成したボイラモデルを利用する。このため、燃焼空気温度・燃焼空気比・排ガス温度を定数として宣言する。なお、排ガス温度に関しては吸収式冷凍機のカタログに記載されていることもある。

プログラム 15.20　定数宣言

Popolo.HVAC.HeatSource.DirectFiredAbsorptionChiller class

```
 1 /// <summary>水の定圧比熱[kJ/(kgK)]</summary>
 2 private const double WATER_SPECIFIC_HEAT = 4.186;
 3
 4 /// <summary>冷温水変流量下限[-]</summary>
 5 private const double MINIMUM_WATERFLOW_RATIO = 0.5;
 6
 7 /// <summary>水溶液変流量下限[-]</summary>
 8 private const double MINIMUM_SOLUTIONFLOW_RATIO = 0.5;
 9
10 /// <summary>負荷率下限値[-]</summary>
```

```
11 private const double MINIMUM_PARTIALLOAD = 0.2;
12
13 /// <summary>周囲の温度・燃焼空気温度[C]</summary>
14 private const double AMB_TEMP = 25;
15
16 /// <summary>燃焼空気比[-]</summary>
17 private const double AIR_RATIO = 1.15;
18
19 /// <summary>排ガス温度[C]</summary>
20 private const double SMK_TEMP = 100;
```

インスタンス変数とプロパティの定義をプログラム 15.21 に示す。温水吸収と同様の情報が多いが、直焚吸収冷温水機の場合には温水製造も行うため、温水関係のプロパティを追加している。ただし実態としては後述の通り、第 13 章で作成した温水ボイラの計算を行う。

プログラム **15.21**　インスタンス変数とプロパティの定義

Popolo.HVAC.HeatSource.DirectFiredAbsorptionChiller class

```
 1 /// <summary>蒸発器伝熱係数[kW/K]</summary>
 2 private double evaporatorKA;
 3
 4 /// <summary>凝縮器伝熱係数[kW/K]</summary>
 5 private double condensorKA;
 6
 7 /// <summary>低温再生器伝熱係数[kW/K]</summary>
 8 private double lowDesorborKA;
 9
10 /// <summary>低温溶液熱交換器伝熱係数[kW/K]</summary>
11 private double thinSolutionHexKA;
12
13 /// <summary>定格水溶液流量[kg/s]</summary>
14 private double solutionFlowRate;
15
16 /// <summary>定格高温再生器投入熱量[kW]</summary>
17 private double desorbHeat;
18
19 /// <summary>缶体からの熱損失係数[kW/K]</summary>
20 private double heatLossCoefficient;
21
22 /// <summary>温水ボイラ</summary>
23 private HotWaterBoiler hBoiler;
24
25 /// <summary>冷却運転モードか否かを設定・取得する</summary>
26 public bool IsCoolingMode { get; set; }
27
28 /// <summary>燃料の種別を取得する</summary>
29 public Boiler.Fuel Fuel { get; private set; }
30
31 /// <summary>出口水温[C]を取得する</summary>
32 public double OutletWaterTemperature { get; private set; }
33
34 /// <summary>出口水温設定値[C]を設定・取得する</summary>
35 public double OutletWaterSetPointTemperature { get; set; }
36
37 /// <summary>入口水温[C]を取得する</summary>
38 public double InletWaterTemperature { get; private set; }
39
40 /// <summary>冷却水出口温度[C]を取得する</summary>
41 public double CoolingWaterOutletTemperature { get; private set; }
42
43 /// <summary>冷却水入口温度[C]を取得する</summary>
44 public double CoolingWaterInletTemperature { get; private set; }
45
46 /// <summary>冷水/温水の水量[kg/s]を取得する</summary>
47 public double WaterFlowRate { get; private set; }
48
49 /// <summary>冷却水流量[kg/s]を取得する</summary>
50 public double CoolingWaterFlowRate { get; private set; }
51
52 /// <summary>冷水上限流量[kg/s]を取得する</summary>
53 public double MaxChilledWaterFlowRate { get; private set; }
54
55 /// <summary>温水上限流量[kg/s]を取得する</summary>
56 public double MaxHotWaterFlowRate { get; private set; }
57
58 /// <summary>冷水下限流量[kg/s]を取得する</summary>
59 public double MinChilledWaterFlowRate { get; private set; }
60
```

```
 61 /// <summary>温水下限流量[kg/s]を取得する</summary>
 62 public double MinHotWaterFlowRate { get; private set; }
 63
 64 /// <summary>冷却水上限流量[kg/s]を取得する</summary>
 65 public double MaxCoolingWaterFlowRate { get; private set; }
 66
 67 /// <summary>冷却水下限流量[kg/s]を取得する</summary>
 68 public double MinCoolingWaterFlowRate { get; private set; }
 69
 70 /// <summary>定格冷却能力[kW]を取得する</summary>
 71 public double NominalCoolingCapacity { get; private set; }
 72
 73 /// <summary>定格加熱能力[kW]を取得する</summary>
 74 public double NominalHeatingCapacity { get; private set; }
 75
 76 /// <summary>定格冷却燃料消費量[Nm3/s, kg/s]を取得する</summary>
 77 public double NominalCoolingFuelConsumption { get; private set; }
 78
 79 /// <summary>定格加熱燃料消費量[Nm3/s, kg/s]を取得する</summary>
 80 public double NominalHeatingFuelConsumption { get; private set; }
 81
 82 /// <summary>容量制御下限値[-]を取得する</summary>
 83 public double MinimumPartialLoadRatio { get; private set; }
 84
 85 /// <summary>定格消費電力[kW]を取得する</summary>
 86 public double NominalElectricConsumption { get; private set; }
 87
 88 /// <summary>消費電力[kW]を取得する</summary>
 89 public double ElectricConsumption { get; private set; }
 90
 91 /// <summary>燃料消費量[Nm3/s, kg/s]を取得する</summary>
 92 public double FuelConsumption { get; private set; }
 93
 94 /// <summary>加熱負荷[kW]を取得する</summary>
 95 public double HeatingLoad { get; private set; }
 96
 97 /// <summary>冷却負荷[kW]を取得する</summary>
 98 public double CoolingLoad { get; private set; }
 99
100 /// <summary>COP[-]を取得する</summary>
101 public double COP
102 {
103   get
104   {
105     if (FuelConsumption == 0) return 0;
106     else if(IsCoolingMode)
107       return 0.001 * CoolingLoad / (FuelConsumption * Boiler.GetHeatingValue(Fuel, true));
108     else return 0.001 * HeatingLoad / (FuelConsumption * Boiler.GetHeatingValue(Fuel, true));
109   }
110 }
111
112 /// <summary>溶液ポンプのインバータ制御があるか</summary>
113 public bool HasSolutionInverterPump { get; set; }
114
115 /// <summary>高濃度溶液の質量分率[-]を取得する</summary>
116 public double ThickSolutionMassFraction { get; private set; }
117
118 /// <summary>低濃度溶液の質量分率[-]を取得する</summary>
119 public double ThinSolutionMassFraction { get; private set; }
120
121 /// <summary>蒸発温度[C]を取得する</summary>
122 public double EvaporatingTemperature { get; private set; }
123
124 /// <summary>凝縮温度[C]を取得する</summary>
125 public double CondensingTemperature { get; private set; }
126
127 /// <summary>再生温度[C]を取得する</summary>
128 public double DesorbTemperature { get; private set; }
129
130 /// <summary>過負荷か否か</summary>
131 public bool IsOverLoad { get; private set; }
```

直焚吸収冷温水機クラスのコンストラクタをプログラム 15.22 に示す。22~42 行で定格性能などの情報をプロパティに保存する。44~49 行でプログラム 15.12 を用いて構成要素の伝熱係数を計算してインスタンス変数に保存する。算出された再生器投入熱量（desorbHeat）と消費燃料の関係に関しては、51~54 行に示すようにボイラモデルを用いて表現する。つまり、式 15.1 の Q_{L1} をボイラモデルで

表現するということである。これにより部分負荷運転時の煙突排ガス熱損失などの特性が表現できるようになる。また、温水製造時は単純なボイラとして動作すると仮定し、57, 58 行で示すようにボイラクラスのインスタンスを作成する。

プログラム 15.22　コンストラクタ

Popolo.HVAC.HeatSource.DirectFiredAbsorptionChiller class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="nominalCoolingFuelConsumption">定格冷却ガス消費[Nm/3, kg/s]</param>
 3 /// <param name="nominalHetingFuelConsumption">定格加熱ガス消費[Nm/3, kg/s]</param>
 4 /// <param name="chilledWaterInletTemperature">冷水入口温度[C]</param>
 5 /// <param name="chilledWaterOutletTemperature">冷水出口温度[C]</param>
 6 /// <param name="coolingWaterInletTemperature">冷却水入口温度[C]</param>
 7 /// <param name="coolingWaterOutletTemperature">冷却水出口温度[C]</param>
 8 /// <param name="hotWaterInletTemperature">温水入口温度[C]</param>
 9 /// <param name="hotWaterOutletTemperature">温水出口温度[C]</param>
10 /// <param name="chilledWaterFlowRate">冷水質量流量[kg/s]</param>
11 /// <param name="coolingWaterFlowRate">冷却水質量流量[kg/s]</param>
12 /// <param name="hotWaterFlowRate">温水質量流量[kg/s]</param>
13 /// <param name="electricConsumption">消費電力[kW]</param>
14 /// <param name="fuel">燃料の種別</param>
15 public DirectFiredAbsorptionChiller
16   (double nominalCoolingFuelConsumption, double nominalHetingFuelConsumption,
17   double chilledWaterInletTemperature, double chilledWaterOutletTemperature,
18   double coolingWaterInletTemperature, double coolingWaterOutletTemperature,
19   double hotWaterInletTemperature, double hotWaterOutletTemperature, double chilledWaterFlowRate,
20   double coolingWaterFlowRate, double hotWaterFlowRate, double electricConsumption, Boiler.Fuel fuel)
21 {
22   this.IsCoolingMode = true;
23   this.OutletWaterSetPointTemperature = chilledWaterOutletTemperature;
24   this.CoolingWaterInletTemperature = coolingWaterInletTemperature;
25   this.InletWaterTemperature = chilledWaterInletTemperature;
26   this.CoolingWaterOutletTemperature = coolingWaterOutletTemperature;
27   this.CoolingWaterFlowRate = coolingWaterFlowRate;
28   this.WaterFlowRate = chilledWaterFlowRate;
29   this.MinCoolingWaterFlowRate = coolingWaterFlowRate * MINIMUM_WATERFLOW_RATIO;
30   this.MaxCoolingWaterFlowRate = coolingWaterFlowRate;
31   this.MaxChilledWaterFlowRate = chilledWaterFlowRate;
32   this.MaxHotWaterFlowRate = hotWaterFlowRate;
33   this.MinChilledWaterFlowRate = chilledWaterFlowRate * MINIMUM_WATERFLOW_RATIO;
34   this.MinHotWaterFlowRate = hotWaterFlowRate * MINIMUM_WATERFLOW_RATIO;
35   this.NominalCoolingCapacity = chilledWaterFlowRate * WATER_SPECIFIC_HEAT
36     * (chilledWaterInletTemperature - chilledWaterOutletTemperature);
37   this.NominalHeatingCapacity = hotWaterFlowRate * WATER_SPECIFIC_HEAT
38     * (hotWaterOutletTemperature - hotWaterInletTemperature);
39   this.NominalCoolingFuelConsumption = nominalCoolingFuelConsumption;
40   this.NominalHeatingFuelConsumption = nominalHetingFuelConsumption;
41   this.Fuel = fuel;
42   this.NominalElectricConsumption = electricConsumption;
43
44   //吸収冷凍サイクルの各種性能を計算
45   AbsorptionRefrigerationCycle.GetHeatTransferCoefficients(chilledWaterInletTemperature,
46     chilledWaterOutletTemperature, chilledWaterFlowRate, coolingWaterInletTemperature,
47     coolingWaterOutletTemperature, coolingWaterFlowRate, out evaporatorKA, out condensorKA,
48     out lowDesorborKA, out thinSolutionHexKA, out solutionFlowRate, out desorbHeat);
49   desorbHeat *= 1.001;  //定格性能を担保するための処理
50
51   //缶体熱損失係数の計算
52   heatLossCoefficient = Boiler.GetHeatLossCoefficient
53     (desorbHeat, AbsorptionRefrigerationCycle.NOM_DSB_LIQ_TEMP, AMB_TEMP, Fuel,
54     SMK_TEMP, AIR_RATIO, NominalCoolingFuelConsumption, AMB_TEMP);
55
56   //温水ボイラ初期化
57   hBoiler = new HotWaterBoiler(hotWaterInletTemperature, hotWaterOutletTemperature, hotWaterFlowRate,
58     NominalHeatingFuelConsumption, 0, AMB_TEMP, AIR_RATIO, Fuel, SMK_TEMP);
59
60   //運転停止
61   ShutOff();
62 }
63
64 /// <summary>運転を停止させる</summary>
65 public void ShutOff()
66 {
67   OutletWaterTemperature = InletWaterTemperature;
68   CoolingWaterOutletTemperature = CoolingWaterInletTemperature;
69   FuelConsumption = CoolingLoad = HeatingLoad = CoolingWaterFlowRate = WaterFlowRate = 0;
70   ElectricConsumption = 0;
71 }
```

状態更新処理をプログラム 15.23 に示す。10~12 行で入力条件をプロパティに保存する。温水吸収冷凍機の場合と同様に、吸収冷凍サイクルが成立しない異常な流量にならないように流量下限値を設ける。14~76 行は冷房時の計算、77~93 行は暖房時の計算、そのいずれでもない場合には 95 行で運転を停止させる。

負荷と冷水出入口温度差が極めて小さい場合には冷凍サイクルが成立しない場合があるため、17~22 行で冷水入口温度を補正する。まず、27 行で成り行きの冷水出口温度計算を行う。得られた出口温度が設定値よりも高く、過負荷の場合には 36, 37 行に示すように成り行きでの出口温度をプロパティに設定する。この場合の燃料消費量は定格燃料消費量となる。熱負荷を処理可能な場合には 42~62 行で消費燃料の計算を行うが、溶液ポンプがインバータ搭載か否かによって計算方法を切り替える。固定速の場合には 57 行でプログラム 15.15 を呼び出して燃料消費量を求める。可変速の場合には、熱源効率が最大となる溶液循環量を収束計算で求める。誤差関数は 44~51 行に示すように、溶液循環量を入力とし、再生器投入熱量を出力する関数である。第 2 章で解説した黄金分割法を用い、53 行で再生器投入熱量が最小値をとる溶液循環量を求める。

暖房運転時は単純な温水ボイラとして計算を行う。86 行に示すように温水ボイラのインスタンスで出口条件などを計算し、プロパティに設定する。

プログラム 15.23　状態更新処理

Popolo.HVAC.HeatSource.DirectFiredAbsorptionChiller class

```
 1 /// <summary>状態を更新する</summary>
 2 /// <param name="coolingWaterInletTemperature">冷却水入口温度[C]</param>
 3 /// <param name="inletWaterTemperature">冷温水入口温度[C]</param>
 4 /// <param name="coolingWaterFlowRate">冷却水質量流量[kg/s]</param>
 5 /// <param name="waterFlowRate">冷温水質量流量[kg/s]</param>
 6 public void Update(double coolingWaterInletTemperature, double inletWaterTemperature,
 7   double coolingWaterFlowRate, double waterFlowRate)
 8 {
 9   //状態値を保存
10   this.InletWaterTemperature = inletWaterTemperature;
11   this.CoolingWaterInletTemperature = coolingWaterInletTemperature;
12   this.CoolingWaterFlowRate = Math.Max(coolingWaterFlowRate, MinCoolingWaterFlowRate);
13
14   //冷却運転
15   if (IsCoolingMode && (OutletWaterSetPointTemperature < InletWaterTemperature))
16   {
17     //極少負荷対応のための入口水温・水量補正
18     this.WaterFlowRate = Math.Max(waterFlowRate, MinChilledWaterFlowRate);
19     double pl = (InletWaterTemperature - OutletWaterSetPointTemperature)
20       * WATER_SPECIFIC_HEAT * WaterFlowRate / NominalCoolingCapacity;
21     double ti = (InletWaterTemperature - OutletWaterSetPointTemperature)
22       / Math.Min(1, pl / MINIMUM_PARTIALLOAD) + OutletWaterSetPointTemperature;
23
24     double dsbH = 0;
25     double tcdo, tdsb, tevp, tcnd, wtn, wtk;
26     //定格の高温再生器投入熱量で出口状態を計算
27     double cht = AbsorptionRefrigerationCycle.GetChilledWaterOutletTemperature
28       (ti, WaterFlowRate, CoolingWaterInletTemperature, CoolingWaterFlowRate, evaporatorKA,
29       condensorKA, lowDesorborKA, thinSolutionHexKA, solutionFlowRate, desorbHeat,
30       out tcdo, out tdsb, out tevp, out tcnd, out wtn, out wtk);
31
32     IsOverLoad = OutletWaterSetPointTemperature <= cht;
33     //過負荷の場合
34     if (IsOverLoad)
35     {
36       OutletWaterTemperature = cht;
37       dsbH = desorbHeat;
38     }
39     //処理可能の場合
40     else
41     {
42       if (HasSolutionInverterPump)
43       {
44         Minimization.MinimizeFunction mFnc = delegate (double sFlow)
```

```
45          {
46            dsbH = AbsorptionRefrigerationCycle.GetDesorbHeat
47            (ti, WaterFlowRate, CoolingWaterInletTemperature, CoolingWaterFlowRate, evaporatorKA, condensorKA,
48            lowDesorborKA, thinSolutionHexKA, sFlow, OutletWaterSetPointTemperature,
49            out tcdo, out tdsb, out tevp, out tcnd, out wtn, out wtk);
50            return dsbH;
51          };
52          double sf = solutionFlowRate * MINIMUM_SOLUTIONFLOW_RATIO;
53          Minimization.GoldenSection(ref sf, solutionFlowRate, mFnc);
54        }
55        else
56        {
57          dsbH = AbsorptionRefrigerationCycle.GetDesorbHeat
58            (ti, WaterFlowRate, CoolingWaterInletTemperature, CoolingWaterFlowRate, evaporatorKA, condensorKA,
59            lowDesorborKA, thinSolutionHexKA, solutionFlowRate, OutletWaterSetPointTemperature,
60            out tcdo, out tdsb, out tevp, out tcnd, out wtn, out wtk);
61        }
62        OutletWaterTemperature = OutletWaterSetPointTemperature;
63      }
64      CoolingWaterOutletTemperature = tcdo;
65      CoolingLoad = (InletWaterTemperature - OutletWaterTemperature) * WATER_SPECIFIC_HEAT * waterFlowRate;
66      HeatingLoad = 0;
67      FuelConsumption = Boiler.GetFuelConsumption
68        (dsbH, tdsb, AMB_TEMP, Fuel, SMK_TEMP, AIR_RATIO, heatLossCoefficient,
69        AMB_TEMP, AbsorptionRefrigerationCycle.NOM_DSB_LIQ_TEMP);
70      ElectricConsumption = CoolingLoad / NominalCoolingCapacity * NominalElectricConsumption;
71      DesorbTemperature = tdsb;
72      EvaporatingTemperature = tevp;
73      CondensingTemperature = tcnd;
74      ThickSolutionMassFraction = wtk;
75      ThinSolutionMassFraction = wtn;
76    }
77    //加熱運転
78    else if (!IsCoolingMode && (InletWaterTemperature < OutletWaterSetPointTemperature))
79    {
80      //水量設定
81      CoolingWaterFlowRate = 0;
82      this.WaterFlowRate = Math.Max(waterFlowRate, MinHotWaterFlowRate);
83
84      //温水ボイラを更新
85      hBoiler.OutletWaterSetPointTemperature = this.OutletWaterSetPointTemperature;
86      hBoiler.Update(InletWaterTemperature, WaterFlowRate);
87      IsOverLoad = hBoiler.IsOverLoad;
88      HeatingLoad = hBoiler.HeatLoad;
89      CoolingLoad = 0;
90      OutletWaterTemperature = hBoiler.OutletWaterTemperature;
91      FuelConsumption = hBoiler.FuelConsumption;
92      ElectricConsumption = HeatingLoad / NominalHeatingCapacity * NominalElectricConsumption;
93    }
94    //運転停止
95    else ShutOff();
96  }
```

【例題 15.6】

例題 15.1 の二重効用吸収冷温水機について負荷率と COP の関係を計算して特性線図を作成せよ。あわせて溶液ポンプ INV の有無による部分負荷性能の違いを示せ。

【解】

プログラム 15.24 に計算処理を示す。3 行で直焚吸収冷温水機クラスのインスタンスを作成する。14~16 行は部分負荷条件の計算であり、冷却水温度を 27~32 ℃ の範囲で負荷率に比例的に変化させる。これは JIS B8622 条件と呼ばれるもので、カタログ記載の負荷特性はこのような条件を前提としている。計算結果をグラフにすると図 15.11 が得られる。溶液循環ポンプを INV 制御すると、負荷の低下に伴って緩やかに COP が向上していくことがわかる。負荷率 30 %程度でピークを示し、定格の COP に比較して 1 割程度、効率が向上する。

プログラム 15.24　二重効用吸収冷温水機の負荷特性の計算

```
 1 private static void DirectFiredAbsorptionChillerTest1()
 2 {
 3   DirectFiredAbsorptionChiller ar = new DirectFiredAbsorptionChiller(103d / 3600, 103d / 3600,
 4     15, 7, 32, 37, 54.7, 60, 189 / 3.6, 500 / 3.6, 189 / 3.6, 0, Boiler.Fuel.Gas13A);
 5
 6   using (StreamWriter sWriter = new StreamWriter
 7     ("DirectFiredAbsorptionChillerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
 8   {
 9     sWriter.WriteLine("負荷率,COP(INV),COP(定速)");
10
```

```
    for (int i = 0; i < 19; i++)
    {
      //JIS B8622 条件
      double pl = (1 - 0.05 * i);
      double tcd = 27 + 5 * pl;
      double tch = 7 + 8 * pl;

      sWriter.Write((pl * 100).ToString("F0"));

      ar.HasSolutionInverterPump = true;
      ar.Update(tcd, tch, 500 / 3.6, 189 / 3.6);
      sWriter.Write("," + ar.COP.ToString("F3"));

      ar.HasSolutionInverterPump = false;
      ar.Update(tcd, tch, 500 / 3.6, 189 / 3.6);
      sWriter.Write("," + ar.COP.ToString("F3"));

      sWriter.WriteLine();
    }
  }
}
```

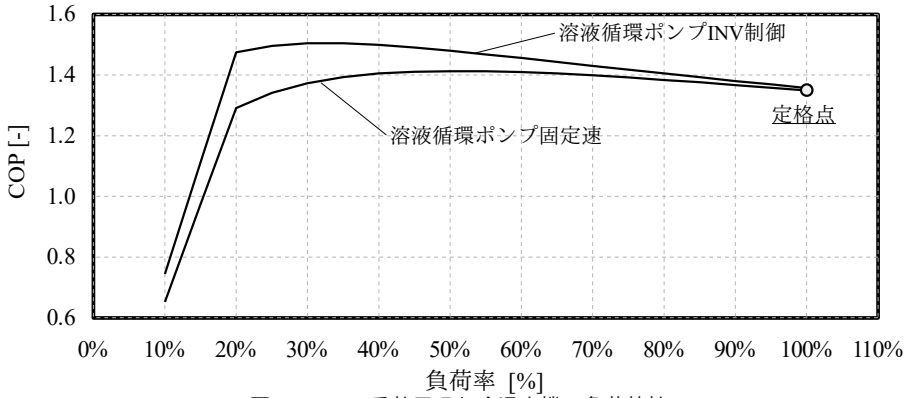


図 15.11　二重効用吸収冷温水機の負荷特性

【例題 15.7】

例題 15.1 の二重効用吸収冷温水機について冷水出入口温度差を半分にした場合、冷却水入口温度を 25°C にした場合、冷水流量を 50 %にした場合、冷水出口温度を 10°C にした場合、それぞれの運転状態を計算し、機器効率について考察せよ。ただし溶液ポンプはインバータ制御有とする。

【解】

プログラム 15.25 に計算処理を示す。表 15.3 に計算結果を示す。冷水出入口温度差が小さくなると、蒸発器への負荷が小さくなるため蒸発温度が上昇する。また熱源全体で処理すべき熱量も減少するため、再生温度を小さくすることができる。この結果、COP は向上する。冷水流量を 50 %とした場合も負荷率が 50 %となり、凝縮温度に変わりは無いが、蒸発温度をより大きく上げることができ、COP はさらに高い値になる。冷却水温度を低くした場合、蒸発温度への影響は無いが、凝縮温度が大きく低下して COP が向上する。冷水温度を高くした場合には蒸発温度が高くなり、効率が向上した結果、凝縮温度もわずかに低下する。図 15.12 に冷水出入口温度差を半分にした場合のデューリング線図上の運転点を示す。破線が定格運転の場合の冷凍サイクル、実践が出入口温度差低下時の冷凍サイクルである。

プログラム 15.25　二重効用吸収冷温水機の運転点の計算

```
private static void DirectFiredAbsorptionChillerTest2()
{
  DirectFiredAbsorptionChiller ar = new DirectFiredAbsorptionChiller(103d / 3600, 103d / 3600,
    15, 7, 32, 37, 54.7, 60, 189 / 3.6, 500 / 3.6, 189 / 3.6, 0, Boiler.Fuel.Gas13A);
  ar.HasSolutionInverterPump = true;

  using (StreamWriter sWriter = new StreamWriter
    ("DirectFiredAbsorptionChillerTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
  {
    sWriter.WriteLine("条件,蒸発温度,凝縮温度,再生温度,COP,稀溶液濃度,濃溶液濃度");

    sWriter.Write("定格性能,");
    ar.Update(32, 15, 500 / 3.6, 189 / 3.6);
    sWriter.WriteLine(ar.EvaporatingTemperature + "," + ar.CondensingTemperature + "," + ar.DesorbTemperature
      + "," + ar.COP + "," + ar.ThinSolutionMassFraction + "," + ar.ThickSolutionMassFraction);
```

```
16
17      sWriter.Write("負荷率 50%,");
18      ar.Update(32, 11, 500 / 3.6, 189 / 3.6);
19      sWriter.WriteLine(ar.EvaporatingTemperature + "," + ar.CondensingTemperature + "," + ar.DesorbTemperature
20        + "," + ar.COP + "," + ar.ThinSolutionMassFraction + "," + ar.ThickSolutionMassFraction);
21
22      sWriter.Write("冷却水温度 25 度,");
23      ar.Update(25, 15, 500 / 3.6, 189 / 3.6);
24      sWriter.WriteLine(ar.EvaporatingTemperature + "," + ar.CondensingTemperature + "," + ar.DesorbTemperature
25        + "," + ar.COP + "," + ar.ThinSolutionMassFraction + "," + ar.ThickSolutionMassFraction);
26
27      sWriter.Write("冷水流量 50%,");
28      ar.Update(32, 15, 500 / 3.6, 189 / 3.6 * 0.5);
29      sWriter.WriteLine(ar.EvaporatingTemperature + "," + ar.CondensingTemperature + "," + ar.DesorbTemperature
30        + "," + ar.COP + "," + ar.ThinSolutionMassFraction + "," + ar.ThickSolutionMassFraction);
31
32      sWriter.Write("冷水出口温度 10 度,");
33      ar.OutletWaterSetPointTemperature = 10;
34      ar.Update(32, 18, 500 / 3.6, 189 / 3.6);
35      sWriter.WriteLine(ar.EvaporatingTemperature + "," + ar.CondensingTemperature + "," + ar.DesorbTemperature
36        + "," + ar.COP + "," + ar.ThinSolutionMassFraction + "," + ar.ThickSolutionMassFraction);
37    }
38 }
```

表 15.3　二重効用吸収冷温水機の運転点の計算

条件	蒸発温度	凝縮温度	再生温度	COP	稀溶液濃度	濃溶液濃度
定格性能	5.0	40.0	159.6	1.36	58.1%	62.6%
冷水出入口温度差半分	6.0	35.9	126.0	1.43	55.0%	58.1%
冷却水温度 25°C	5.0	32.8	126.9	1.48	53.8%	58.0%
冷水流量 50%	6.7	35.9	123.7	1.44	54.6%	57.6%
冷水出口温度 10°C	8.0	39.9	148.1	1.41	56.1%	60.4%

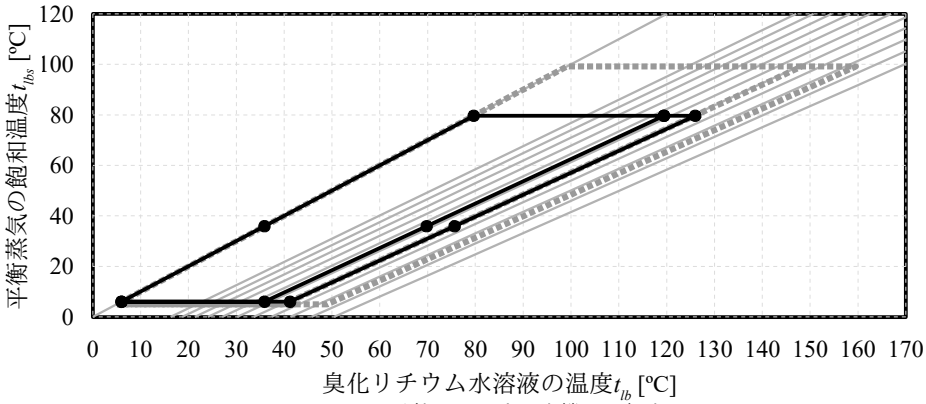


図 15.12　二重効用吸収冷温水機の運転点

【第 15 章 記号表】

A,B	：近似係数	NTU	：移動単位数 **[-]**
a,b,c,d,e	：近似係数	Q	：熱量 **[kW]**
a_w	：溶液循環比 **[-]**	r_H	：高温側溶液配分比 **[-]**
c_{pw}	：水の定圧比熱 **(4.186 kJ/(kg·K))**	T	：温度 **[K]**
h	：比エンタルピー **[kJ/kg]**	t	：温度 **[°C]**
KA	：伝熱係数 **[kW/K]**	w	：質量分率 **[kg/kg]**
m	：質量流量 **[kg/s]**	ε	：熱通過有効度 **[-]**
sub scripts			
A	：吸収器	L	：熱損失
CD	：凝縮器	lb	：臭化リチウム
cd	：冷却水	o	：出口
ch	：冷水	RL	：冷媒（液体）
D	：再生器	RV	：冷媒（蒸気）
E	：蒸発器	S	：臭化リチウム水溶液
H	：高温側	s	：飽和状態
H	：燃料	thk	：濃溶液
i	：入口	thn	：稀溶液
L	：低温側	X	：熱交換器

【第 15 章 参考文献】

15.1） 井上修行: 吸収サイクルの特性解析とその応用に関する研究, 早稲田大学博士論文, 2004.10

15.2） 村上和彦, 佐藤春樹, 渡部康一: 臭化リチウム水溶液の熱力学性質 第 2 報:デューリング線図およびデューリング式の作成, 日本冷凍協会論文集, 12 巻 3 号, pp.307-311, 1995.11

15.3） L. A. McNeely: Thermodynamic properties of aqueous solutions of lithium bromide, ASHRAE Trans. 85, Part 1, 413, 1979

15.4） American Society of Heating, Refrigerating and Air-Conditioning Engineers. ASHRAE Fundamentals Handbook, Chapter 19 Thermophysical Properties of Refrigerants, pp.19.84, 1997

15.5） 内田修一郎, 西口章: 水-臭化リチウム系混合冷媒低温吸収式冷凍機 (小特集:水や空気を冷媒として用いた冷凍空調技術), 冷凍 81(946), pp.618~621, 日本冷凍空調学会, 2006.08

第16章　流体機械
(Fluid Machinery)

16.1　概要

流体に対してエネルギーを与える機械を流体機械と呼び、建築設備分野においてはポンプおよび送風機（ファン）が代表的である。また、圧力というエネルギーを与えるという意味では、ヒートポンプに組み込まれる圧縮機なども流体機械に分類される。

本章ではポンプおよび送風機が運転時に示す流量と圧力の関係をモデル化する。また、流体機械の機器特性と流路の抵抗特性との関係からエネルギー消費量を計算する手法を示す。流路の抵抗特性に関しては流量の 2 次式で表現する単純なモデルを採用することとする。この他に複雑な回路網として捉える方法もあるが、これに関しては次章で述べることにする。エネルギー計算という目的であれば抵抗特性を用いるモデルでも十分な精度であるため、既存のエネルギーシミュレーションプログラムの多くは、本章に示すようなモデルで計算を行っている。

熱源空調システムが設計値通りの条件で稼働することは年間を通じて殆ど無く、多くの時間帯は低負荷で運転することになる。ポンプやファンの流量は設計値よりも遥かに小さく絞られることが通常であり、このような状態におけるエネルギー消費量予測が求められる。特に近年ではインバータを搭載する事例が増加しているため、本書ではポンプ（ファン）効率とモーター効率に加えインバータ効率を反映したモデルの作成方法を示す。

写真 16.1　ゐのくち式渦巻きポンプ
（遠心ポンプの理論を体系化した井口在屋によるポンプ：明治村にて撮影）

16.2　理論

16.2.1　理論動力

ポンプおよびファンの理論動力 W_T [kW (=kN·m/s)]は、揚程（または全圧）P_F [kPa (=kN/m^2)]と体積流量 Q [m^3/s]の積であり、式 16.1 で計算できる。m [kg/m^3]は質量流量、ρ [kg/m^3]は流体の比重量である。定格の回転数における揚程 $P_{F,N}$ [kPa]と体積流量 Q_N [m^3/s]の関係は式 16.2 で示されるように多項式近似が可能である。通常は 3 次式程度で十分な近似性能が得られる。式 16.2 にもとづき、横軸に体積流量、縦軸に揚程を表現したグラフは PQ 特性線図と呼ばれ、流体機械の性能を捉えるための基本的な図表である。

$$W_T = P_F Q = P_F m / \rho \tag{16.1}$$

$$P_{F,N} = \sum_{n=0}^{N} a_{P,n} Q_N^n \tag{16.2}$$

現実の機械には各種の損失があるため、モーターの消費電力 W_M [kW]は流体機械の理論動力 W_T [kW]に一致しない。通常はポンプ（ファン）効率 η_F [-]、モーター効率 η_M [-]、インバータ効率 η_{Inv} [-]による損失を考慮して式 16.3 で計算する。3 つの効率を相乗した値を流体機械の総合効率と呼ぶ。

$$W_M = \frac{W_T}{\eta_F \eta_M \eta_{Inv}} \tag{16.3}$$

定格条件におけるポンプ（ファン）効率 $\eta_{F,N}$ [-]は体積流量の関数として式 16.4 で近似する。PQ 特性と同様に 3 次式程度で十分な精度が得られる。通常はポンプ（ファン）効率は 0.3~0.7 程度の値をとり、定格流量が大きい機種の方が効率も高い傾向にある。

$$\eta_{F,N} = \sum_{n=0}^{N} a_{\eta,n} Q_N^n \tag{16.4}$$

モーター効率はモーター出力（理論動力 W_T [-]）の関数として式 16.5 で近似する。モーター効率は 0.60~0.95 程度の値をとり、出力が大きいほど効率が高い傾向にある。後述のようにメーカー技術資料にはモーター効率が記載されているため、モデル化の度に近似係数を求めても良いが、あまり精度を求めないのであれば $a_{M,0}$ = -8.724 × 10^{-3}、$a_{M,1}$ = 6.567 × 10^{-2}、$a_{M,2}$ = 8.015 × 10^{-1} としておけば概ね妥当な値が得られる。この係数は 50Hz 地域用の 0.2~45kW のモーターを持つポンプおよびファンについて回帰したものである。

$$\eta_M = a_{M,0} \ln(W_T)^2 + a_{M,1} \ln(W_T) + a_{M,2} \tag{16.5}$$

インバータ効率は負荷率 L_P [-]の関数として式 16.6 で近似する。負荷率は定格条件における理論動力に対する運転条件における理論動力の比率である。R_N は定格条件における回転数 N_N [rps]に対する実際の回転数 N [rps]の比率（N/N_N）である。インバータ効率は負荷率の低下とともに低下する傾向がある。近似係数 $\alpha_{Inv,n,m}$ の値を表 16.1 に示す[†1)]。式 16.6 にもとづいてインバータ効率を計算した結果を図 16.1 に示す。

$$\eta_{Inv} = \sum_{n=0}^{2} \sum_{m=0}^{2} a_{Inv,n,m} L_P^n R_N^m \tag{16.6}$$

†1　参考文献 16.7)に記載の図に基いて近似係数を計算した。

表 16.1　インバータ効率近似係数

$a_{Inv,n,m}$	m=2	m=1	m=0
n=2	-0.2287	0.3425	-0.4923
n=1	0.4690	-0.8248	0.9172
n=0	-0.3356	0.6319	0.4756

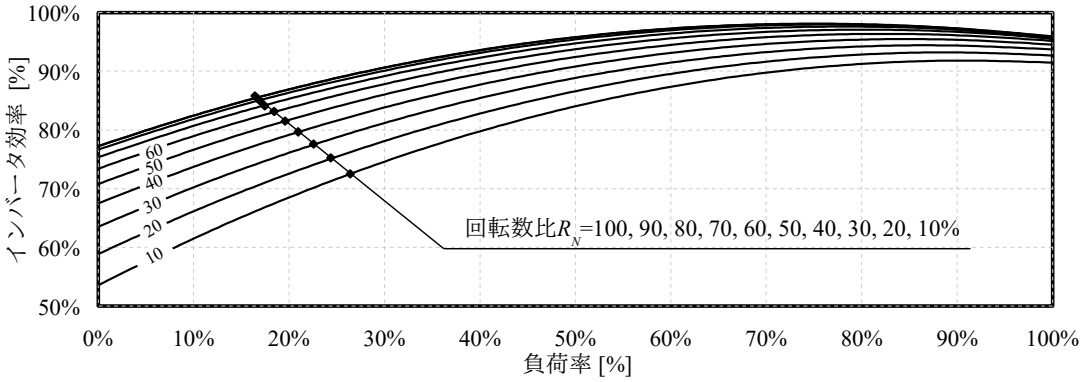


図 16.1　負荷率・回転数比とインバータ効率の関係

回転数を変化させた場合の体積流量と揚程の関係は回転数比 R_N (=N/N_N) を用いて式 16.7 と式 16.8 で表現できる。これらを式 16.2 と式 16.4 に代入すると式 16.9 と式 16.10 が得られ、任意の回転数と体積流量に対する揚程および効率を計算することができる。

$$Q = R_N Q_N \tag{16.7}$$

$$P_F = R_N^2 P_{F,N} \tag{16.8}$$

$$P_F = R_N^2 \sum_{n=0}^{N} a_{P,n} (Q / R_N)^n \tag{16.9}$$

$$\eta_F = \sum_{n=0}^{N} a_{\eta,n} (Q / R_N)^n \tag{16.10}$$

【例題 16.1】

ポンプとファンのそれぞれについて熱搬送の効率を計算せよ。ただし単位距離あたりの圧力損失を、配管は 1 kPa/m、ダクトは 1 Pa/m で計画し、温度差はいずれも 10 ºC、搬送距離は 500 m とする。ポンプ効率、ファン効率、モーター効率、インバータ効率をかけ合わせた総合効率は 50 %とし、ポンプとファンとで等しいとする。また、水と空気の比熱と比重量はそれぞれ、4.186 kJ/(kg·K)、1,000 kg/m³、1.006 kJ/(kg·K)、1.2 kg/m³ とする。

【解】

1 kW の熱を搬送するために必要な水量は、1 ÷ 4.186 ÷ 10 = 0.0239 kg/s である。体積流量に変換すると 0.0239 ÷ 1,000 = 2.39×10^{-6} m³/s となる。式 16.1 により理論動力は 2.39×10^{-6} × 1 kPa/m × 500 m ÷ 50 % = 0.0239 kW である。従って熱搬送効率は 1 ÷ 0.0239 = 41.9 となる。

同様に空気の熱搬送効率を計算する。1 kW の熱を搬送するために必要な風量は、1 ÷ 1.006 ÷ 10 = 0.0994 kg/s である。体積流量に変換すると 0.0994 ÷ 1.2 =8.28×10^{-2} m³/s となる。理論動力は 8.28×10^{-2} × 1 kPa/m × 500 m ÷ 50 % = 0.0828 kW である。従って熱搬送効率は 1 ÷ 0.0828 = 12.1 となる。

水の熱搬送効率を WTF（Water Transfer Factor）、空気の熱搬送効率を ATF（Air Transfer Factor）と呼ぶ。実運転データの分析に際しては横軸に負荷率、縦軸に熱搬送効率（WTF or ATF）をとった散布図を描画し、温度差制御が機能していることを確認することが多い。また、上記の通り、一般に WTF は ATF よりも高い。熱源空調システムにおいて熱源機から直接に空気を媒体として熱を取り出さず、一旦、水を経由することの理由はここにある。

16.2.2　ポンプ

1）機器特性

図 16.2 に示すような特性線図が技術資料として用意されており、メーカーから入手可能であるため、これを用いて近似係数の推定を行う。流量は L/min や m³/h のように体積流量で表現されることが通常であり比重量を用いて質量流量に変換する必要がある。また、全揚程が m 単位で表現されている場合には重力加速度 g (=9.8 m/s²)を乗じて kPa に変換する。

試験点 No.		1	2	3	4	5
ポンプ	吐出量 [L/min]	0	833	1,670	2,500	2,800
	全揚程 [m]	35.1	34.0	28.5	18.5	13.2
	効率 [%]	0.0%	60.1%	80.8%	72.4%	59.0%
電動機	入力 [kW]	6.95	8.83	11.13	12.15	11.89
	効率 [%]	86.9%	86.9%	86.2%	85.7%	85.8%
	出力 [kW]	6.04	7.67	9.59	10.41	10.2

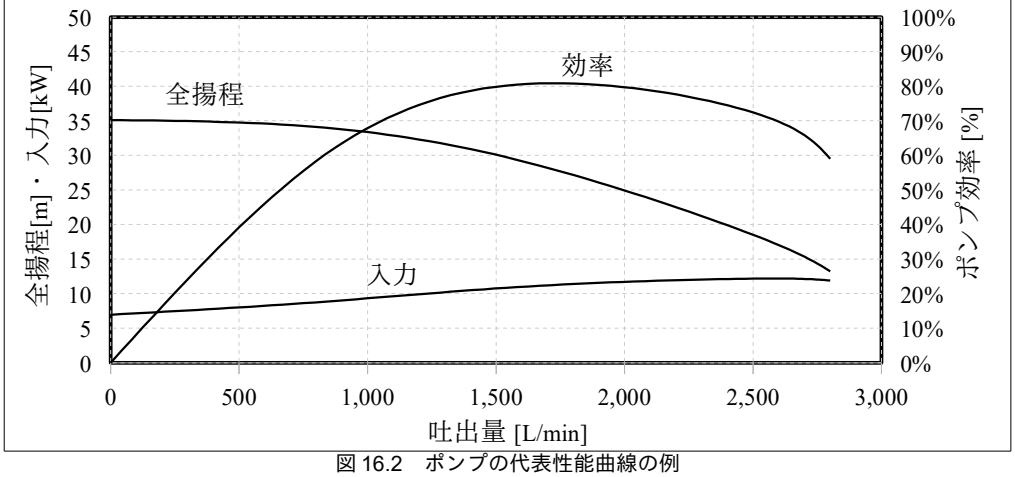


図 16.2　ポンプの代表性能曲線の例

一般に揚程は流量の減少に伴って増加する凸形状をしている。ポンプ効率は極大値を持ち、流量が 0 のときに 0 となる。入力は流量の増大とともに増加する。図 16.2 は空調で用いられることが多い渦巻ポンプの特性であり、ポンプの種類によっては、これとは異なる特性もあることに注意する。

技術資料から式 16.4 と式 16.5 の近似係数を推定する作業は煩わしい。設計水量と揚程から近似係数を簡易に推定する手法が存在するため[16.5) 16.6)]、あまり精度が求められない検討段階において適用すると有効である。推定式を式 16.11 と式 16.12 に示す。$P_{F,M}$ [kPa]、$\eta_{F,M}$ [-]、Q_M [m³/s]はそれぞれポンプ効率が最大となる点における揚程、ポンプ効率、体積流量であり、設計によって理想的な運転点でポンプが選定されたと考えれば定格条件での値を設定すれば良い（$P_{F,M} = P_{F,N}$、$\eta_{F,M} = \eta_{F,N}$、$Q_M = Q_N$）。図 16.2 の例では 260 kPa、0.03 m³/s (=1,800 L/min)、80 %である。$\beta_{P,n}$ と $\beta_{\eta,n}$ の値を表 16.2 に示す。

$$\alpha_{P,n} = \beta_{P,n} P_{F,M} / Q_M^n \tag{16.11}$$

$$\alpha_{\eta,n} = \beta_{\eta,n} \eta_{F,M} / Q_M^n \tag{16.12}$$

表 16.2　ポンプ特性推定式の係数

n	0	1	2
$\beta_{P,n}$	1.294	0.096	-0.395
$\beta_{\eta,n}$	0.109	1.795	-0.929

最大のポンプ効率は式 16.13 を用いて体積流量から推定する[†1)]。推定式の適用可能範囲は 100~8,000

†1　JIS において吐出量に応じた最高効率（A 効率と呼ばれる）が定められているが、式 16.13 の効率はこれを上回る。市場の実機の効率は JIS 規定値よりも平均で 5%程度高いようである。

L/min 程度である。

$$\eta_{F,M} = -0.01618\ln(Q_M)^2 - 0.05662\ln(Q_M) + 0.78179 \tag{16.13}$$

【例題 16.2】

式 16.11 と式 16.13 で示した特性推定式を用いて設計水量 1,800 L/min、設計揚程 260 kPa のポンプの特性を計算し、図 16.2 の特性と比較せよ。

【解】

水の比重量は 1 kg/L であるため、体積流量は 0.03 m^3/s である。式 16.13 を用いると最高効率は、$\eta_{F,M}$ = -0.01618×ln(0.03)^2 + 0.05662 ln(0.03) + 0.78179 = 78.1 %となる。従って、特性式近似係数は、

$\alpha_{\eta,2}$ = -0.929 × 78.1 % / 0.03^2 = -8.419 × 10^{-4}　　$\alpha_{P,2}$ = -0.395 × 260 / 0.03^2 = -0.114

$\alpha_{\eta,1}$ = 1.795 × 78.1 % / 0.03^1 = 4.879 × 10^{-2}　　$\alpha_{P,1}$ = 0.096 × 260 / 0.03^1 = 0.8349

$\alpha_{\eta,0}$ = 0.109 × 78.1 % / 0.03^0 = 8.917 × 10^{-2}　　$\alpha_{P,0}$ = 1.294 × 260 / 30.03^0 = 336.3

である。図 16.3 にこれらの近似係数を用いて計算した特性線図を示す。メーカーによる特性式と比較して最大でも 1 割程度の誤差内に納まっている。

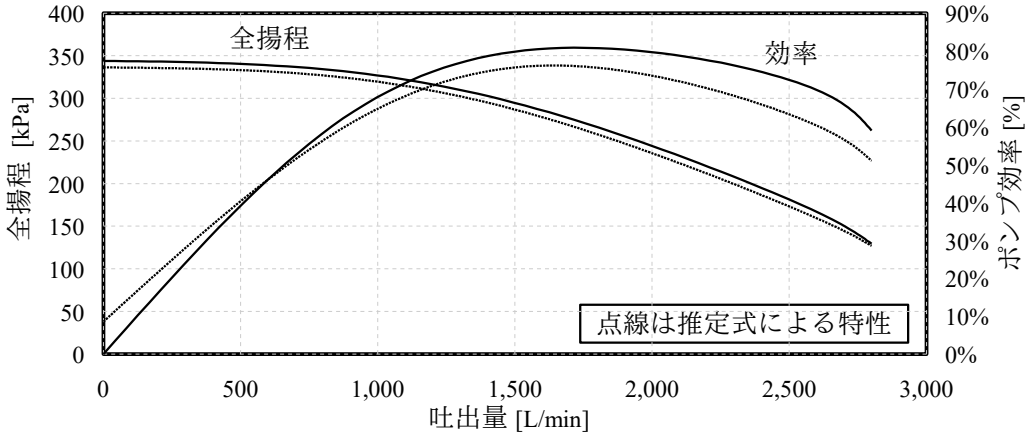


図 16.3　ポンプ性能特性の推定結果

2）制御方式と運転点

式 16.9 では回転数比 R_N と揚程 P_F の二つが未知数であり、具体的な運転点を特定するためには、式がもう一つ必要である。エネルギーシミュレーションでは、流路の抵抗特性を式 16.14 で近似することが多い。P_F [kPa]は流路の圧力損失であり、ポンプの揚程に一致する。α_R [kPa/(kg/s)^2]は流路の抵抗係数、α_0 [kPa]は流路の実揚程である。抵抗係数 α_R は流路を検討して積み上げ方式で求めることができるが、煩雑であるため通常は設計流量と設計揚程から逆算して設定する。実揚程 α_0 は、冷却水系統における冷却塔や蓄熱系統における汲み上げのためのエネルギーなどが存在する場合に設定する。

$$P_F = \alpha_R Q^2 + \alpha_0 \tag{16.14}$$

図 16.4 に流量制御方式に応じたポンプの運転点を示す。図中、定格点と記した運転点は、メーカーがポンプの性能を測定した点であり、技術資料にはこの点が掲載されている。設計点は設計者が二次側配管等の計画をもとに設定した運転点であり、ピーク流量のときにはこの点で動作する。当然、定格点を通る PQ 特性は設計点を内包する。

左の図はヘッダ間のバイパス弁開閉により吐出圧を一定に保った場合である。二次側負荷の減少により AHU 二方弁開度が小さくなり、二次側抵抗は図の二点鎖線となる。このままでは吐出圧が高くなるため、バイパス弁を開いて流体をバイパスさせる。結果としてポンプの運転点は設計運転点で変わらず、消費電力量は定格消費電力量となる。

中央の図は回転数制御によって吐出圧を一定に保った場合である。二次側抵抗の減少に対して、回転数を下げることによって圧力を一定に保つ。式 16.1 に示されるとおり、ポンプの流量に比例して消費電力量が減少する。

右の図は回転数制御によって吐出圧を最小化した場合である。ポンプの運転点は流量の減少に合わせて流路の抵抗曲線をくだる。回転数比が最小（図では 60%）となった後はバイパス制御に移り、以降の消費電力量は一定となる。式 16.14 により圧力損失 P_F は流量の 2 乗にほぼ比例するため、式 16.1 と合わせると、消費電力量は流量の 3 乗にほぼ比例して減少することがわかる。ただし、現実には全ての流路の流量が同じ比率で減少しない限りは抵抗係数は一定とならないため、厳密には正しくない。この点を評価するためには次章に述べる回路網の計算が必要となる。しかし計算が煩雑になるため、多くのシミュレーションプログラムはこの簡単なモデルで最小吐出圧制御を表現している。

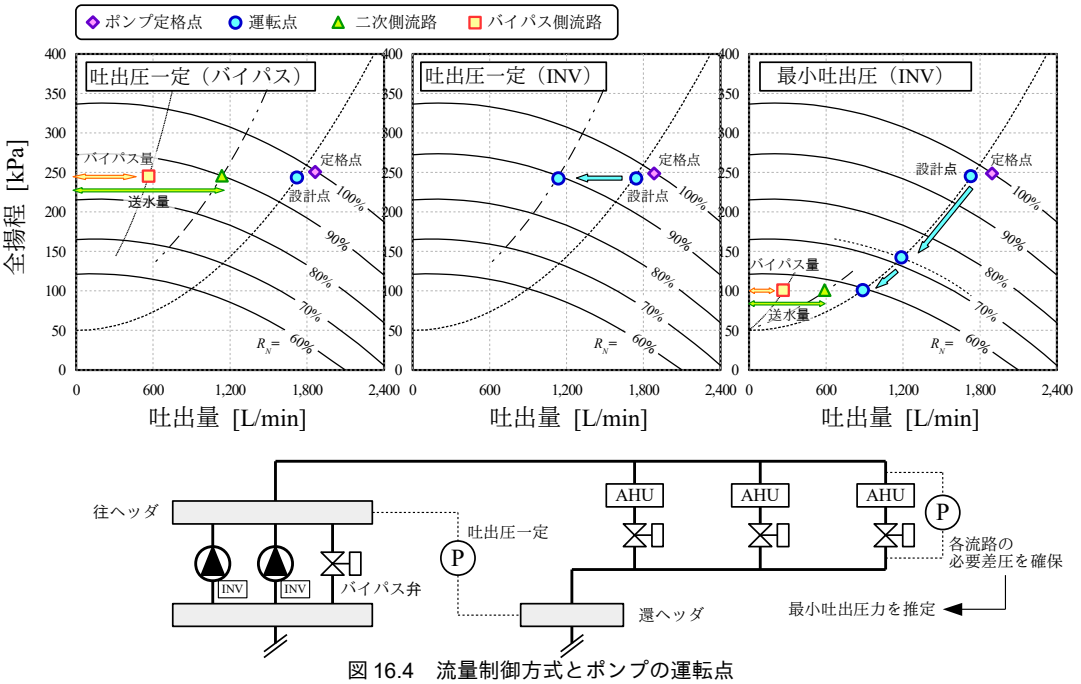


図 16.4　流量制御方式とポンプの運転点

3）台数制御

複数台のポンプを並列に接続した場合、PQ 特性は図 16.5 のように合成される。1 台運転の時の PQ 特性を吐出量方向に重ね合わせた特性となる。直列に接続した場合には PQ 特性を揚程方向に重ね合わせた特性となるが、このような構成は熱源空調設備ではあまり計画することが無い。以降、このような複数のポンプによって構成されたシステムをポンプシステムと呼ぶことにする。

揚程が一定の場合、台数を 1 台追加することによって増加する吐出量は一定である。従って吐出圧一定制御の場合には 1 台運転の場合の運転点を計算し、その場合の吐出量で所要流量を割れば運転台数が求まる。一方、最小吐出圧制御の場合には 1 台増段することによって増加する水量が異なるため、都度、計算をして必要台数を確認する必要がある。また、図の最小吐出圧制御の 1 台運転の運転点を確認すると、定格運転（1,800 L/min, 260 kPa）に比較して水量が多く揚程が低い点（3,000 L/min, 100 kPa）での運転が行われている。図 16.2 に示したとおり、過大な流量ではポンプ効率が低いため、エネルギー消費量が増大する。そこでこのような場合には並列運転を行うポンプとは別に、図の

二点鎖線で示すような PQ 特性を持つ小流量ポンプを計画すると良い。

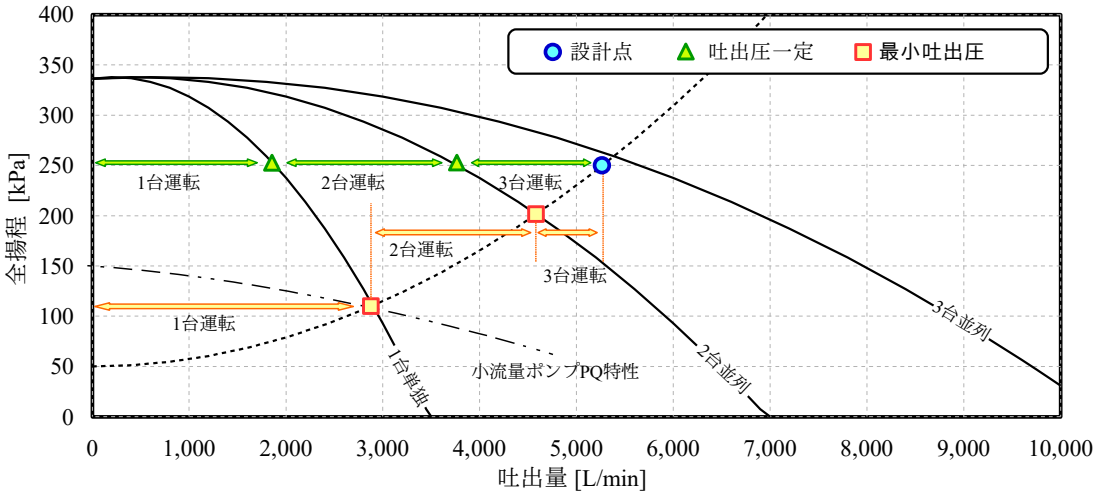


図 16.5　並列運転を行った場合の PQ 特性

16.2.3　送風機

1）機器特性

ファンの場合にもポンプと同様に、風量と全圧や効率との関係性が性能曲線としてメーカーから提供されていることが通常である。しかし技術資料から特性式を推定する作業は煩雑であるため、ポンプと同様に簡易に特性係数を推定する手法を以下に述べる。

慣例として、ファンは通常はサイズを番手で表現する。160mm の羽根直径を基準とし、160mm のファンを 1 番手、320mm のファンを 2 番手...と呼ぶ†1)。番手 ct と全圧 $P_{F,M}$ [kPa]が与えられれば最大ファン効率 $\eta_{F,M}$ [-]は式 16.15 で計算できる†2)。式 16.15 の近似係数を表 16.3 に示す。式 16.15 にもとづいて静圧と最大効率の関係を番手別に計算した結果を図 16.6 に示す。番手が大きく、静圧が高いほど最大効率は高い傾向にある。

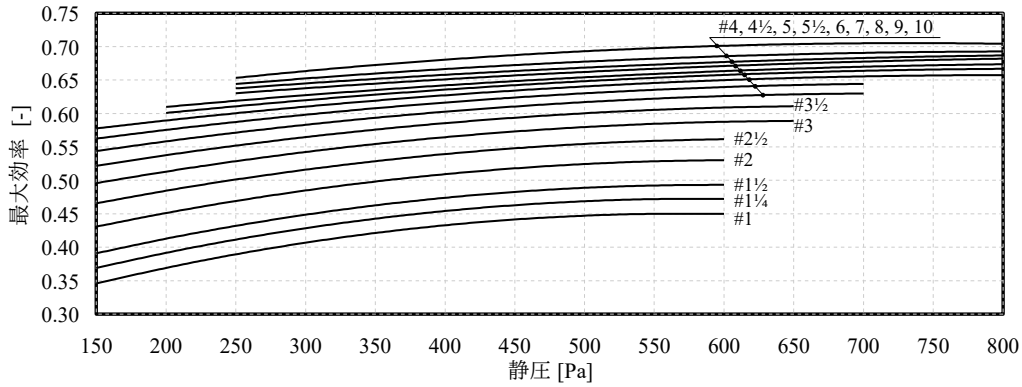


図 16.6　ファンの最大効率特性

$$\eta_{F,M}=\sum_{j=0}^{J}\sum_{k=0}^{K} a_{FM,j,k}\, ct^{j}\cdot P_{F,M}^{k} \tag{16.15}$$

†1　HVACSIM+(J)などでは、羽根直径を用いて無次元化した効率特性式、圧力特性式を作成するというモデルが採用されている。番手から羽根直径が推定できるため、同モデルを用いれば汎用のファンモデルを作成できるかと思ったが、イマイチ芳しい結果が得られなかった。

†2　参考文献 16.9 に記載の効率特性図から作成した。同図表を用いて近似式を作成した BEST プログラムの計算結果と比較して平均誤差率 0.4 %、最大誤差率 1.3 %の近似性能である。

表 16.3　ファンの最大効率の近似係数 $a_{F,M}$

		j			
		0	1	2	3
k	0	1.397E-01	1.303E-01	-1.179E-02	3.219E-04
	1	7.649E-01	-9.105E-02	-5.345E-03	1.006E-03
	2	-7.141E-01	1.325E-01	-2.639E-03	-5.690E-04

効率および全圧の近似係数推定式はポンプと同様に式 16.11 と式 16.12 を用いる。ファンの場合の係数を表 16.4 に示す[†1)]。

表 16.4　ファン特性推定式の係数

n	0	1	2	3
$\beta_{P,n}$	0.767	0.517	-0.300	-
$\beta_{\eta,n}$	-	2.350	-1.700	0.337

2）制御

風量を制御するためには、ダンパによって流路の抵抗を上げる方法と、インバータによってファン自体の回転数を変化させる方法がある。原理はポンプのバルブ制御とインバータ制御と同様である。ポンプの場合は台数制御を行うことが一般的であるが、ファンの場合には 1 系統に対して複数のファンで給気を行うことは余り多くない。従って、台数制御の計算法は省略するが、原理はポンプと同じである[†2)]。

【例題 16.3】

下記の定格仕様を持つファンについて PQ 特性と効率特性線図（回転数比 R_N は 100 %と 80 %とする）を作成せよ。

番手：#4　　全圧：550 Pa　　風量：13,500CMH

【解】

式 16.15 の ct に 4 を、$P_{F,M}$ に 0.55 kPa を代入することで、定格条件におけるファン効率は 62.2 %となる。また、空気の比重を 1.2kg/s とすれば空気の質量流量は 13,500CMH×1.2÷3,600=4.5 kg/s である。従って、式 16.11、式 16.12、表 16.4 を用いることで、特性式の係数は、

$\alpha_{\eta,3} = 0.337 \times 62.2\ \% / 4.5^3 = 0.002$　　$\alpha_{P,2} = -0.300 \times 0.55 / 4.5^2 = -0.008$

$\alpha_{\eta,2} = -1.700 \times 62.2\ \% / 4.5^2 = -0.052$　　$\alpha_{P,1} = 0.517 \times 0.55 / 4.5^1 = 0.063$

$\alpha_{\eta,1} = 2.350 \times 62.2\ \% / 4.5^1 = 0.325$　　$\alpha_{P,0} = 0.767 \times 0.55 / 4.5^0 = 0.422$

となる。これらを式 16.9 と式 16.10 に代入して風量別に全圧と効率を計算すると図 16.7 が得られる。

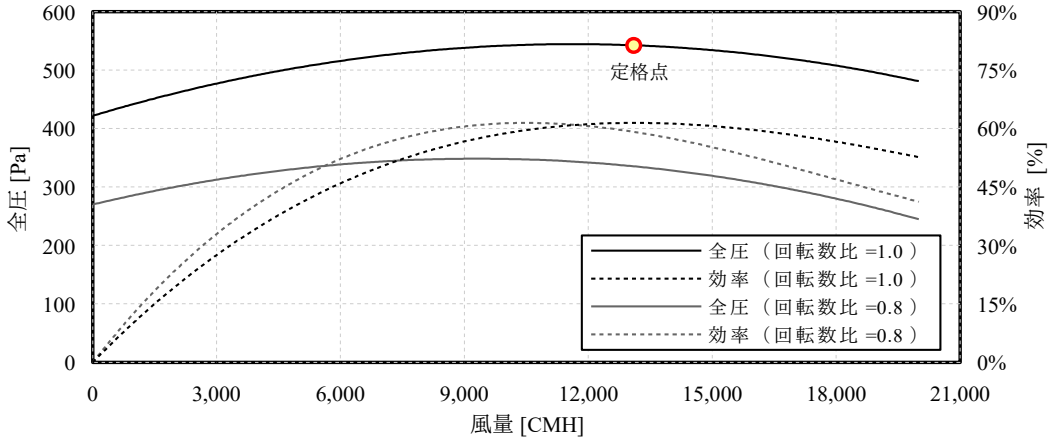


図 16.7　ファンの PQ 特性と効率特性

†1　E 社の片吸込遠心ファン 83 台（#1~#8）から作成した特性近似係数である。

†2　流体搬送の原理から言えば、複数の AHU 系統を同一のダクトに集合させるほうが、同時負荷率が小さくなり、実運用時のダクト抵抗が下がる結果、エネルギー消費量が削減できるように思う。しかし、系統分割を行えば系統ごとに給気の温湿度設定が可能となるというメリットもある。また、冷温水配管に比較するとダクトの方がメンテナンスが生じやすく、メンテナンスによる使用停止の影響を多系統に及ぼしたくないという都合もある。なお、機器の堅牢性向上という意味も兼ねて、1 台の AHU の中に 2 台のファンを設置する機器は存在する。

16.3 計算法

16.3.1 「流体機械クラス」の作成

ポンプとファンはともに流体機械であるため、いくつかの計算処理は共通する[†1)]。そこで両者に共通する処理をまとめて抽象クラスである「流体機械クラス」を作成する。プログラム 16.1 に流体機械効率、モーター効率、インバータ効率の計算処理を示す。式 16.4~16.6の実装である。モーター効率とインバータ効率はポンプとファンで同じ特性を用いるが、流体機械効率は別のため、1~6 行のメソッドでは近似係数を変数として扱い、具体的な値は派生クラスでそれぞれに設定する。

プログラム 16.1　インバータおよびモーター効率の計算

Popolo.HVAC.Circuit.FluidMachinery class

```
 1 /// <summary>流体機械効率[-]を計算する</summary>
 2 /// <param name="flowRate">体積流量[m3/s]</param>
 3 /// <param name="efficiencyCoef">機械効率[-]</param>
 4 /// <returns>流体機械効率[-]</returns>
 5 public static double GetFluidMachineryEfficiency(double flowRate, double[] efficiencyCoef)
 6 { return getPolynomial(flowRate, efficiencyCoef); }
 7
 8 /// <summary>モーター効率[-]を計算する</summary>
 9 /// <param name="shaftPower">軸動力[kW]</param>
10 /// <returns>モーター効率[-]</returns>
11 public static double GetMotorEfficiency(double shaftPower)
12 {
13   double lnWT = Math.Log(shaftPower);
14   return 8.015e-1 + (6.567e-2 + -8.724e-3 * lnWT) * lnWT;
15 }
16
17 /// <summary>インバータ効率[-]を計算する</summary>
18 /// <param name="loadRate">負荷率[-]</param>
19 /// <param name="rotationRatio">回転数比[-]</param>
20 /// <returns>インバータ効率[-]</returns>
21 public static double GetInverterEfficiency(double loadRate, double rotationRatio)
22 {
23   double r2 = rotationRatio * rotationRatio;
24   double a1 = -0.2287 * r2 + 0.3425 * rotationRatio - 0.4923;
25   double a2 = 0.4690 * r2 - 0.8248 * rotationRatio + 0.9172;
26   double a3 = -0.3356 * r2 + 0.6319 * rotationRatio + 0.4756;
27   return loadRate * (loadRate * a1 + a2) + a3;
28 }
29
30 /// <summary>多項式 y=Σa[n]*x^(N-n)の計算を行う</summary>
31 /// <param name="x">入力変数</param>
32 /// <param name="a">係数配列</param>
33 /// <returns>y=Σa[n]*x^n</returns>
34 private static double getPolynomial(double x, double[] a)
35 {
36   double y = a[0];
37   for (int i = 1; i < a.Length; i++) y = y * x + a[i];
38   return y;
39 }
```

プログラム 16.2 は PQ 特性線上の運転点の計算処理である。

1~11 行は任意の流量と圧力から回転数比を求めるメソッドである。式 16.9 を回転数比 R_N について解く処理であるが、解析的に解を求めることはできないため、ニュートン・ラフソン法を用いた反復計算を行う。8, 9 行が誤差関数であり、与えられた体積流量と仮定した回転数にもとづいて 62~68 行のメソッドで圧力を計算（式 16.9）し、入力条件である圧力との差分を誤差として出力する。

13~34 行は圧力と回転数比から流量を求めるメソッドであり、式 16.9 を流量 m について解く処理である。これも特性式の次数が大きい場合には解析的に解けず、反復計算が必要となるため、ニュートン・ラフソン法を適用する[†2)]。ただし、多項式近似であり微分値を解析的に得ることができる

†1　オブジェクト指向プログラムの特徴である「継承」を活用できる典型的な場面である。

†2　国土交通省大臣官房官庁営繕部の LCEM のためのシミュレーションツールでは、2 次式で表現した上で解の公式で反復計算なしに解く方法を採用している。計算が安定するというメリットがあるため、このような割り切りは設計初期段階の計算とし

（70~81 行）ため、数値微分ではなくこれを用いて反復計算を行う。

36~60 行は抵抗曲線と PQ 特性の交点を計算する関数であり、式 16.9 と式 16.14 を連立させて流量 m および圧力 P_F を求める処理である。

プログラム 16.2　PQ 特性上の運転点計算処理

Popolo.HVAC.Circuit.FluidMachinery class

```
 1 /// <summary>体積流量と圧力から回転数比を計算する</summary>
 2 /// <param name="flowRate">体積流量[m3/s]</param>
 3 /// <param name="pressure">圧力[kPa]</param>
 4 /// <param name="pressureCoef">圧力特性係数</param>
 5 /// <returns>回転数比[-]</returns>
 6 public static double GetRotationRatio(double flowRate, double pressure, double[] pressureCoef)
 7 {
 8   Roots.ErrorFunction eFnc = delegate (double rRate)
 9   { return GetPressure(flowRate, rRate, pressureCoef) - pressure; };
10   return Roots.Newton(eFnc, 0.8, 1e-5, 1e-4, 1e-4, 20);
11 }
12
13 /// <summary>設定圧力と PQ 特性が交わる体積流量を計算する</summary>
14 /// <param name="rotationRatio">回転数比[-]</param>
15 /// <param name="pressureCoef">圧力特性係数</param>
16 /// <param name="pressure">設定圧力[kPa]</param>
17 /// <param name="initialFlowRate">体積流量初期値[m3/s]</param>
18 /// <returns>体積流量[m3/s]</returns>
19 public static double GetFlowRate
20   (double rotationRatio, double[] pressureCoef, double pressure, double initialFlowRate)
21 {
22   if (rotationRatio <= 0) return 0;
23   double r2 = rotationRatio * rotationRatio;
24
25   //締切り揚程で不足する場合には 0 を出力
26   if (r2 * getPolynomial(0, pressureCoef) < pressure) return 0;
27
28   Roots.ErrorFunction eFnc = delegate (double vf)
29   { return r2 * getPolynomial(vf / rotationRatio, pressureCoef) - pressure; };
30   Roots.ErrorFunction eFncD = delegate (double vf)
31   { return r2 * getPolynomialD(vf, 1 / rotationRatio, pressureCoef); };
32
33   return Roots.Newton(eFnc, eFncD, initialFlowRate, 1e-4, 1e-4, 20);
34 }
35
36 /// <summary>抵抗曲線と PQ 特性が交わる体積流量を計算する</summary>
37 /// <param name="rotationRatio">回転数比[-]</param>
38 /// <param name="pressureCoef">圧力特性係数</param>
39 /// <param name="resistanceCoefficient">抵抗係数[kPa/(m3/s)^2]</param>
40 /// <param name="actualHead">実揚程[kPa]</param>
41 /// <param name="operatingNumber">運転台数</param>
42 /// <param name="initialFlowRate">体積流量初期値[m3/s]</param>
43 /// <returns>体積流量[m3/s]</returns>
44 public static double GetFlowRate
45   (double rotationRatio, double[] pressureCoef, double resistanceCoefficient,
46   double actualHead, int operatingNumber, double initialFlowRate)
47 {
48   double r2 = rotationRatio * rotationRatio;
49   Roots.ErrorFunction eFnc = delegate (double vf)
50   {
51     return r2 * getPolynomial
52       (vf / (operatingNumber * rotationRatio), pressureCoef) - (vf * vf * resistanceCoefficient + actualHead);
53   };
54   Roots.ErrorFunction eFncD = delegate (double vf)
55   {
56     return r2 * getPolynomialD(vf, 1 / (operatingNumber * rotationRatio), pressureCoef)
57       - 2 * vf * resistanceCoefficient;
58   };
59   return Roots.Newton(eFnc, eFncD, initialFlowRate, 1e-4, 1e-4, 20);
60 }
61
62 /// <summary>体積流量と回転数比から圧力を計算する</summary>
63 /// <param name="flowRate">体積流量[m3/s]</param>
64 /// <param name="rotationRatio">回転数比[-]</param>
65 /// <param name="pressureCoef">圧力特性係数</param>
66 /// <returns>圧力[kPa]</returns>
67 public static double GetPressure(double flowRate, double rotationRatio, double[] pressureCoef)
68 { return rotationRatio * rotationRatio * getPolynomial(flowRate / rotationRatio, pressureCoef); }
```

て有効である。この発展形として、筆者は 3 次式で表現した上でカルダーノの公式を用いて解析的に求根する手段を検討したことがあるが、複数の実数解から適切なものを選択する処理などが生じ、かえって煩雑であった。

```
69
70 /// <summary>多項式 y=Σa[n]*x^(N-n)*b^(N-n)の微分値の計算を行う</summary>
71 /// <param name="x">入力変数</param>
72 /// <param name="b">係数 b</param>
73 /// <param name="a">係数配列</param>
74 /// <returns>多項式 y=Σa[n]*x^(N-n)*b^(N-n)の微分値</returns>
75 private static double getPolynomialD(double x, double b, double[] a)
76 {
77   int pow = a.Length - 1;
78   double y = b * pow * a[0];
79   for (int i = 1; i < a.Length - 1; i++) y = b * (x * y + (pow - i) * a[i]);
80   return y;
81 }
```

流体機械クラスのインスタンス変数、プロパティ、コンストラクタの定義をプログラム 16.3 に示す。流体機械効率特性（2 行）、PQ 特性（5 行）、回路抵抗特性（8 行）に関しては派生クラスから値の書き換えができるようにアクセス修飾子を protected にする。コンストラクタでは、設計条件である圧力と流量、実揚程を用いて式 16.14 で回路の抵抗係数を推定する（64 行）。

プログラム 16.3　流体機械クラスのインスタンス変数、プロパティ、コンストラクタの定義

Popolo.HVAC.Circuit.FluidMachinery class

```
 1 /// <summary>効率特性係数</summary>
 2 protected double[] efficiencyCoefficient = new double[3];
 3
 4 /// <summary>揚程特性係数</summary>
 5 protected double[] pressureCoefficient = new double[3];
 6
 7 /// <summary>抵抗係数[kPa/(m3/s)^2]</summary>
 8 protected double resistanceCoefficient;
 9
10 /// <summary>最小回転数比[-]</summary>
11 private double minimumRotationRatio = 0.4;
12
13 /// <summary>体積流量[m3/s]を設定・取得する</summary>
14 public double VolumetricFlowRate { get; internal set; }
15
16 /// <summary>回転数比[-]を設定・取得する</summary>
17 public double RotationRatio { get; set; } = 1.0;
18
19 /// <summary>最小回転数比[-]を設定・取得する</summary>
20 public double MinimumRotationRatio
21 {
22   get { return minimumRotationRatio; }
23   set { minimumRotationRatio = Math.Max(Math.Min(value, 1.0), 0.05); }
24 }
25
26 /// <summary>全圧・揚程[kPa]を取得する</summary>
27 public double Pressure { get; protected set; }
28
29 /// <summary>実揚程[kPa]を取得する</summary>
30 public double ActualHead { get; private set; }
31
32 /// <summary>インバータ機か否か</summary>
33 public bool HasInverter { get; private set; }
34
35 /// <summary>設計流量[m3/s]を取得する</summary>
36 public double DesignFlowRate { get; private set; }
37
38 /// <summary>定格軸動力[kW]を取得する</summary>
39 public double NominalShaftPower { get; private set; }
40
41 /// <summary>モーター効率[-]を取得する</summary>
42 public double MotorEfficiency { get; private set; }
43
44 /// <summary>インスタンスを初期化する</summary>
45 /// <param name="nominalPressure">定格全圧・揚程[kPa]</param>
46 /// <param name="nominalFlowRate">定格流量[m3/s]</param>
47 /// <param name="designPressure">設計全圧・揚程[kPa]</param>
48 /// <param name="designFlowRate">設計流量[m3/s]</param>
49 /// <param name="actualHead">実揚程[kPa]</param>
50 /// <param name="hasInverter">インバータ機か否か</param>
51 protected FluidMachinery
52   (double nominalPressure, double nominalFlowRate, double designPressure,
53   double designFlowRate, double actualHead, bool hasInverter)
54 {
55   DesignFlowRate = designFlowRate;
```

```
56   ActualHead = actualHead;
57   HasInverter = hasInverter;
58
59   //定格軸動力[kW]を計算
60   NominalShaftPower = nominalPressure * nominalFlowRate;
61   MotorEfficiency = GetMotorEfficiency(NominalShaftPower);
62
63   //抵抗係数[kPa/(m3/s)^2]を計算
64   resistanceCoefficient = (designPressure - actualHead) / (designFlowRate * designFlowRate);
65 }
```

流体機械クラスのインスタンスメソッドをプログラム 16.4 に示す。

1~12 行は流体機械効率とインバータ効率の計算処理であり、プログラム 16.1 の static メソッドを用いる。これらを用いて総合効率を計算するメソッドを 14~16 行に、総合効率から消費電力を計算するメソッドを 18~20 行に定義する（式 16.3）。22~77 行はプログラム 16.1 の static メソッドと同じ処理であるが、入力条件と計算結果をインスタンスのプロパティに保存する点が異なる。

プログラム 16.4　流体機械クラスのインスタンスメソッド

Popolo.HVAC.Circuit.FluidMachinery class

```
 1 /// <summary>流体機械効率[-]を取得する</summary>
 2 public virtual double GetFluidMachineryEfficiency()
 3 { return Math.Max(GetFluidMachineryEfficiency(VolumetricFlowRate, efficiencyCoefficient), 0.05); }
 4
 5 /// <summary>インバータ効率[-]を取得する</summary>
 6 public virtual double GetInverterEfficiency()
 7 {
 8   if (HasInverter)
 9     return GetInverterEfficiency
10       (VolumetricFlowRate * Pressure / NominalShaftPower, Math.Min(1, RotationRatio));
11   else return 1.0;
12 }
13
14 /// <summary>総合効率[-]を取得する</summary>
15 public virtual double GetTotalEfficiency()
16 { return MotorEfficiency * GetFluidMachineryEfficiency() * GetInverterEfficiency(); }
17
18 /// <summary>消費電力[kW]を取得する</summary>
19 public virtual double GetElectricConsumption()
20 { return VolumetricFlowRate * Pressure / GetTotalEfficiency(); }
21
22 /// <summary>体積流量と圧力から回転数比を計算する</summary>
23 /// <param name="flowRate">体積流量[m3/s]</param>
24 /// <param name="pressure">圧力[kPa]</param>
25 internal void updateWithFlowRateAndPressure(double flowRate, double pressure)
26 {
27   VolumetricFlowRate = flowRate;
28   Pressure = pressure;
29   RotationRatio = GetRotationRatio(flowRate, pressure, pressureCoefficient);
30 }
31
32 /// <summary>設定圧力と PQ 特性が交わる体積流量を計算する</summary>
33 /// <param name="rotationRatio">回転数比[-]</param>
34 /// <param name="pressure">設定圧力[kPa]</param>
35 internal void updateWithRotationRatioAndPressure(double rotationRatio, double pressure)
36 {
37   RotationRatio = rotationRatio;
38   Pressure = pressure;
39   VolumetricFlowRate = GetFlowRate(rotationRatio, pressureCoefficient, pressure, DesignFlowRate);
40 }
41
42 /// <summary>抵抗曲線と PQ 特性が交わる体積流量を計算する</summary>
43 /// <param name="rotationRatio">回転数比[-]</param>
44 /// <param name="resistanceCoefficient">抵抗係数[kPa/(m3/s)^2]</param>
45 /// <param name="actualHead">実揚程[kPa]</param>
46 /// <param name="operatingNumber">運転台数</param>
47 /// <param name="totalFlowRate">総流量[m3/s]</param>
48 internal void updateWithResistanceAndRotationRatio
49   (double rotationRatio, double resistanceCoefficient,
50   double actualHead, int operatingNumber, out double totalFlowRate)
51 {
52   RotationRatio = rotationRatio;
53   totalFlowRate = GetFlowRate(RotationRatio, pressureCoefficient, resistanceCoefficient,
54     actualHead, operatingNumber, DesignFlowRate * operatingNumber);
55   VolumetricFlowRate = totalFlowRate / operatingNumber;
56   Pressure = resistanceCoefficient * totalFlowRate * totalFlowRate + actualHead;
```

```
57 }
58
59 /// <summary>抵抗曲線と PQ 特性が交わる体積流量を計算する</summary>
60 /// <param name="rotationRatio">回転数比[-]</param>
61 internal void updateWithResistanceAndRotationRatio(double rotationRatio)
62 {
63   RotationRatio = rotationRatio;
64   VolumetricFlowRate = GetFlowRate(RotationRatio, pressureCoefficient,
65     resistanceCoefficient, ActualHead, 1, DesignFlowRate);
66   Pressure = resistanceCoefficient * Math.Pow(VolumetricFlowRate, 2) + ActualHead;
67 }
68
69 /// <summary>体積流量と回転数比から圧力を計算する</summary>
70 /// <param name="flowRate">体積流量[m3/s]</param>
71 /// <param name="rotationRatio">回転数比[-]</param>
72 internal void updateWithFlowRateAndRotationRatio(double flowRate, double rotationRatio)
73 {
74   VolumetricFlowRate = flowRate;
75   RotationRatio = rotationRatio;
76   Pressure = GetPressure(flowRate, rotationRatio, pressureCoefficient);
77 }
78
79 /// <summary>停止させる</summary>
80 public virtual void ShutOff()
81 {
82   VolumetricFlowRate = 0;
83   Pressure = 0;
84   RotationRatio = 0;
85 }
```

16.3.2　ポンプの計算

1）「遠心ポンプクラス（単体）」の作成

流体機械クラスの派生クラスとして遠心ポンプクラスを作成する。本書のモデルは、バイパスと INV による吐出圧一定制御ならびに最小吐出圧制御に対応したモデルとする。プログラム 16.5 にポンプの流量制御方式を表わす列挙型の定義を示す。

プログラム 16.5　ポンプ流量制御方式を表わす列挙型の定義

Popolo.HVAC.Circuit.CentrifugalPump class

```
1 /// <summary>流量制御方式</summary>
2 public enum ControlMethod
3 {
4   /// <summary>吐出圧一定制御（バイパス）</summary>
5   ConstantPressureWithBypass,
6   /// <summary>吐出圧一定制御（INV）</summary>
7   ConstantPressureWithInverter,
8   /// <summary>最小吐出圧制御</summary>
9   MinimumPressure
10 }
```

プログラム 16.6 にプロパティを示す。ポンプクラスは流体機械クラスの派生クラスとするため、計算に必要となる多くのプロパティは既に定義済である。

プログラム 16.6　ポンプ流量制御方式を示す列挙型の定義

Popolo.HVAC.Circuit.CentrifugalPump class

```
1 /// <summary>バイパス量[m3/s]を取得する</summary>
2 public double BypassFlowRate { get; private set; }
3
4 /// <summary>制御方式を取得する</summary>
5 public ControlMethod Control { get; private set; }
6
7 /// <summary>圧力設定値[kPa]を設定・取得する</summary>
8 public double PressureSetpoint { get; set; }
```

プログラム 16.7 に初期化処理を示す。1~21 行はコンストラクタであり、ポンプ効率特性と PQ 特性の特性係数の計算が主たる処理である。それぞれ式 16.11 と式 16.12 にもとづき、23~44 行で定義したメソッドで計算する。制御のために必要な圧力設定値の初期値は設計圧力を用いる（20 行）。

プログラム 16.7　ポンプの PQ 特性およびポンプ効率の計算

Popolo.HVAC.Circuit.CentrifugalPump class

```
1 /// <summary>インスタンスを初期化する</summary>
```

```
 2 /// <param name="nomPressure">定格揚程[kPa]</param>
 3 /// <param name="nomFlowRate">定格水量[m3/s]</param>
 4 /// <param name="designPressure">設計揚程[kPa]</param>
 5 /// <param name="designFlowRate">設計水量[m3/s]</param>
 6 /// <param name="control">制御方式</param>
 7 /// <param name="actualHead">実揚程[kPa]</param>
 8 public CentrifugalPump(double nomPressure, double nomFlowRate, double designPressure,
 9   double designFlowRate, ControlMethod control, double actualHead) :
10    base(nomPressure, nomFlowRate, designPressure, designFlowRate, actualHead,
11      control != ControlMethod.ConstantPressureWithBypass)
12 {
13   //制御方式を保存
14   Control = control;
15
16   //効率・圧力特性係数を計算
17   efficiencyCoefficient = new double[3];
18   pressureCoefficient = new double[3];
19   getGeneralParameters(nomFlowRate, nomPressure, ref efficiencyCoefficient, ref pressureCoefficient);
20   PressureSetpoint = designPressure;
21 }
22
23 /// <summary>汎用ポンプの特性係数を計算する</summary>
24 /// <param name="flowRate">設計流量[m3/s]</param>
25 /// <param name="pressure">設計揚程[kPa]</param>
26 /// <param name="efficiencyCoef">出力：効率特性係数</param>
27 /// <param name="pressureCoef">出力：揚程特性係数</param>
28 private static void getGeneralParameters(double flowRate, double pressure,
29   ref double[] efficiencyCoef, ref double[] pressureCoef)
30 {
31   double vf2 = flowRate * flowRate;
32
33   //効率特性
34   double lnm = Math.Log(flowRate);
35   double maxEff = -0.01618 * lnm * lnm - 0.05662 * lnm + 0.78179;
36   efficiencyCoef[0] = -0.929 * maxEff / vf2;
37   efficiencyCoef[1] = 1.795 * maxEff / flowRate;
38   efficiencyCoef[2] = 0.109 * maxEff;
39
40   //圧力特性
41   pressureCoef[0] = -0.395 * pressure / vf2;
42   pressureCoef[1] = 0.096 * pressure / flowRate;
43   pressureCoef[2] = 1.294 * pressure;
44 }
```

プログラム 16.8 にポンプの状態更新処理を示す。引数は体積流量であり、必要流量が 0 以下の場合（5~11 行）には 44~49 行のメソッドで機器を停止させる。

13~18 行はバイパスによる吐出圧一定制御の場合の処理である。この場合には各ポンプは回転数比 100%で運転するため、単純に設計揚程と PQ 特性との交点を求めれば、それがポンプの運転点となる。設計揚程と PQ 特性の交点はプログラム 16.2 を用いて計算する。

19~29 行は INV による吐出圧一定制御の場合の処理である。この場合も圧力は設計圧力で一定である。しかし、バイパス制御の場合とは異なり回転数比を絞ることによってバイパス流量が 0 になるように制御する。この運転点を求めるためには、揚程と水量から回転数比を計算する処理が必要となる。プログラム 16.2 に示した getRotationRatio メソッドを用いる。ただし、最小回転数未満となるときはバイパス制御が働くため、24 行でこの条件分岐を行う。この場合には最小回転数での PQ 特性と設計圧力との交点を計算する。

30~41 行は最小吐出圧制御の場合の処理である。この場合には抵抗曲線と PQ 特性の交点で運転点が定まる。INV による吐出圧一定制御の場合と同じように回転数比を計算する必要があり、プログラム 16.2 の getRotationRatio メソッドを用いて反復計算で求める。

プログラム 16.8　ポンプの状態更新処理

Popolo.HVAC.Circuit.CentrifugalPump class

```
 1 /// <summary>状態を更新する</summary>
 2 /// <param name="flowRate">体積流量[m3/s]</param>
```

```
 3 public void UpdateState(double flowRate)
 4 {
 5   //停止判定
 6   VolumetricFlowRate = flowRate;
 7   if (VolumetricFlowRate <= 0)
 8   {
 9     ShutOff();
10     return;
11   }
12
13   //バイパスによる吐出圧一定制御の場合：最大回転数・設計圧力で状態更新
14   if (Control == ControlMethod.ConstantPressureWithBypass)
15   {
16     updateWithRotationRatioAndPressure(1.0, PressureSetpoint);
17     BypassFlowRate = VolumetricFlowRate - flowRate;
18   }
19   //INVによる吐出圧一定制御の場合：必要流量・圧力を満たすように回転数を調整
20   else if (Control == ControlMethod.ConstantPressureWithInverter)
21   {
22     updateWithFlowRateAndPressure(flowRate, PressureSetpoint);
23     //最小回転数比[-]未満の場合は最小回転数にしてバイパス流量を計算
24     if (RotationRatio < MinimumRotationRatio)
25     {
26       updateWithRotationRatioAndPressure(MinimumRotationRatio, PressureSetpoint);
27       BypassFlowRate = VolumetricFlowRate - flowRate;
28     }
29   }
30   //INVによる最小吐出圧制御の場合：抵抗曲線に合う圧力と流量で計算
31   else
32   {
33     double ps = flowRate * flowRate * resistanceCoefficient + ActualHead;
34     updateWithFlowRateAndPressure(flowRate, ps);
35     //最小回転数比[-]未満の場合は最小回転数にしてバイパス流量を計算
36     if (RotationRatio < MinimumRotationRatio)
37     {
38       updateWithResistanceAndRotationRatio(MinimumRotationRatio);
39       BypassFlowRate = VolumetricFlowRate - flowRate;
40     }
41   }
42 }
43
44 /// <summary>停止させる</summary>
45 public override void ShutOff()
46 {
47   base.ShutOff();
48   BypassFlowRate = 0;
49 }
```

2）「ポンプシステムクラス」の作成

多くの設備システムでは省エネルギー性や制御性の向上を目的に、ポンプを並列に接続させて複数台数での運転を行う。そこでポンプの台数運転に対応させたポンプシステムのクラスを作成する。プログラム 16.9 にポンプシステムクラスのインスタンスおよびプロパティを示す。単体のポンプクラスとほぼ同様のプロパティを持つが、17 行と 20 行で台数に関する情報を保持する点が特徴である。また、コンストラクタで設計流量と設計圧力を指定するが、複数台のポンプを同時稼働させた流量である点に注意する必要がある。

プログラム 16.9　ポンプシステムクラスのインスタンス変数、プロパティ、コンストラクタの定義

Popolo.HVAC.Circuit.PumpSystem class

```
 1 /// <summary>遠心ポンプ</summary>
 2 private CentrifugalPump pump;
 3
 4 /// <summary>遠心ポンプを取得する</summary>
 5 public ImmutableCentrifugalPump Pump { get { return pump; } }
 6
 7 /// <summary>抵抗係数[kPa/(m3/s)^2]</summary>
 8 private double resistanceCoefficient;
 9
10 /// <summary>総流量[m3/s]を設定・取得する</summary>
11 public double TotalFlowRate { get; set; }
12
13 /// <summary>バイパス量[m3/s]を取得する</summary>
14 public double BypassFlowRate { get; private set; }
15
```

```
16 /// <summary>運転台数[台]を取得する</summary>
17 public int OperatingNumber { get; private set; }
18
19 /// <summary>ポンプ台数[台]を取得する</summary>
20 public int PumpNumber { get; private set; }
21
22 /// <summary>実揚程[kPa]を取得する</summary>
23 public double ActualHead { get; private set; }
24
25 /// <summary>圧力設定値[kPa]を設定・取得する</summary>
26 public double PressureSetpoint
27 {
28   get { return pump.PressureSetpoint; }
29   set { pump.PressureSetpoint = value; }
30 }
31
32 /// <summary>インスタンスを初期化する</summary>
33 /// <param name="pump">遠心ポンプ</param>
34 /// <param name="designPressure">設計圧力[kPa]</param>
35 /// <param name="designFlowRate">設計流量[m3/s]</param>
36 /// <param name="actualHead">実揚程[kPa]</param>
37 /// <param name="pumpNumber">ポンプの台数</param>
38 public PumpSystem
39   (CentrifugalPump pump, double designPressure, double designFlowRate, double actualHead, int pumpNumber)
40 {
41   this.pump = pump;
42   PumpNumber = pumpNumber;
43   PressureSetpoint = designPressure;
44   ActualHead = actualHead;
45
46   //抵抗係数[kPa/(m3/s)^2]を計算
47   resistanceCoefficient = (designPressure - actualHead) / (designFlowRate * designFlowRate);
48 }
```

プログラム 16.10 にポンプの必要台数の計算処理を示す。ポンプシステムに必要とされる水量を入力として与え、必要運転台数を出力する。

8~27 行は最小吐出圧制御の場合である。14, 15 行でそもそも締切揚程が必要揚程を上回ることを確かめた後、17~25 行で 1 台ずつ運転台数を増やしていき、必要揚程を上回ったところで計算を打ち切って台数を確定する。

28~33 行は吐出圧一定制御の場合である。必要揚程での 1 台あたりの吐出量を計算し、必要水量を除することで台数を決定する。

プログラム 16.10　ポンプの必要台数の計算

Popolo.HVAC.Circuit.PumpSystem class

```
 1 /// <summary>ポンプの必要運転台数[台]を計算する</summary>
 2 /// <param name="flowRate">必要水量[m3/s]</param>
 3 /// <returns>必要運転台数[台]</returns>
 4 private int getOperatingNumber(double flowRate)
 5 {
 6   if (flowRate <= 0) return 0;
 7
 8   //最小吐出圧制御の場合：1台ずつ増やして確認
 9   if (Pump.Control == CentrifugalPump.ControlMethod.MinimumPressure)
10   {
11     double r2 = flowRate * flowRate;
12     double ps = resistanceCoefficient * r2 + ActualHead;
13     //締切揚程が必要揚程を上回ることの確認
14     pump.updateWithFlowRateAndRotationRatio(0, 1.0);
15     if (pump.Pressure < ps) throw new Exception("Pump Pressure Error");
16     int opNum = 1;
17     while (true)
18     {
19       //最大回転数比で水量が足りるか確認
20       double tf;
21       pump.updateWithResistanceAndRotationRatio(1.0, resistanceCoefficient, ActualHead, opNum, out tf);
22       if (flowRate < tf) break;
23       else opNum++;
24       if (50 < opNum) throw new Exception("Pump Number Error");
25     }
26     return opNum;
27   }
28   //吐出圧一定制御・バイパス制御の場合には台数で流量均等分割
29   else
```

```
30   {
31     pump.updateWithRotationRatioAndPressure(1.0, PressureSetpoint);
32     return (int)Math.Ceiling(flowRate / Pump.VolumetricFlowRate);
33   }
34 }
```

プログラム 16.11 にポンプシステムの状態更新処理を示す。

5 行はプログラム 16.10 で示した台数計算であり、0 台の場合には 43~50 行の ShutOff メソッドで機器を停止させる。また、必要な運転台数が最大台数を上回る過負荷状態の場合には、最大台数と最大回転数での流量を計算する（11~19 行）。過負荷とならない場合には 22 行でポンプ 1 台あたりの必要流量を計算し、23~37 行で制御方式別にポンプの状態を更新する。ポンプ単体の処理と同様である。運転台数と運転点が確定すれば 52~54 行のメソッドでシステムの消費電力を計算できる。

プログラム 16.11　ポンプシステムの状態更新処理

Popolo.HVAC.Circuit.PumpSystem class

```
 1 /// <summary>状態を更新する</summary>
 2 public void UpdateState()
 3 {
 4   //運転台数を確定
 5   OperatingNumber = getOperatingNumber(TotalFlowRate);
 6   if (OperatingNumber == 0)
 7   {
 8     ShutOff();
 9     return;
10   }
11   //過負荷の場合
12   if (PumpNumber < OperatingNumber)
13   {
14     OperatingNumber = PumpNumber;
15     double tf;
16     pump.updateWithResistanceAndRotationRatio(1.0, resistanceCoefficient, ActualHead, PumpNumber, out tf);
17     TotalFlowRate = tf;
18     BypassFlowRate = 0;
19   }
20   else
21   {
22     double vFlow = TotalFlowRate / OperatingNumber;
23     //最小吐出圧制御の場合：
24     if (Pump.Control == CentrifugalPump.ControlMethod.MinimumPressure)
25     {
26       double pressure = TotalFlowRate * TotalFlowRate * resistanceCoefficient + ActualHead;
27       pump.updateWithFlowRateAndPressure(vFlow, pressure);
28       //最小回転数比[-]未満の場合
29       if (Pump.RotationRatio < Pump.MinimumRotationRatio)
30       {
31         double tf;
32         pump.updateWithResistanceAndRotationRatio
33           (Pump.MinimumRotationRatio, resistanceCoefficient, ActualHead, OperatingNumber, out tf);
34       }
35     }
36     //吐出圧一定制御・バイパス制御の場合
37     else pump.UpdateState(vFlow);
38
39     BypassFlowRate = (Pump.VolumetricFlowRate * OperatingNumber) - TotalFlowRate;
40   }
41 }
42
43 /// <summary>停止させる</summary>
44 public void ShutOff()
45 {
46   pump.ShutOff();
47   BypassFlowRate = 0;
48   TotalFlowRate = 0;
49   OperatingNumber = 0;
50 }
51
52 /// <summary>消費電力[kW]を取得する</summary>
53 public double GetElectricConsumption()
54 { return Pump.GetElectricConsumption() * OperatingNumber; }
```

【例題 16.4】

例題 16.2 のポンプを 3 台並列接続したシステムの、水量・消費電力・総合効率の関係を計算せよ。

【解】

プログラム 16.12 に計算処理を示す。3~8 行で 3 種類の制御方式のポンプインスタンスを生成する。定格の流量と揚程、設計流量と揚程はいずれも同じである。設計流量である 5,400 L/min から 25 L/min ずつ減らしていき 20~24 行で状態を更新し、26~30 行で書き出し処理を行う。計算結果を図 16.8 に示す。バイパス制御の場合の効率は一定のため省略した。3 種の制御方式の中では最小吐出圧制御が最も消費電力を低減できることがわかる。ただし最小吐出圧制御の場合、運転台数減段時に消費電力が増加する。これは減段時には揚程が低く吐出量が大きい運転点に移りポンプ効率が低下するためであり、グラフからもその様子が読み取れる。この問題への対処としては、先に記したように極低負荷に対応するための小流量ポンプを導入するという方法がある。

プログラム 16.12　吐出流量[L/min]とポンプ消費電力[kW]・効率[-]の関係の計算（制御方式別）

```
 1 private static void CentrifugalPumpTest()
 2 {
 3   CentrifugalPump p1 = new CentrifugalPump
 4     (260, 0.03, 250, 0.03, CentrifugalPump.ControlMethod.ConstantPressureWithBypass, 40);
 5   CentrifugalPump p2 = new CentrifugalPump
 6     (260, 0.03, 250, 0.03, CentrifugalPump.ControlMethod.ConstantPressureWithInverter, 40);
 7   CentrifugalPump p3 = new CentrifugalPump
 8     (260, 0.03, 250, 0.03, CentrifugalPump.ControlMethod.MinimumPressure, 40);
 9   PumpSystem[] pss = new PumpSystem[3];
10   pss[0] = new PumpSystem(p1, 250, 0.09, 40, 3);
11   pss[1] = new PumpSystem(p2, 250, 0.09, 40, 3);
12   pss[2] = new PumpSystem(p3, 250, 0.09, 40, 3);
13
14   using (StreamWriter sWriter =
15   new StreamWriter("CentrifugalPumpTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
16   {
17     for (int i = 0; i < 216; i++)
18     {
19       double wf = (5400 - 25 * i) / 60d / 1000d;
20       for (int j = 0; j < pss.Length; j++)
21       {
22         pss[j].TotalFlowRate = wf;
23         pss[j].UpdateState();
24       }
25
26       sWriter.Write(wf * 60d * 1000);
27       for (int j = 0; j < pss.Length; j++) sWriter.Write(", " + pss[j].GetElectricConsumption());
28       for (int j = 0; j < pss.Length; j++) sWriter.Write(", " + pss[j].Pump.GetTotalEfficiency());
29       for (int j = 0; j < pss.Length; j++) sWriter.Write(", " + pss[j].BypassFlowRate * (60 * 1000));
30       sWriter.WriteLine();
31     }
32   }
33 }
```

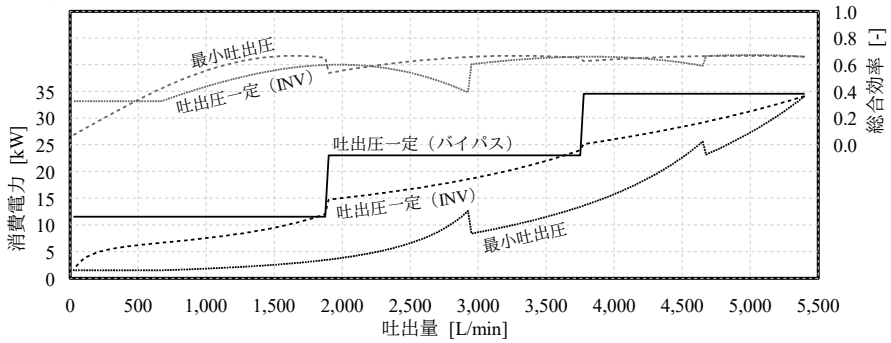


図 16.8　吐出流量[L/min]とポンプ消費電力[kW]・効率[-]の関係（制御方式別）

16.3.3　送風機の計算

ポンプと同様に流体機械クラスの派生クラスとする。

プログラム 16.13 に遠心ファンクラスの初期化処理を示す。定格点と設計点での風量と全圧、インバータの有無、番手などを与えて初期化を行う。効率特性と PQ 特性は式 16.11 と式 16.12 に従い、32~66 行のメソッドを用いて 13~16 行で初期化する。制気口まわりで動圧が必要になるが、通常は 20 Pa 程度あれば十分であるため、10 行に示すように標準では 0.02 kPa で初期化する。細かく設定

が必要な場合には19~30行のオーバーロードされたコンストラクタを用いる。

プログラム 16.13　遠心ファンクラスの初期化処理

Popolo.HVAC.Circuit.CentrifugalFan class

```
/// <summary>インスタンスを初期化する</summary>
/// <param name="nomPressure">定格全圧[kPa]</param>
/// <param name="nomFlowRate">定格風量[m3/s]</param>
/// <param name="designPressure">設計全圧[kPa]</param>
/// <param name="designFlowRate">設計風量[m3/s]</param>
/// <param name="number">番手[-]</param>
/// <param name="hasInverter">インバータ機か否か</param>
public CentrifugalFan(double nomPressure, double nomFlowRate, double designPressure,
  double designFlowRate, double number, bool hasInverter) :
  base(nomPressure, nomFlowRate, designPressure, designFlowRate, 0.02, hasInverter)
{
  //効率・圧力特性係数を計算
  efficiencyCoefficient = new double[4];
  pressureCoefficient = new double[3];
  getGeneralParameters
    (nomFlowRate, nomPressure, number, ref efficiencyCoefficient, ref pressureCoefficient);
}

/// <summary>インスタンスを初期化する</summary>
/// <param name="nomPressure">定格全圧[kPa]</param>
/// <param name="nomFlowRate">定格風量[m3/s]</param>
/// <param name="designPressure">設計全圧[kPa]</param>
/// <param name="designFlowRate">設計風量[m3/s]</param>
/// <param name="number">番手[-]</param>
/// <param name="dynamicPressure">設計動圧[kPa]</param>
/// <param name="hasInverter">インバータ機か否か</param>
public CentrifugalFan(double nomPressure, double nomFlowRate, double designPressure,
  double designFlowRate, double number, double dynamicPressure, bool hasInverter) :
  this(nomPressure, nomFlowRate, designPressure, designFlowRate, dynamicPressure, hasInverter)
{ }

/// <summary>汎用ファンの特性係数を計算する</summary>
/// <param name="flowRate">風量[m3/s]</param>
/// <param name="pressure">全圧[kPa]</param>
/// <param name="number">ファンの番手</param>
/// <param name="efficiencyCoef">出力：効率特性係数</param>
/// <param name="pressureCoef">出力：揚程特性係数</param>
private static void getGeneralParameters(double flowRate, double pressure,
  double number, ref double[] efficiencyCoef, ref double[] pressureCoef)
{
  //最大効率近似係数
  double[][] aj = new double[3][];
  aj[0] = new double[] { 1.397E-01, 1.303E-01, -1.179E-02, 3.219E-04 };
  aj[1] = new double[] { 7.649E-01, -9.105E-02, -5.345E-03, 1.006E-03 };
  aj[2] = new double[] { -7.141E-01, 1.325E-01, -2.639E-03, -5.690E-04 };

  //最大効率[-]の計算
  double[] act = new double[3];
  for (int i = 0; i < aj.Length; i++)
    for (int j = aj[0].Length - 1; 0 <= j; j--)
      act[i] = (number * act[i] + aj[i][j]);
  double maxEff = act[0] + pressure * (act[1] + act[2] * pressure);

  //効率特性
  double m2 = flowRate * flowRate;
  double m3 = m2 * flowRate;
  efficiencyCoef[0] = 0.337 * maxEff / m3;
  efficiencyCoef[1] = -1.700 * maxEff / m2;
  efficiencyCoef[2] = 2.350 * maxEff / flowRate;
  efficiencyCoef[3] = 0;

  //圧力特性
  pressureCoef[0] = -0.300 * pressure / m2;
  pressureCoef[1] = 0.517 * pressure / flowRate;
  pressureCoef[2] = 0.767 * pressure;
}
```

プログラム 16.14 に送風機の状態更新処理を示す。風量が 0 以下の場合にはプログラム 16.4 で示した抽象クラスの ShutOff メソッドを呼び出して機器を停止させる（10 行）。

17~32 行は回転数制御を行う場合であり、プログラム 16.2 を用いて圧力と流量から必要な回転数を計算し、最大回転数を上回る場合には最大回転数での成行計算（21~28 行）、最小回転数未満の場合

には最小回転数でのダンパ制御（29~31 行）に移る。

17~32 行は回転数制御を行わない場合である。PQ 特性上の全圧と抵抗曲線上の全圧を求めて比較し、前者が後者を上回っている場合にはダンパ制御によって要求風量を実現可能とする。過負荷で要求風量を満足できない場合には PQ 特性と抵抗曲線の交点を計算する（42 行）。

プログラム 16.14　遠心ファンの運転点の計算

Popolo.HVAC.Circuit.CentrifugalFan class

```
 1 /// <summary>状態を更新する</summary>
 2 /// <param name="flowRate">体積流量[m3/s]</param>
 3 public void UpdateState(double flowRate)
 4 {
 5   VolumetricFlowRate = flowRate;
 6
 7   //停止判定
 8   if (VolumetricFlowRate <= 0)
 9   {
10     ShutOff();
11     return;
12   }
13
14   //抵抗曲線上の圧力
15   double ps = VolumetricFlowRate * VolumetricFlowRate * resistanceCoefficient + ActualHead;
16
17   //インバータ有り
18   if (HasInverter)
19   {
20     updateWithFlowRateAndPressure(VolumetricFlowRate, ps);
21     //回転数上限の場合には最大回転数で成り行き計算
22     if (1.0 < RotationRatio)
23     {
24       RotationRatio = 1.0;
25       VolumetricFlowRate =
26         GetFlowRate(RotationRatio, pressureCoefficient, resistanceCoefficient, ActualHead, 1, DesignFlowRate);
27       Pressure = VolumetricFlowRate * VolumetricFlowRate * resistanceCoefficient + ActualHead;
28     }
29     //回転数下限の場合には最小回転数でダンパ制御
30     else if (RotationRatio < MinimumRotationRatio)
31       updateWithFlowRateAndRotationRatio(VolumetricFlowRate, MinimumRotationRatio);
32   }
33   //インバータ無し
34   else
35   {
36     //過負荷判定
37     updateWithFlowRateAndRotationRatio(VolumetricFlowRate, 1.0);
38     if (Pressure < ps)
39     {
40       RotationRatio = 1.0;
41       VolumetricFlowRate =
42         GetFlowRate(1.0, pressureCoefficient, resistanceCoefficient, ActualHead, 1, DesignFlowRate);
43       Pressure = VolumetricFlowRate * VolumetricFlowRate * resistanceCoefficient + ActualHead;
44     }
45   }
46 }
```

【例題 16.5】

例題 16.3 のファンについて風量と消費電力および総合効率の関係を計算せよ。

【解】

プログラム 16.15 に計算処理を示す。5, 6 行で定速機と INV 機のファンを作成する。14, 15 行で状態を更新し、16 行で計算結果を書き出す。　図 16.9 に計算結果のグラフを示す。総合効率自体は INV 効率の影響により定速機の方が高いが、そもそもの圧力は INV 機の方が小さく、理論動力も小さいため、消費電力に関しては INV 機が定速機を下回ることがわかる。なお、計算上は 0 CMH まで連続的に風量を絞ることができるとしているが、現実にはある点でサージング領域に入る。この場合には温度制御などで発停状態となるが、時間平均のシミュレーションを目的とする場合には、連続的に風量を低下できるとしても誤差は小さいと推測する。

プログラム 16.15　風量と消費電力・総合効率の関係

```
 1 private static void CentrifugalFanTest()
 2 {
 3   double nf = 13500d / 3600;
 4   double df = 13000d / 3600;
```

```
 5   CentrifugalFan f1 = new CentrifugalFan(0.55, nf, 0.53, df, 4, false);
 6   CentrifugalFan f2 = new CentrifugalFan(0.55, nf, 0.53, df, 4, true);
 7
 8   using (StreamWriter sWriter =
 9   new StreamWriter("CentrifugalFanTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
10   {
11 for (int i = 0; i < 100; i++)
12 {
13   double vf = (13000 - 130 * i) / 3600d;
14   f1.UpdateState(vf);
15   f2.UpdateState(vf);
16   sWriter.WriteLine(vf * 3600d +
17     ", " + f1.GetElectricConsumption() + ", " + f2.GetElectricConsumption() +
18     ", " + f1.GetTotalEfficiency() + ", " + f2.GetTotalEfficiency());
19     }
20   }
21 }
```

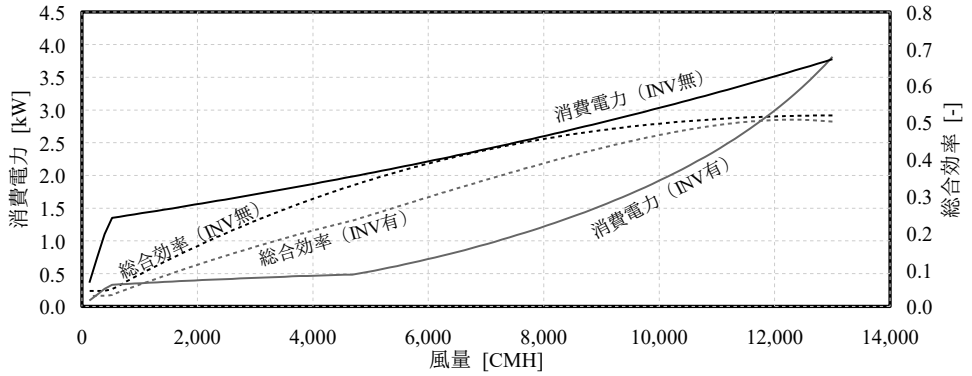


図 16.9　INV 制御の有無による風量と消費電力・総合効率の関係

【第 16 章 記号表】

ct	：番手 **[番]**	W_M	：モーターの消費電力 **[kW]**
g	：重力加速度 **[m²/s]**	W_T	：流体機械の理論動力 **[kW = kN·(m/s)]**
L_P	：負荷率 **[-]**	α	：特性式の係数
m	：質量流量 **[kg/s]**	ρ	：比重量 **[kg/m³]**
N	：モーターの回転数 **[rps]**	η_F	：流体機械の効率 **[-]**
P_F	：揚程・静圧 **[kPa]**	η_M	：モーター効率 **[-]**
Q	：体積流量 **[m³/s]**	η_{Inv}	：インバータ効率 **[-]**
R_N	：モーターの回転数比 **[-]**		
sub scripts			
D	：設計条件	N	：定格条件
M	：最大効率点		

【第 16 章 参考文献】

16.1） 津島孝雄, 森川敬信: 配管網の定常流れ解析 第 1 報 T 形分流基本回路の流れ解析, 空気調和・衛生工学会論文集 No.28, pp.21-31, 1985

16.2） 坂東修: Excel で解く配管とポンプの流れ, 株式会社工業調査会, 2008

16.3） 日本工業規格 JIS B8301 遠心ポンプ, 斜流ポンプ及び軸流ポンプ -試験方法, 2000

16.4） 日本工業規格 JIS B8313 小型渦巻ポンプ, 2003

16.5） 三毛正仁, 中尾正喜, 西岡真稔, 鍋島美奈子: 渦巻ポンプの特性近似式の提案, 空気調和・衛生工学会論文集, Vol.126, pp.19-26, 2007

16.6） 何原一平, 丹羽英治, 田中英紀, 富樫英介, 佐々木裕文, 宮島裕二, 伊藤誠恭, 杉原義文, 渡邊剛: ライフサイクルエネルギーマネージメントのための空調システムシミュレーション開発 : (第 22 報)LCEM ツール Ver.3 のオブジェクト開発・整備, 空気調和・衛生工学会 学術講演会論文集, pp.2275-2278, 2009

16.7） 竹谷是幸, 八本輝: インバータと高調波(3) 空調・衛生分野におけるインバータ利用機器とその応用 , 空気調和・衛生工学 Vol.75, No.2, pp.109-115, 2001

16.8） 黒田尚紀: インバータ制御による消費電力, 空気調和・衛生工学会大会学術講演論文集, pp1309-1312, 2005

16.9） 建築設備設計基準 平成 18 年版, 第 2 章 空調機器, 国土交通省大臣官房官庁営繕部設備・環境課監修, 社団法人 公共建築協会, pp.339 図 2-32, 2006

第17章　回路網
(Circuit Network)

17.1　概要

第 16 章では流路の抵抗特性が流体機械の吐出体積流量の二次式で表現できるとして計算を行った。しかし現実の流路は複雑であり、厳密にはこのような仮定は成立しない。例えば空調機が 2 台ある単純なシステムを想定する。片方の空調機の水量のみが 0 となった場合と両方の空調機の水量が 50%となった場合では、いずれも全体としての流量は半分となり同じであるが、回路全体での圧力損失は異なる。前者の場合には 100 %負荷が維持される流路で大きな圧力損失が生じるため、後者に比較して必要なエネルギーが大きい。このような特性の違いを表現するためには二次式によるモデル化は不十分であり、複雑な配管路の中のどこにどれだけの水が流れ、また、どの部位でどの程度の圧力損失が生じるのかを推定しなければならない。このために必要となる計算が、回路網の計算である。設備システムの年間エネルギーシミュレーションにおいてこのレベルの計算が行われることは稀であるが、部位別の動的シミュレーションや制御系のシミュレーションへの発展を見据えた場合には必須の計算技術である。本章では回路を構成する配管、ダクト、ダンパ、バルブなどの計算法を記すとともに、これらの要素によって構成された回路網を解く方法について解説する。

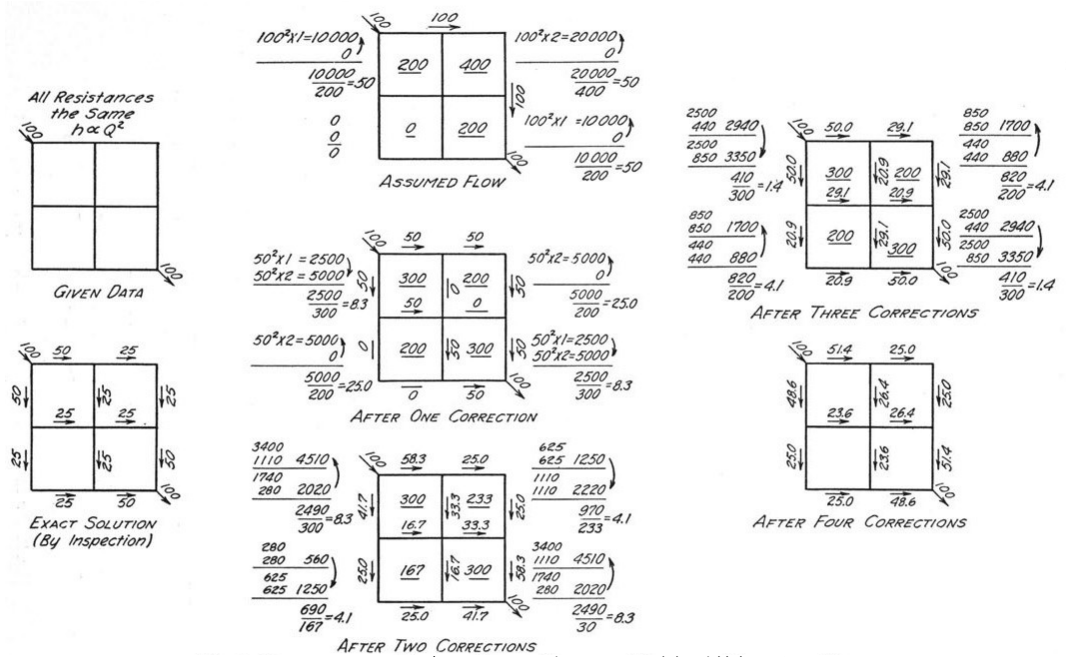


図 17.1　ハーディ・クロス法による回路網計算例[17.1)]

（手計算でかなりの規模の問題が解けたるため、コンピュータ登場前の主流な解法であった）

17.2 理論

17.2.1 回路網（流路と節点）

建築設備システムではダクトや配管などの中に空気や水を流すことで熱を搬送する。このような搬送経路を一般に「流路」と呼ぶ。多くの場合、流路は一筆書きができるような単純な形をしておらず、合流や分岐を繰り返して複雑な網状になっている。このような網状の複雑な流路を「回路網」と呼ぶ[†1)]。

図 17.2 に回路網の例を示す。左は外部への流出入が無い閉回路の例、右は外部への流出入がある開回路の例である。閉回路は主には空調配管設備[†2)]、開回路は主には給排水設備で問題となる。N_x と記載した点は、ある圧力を持った点であり、節点（Node）と呼ぶ。CH_x は流路（CHannel）であり、節点と節点をつなぐように配置され、両端の節点の圧力差に依存して内部を流れる流体の流量が唯一つに定まる。この意味で言えば第 16 章で解説した流体機械も流路の一つであり、図 17.2 の CH_{01} がこれである。

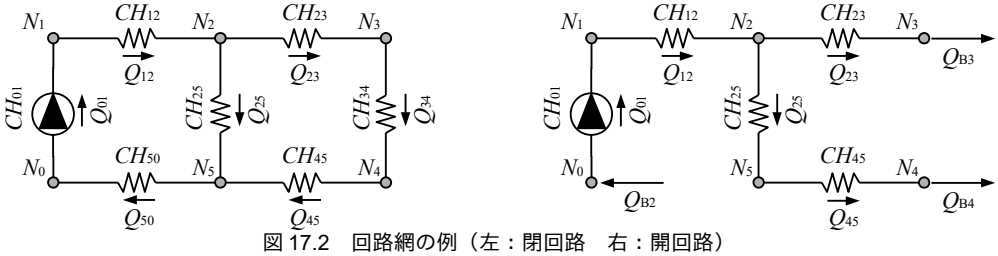


図 17.2 回路網の例（左：閉回路　右：開回路）

節点には複数の流路が接続されうるが、質量保存則により各節点から流入する、あるいは流出する流量の積算値は 0 となり、式 17.1 を満たす必要がある。ここで $Q_{m,k}$ [m³/s]は m 番の節点に接続された k 番目の流路の流量である。

$$0=\sum_{k=0}^{K} Q_{m,k} \tag{17.1}$$

「回路網を解く」とは上記の質量保存則を満たす節点圧力と流路流量の組み合わせを計算することである。ただし、設備システムシミュレーションにおいては流体機械による投入エネルギー量を固定して成り行き状態の計算を行うだけでは不十分であり、逆に、特定の流路の流量や節点の圧力をある値に制御するために必要となる流体機械の投入エネルギー量を推定する必要がある。

17.2.2 流路の基礎式

1）流動抵抗係数

回路網を解くためには各流路でどれだけの圧力損失が生じるかを予測する必要がある。一般に流路で発生する圧力損失 ΔP [kPa]は、式 17.2 に示すように流体の体積流量 Q の 2 乗に比例する。比例定数 K [kPa/(m³/s)²]を流動抵抗係数と呼ぶ。

$$\Delta P = K Q^2 \tag{17.2}$$

図 17.3 に示すような単純な並列回路と直列回路であれば式 17.3 および式 17.4 を用いて合成した流動抵抗係数 K_{12} を計算することができる。後述するように複雑な回路網は数値計算的に解くしか方法がないが、事前にできるだけ合成抵抗に置き換えて変数の数を減らした方が計算が安定し、また収束

†1 本書では水と空気の流動について解説するが、回路網という考え方は電流、熱流、物質移動などの問題にも適用できる。回路網全般のモデル化と解法については奥山が詳細に報告しており[17.8)]、プログラム（NETS）を公開している。

†2 ただし、水蓄熱槽や開放式冷却塔などで配管網を開回路として解く場合もある。

に要する時間も短くなる。

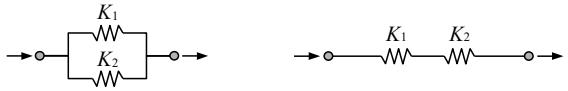


図 17.3　直列回路と並列回路

$$K_{12}=1/\left(\sqrt{1/K_1}+\sqrt{1/K_2}\right)^2 \quad \text{（並列）} \tag{17.3}$$

$$K_{12}=K_1+K_2 \quad \text{（直列）} \tag{17.4}$$

以下に代表的な要素の流動抵抗係数を計算する方法を示す。

2）ダルシー・ワイスバッハの式

ダクトと配管を問わず、流路全般について適用可能な圧力損失 ΔP の推定式としてダルシー・ワイスバッハの式（式 17.5）がある。λ [-]は管摩擦係数と呼ばれる無次元の数であり、計算法は後述する。ρ [kg/m^3]は流体の密度、l [m]は流路の長さ、v [m/s]は流体の速度、d [m]は流路の内径である。ダルシー・ワイスバッハの式は円柱管を想定しており、角形ダクトなど、円柱形状とは異なる場合の適用方法については後述する。式 17.2 に式 17.5 を代入すると流動抵抗係数 K は式 17.6 となる。

$$\Delta P=\lambda\frac{\rho l v^2}{2d} \tag{17.5}$$

$$K=\frac{8\lambda\rho l}{\pi^2 d^5} \tag{17.6}$$

3）管摩擦係数

管摩擦係数 λ は、流体のレイノルズ数 Re [-]と流路表面の粗さ ε [m]（「絶対粗度」と呼ぶ）に依存する。流路が広い場合には流路表面の粗度の影響は相対的に小さく、逆に流路が狭い場合には流路表面の僅かな凹凸も流れに大きく影響を与える。このため、流路表面の絶対粗度 ε を内径 d で除した無次元の数値を相対粗度(ε/d) [-]と定義し、これを用いて管摩擦係数 λ を推定する。

レイノルズ数 Re が小さく、層流の場合には、管摩擦係数 λ は相対粗度 ε/d に影響は受けず、式 17.7 で計算できる。これは具体的には Re < 4,000 となる範囲である。

$$\lambda=64/Re \tag{17.7}$$

レイノルズ数が大きく流れが非常に乱れている場合には、管摩擦係数 λ は相対粗度(ε/d)のみに依存し、式 17.8 に示すカルマン・ニクラッゼの式で計算できる。これは式 17.9 が成立する範囲である。

$$1/\sqrt{\lambda}=-2.0\log\left(\varepsilon/d\right)+1.14 \tag{17.8}$$

$$Re\sqrt{\lambda}(\varepsilon/d)>200 \tag{17.9}$$

上記の 2 つの状態の間（遷移域および乱れの小さい乱流）の場合には、管摩擦係数 λ はレイノルズ数 Re と相対粗度(ε/d)の両方に依存し、式 17.10 に示すコール・ブルックの式で計算できる。ただし、式 17.10 は両辺に管摩擦係数 λ があるため、収束計算が必要になる。

$$1/\sqrt{\lambda}=2.0\log\left(\frac{\varepsilon}{d}+\frac{9.34}{Re\sqrt{\lambda}}\right)+1.14 \tag{17.10}$$

4）絶対粗度

表面の絶対粗度 ε に関しては様々な実測値が報告されている。表 17.1 に建築環境に関連する材料の粗度を示す[17.2)]。

表 17.1　代表的な材料の粗度

材料	粗度 [mm]	材料	粗度 [mm]
銅	0.0015	ガルバリウム	0.15
ステンレス	0.015	グラスウールダクト	0.09
炭素鋼	0.03	吹付グラスウール	3.0
ポリ塩化ビニル	0.03	フレキシブルダクト	3.0
アルミニウム	0.03	コンクリート	3.0

4）等価直径

配管断面は通常は円形であるが、ダクトに関しては角形や楕円形などの断面形状をとることがある。この場合に式 17.5 を適用するために等価直径 d_e [m]という概念を導入する。等価直径 d_e は、任意の断面形状の流路について流体の流れの観点から等価とみなせる円管の直径である。一般的には等価直径は式 17.11 で示されるように流路の周長 L_p [m]と断面積 A_f [m^2]によって計算できる。

$$d_e = 4\frac{A_f}{L_p} \tag{17.11}$$

角形ダクトおよび楕円形ダクト（オーバルダクトと呼ぶ）に関しては精度の高い計算法が提案されており、それぞれ式 17.12 と式 17.13 で計算する[17.3) 17.4)]。l_a [m]と l_b [m]はそれぞれ角形ダクトの 2 つの辺の長さ、A_{ovl} [m^2]、P_{ovl} [m]、l_{mj} [m]、l_{mn} [m]はそれぞれオーバルダクトの断面積、周長、長径と短径の長さである。

$$d_e = 1.3\frac{(l_a l_b)^{0.625}}{(l_a + l_b)^{0.250}} \tag{17.12}$$

$$d_e = 1.55\frac{A_{ovl}}{P_{ovl}} = 1.55\frac{\left((\pi l_{mn}^2/4) + l_{mn}(l_{mj} - l_{mn})\right)^{0.625}}{\left(\pi l_{mn} + 2(l_{mj} - l_{mn})\right)^{0.250}} \tag{17.13}$$

以上の方法で計算した等価直径 d_e を式 17.5 の直径 d に代入することで、任意断面の流路について式 17.5 を適用することができる。

5）局部抵抗

流路が折れ曲がる箇所、断面積が変わる箇所、分流や合流が存在する箇所など、流路の形状が変化する場合にも圧力損失が発生する。この圧力損失を局部抵抗と呼び、式 17.14 で計算する。ζ [-]は損失係数または局部抵抗係数と呼ばれる無次元数であり、様々な形状の流路について図表が提供されている[17.6) 17.7)]。直径 d の円管に接続されていると仮定すれば流動抵抗係数 K は式 17.15 となる。

$$\Delta P = \zeta \frac{\rho v^2}{2} \tag{17.14}$$

$$K = \frac{8\zeta\rho}{\pi d^4} \tag{17.15}$$

6）配管径

通常、配管径は「100A」や「150A」というような「呼び径」と呼ばれる方法で表現される。A の前の数値はミリメートル単位で表示した配管の内径とほぼ等しい値である[†1)]。様々な管種の内径、外径、配管厚は JIS に規定されており、厳密な値が必要な場合はそれらを参照すると良い。エネルギーシミュレーションに用いるレベルの精度であれば、式 17.16 の近似式で配管厚を推定しても良いだろう。d_{th} [mm]は配管厚、a_{dp} と b_{dp} は近似係数、d_i [mm]は内径である。空調・衛生設備で使われること

†1　厳密にはわずかに異なる。また、ミリメートル単位ではなくインチ単位で表現した B 呼称と呼ばれる表示体系もある。

の多い配管用炭素鋼鋼管、硬質塩化ビニル管、配管用ステンレス鋼管の近似係数を表 17.2 に示す。式 17.16 で推定した厚みと JIS による値の関係を図 17.4 に示す。概ね傾向が捉えられることがわかる。

$$d_{th}=a_{dp}d_i^{b_{dp}} \tag{17.16}$$

表 17.2　配管の厚み近似式の係数

配管種別	近似係数 a_{dp}	近似係数 b_{dp}
配管用炭素鋼鋼管 SGP	1.0796	0.3181
硬質塩化ビニル管 VP	0.5196	0.5746
配管用ステンレス鋼管 SUS SC40	0.7250	0.4581

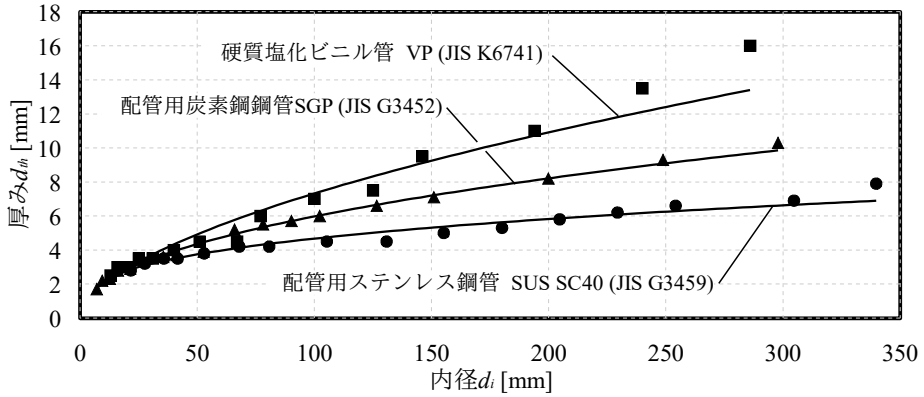


図 17.4　配管の内径と厚みの関係

7）バルブとダンパ

流路に流れる流体の流量を制御するために、開閉の状態を調整可能な抵抗を設けることが多い。流体が水の場合にはバルブ（二方弁）、空気の場合にはダンパと呼ばれる。バルブやダンパは現地で手動で開閉するものと、遠隔で自動開閉させることができるものがある。前者は主に機器のメンテナンスや固定流量が求められる流路の初期調整のために用いられ、後者は主にシステムの制御に用いられる。式 17.17 にバルブおよびダンパによる流動抵抗係数の計算式を示す。W_L [-]は 0~1 までの値をとるリニア特性の重み係数、R_a [-]はレンジアビリティ、K_0 [kPa/(m³/s)²]は全開時の流動抵抗係数、L_f [-]は開度である。レンジアビリティとは制御可能な最小流量と最大流量の比率である。

$$K=\frac{W_L K_0}{\left[(1-1/R_a)L_f+1/R_a\right]^2}+\frac{(1-W_L)K_0}{R_a^{2L_f-2}} \tag{17.17}$$

一般にバルブやダンパを流れる流量は図 17.5 左に示すように開度が開くにつれて大きくなる。リニア特性の重み係数 W_L はこの特性を表現するための係数であり、0 のときにイコールパーセント特性、1 のときにリニア特性となる。現実にはこの他にも図に点線で示すようなクイックオープニング特性と呼ばれる傾向を持つ機器がある。これは迅速な応答の求められる二値制御あるいは ON/OFF 制御用の特性である。一般に熱交換器の流量比と能力比は図 17.5 右に示すような関係を持つため、適切なイコールパーセント特性を持つバルブを選定することで、バルブ開度と熱交換器の能力を比例的にすることができて制御性が増す。これがイコールパーセント特性の意味である。

多くの場合、バルブの仕様は Cv 値を用いて表現される。Cv 値とは前後に 1 lbf/in² の差圧をかけたときに流れる流体の流量を US gal/min で示したものであり、流体の温度は 60 °F である。なお、1 US gal/min は 6.31×10^{-5} m³/s、1 lbf/in² は 6.89 kPa である。

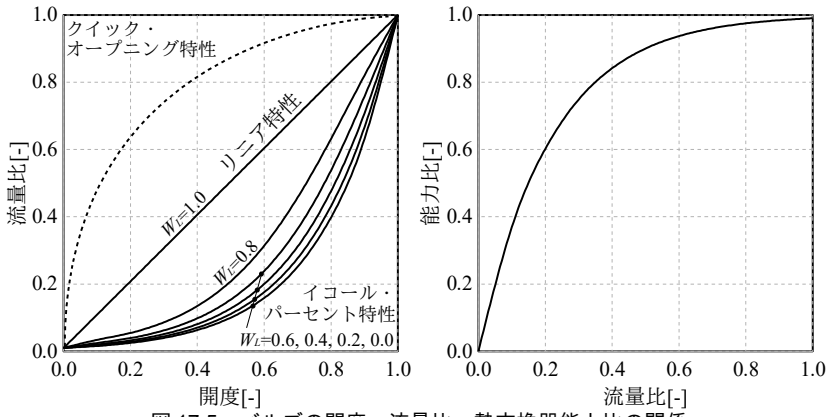


図 17.5　バルブの開度、流量比、熱交換器能力比の関係

17.2.3　モデル化

具体的な熱源システムを例に、回路網によるモデル化の方法を解説する。

図 17.6 に熱源システム系統図および回路網による表現を示す。一次側には 2 台の熱源、二次側には 4 台の AHU を計画している。AHU は別々の階に設置しており、図の左に記載の長さは各階までの配管距離である。二次側は高層系統（AHU3 と AHU4）と低層系統（AHU1 と AHU2）に分けており、高層系統の方が配管経路が長い。ポンプは一次側の熱源用に 2 台、二次側搬送用に 3 台設置されている。このように一次側と二次側でポンプをわけるシステムを複式ポンプ方式と呼ぶ。これとは異なり、一次側と二次側でポンプを分けない方式を単式ポンプ方式と呼ぶ。複式ポンプ方式は単式ポンプ方式よりもスペースとコストが必要となるが、単式ポンプ方式よりも細かな制御がしやすく、エネルギーを減らしやすい。複式ポンプ方式の場合には、一次側ヘッダ間の水搬送のエネルギーは一次ポンプによって行われるため、基本的には往一次ヘッダ（N_0）の圧力と還ヘッダ（N_{14}）の圧力は等しくなると考えて良い[†1]。従って、この熱源システムを回路網に表すと図 17.6 右となる。各所の配管に加え、AHU とこれを制御する二方弁を流路と捉えている。

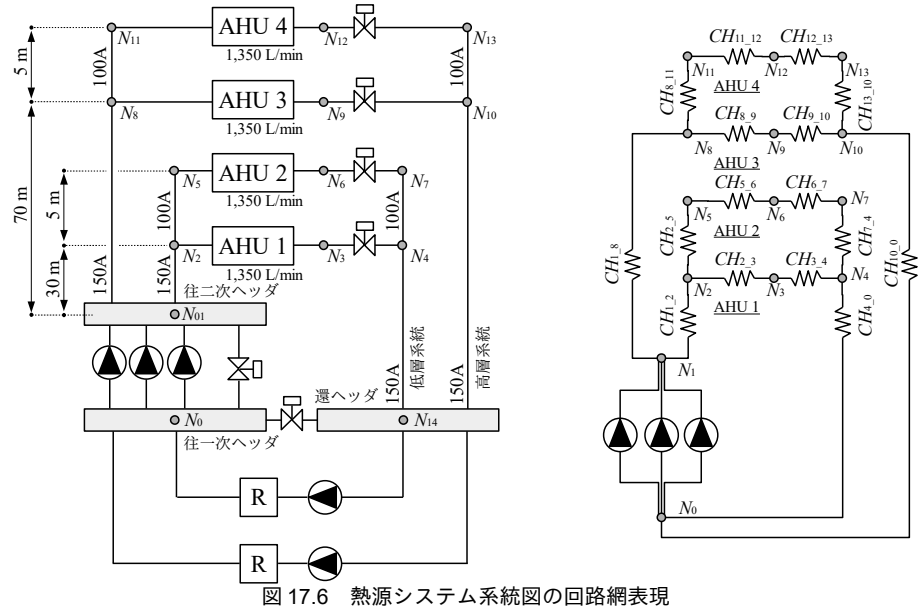


図 17.6　熱源システム系統図の回路網表現

†1　二次側負荷流量が極めて小さく、熱源循環流量の下限値を下回るときには往還ヘッダ間バイパスで冷温水をバイパスさせるため、このような場合には $P_{10} < P_0$ となる。

17.3　計算法

17.3.1　「管内流れ計算クラス」の作成

ダクトと配管に共通する処理を提供する static クラス（Conduit class）を作成する。代表的な管材料を表す列挙型と当該管材料の絶対粗度 ε を出力する関数をプログラム 17.1 に示す。表 17.1 に記載の値である。

プログラム 17.1　管材列挙型の定義と粗度の計算

Popolo.HVAC.Circuit.Conduit class

```
/// <summary>管材</summary>
public enum Material
{
  /// <summary>炭素鋼</summary>
  UncoatedCarbonSteel_Clean,
  /// <summary>ポリ塩化ビニル</summary>
  PVC_PlasticPipe,
  /// <summary>アルミニウム</summary>
  Aluminum,
  /// <summary>ガルバリウム</summary>
  GalvanizedSteel,
  /// <summary>ステンレス</summary>
  StainlessSteel,
  /// <summary>グラスウール吹付</summary>
  FibrousGlassSpray,
  /// <summary>フレキシブルダクト</summary>
  FlexibleDuct,
  /// <summary>グラスウールダクト</summary>
  FibrousGlassDuct,
}

/// <summary>素材の粗度[m]を取得する</summary>
/// <param name="mat">素材</param>
/// <returns>粗度[m]</returns>
/// <remarks>管摩擦係数の計算では内径[m]で除した相対粗度[-]を用いる点に注意</remarks>
public static double GetRoughness(Material mat)
{
  switch (mat)
  {
    case Material.Aluminum:
  return 0.03e-3;
    case Material.FibrousGlassSpray:
  return 3.0e-3;
    case Material.FibrousGlassDuct:
  return 0.9e-3;
    case Material.FlexibleDuct:
  return 3.0e-3;
    case Material.GalvanizedSteel:
  return 0.09e-3;
    case Material.StainlessSteel:
  return 0;
    case Material.UncoatedCarbonSteel_Clean:
  return 0.03e-3;
    case Material.PVC_PlasticPipe:
  return 0.03e-3;
    default:
  return 0;
  }
}
```

プログラム 17.2 に等価直径の計算処理を示す。3 つのメソッドを記したが、それぞれ式 17.11、式 17.12、式 17.13 の実装である。

プログラム 17.2　等価直径の計算処理

Popolo.HVAC.Circuit.Conduit class

```
/// <summary>等価直径[m]を計算する</summary>
/// <param name="flowArea">流路面積[m2]</param>
/// <param name="perimeterLength">周長[m]</param>
/// <returns>等価直径[m]</returns>
public static double GetEquivalentDiameter(double flowArea, double perimeterLength)
{ return 4 * flowArea / perimeterLength; }

/// <summary>角ダクトの等価直径[m]を計算する</summary>
```

```
 9 /// <param name="side1Length">1 側の長さ[m]</param>
10 /// <param name="side2Length">2 側の長さ[m]</param>
11 /// <returns>角ダクトの等価直径[m]</returns>
12 /// <remarks>Huebscher(1948)による</remarks>
13 public static double GetEquivalentDiameterOfRectangularDuct (double side1Length, double side2Length)
14 {
15   double a = side1Length / 1000d;
16   double b = side2Length / 1000d;
17   return (1.30 * Math.Pow(a * b, 0.625) / Math.Pow(a + b, 0.250)) / 1000d;
18 }
19
20 /// <summary>オーバルダクトの等価直径[m]を計算する</summary>
21 /// <param name="majorLength">長径[m]</param>
22 /// <param name="minorLength">短径[m]</param>
23 /// <returns>オーバルダクトの等価直径[m]</returns>
24 /// <remarks>Heyt and Diaz(1975)による</remarks>
25 public static double GetEquivalentDiameterOfOvalDuct (double majorLength, double minorLength)
26 {
27   double a = majorLength / 1000d;
28   double b = minorLength / 1000d;
29   double aa = (Math.PI * b * b / 4d) + b * (a - b);
30   double p = Math.PI * b + 2d * (a - b);
31   return 1.55 * Math.Pow(aa, 0.625) / Math.Pow(p, 0.250);
32 }
```

管摩擦係数の計算処理をプログラム 17.3 に示す。式 17.7、式 17.8、式 17.10 の実装である。レイノルズ数が小さい層流の場合には 8 行に示すように式 17.7 を用いる。式 17.8 と式 17.10 のいずれを用いるかの判定は式 17.9 で行うが、そもそも式 17.9 に含まれる管摩擦係数λが未知であるため、一旦、式 17.10 が成立すると仮定してλを求める。式 17.10 は両辺にλが含まれるため、11~18 行に記載のように収束計算が必要となる。得られたλに対して 19 行で式 17.9 による判定を行い、条件を満たさない場合には式 17.8 によりλを再計算する。

プログラム 17.3　管摩擦係数の計算処理

Popolo.HVAC.Circuit.Conduit class

```
 1 /// <summary>管摩擦係数[-]を計算する</summary>
 2 /// <param name="reynoldsNumber">レイノルズ数[-]</param>
 3 /// <param name="relRoughness">相対粗度[-] (粗度[m]÷内径[m]) </param>
 4 /// <returns>管摩擦係数[-]</returns>
 5 public static double GetFrictionFactor(double reynoldsNumber, double relRoughness)
 6 {
 7   //層流の場合はレイノルズ数のみに依存
 8   if (reynoldsNumber < 4000) return 64d / reynoldsNumber;
 9   else
10   {
11     //遷移領域と仮定して Cole brook の式を解く
12     Roots.ErrorFunction eFnc = delegate (double fc)
13     {
14       double fcc = -2.0 * Math.Log10
15       (relRoughness + 9.34 / (reynoldsNumber * Math.Sqrt(fc))) + 1.14;
16       return fc - 1d / (fcc * fcc);
17     };
18     double fCoef = Roots.Newton(eFnc, 0.02, 0.0001, 0.000001, 0.0001, 10);
19     double bnd = reynoldsNumber * Math.Sqrt(fCoef) * relRoughness;
20     if (bnd <= 200) return fCoef;
21     else
22     {
23       //完全に粗い場合 (fully rough) には Nikuradse の式を解く
24       fCoef = -2.0 * Math.Log10(relRoughness) + 1.14;
25       return 1d / (fCoef * fCoef);
26     }
27   }
28 }
```

ダルシー・ワイスバッハの式関連の計算処理をプログラム 17.4 に示す。1 つめのメソッドは式 17.5 の実装であり、2 つめのメソッドは式 17.5 を平均流速vについて解く処理である。

プログラム 17.4　ダルシー・ワイスバッハの式関連の計算処理

Popolo.HVAC.Circuit.Conduit class

```
 1 /// <summary>ダルシーワイスバッハの式で圧力損失[Pa]を計算する</summary>
 2 /// <param name="frictionFactor">管摩擦係数[-]</param>
 3 /// <param name="density">密度[kg/m3]</param>
```

```
 4 /// <param name="length">管長[m]</param>
 5 /// <param name="diameter">管内径[m]</param>
 6 /// <param name="velocity">管内平均流速[m/s]</param>
 7 /// <returns>圧力損失[Pa]</returns>
 8 public static double GetPressureDrop
 9   (double frictionFactor, double density, double length, double diameter, double velocity)
10 { return frictionFactor * density * length * velocity * velocity / (2 * diameter); }
11
12 /// <summary>ダルシーワイスバッハの式で管内平均流速[m/s]を計算する</summary>
13 /// <param name="frictionFactor">管摩擦係数[-]</param>
14 /// <param name="density">密度[kg/m3]</param>
15 /// <param name="length">管長[m]</param>
16 /// <param name="diameter">管内径[m]</param>
17 /// <param name="pressureDrop">圧力損失[Pa]</param>
18 /// <returns>管内平均流速[m/s]</returns>
19 public static double GetVelocity
20   (double frictionFactor, double density, double length, double diameter, double pressureDrop)
21 {
22   double pda = Math.Abs(pressureDrop);
23   double vel = (2 * pda * diameter) / (frictionFactor * density * length);
24   return Math.Sign(pressureDrop) * Math.Sqrt(vel);
25 }
```

【例題 17.1】

レイノルズ数 Re と管摩擦係数 λ の関係を相対粗度別に計算して図示せよ。

【解】

計算処理をプログラム 17.5 に示す。計算結果を図にすると図 17.7 が得られる。このように横軸にレイノルズ数 Re、縦軸に管摩擦係数 λ をとり、相対粗度別の管摩擦係数を読み取れるようにした図をムーディ線図と呼ぶ。

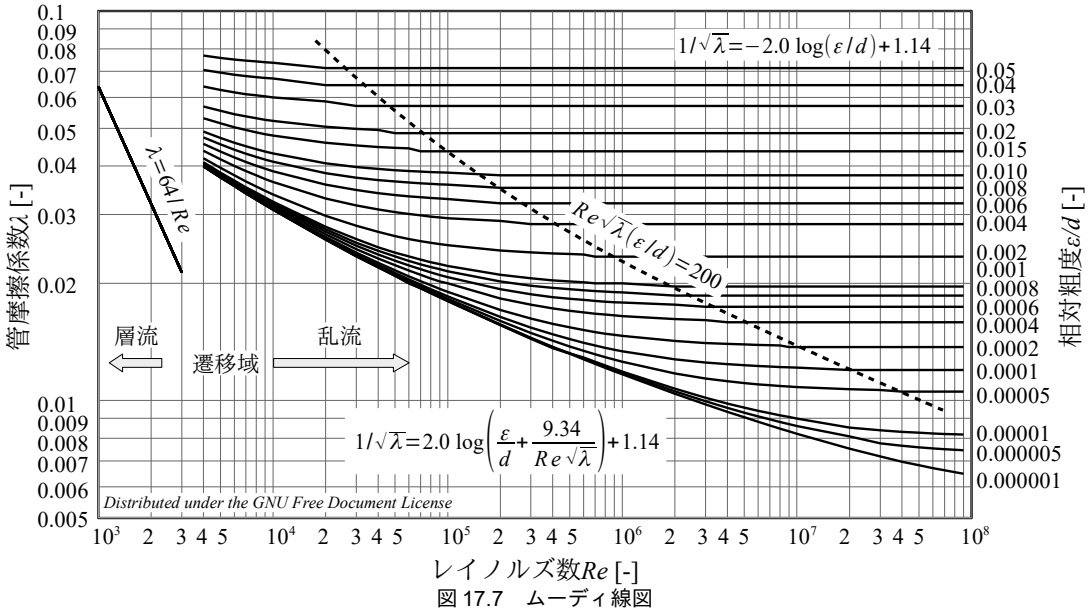


図 17.7　ムーディ線図

プログラム **17.5**　ムーディ線図の作成処理

```
 1 private static void PipeFlowTest()
 2 {
 3   //レイノルズ数[-]
 4   double[] rn = new double[6 * 9];
 5   for (int i = 0; i < 6; i++)
 6     for (int j = 0; j < 9; j++)
 7       rn[i * 9 + j] = (j + 1) * 100 * Math.Pow(10, i);
 8
 9   //相対粗度[-]
10   double[] rf = new double[]
11   { 1e-6, 5e-6, 1e-5, 5e-5, 1e-4, 2e-4, 4e-4, 6e-4, 8e-4,
12     1e-3, 2e-3, 4e-3, 6e-3, 8e-3, 0.01, 0.015, 0.02, 0.03, 0.04, 0.05 };
13
14   using (StreamWriter sWriter =
15     new StreamWriter("PipeFlowTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
16   {
17     for (int i = 0; i < rf.Length; i++) sWriter.Write("," + rf[i]);
```

```
18     sWriter.WriteLine();
19     for (int i = 0; i < rn.Length; i++)
20     {
21       sWriter.Write(rn[i]);
22       for (int j = 0; j < rf.Length; j++)
23       {
24         double fc = Conduit.GetFrictionFactor(rn[i], rf[j]);
25         sWriter.Write("," + fc);
26       }
27       sWriter.WriteLine();
28     }
29   }
30 }
```

17.3.2 「回路網クラス」の作成

回路網をモデル化して解くためには、それぞれの節点と流路がどのように接続されており、どのような状態量（圧力と流量）を持つのかを管理する必要がある。オブジェクト指向言語の特徴をいかせば、このような状態管理をわかりやすく簡易に行うことができる。まず、節点と流路を表現するクラスとインターフェースを作成する。

1）節点と流路のクラス

プログラム 17.6 に節点を表すクラス（CircuitNode class）を示す。

5 行と 8 行は接続されている流路のリストである。流路は、ある節点から別の節点へつなぐことで計画されるため、接続元と接続先の 2 種類の節点が存在する。そこで、節点側でも接続元として使われているのか接続先として使われているのかを判別するため、2 種類のリストで管理する。ただし、接続先と接続元という呼称は回路網を構築するための便宜的なものであり、流れの方向とは無関係であることに注意する。

11 行は境界条件情報であり、圧力が固定されているか否かの情報を持つ。例えば差圧一定制御を行う場合や開放型水槽で圧力が大気圧になる場合などにはこれを true に設定し、境界条件として圧力値与える。この場合には外部からの流入流量が状態変数になる。

14 行と 17 行は圧力と外部から節点に流れ込む流量の値である。境界条件として値を固定する場合には具体的な値を設定する。

22~36 行は流路の接続および解除の処理である。38~47 行は節点に接続された流路の流量の合算処理である。境界条件として外部から流れ込む流量である Inflow も合算する点に注意する。式 17.1 で示したように、質量保存則により、最終的にはモデル内の総ての節点で、このメソッドの返り値が 0 となる必要がある。

プログラム 17.6　節点クラスの定義

Popolo.HVAC.Circuit.CircuitNode class

```
1 /// <summary>回路網の節点</summary>
2 public class CircuitNode
3 {
4   /// <summary>接続元流路リスト</summary>
5   private List<ICircuitBranch> outFlowBrchs = new List<ICircuitBranch>();
6
7   /// <summary>接続先流路リスト</summary>
8   private List<ICircuitBranch> inFlowBrchs = new List<ICircuitBranch>();
9
10  /// <summary>圧力固定か否か</summary>
11  public bool IsPressureFixed { get; set; }
12
13  /// <summary>圧力[kPa]を設定・取得する</summary>
14  public double Pressure { get; set; }
15
16  /// <summary>流入量[m3/s]を設定・取得する</summary>
17  public double Inflow { get; set; }
18
19  /// <summary>インスタンスを初期化する</summary>
```

```
20   internal CircuitNode() { IsInflowFixed = true; }
21
22   /// <summary>接続元流路を追加する</summary>
23   /// <param name="branch">接続元流路</param>
24   internal void addOutFlowBranch(ICircuitBranch branch) { outFlowBrchs.Add(branch); }
25
26   /// <summary>接続先流路を追加する</summary>
27   /// <param name="branch">接続先流路</param>
28   internal void addInFlowBranch(ICircuitBranch branch) { inFlowBrchs.Add(branch); }
29
30   /// <summary>流路を削除する</summary>
31   /// <param name="branch">流路</param>
32   internal void removeBranch(ICircuitBranch branch)
33   {
34     outFlowBrchs.Remove(branch);
35     inFlowBrchs.Remove(branch);
36   }
37
38   /// <summary>流出入収支[m3/s]を積算する</summary>
39   /// <returns>流出入収支[m3/s]</returns>
40   /// <remarks>質量保存則判定用</remarks>
41   public double IntegrateFlow()
42   {
43     double sum = Inflow;
44     foreach (ICircuitBranch br in outFlowBrchs) sum -= br.VolumetricFlowRate;
45     foreach (ICircuitBranch br in inFlowBrchs) sum += br.VolumetricFlowRate;
46     return sum;
47   }
48
49   /// <summary>接続先流路リストを取得する</summary>
50   /// <returns>接続先流路リスト</returns>
51   public ImmutableCircuitBranch[] GetInFlowBranches() { return inFlowBrchs.ToArray(); }
52
53   /// <summary>接続元流路リストを取得する</summary>
54   /// <returns>接続元流路リスト</returns>
55   public ImmutableCircuitBranch[] GetOutFlowBranches() { return outFlowBrchs.ToArray(); }
56 }
```

流路の種類は色々とあり、実装方法も様々に考えられるため、流路一般を表わすインターフェースを定義する。プログラム 17.7 に流路を表現するインターフェース（ICircuitBranch interface）を示す。5 行と 8 行は上流と下流の節点の特定処理、12 行は節点差圧による体積流量計算処理である。

プログラム 17.7　流路インターフェース

Popolo.HVAC.Circuit.ICircuitBranch interface

```
 1 /// <summary>回路網の流路</summary>
 2 public interface ICircuitBranch: IImmutableCircuitBranch
 3 {
 4   /// <summary>上流節点を設定・取得する</summary>
 5   new CircuitNode UpStreamNode { get; set; }
 6
 7   /// <summary>下流節点を設定・取得する</summary>
 8   new CircuitNode DownStreamNode { get; set; }
 9
10   /// <summary>前後差圧から体積流量[m3/s]を計算する</summary>
11   /// <returns>体積流量[m3/s]</returns>
12   void UpdateFlowRateFromNodePressureDifference();
13 }
```

ICircuitBranch インターフェースの実装例をプラグラム 17.8 に示す。流動抵抗係数を一定とみなし、式 17.2 に従って圧力損失を計算する単純な流路のクラスである。20 行のコンストラクタで、式 17.2 と初期化用に与えられた体積流量と圧力損失から流動抵抗係数を求めてプロパティに保存する。32~38 行が ICircuitBranch のメソッドの実装である。

プログラム 17.8　ICircuitBranch の実装例

Popolo.HVAC.Circuit.SimpleCircuitBranch class

```
 1 /// <summary>圧力損失が体積流量の 2 乗に比例する単純な流路</summary>
 2 public class SimpleCircuitBranch : ICircuitBranch
 3 {
 4
 5   /// <summary>上流節点を設定・取得する</summary>
 6   public CircuitNode UpStreamNode { get; set; }
 7
```

```
 8   /// <summary>下流節点を設定・取得する</summary>
 9   public CircuitNode DownStreamNode { get; set; }
10
11   /// <summary>体積流量[m3/s]を設定・取得する</summary>
12   public double VolumetricFlowRate { get; set; }
13
14   /// <summary>流動抵抗係数[kPa/(m/s)2]を設定・取得する</summary>
15   public double Resistance { get; set; }
16
17   /// <summary>インスタンスを初期化する</summary>
18   /// <param name="flowRate">体積流量[m3/s]</param>
19   /// <param name="pressure">圧力損失[kPa]</param>
20   public SimpleCircuitBranch(double flowRate, double pressure)
21   { Resistance = pressure / (flowRate * flowRate); }
22
23   /// <summary>圧力損失[kPa]を計算する</summary>
24   /// <param name="flowRate">体積流量[m3/s]</param>
25   /// <returns>圧力損失[kPa]</returns>
26   public double GetPressureDrop(double flowRate)
27   {
28     VolumetricFlowRate = flowRate;
29     return flowRate * flowRate * Resistance;
30   }
31
32   /// <summary>前後差圧から体積流量[m3/s]を計算する</summary>
33   /// <returns>体積流量[m3/s]</returns>
34   public void UpdateFlowRateFromNodePressureDifference()
35   {
36     double dp = UpStreamNode.Pressure - DownStreamNode.Pressure;
37     VolumetricFlowRate = Math.Sign(dp) * Math.Sqrt((Math.Abs(dp) / Resistance));
38   }
39 }
```

流量制御バルブ・ダンパのクラスをプログラム 17.9 に示す。式 17.17 に従ってバルブおよびダンパの開度によって流動抵抗係数を変化させるモデルとする。43~55 行がコンストラクタであり、開度が全開の時の流量と圧力損失から全開時抵抗を計算する。57~68 行でコンストラクタをオーバーロードし、Cv 値による初期化も可能とする。70~92 行は ICircuitBranch の実装である。82~92 行は体積流量の計算処理であり、94~103 行の関数で開度に応じて流動抵抗係数を更新してから計算する。105~131 行は開度の調整処理である。前後差圧と流量設定値にもとづき、117 行で必要な抵抗を計算する。この抵抗に合致するように 123~129 行で収束計算でバルブ開度を求める。ただし必要な抵抗が最大開度における抵抗を下回る場合には最大開度とする（118 行）。

プログラム 17.9　流量制御バルブ・ダンパクラスの定義

Popolo.HVAC.Circuit.Regulator class

```
 1 /// <summary>流量制御バルブ・ダンパ</summary>
 2 public class Regulator: ICircuitBranch
 3 {
 4   /// <summary>全開時抵抗係数[kPa/(m3/s)^2]</summary>
 5   private double minResistance;
 6
 7   /// <summary>開度[-]</summary>
 8   private double lift = 1;
 9
10   /// <summary>リニア特性重み係数[-]</summary>
11   private double lWeight;
12
13   /// <summary>レンジアビリティ[-]</summary>
14   private double rangeAbility = 100;
15
16   /// <summary>開度[-]を設定・取得する</summary>
17   public double Lift
18   {
19     get { return lift; }
20     set { lift = Math.Max(0, Math.Min(1, value)); }
21   }
22
23   /// <summary>リニア特性の重み係数[-]を設定・取得する</summary>
24   public double LinearCharactaristicWeight
25   {
26     get { return lWeight; }
```

```
27     set { lWeight = Math.Max(0, Math.Min(1, value)); }
28   }
29
30   /// <summary>レンジアビリティ[-]を設定・取得する</summary>
31   public double RangeAbility
32   {
33     get { return rangeAbility; }
34     set { if (0 < value) rangeAbility = value; }
35   }
36
37   /// <summary>流量設定値[m3/s]を設定・取得する</summary>
38   public double VolumetricFlowRateSetPoint { get; set; }
39
40 /// <summary>全閉可能か否かを設定・取得する</summary>
41 public bool IsTotallyClosable { get; set; } = false;
42
43   /// <summary>インスタンスを初期化する</summary>
44   /// <param name="flowRate">全開時流量[m3/s]</param>
45   /// <param name="pressureDrop">全開時圧力損失[kPa]</param>
46   /// <param name="rangeAbility">レンジアビリティ[-]</param>
47   /// <param name="linearWeight">リニア特性重み係数[-]</param>
48   public Regulator(double flowRate, double pressureDrop, double rangeAbility, double linearWeight)
49   {
50     DesignFlowRate = flowRate;
51     VolumetricFlowRateSetPoint = flowRate;
52     minResistance = pressureDrop / (flowRate * flowRate);
53     RangeAbility = rangeAbility;
54     LinearCharactaristicWeight = linearWeight;
55   }
56
57   /// <summary>インスタンスを初期化する</summary>
58   /// <param name="cvValue">CV 値[USgal/min]</param>
59   /// <param name="rangeAbility">レンジアビリティ[-]</param>
60   /// <param name="linearWeight">リニア特性重み係数[-]</param>
61   public Regulator(double cvValue, double rangeAbility, double linearWeight)
62   {
63     DesignFlowRate = cvValue * 6.31e-5;
64     VolumetricFlowRateSetPoint = DesignFlowRate;
65     minResistance = 6.89 / (DesignFlowRate * DesignFlowRate);
66     RangeAbility = rangeAbility;
67     LinearCharactaristicWeight = linearWeight;
68   }
69
70   /// <summary>流量[m3/s]を取得する</summary>
71   public double VolumetricFlowRate { get; private set; }
72
73   /// <summary>設計流量[m3/s]を取得する</summary>
74   public double DesignFlowRate { get; private set; }
75
76   /// <summary>上流節点を設定・取得する</summary>
77   public CircuitNode UpStreamNode { get; set; }
78
79   /// <summary>下流節点を設定・取得する</summary>
80   public CircuitNode DownStreamNode { get; set; }
81
82 /// <summary>前後差圧から体積流量[m3/s]を計算する</summary>
83 /// <returns>体積流量[m3/s]</returns>
84 public void UpdateFlowRateFromNodePressureDifference()
85 {
86   if (IsTotallyClosable && Lift == 0) VolumetricFlowRate = 0;
87   else
88   {
89     double dp = UpStreamNode.Pressure - DownStreamNode.Pressure;
90     VolumetricFlowRate = Math.Sign(dp) * Math.Sqrt(Math.Abs(dp) / GetResistance());
91   }
92 }
93
94 /// <summary>流動抵抗係数[kPa/(m3/s)^2]を取得する</summary>
95 /// <returns>流動抵抗係数[kPa/(m3/s)^2]</returns>
96 public double GetResistance()
97 {
98   if (IsTotallyClosable && Lift == 0) return double.PositiveInfinity;
99   double wf = LinearCharactaristicWeight;
100  double lam = 1d / RangeAbility;
101  return wf* minResistance / Math.Pow((1 - lam) * Lift + lam, 2)
102   + (1d - wf) * minResistance * Math.Pow(lam, 2 * Lift - 2);
103 }
104
105  /// <summary>現在の前後差圧[kPa]を前提に開度を調整する</summary>
106  public void UpdateLift()
```

```
107   {
108     if (VolumetricFlowRateSetPoint == 0) Lift = 0;
109     double dp = Math.Abs(UpStreamNode.Pressure - DownStreamNode.Pressure);
110     UpdateLift(dp);
111   }
112
113   /// <summary>前後差圧[kPa]にもとづいて開度を調整する</summary>
114   /// <param name="pressure">前後差圧[kPa]</param>
115   public void UpdateLift(double pressure)
116   {
117     double res = pressure / (VolumetricFlowRateSetPoint * VolumetricFlowRateSetPoint);
118     if (res < minResistance) Lift = 1.0;
119     else
120     {
121       double wf = LinearCharactaristicWeight;
122       double lam = 1d / RangeAbility;
123       Roots.ErrorFunction eFnc = delegate (double c)
124       {
125         Lift = c;
126         return wf * minResistance / Math.Pow((1 - lam) * Lift + lam, 2)
127         + (1d - wf) * minResistance * Math.Pow(lam, 2 * Lift - 2) - res;
128       };
129       Lift = Roots.Newton(eFnc, 0.5, 1e-4, 1e-4, 1e-4, 20);
130     }
131   }
132 }
```

ICircuitBranch インターフェースの実装は自由であり、例えば第 1 章で示した WaterPipe クラスにプラグラム 17.10 の処理を追加すれば回路網モデルに取り込むことができる。 10~31 行が体積流量の計算処理であり、温度情報をもとに配管内の水のレイノルズ数を求め、プログラム 17.4 で示したダルシーワイスバッハ式に関する処理を用いて流速および体積流量を推定する。

プログラム 17.10　WaterPipe クラスの ICircuitBranch 実装

Popolo.HVAC.Circuit.WaterPipe class

```
 1 /// <summary>流量[m3/s]を取得する</summary>
 2 public double VolumetricFlowRate { get; private set; }
 3
 4 /// <summary>上流節点を設定・取得する</summary>
 5 public CircuitNode UpStreamNode { get; set; }
 6
 7 /// <summary>下流節点を設定・取得する</summary>
 8 public CircuitNode DownStreamNode { get; set; }
 9
10 /// <summary>前後差圧から体積流量[m3/s]を計算する</summary>
11 /// <returns>体積流量[m3/s]</returns>
12 public void UpdateFlowRateFromNodePressureDifference()
13 {
14   //水物性（動粘性係数[m2/s], 熱拡散率[m2/s]）を計算
15   double v = Water.GetLiquidDynamicViscosity(InletWaterTemperature);
16   double rho = Water.GetLiquidDensity(InletWaterTemperature);
17
18   //流路面積[m2]と前後差圧[kPa]
19   double fArea = InnerDiameter * InnerDiameter / 4d * Math.PI;
20   double dp = (UpStreamNode.Pressure - DownStreamNode.Pressure) * 1000;
21
22   //流速[m/s]を収束計算
23   Roots.ErrorFunction eFnc = delegate (double vel)
24   {
25     double reNumber = vel * InnerDiameter / v;
26     double ff = Conduit.GetFrictionFactor(reNumber, Roughness / InnerDiameter);
27     return Conduit.GetVelocity(ff, rho, Length, InnerDiameter, dp) - vel;
28   };
29   //初期値はレイノルズ数（流速大）が大きめの点とすると収束がうまくいく
30   VolumetricFlowRate = Roots.Newton(eFnc, 5, 0.0001, 1e-10, 1e-9, 30) * fArea;
31 }
```

2）回路網クラス

以上に記した節点クラスおよび流路インターフェース用いて回路網を組み立て、これを解くクラス（CircuitNetwork class）を作成する。プログラム 17.11 に CircuitNetwork class のインスタンス変数およびプロパティを示す。流路と節点のリストを持つ。回路内に流れる流量は節点同士の差圧に依存し、絶対圧力には影響を受けないため、回路内のどこかに基準となる絶対圧力を定める必要がある。この

圧力の基準点および基準圧力を保持するプロパティが 8 行と 11 行である。回路を解くためには反復収束計算が必要となるため、反復計算の情報を 14 行と 17 行で保持する。

プログラム 17.11　インスタンス変数およびプロパティの定義

Popolo.HVAC.Circuit.CircuitNetwork class

```
 1 /// <summary>流路リスト</summary>
 2 private List<ICircuitBranch> branches = new List<ICircuitBranch>();
 3
 4 /// <summary>節点リスト</summary>
 5 private List<CircuitNode> nodes = new List<CircuitNode>();
 6
 7 /// <summary>圧力基準となる節点を取得する</summary>
 8 public CircuitNode BasePressureNode { get; private set; }
 9
10 /// <summary>基準圧力[kPa]を取得する</summary>
11 public double BasePressure { get; private set; }
12
13 /// <summary>反復計算回数を取得する</summary>
14 public int Iteration { get; private set; }
15
16 /// <summary>誤差を取得する</summary>
17 public double Error { get; private set; }
```

モデル構築に関連する処理をプログラム 17.12 に示す。1~26 行のメソッドで節点を追加・削除し、28~50 行のメソッドで節点の接続と流路の削除を行う。52~59 行は基準圧力の設定処理である。

プログラム 17.12　モデル構築関連処理

Popolo.HVAC.Circuit.CircuitNetwork class

```
 1 /// <summary>節点を追加する</summary>
 2 /// <returns>節点番号</returns>
 3 public CircuitNode AddNode()
 4 {
 5   CircuitNode nd = new CircuitNode();
 6   nodes.Add(nd);
 7   if (nodes.Count == 1) SetBasePressure(nd, 0);
 8   return nd;
 9 }
10
11 /// <summary>節点を削除する</summary>
12 /// <param name="node"></param>
13 /// <returns>削除成功の真偽</returns>
14 public bool RemoveNode(CircuitNode node)
15 {
16   if (nodes.Contains(node))
17   {
18     //接続中の節点は削除不可
19     foreach (ICircuitBranch br in branches)
20       if (br.UpStreamNode == node || br.DownStreamNode == node) return false;
21     nodes.Remove(node);
22     BasePressureNode = null;
23     return true;
24   }
25   else return false;
26 }
27
28 /// <summary>流路で節点を接続する</summary>
29 /// <param name="branch">流路</param>
30 /// <param name="ndFrom">接続元</param>
31 /// <param name="ndTo">接続先</param>
32 public void ConnectNode(ICircuitBranch branch, CircuitNode ndFrom, CircuitNode ndTo)
33 {
34   if (!nodes.Contains(ndFrom) || !nodes.Contains(ndTo)) throw new Exception("Invalid Node Error");
35   if (!branches.Contains(branch)) branches.Add(branch);
36   //接続処理
37   branch.UpStreamNode = ndFrom;
38   branch.DownStreamNode = ndTo;
39   ndFrom.addOutFlowBranch(branch);
40   ndTo.addInFlowBranch(branch);
41 }
42
43 /// <summary>流路を削除する</summary>
44 /// <param name="branch">流路</param>
45 public void RemoveBranch(ICircuitBranch branch)
46 {
47   branch.UpStreamNode.removeBranch(branch);
48   branch.DownStreamNode.removeBranch(branch);
```

```
49   branches.Remove(branch);
50 }
51
52 /// <summary>基準圧力を設定する</summary>
53 /// <param name="node">節点</param>
54 /// <param name="pressure">基準圧力[kPa]</param>
55 public void SetBasePressure(CircuitNode node, double pressure)
56 {
57   BasePressureNode = node;
58   BasePressure = pressure;
59 }
```

回路網の計算方法としては、圧力仮定法と流量仮定法がある。圧力仮定法は、すべての節点の圧力を未知変数とした上で、式 17.1 の流量保存が成立するように圧力調整を行う方法である。逆に流量仮定法は、各流路の流量を未知変数として上で、流路抵抗から計算される各節点の圧力が等しくなるように流量調整を行う方法である。多くの空調配管やダクトには分岐があり、1 つの節点に複数の流路が接続されているため、流路の方が節点よりも数が大きい。従って、未知変数の数を最小化するという観点からは圧力仮定法が有利であるため、本書では圧力仮定法を採用する。

圧力仮定法を用いて回路網を解く処理をプログラム 17.13 に示す。収束計算には第 2 章で解説したニュートン法を用いる。5~12 行で未知変数である入力ベクトルを作成する。境界条件ではない節点の圧力（8 行）に加え、外部の系への流出入を考慮するために、自然流入量または自然流出量を未知変数とする（9 行）。この未知変数（入力ベクトル）を受け取って誤差を出力する誤差関数を 26~57 行に定義する。31~45 行は入力ベクトルの設定処理である。48 行で各流路の流量を更新した後、50~56 行で式 17.1 の質量保存則による誤差を積算する。収束計算終了後、22, 23 行で回路網内の基準点に合わせて絶対圧力を調整する。

プログラム 17.13　回路網を解く処理

Popolo.HVAC.Circuit.CircuitNetwork class

```
 1 /// <summary>回路網を解く</summary>
 2 public void Solve()
 3 {
 4   //入力値ベクトルを作成する
 5   List<double> inp = new List<double>();
 6   for (int i = 0; i < nodes.Count; i++)
 7   {
 8     if (nodes[i].IsPressureFixed) inp.Add(nodes[i].Inflow);
 9     else inp.Add(nodes[i].Pressure);
10   }
11   IVector inputs = new Vector(inp.Count);
12   for (int i = 0; i < inp.Count; i++) inputs[i] = inp[i];
13
14   //ニュートン法で解く
15   int iter;
16   double err;
17   MultiRoots.Newton(errorFNC, ref inputs, 1e-5, 1e-5, 100, out iter, out err);
18   Iteration = iter;
19   Error = err;
20
21   //基準点に合わせて絶対圧力を調整する
22   double dp = BasePressure - BasePressureNode.Pressure;
23   foreach (CircuitNode nd in nodes) nd.Pressure += dp;
24 }
25
26 /// <summary>誤差関数</summary>
27 /// <param name="inputs">入力値</param>
28 /// <param name="outputs">出力値</param>
29 private void errorFNC(IVector inputs, ref IVector outputs)
30 {
31   //節点圧力および固定流出入量を設定
32   int indx = 0;
33   for (int i = 0; i < nodes.Count; i++)
34   {
35     if (nodes[i].IsPressureFixed)
36     {
37       nodes[i].Inflow = inputs[indx];
```

```
38        indx++;
39      }
40      else
41      {
42        nodes[i].Pressure = inputs[indx];
43        indx++;
44      }
45    }
46
47    //流路流量を更新
48    for (int i = 0; i < branches.Count; i++) branches[i].UpdateFlowRateFromNodePressureDifference();
49
50    //節点の質量保存誤差を計算
51    indx = 0;
52    for (int i = 0; i < nodes.Count; i++)
53    {
54      outputs[i] = nodes[i].IntegrateFlow();
55      indx++;
56    }
57 }
```

プログラム 17.13 は汎用的な回路網の計算処理であるため、現実の問題を解く際には、具体的な問題に即応した適切な回路網のモデル化を行う必要がある。図 17.8 は図 17.6 の回路網について、吐出圧一定制御、末端差圧一定制御、DDC 連携制御をどのように捉えれば良いかを示した図である。図中、枠線で囲った流量は境界条件とする状態値である。

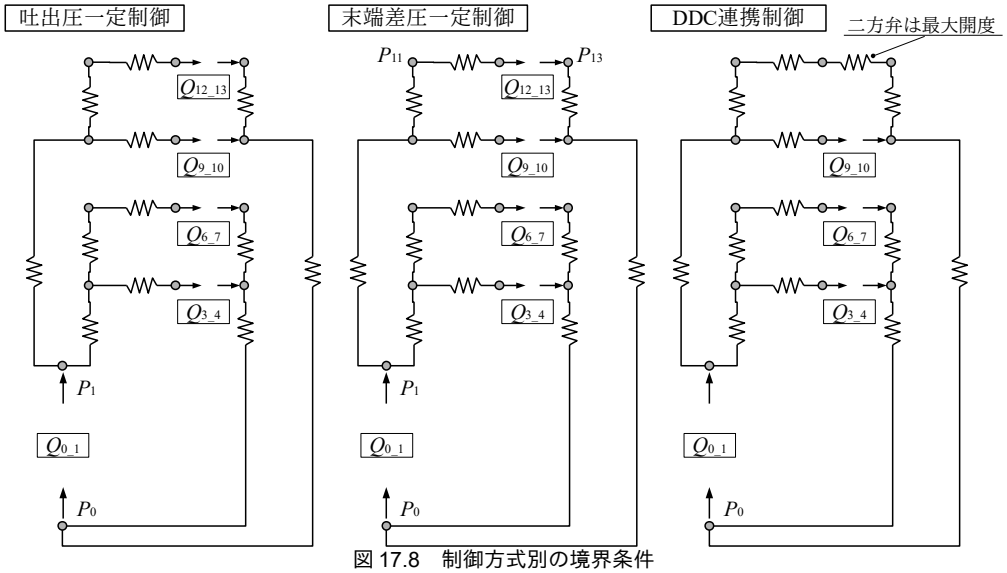


図 17.8　制御方式別の境界条件

図 17.8 左は吐出圧一定制御の場合である。各 AHU の流量は制御対象であるため、境界条件とする。また、これらを合算した流量を送り出す必要があるため、ヘッダにおける流量も境界条件となる。AHU を挟んで 2 つの回路網に分割できることがわかる。ポンプ前後差圧は一定に保つため、ポンプの出入口である P_0 と P_1 は基準圧力となる。例えば、前後差圧を 250 kPa に制御するのであれば P_0 を 0kPa、P_1 を 250 kPa として 2 つの回路網を解くことになる。ただし、この制御方式の場合には計算条件としてポンプの前後差圧と流量が与えられるため、ポンプのエネルギー消費量を求めるという目的からは回路網を解く意味は無い。

図 17.8 中は末端差圧一定制御の場合である。境界条件は吐出圧一定制御と同じであるが、末端の P_{11} と P_{13} の差圧を一定に保つ点が異なる。この場合には、まず P_0 を基準圧力（例えば 0 kPa）にとり、下流の回路網を解くことで P_{13} の圧力を計算する。この P_{13} の圧力に対して末端差圧を加えた値を P_{11} に設定し、これを基準圧力として上流の回路を解く。計算結果である P_1 の圧力と当初想定した P_0

の圧力の差からポンプの消費エネルギーを計算する。

図 17.8 右は DDC 連携制御の場合である。DDC 連携制御は、回路内に含まれるすべてのバルブ開度を確認し、開度が最大になるようにポンプの回転数を調整する制御である。そこで、最も流量需要が大きい AHU の二方弁開度を最大に設定し、残余の AHU の流量を境界条件とする。多くの場合には図 17.8 右に示すように最遠端の AHU の流量需要が大きいため、この流路の二方弁開度が最大になる。しかし、手前にある AHU の負荷が極端に大きい場合などに、このモデルを適用してしまうと、手前の AHU に十分な冷温水が流れない（二方弁開度を最大にしても流量が不足する）ことになる。そこで一旦、図 17.8 左の吐出圧一定制御で計算を行い、最も二方弁開度の大きい系統を特定した後、当該系統の二方弁開度を最大にして図 17.8 右のモデルを適用する。

【例題 17.2】

図 17.8 に示した吐出圧一定制御と末端差圧一定制御の回路網について、各節点の圧力を計算せよ。ただし吐出圧一定制御の圧力は 250 kPa、末端差圧一定制御の圧力は 160 kPa とする。流量制御バルブの Cv 値は 340、定格流量時（0.0225m^3/s）の AHU の圧力損失は 150 kPa とする。配管長は図 17.6 に記載のとおりとするが、この他に局部抵抗として配管長の 0.5 倍を見込むこととする。

【解】

計算処理をプログラム 17.14 に示す。

43~89 行は回路網のモデル化処理である。AHU を境に上流側と下流側で 2 つの回路網モデルを作成する。48~71 行が上流側回路の作成処理である。50~57 行と 58, 59 行で流路と節点を作成し、60~67 行で節点を接続する。69, 70 行は境界条件であり、AHU とヘッダの流量を指定する。71 行は未知変数の初期値である。経験則であるが圧力はすべて 0 とすると収束がうまくいくことが多い。73~88 行は下流側のモデル化であり、処理内容は上流側と同様である。

4 行で上記のメソッドを呼び出して回路網モデルを作成する。8~24 行は吐出圧一定制御の場合の処理であり、上流と下流の基準圧力をそれぞれ 0 kPa と 250 kPa として計算を行う。15~24 行は書き出し処理である。26~40 行は末端差圧一定制御であり、この場合には下流側で計算した結果を利用して上流側モデル末端の基準圧力を設定する（31 行）。計算結果を図示すると図 17.9 となる。

なお、本問はすべての流路の流量が既知であるため、ヘッダから順に計算を進めていけば実は手計算でも解を得ることができる。既に述べたように非線形連立方程式の数値計算は、1 変数とは異なり確実な収束が保証されていないため、解析的に解を得ることができないかについては十分に検討する必要がある。

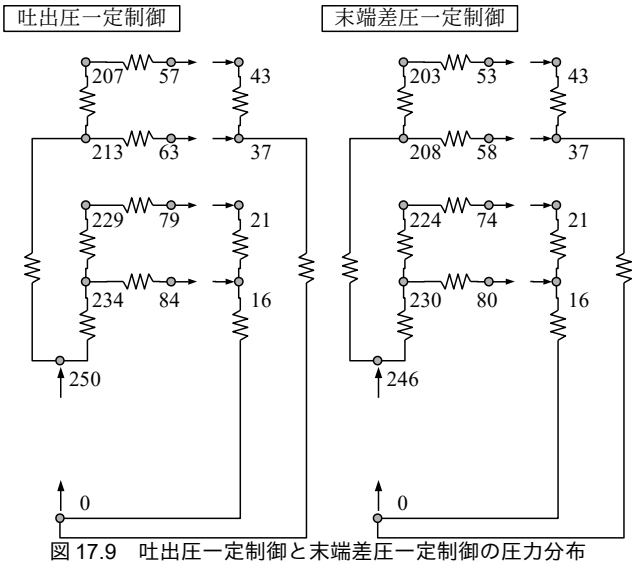


図 17.9　吐出圧一定制御と末端差圧一定制御の圧力分布

プログラム **17.14**　吐出圧一定制御と末端差圧一定制御の計算処理

```
 1 private static void CircuitNetworkTest1()
 2 {
 3   CircuitNetwork cnetUStrm, cnetDStrm;
 4   makePumpNetwork1(out cnetUStrm, out cnetDStrm);
 5   CircuitNode[] nodesUStrm = cnetUStrm.Nodes;
 6   CircuitNode[] nodesDStrm = cnetDStrm.Nodes;
 7
 8   //吐出圧一定制御//////////////////////
 9   //上流側の計算
10   cnetUStrm.SetBasePressure(nodesUStrm[0], 250);
11   cnetUStrm.Solve();
12   //下流側の計算
13   cnetDStrm.SetBasePressure(nodesDStrm[0], 0);
14   cnetDStrm.Solve();
15   //出力
16   using (StreamWriter sWriter = new StreamWriter
17     ("CircuitNetworkTest1.csv", false, Encoding.GetEncoding("Shift_JIS")))
18   {
19     sWriter.WriteLine("節点, 01, 02, 05, 08, 11, 03, 06, 09, 12, 00, 04, 07, 10, 13");
20     sWriter.Write("吐出圧一定制御:圧力[kPa]");
21     for (int i = 0; i < nodesUStrm.Length; i++) sWriter.Write("," + nodesUStrm[i].Pressure);
22     for (int i = 0; i < nodesDStrm.Length; i++) sWriter.Write("," + nodesDStrm[i].Pressure);
23     sWriter.WriteLine();
24   }
25
26   //末端差圧一定制御//////////////////////
27   //下流側の計算
28   cnetDStrm.SetBasePressure(nodesDStrm[0], 0);
29   cnetDStrm.Solve();
30   //上流側の計算
31   cnetUStrm.SetBasePressure(nodesUStrm[4], nodesDStrm[4].Pressure + 160);
32   cnetUStrm.Solve();
33   //出力
34   using (StreamWriter sWriter = new StreamWriter
35     ("CircuitNetworkTest1.csv", true, Encoding.GetEncoding("Shift_JIS")))
36   {
37     sWriter.Write("末端差圧一定制御:圧力[kPa]");
38     for (int i = 0; i < nodesUStrm.Length; i++) sWriter.Write("," + nodesUStrm[i].Pressure);
39     for (int i = 0; i < nodesDStrm.Length; i++) sWriter.Write("," + nodesDStrm[i].Pressure);
40   }
41 }
42
43 private static void makePumpNetwork1
44   (out CircuitNetwork cnetUStrm, out CircuitNetwork cnetDStrm)
45 {
46   double lcf = 1.5; //局部抵抗係数
47
48   //上流型回路
49   cnetUStrm = new CircuitNetwork();
50   WaterPipe ch01_02 = new WaterPipe(30 * lcf, 0.150, WaterPipe.Material.CarbonSteel);
51   WaterPipe ch02_05 = new WaterPipe(5 * lcf, 0.100, WaterPipe.Material.CarbonSteel);
52   WaterPipe ch01_08 = new WaterPipe(70 * lcf, 0.150, WaterPipe.Material.CarbonSteel);
53   WaterPipe ch08_11 = new WaterPipe(5 * lcf, 0.100, WaterPipe.Material.CarbonSteel);
54   SimpleCircuitBranch ch02_03_ahu1 = new SimpleCircuitBranch(0.0225, 150);
55   SimpleCircuitBranch ch05_06_ahu2 = new SimpleCircuitBranch(0.0225, 150);
56   SimpleCircuitBranch ch08_09_ahu3 = new SimpleCircuitBranch(0.0225, 150);
57   SimpleCircuitBranch ch11_12_ahu4 = new SimpleCircuitBranch(0.0225, 150);
58   CircuitNode[] nodesUStrm = new CircuitNode[9]; //01, 02, 05, 08, 11, 03, 06, 09, 12 の順
59   for (int i = 0; i < nodesUStrm.Length; i++) nodesUStrm[i] = cnetUStrm.AddNode();
60   cnetUStrm.ConnectNode(ch01_02, nodesUStrm[0], nodesUStrm[1]);
61   cnetUStrm.ConnectNode(ch02_05, nodesUStrm[1], nodesUStrm[2]);
62   cnetUStrm.ConnectNode(ch01_08, nodesUStrm[0], nodesUStrm[3]);
63   cnetUStrm.ConnectNode(ch08_11, nodesUStrm[3], nodesUStrm[4]);
64   cnetUStrm.ConnectNode(ch02_03_ahu1, nodesUStrm[1], nodesUStrm[5]);
65   cnetUStrm.ConnectNode(ch05_06_ahu2, nodesUStrm[2], nodesUStrm[6]);
66   cnetUStrm.ConnectNode(ch08_09_ahu3, nodesUStrm[3], nodesUStrm[7]);
67   cnetUStrm.ConnectNode(ch11_12_ahu4, nodesUStrm[4], nodesUStrm[8]);
68   //境界条件, 初期値設定
69   nodesUStrm[0].Inflow = 0.0225 * 4;  //流入量
70   for (int i = 5; i < 9; i++) nodesUStrm[i].Inflow = -0.0225; //流出量
71   for (int i = 0; i < nodesUStrm.Length; i++) nodesUStrm[i].Pressure = 0;
72
73   //下流側回路
74   cnetDStrm = new CircuitNetwork();
75   WaterPipe ch04_00 = new WaterPipe(30 * lcf, 0.150, WaterPipe.Material.CarbonSteel);
76   WaterPipe ch07_04 = new WaterPipe(5 * lcf, 0.100, WaterPipe.Material.CarbonSteel);
77   WaterPipe ch10_00 = new WaterPipe(70 * lcf, 0.150, WaterPipe.Material.CarbonSteel);
78   WaterPipe ch13_10 = new WaterPipe(5 * lcf, 0.100, WaterPipe.Material.CarbonSteel);
79   CircuitNode[] nodesDStrm = new CircuitNode[5]; //00, 04, 07, 10, 13 の順
```

```
80   for (int i = 0; i < nodesDStrm.Length; i++) nodesDStrm[i] = cnetDStrm.AddNode();
81   cnetDStrm.ConnectNode(ch04_00, nodesDStrm[1], nodesDStrm[0]);
82   cnetDStrm.ConnectNode(ch07_04, nodesDStrm[2], nodesDStrm[1]);
83   cnetDStrm.ConnectNode(ch10_00, nodesDStrm[3], nodesDStrm[0]);
84   cnetDStrm.ConnectNode(ch13_10, nodesDStrm[4], nodesDStrm[3]);
85   //境界条件, 初期値設定
86   for (int i = 1; i < 5; i++) nodesDStrm[i].Inflow = 0.0225; //流出量
87   nodesDStrm[0].Inflow = -0.0225 * 4;  //流入量
88   for (int i = 0; i < nodesUStrm.Length; i++) nodesUStrm[i].Pressure = 0;
89 }
```

【例題 17.3】

図 17.8 の DDC 連携制御について流量とポンプ消費電力の関係を図示せよ。ただし流量の変化は、1）4 台の AHU の負荷が均等に下がる場合、2）遠方の AHU から順に負荷が下がる場合、3）近傍の AHU から順に負荷が下がる場合、の 3 通りとする。3 台のポンプは同型とし、仕様は第 16 章の例題 16.2 に記載のものと同じとする。その他の情報は例題 17.2 と同一とする。

【解】

計算処理をプログラム 17.15 に示す。119~151 行で回路網モデルを作成する。二方弁を経由して上流と下流を接続するため、例題 17.2 とは異なり、全体を 1 つの回路網モデルとする。3~5 行で吐出圧一定制御用のモデルと最小吐出圧制御用のモデルを作成する。6~8 行は消費電力計算用のポンプである。9 行の配列は各 AHU に流す流量を保持する配列である。15~27 行、28~45 行、46~63 行でそれぞれ負荷を均等に低減した場合、近傍の AHU から負荷率を低減した場合、遠方の AHU から負荷率を低減した場合、の計算を行う。

67~117 行がメインの処理である。吐出圧一定制御と最小吐出圧制御の両モデルについて、73~76 行で境界条件を設定する節点を配列に整理し、79~88 行で具体的な流量を設定する。まず、90~94 行で吐出圧一定制御の計算を行い、二方弁モデルを使って 96~108 行で最も開度の大きくなる最大負荷系統を求める。110~116 行で最大負荷系統に開度 100 %の二方弁を設定して回路網を解く。

計算結果を図 17.10 に示す。すべての AHU の負荷が均等に低下した場合に比べて、負荷が偏在する場合には消費電力の低下幅が小さいことがわかる。特に最遠端の AHU 負荷が大きい場合には、最遠端の流路に対して大きな送水を続ける必要があるため、消費電力の低下が小さい。最遠端の AHU 負荷が小さい場合には、最初は大きく消費電力が下がるが、負荷率の高い流路が残るため次第に消費電力の低下幅が小さくなる。このような傾向は回路網を解かなければつかむことができず、第 16 章で示した簡易計算法にはある程度の誤差が存在することを理解しておく必要がある。

プログラム 17.15　流量とポンプ消費電力の関係（制御方式別）

```
 1 private static void CircuitNetworkTest2()
 2 {
 3   CircuitNetwork cnetUStrm, cnetDStrm, cnetDDC;
 4   makePumpNetwork1(out cnetUStrm, out cnetDStrm);
 5   makePumpNetwork2(out cnetDDC);
 6   CentrifugalPump pump = new CentrifugalPump
 7     (260, 0.03, 250, 0.03, CentrifugalPump.ControlMethod.ConstantPressureWithInverter, 40);
 8   PumpSystem ps = new PumpSystem(pump, 250, 0.09, 40, 3);
 9   double[] flows = new double[4];
10
11   using (StreamWriter sWriter = new StreamWriter
12     ("CircuitNetworkTest2.csv", false, Encoding.GetEncoding("Shift_JIS")))
13   {
14     sWriter.WriteLine("負荷率, ポンプ揚程[kPa], 消費電力[kW]");
15     sWriter.WriteLine("負荷率均等低減");
16     for (int i = 0; i <= 30; i++)
17     {
18       //負荷率を均等に低減
19       double pl = 1.0 - 0.03 * i;
20       sWriter.Write(pl);
21       ps.TotalFlowRate = pl * 0.09;
22       for (int j = 0; j < flows.Length; j++) flows[j] = pl * 0.0225;
23       CircuitNetworkTest2_DDCCntrl(cnetUStrm, cnetDStrm, cnetDDC, flows);
24       ps.PressureSetpoint = cnetDDC.Nodes[1].Pressure;
25       ps.UpdateState();
26       sWriter.WriteLine(", " + cnetDDC.Nodes[1].Pressure + ", " + ps.GetElectricConsumption());
27     }
28     sWriter.WriteLine("負荷率不均等低減 1");
29     for (int i = 0; i <= 30; i++)
30     {
31       //手前側の AHU から負荷を低減
32       double pl = 4 * 0.03 * i;
33       sWriter.Write(1 - pl / 4);
```

```
34        ps.TotalFlowRate = (1 - pl / 4) * 0.09;
35        for (int j = 0; j < 4; j++)
36        {
37          flows[j] = Math.Max(0.1, 1 - pl);
38          pl = pl - (1 - flows[j]);
39          flows[j] *= 0.0225;
40        }
41        CircuitNetworkTest2_DDCCntrl(cnetUStrm, cnetDStrm, cnetDDC, flows);
42        ps.PressureSetpoint = cnetDDC.Nodes[1].Pressure;
43        ps.UpdateState();
44        sWriter.WriteLine(", " + cnetDDC.Nodes[1].Pressure + ", " + ps.GetElectricConsumption());
45      }
46      sWriter.WriteLine("負荷率不均等低減 2");
47      for (int i = 0; i <= 30; i++)
48      {
49        //末端側の AHU から負荷を低減
50        double pl = 4 * 0.03 * i;
51        sWriter.Write(1 - pl / 4);
52        ps.TotalFlowRate = (1 - pl / 4) * 0.09;
53        for (int j = 3; 0 <= j; j--)
54        {
55          flows[j] = Math.Max(0.1, 1 - pl);
56          pl = pl - (1 - flows[j]);
57          flows[j] *= 0.0225;
58        }
59        CircuitNetworkTest2_DDCCntrl(cnetUStrm, cnetDStrm, cnetDDC, flows);
60        ps.PressureSetpoint = cnetDDC.Nodes[1].Pressure;
61        ps.UpdateState();
62        sWriter.WriteLine(", " + cnetDDC.Nodes[1].Pressure + ", " + ps.GetElectricConsumption());
63      }
64    }
65 }
66
67 private static void CircuitNetworkTest2_DDCCntrl
68    (CircuitNetwork cnetUStrm, CircuitNetwork cnetDStrm, CircuitNetwork cnetDDC, double[] flows)
69 {
70    CircuitNode[] ndUStrm = cnetUStrm.Nodes;
71    CircuitNode[] ndDStrm = cnetDStrm.Nodes;
72    CircuitNode[] ndDDC = cnetDDC.Nodes;
73    CircuitNode[] ndBND_CNST = new CircuitNode[]
74    { ndUStrm[5], ndUStrm[6], ndUStrm[7], ndUStrm[8], ndDStrm[1], ndDStrm[2], ndDStrm[3], ndDStrm[4] };
75    CircuitNode[] ndBND_DDC = new CircuitNode[]
76    { ndDDC[3], ndDDC[6], ndDDC[9], ndDDC[12], ndDDC[4], ndDDC[7], ndDDC[10], ndDDC[13] };
77    Regulator reg = new Regulator(340, 100, 0.5);
78
79    //AHU の流量を設定
80    double sum = 0;
81    for (int i = 0; i < 4; i++)
82    {
83      sum += flows[i];
84      ndBND_CNST[i].Inflow = ndBND_DDC[i].Inflow = -flows[i];
85      ndBND_CNST[i + 4].Inflow = ndBND_DDC[i + 4].Inflow = flows[i];
86    }
87    ndUStrm[0].Inflow = ndDDC[1].Inflow = sum;
88    ndDStrm[0].Inflow = ndDDC[0].Inflow = -sum;
89
90    //吐出圧一定制御で計算
91    cnetUStrm.SetBasePressure(ndUStrm[0], 250);
92    cnetDStrm.SetBasePressure(ndDStrm[0], 0);
93    cnetUStrm.Solve();
94    cnetDStrm.Solve();
95
96    //二方弁開度を計算//最大負荷系統を特定
97    int index = 0;
98    double max = 0;
99    for (int i = 0; i < 4; i++)
100   {
101     reg.UpdateLift(ndBND_CNST[i].Pressure - ndBND_CNST[i + 4].Pressure);
102     if (max < reg.Lift)
103     {
104       index = i;
105       max = reg.Lift;
106     }
107   }
108
109   //最大負荷系統に開度 100%の二方弁を設定して計算
110   reg.Lift = 1.0;
111   cnetDDC.ConnectNode(reg, ndBND_DDC[index], ndBND_DDC[index + 4]);
112   ndBND_DDC[index].Inflow = ndBND_DDC[index + 4].Inflow = 0;
113   cnetDDC.SetBasePressure(ndDDC[0], 0);
```

```
115   cnetDDC.Solve();
116   cnetDDC.RemoveBranch(reg);
117 }
118
119 private static void makePumpNetwork2(out CircuitNetwork cnetDDC)
120 {
121   double lcf = 1.5; //局部抵抗係数
122
123   cnetDDC = new CircuitNetwork();
124   WaterPipe ch01_02_DDC = new WaterPipe(30 * lcf, 0.150, WaterPipe.Material.CarbonSteel);
125   WaterPipe ch02_05_DDC = new WaterPipe(5 * lcf, 0.100, WaterPipe.Material.CarbonSteel);
126   WaterPipe ch01_08_DDC = new WaterPipe(70 * lcf, 0.150, WaterPipe.Material.CarbonSteel);
127   WaterPipe ch08_11_DDC = new WaterPipe(5 * lcf, 0.100, WaterPipe.Material.CarbonSteel);
128   WaterPipe ch04_00_DDC = new WaterPipe(30 * lcf, 0.150, WaterPipe.Material.CarbonSteel);
129   WaterPipe ch07_04_DDC = new WaterPipe(5 * lcf, 0.100, WaterPipe.Material.CarbonSteel);
130   WaterPipe ch10_00_DDC = new WaterPipe(70 * lcf, 0.150, WaterPipe.Material.CarbonSteel);
131   WaterPipe ch13_10_DDC = new WaterPipe(5 * lcf, 0.100, WaterPipe.Material.CarbonSteel);
132   SimpleCircuitBranch ch02_03_ahu1_DDC = new SimpleCircuitBranch(0.0225, 150);
133   SimpleCircuitBranch ch05_06_ahu2_DDC = new SimpleCircuitBranch(0.0225, 150);
134   SimpleCircuitBranch ch08_09_ahu3_DDC = new SimpleCircuitBranch(0.0225, 150);
135   SimpleCircuitBranch ch11_12_ahu4_DDC = new SimpleCircuitBranch(0.0225, 150);
136   CircuitNode[] nodesDDC = new CircuitNode[14];
137   for (int i = 0; i < nodesDDC.Length; i++) nodesDDC[i] = cnetDDC.AddNode();
138   cnetDDC.ConnectNode(ch01_02_DDC, nodesDDC[1], nodesDDC[2]);
139   cnetDDC.ConnectNode(ch02_05_DDC, nodesDDC[2], nodesDDC[5]);
140   cnetDDC.ConnectNode(ch01_08_DDC, nodesDDC[1], nodesDDC[8]);
141   cnetDDC.ConnectNode(ch08_11_DDC, nodesDDC[8], nodesDDC[11]);
142   cnetDDC.ConnectNode(ch04_00_DDC, nodesDDC[4], nodesDDC[0]);
143   cnetDDC.ConnectNode(ch07_04_DDC, nodesDDC[7], nodesDDC[4]);
144   cnetDDC.ConnectNode(ch10_00_DDC, nodesDDC[10], nodesDDC[0]);
145   cnetDDC.ConnectNode(ch13_10_DDC, nodesDDC[13], nodesDDC[10]);
146   cnetDDC.ConnectNode(ch02_03_ahu1_DDC, nodesDDC[2], nodesDDC[3]);
147   cnetDDC.ConnectNode(ch05_06_ahu2_DDC, nodesDDC[5], nodesDDC[6]);
148   cnetDDC.ConnectNode(ch08_09_ahu3_DDC, nodesDDC[8], nodesDDC[9]);
149   cnetDDC.ConnectNode(ch11_12_ahu4_DDC, nodesDDC[11], nodesDDC[12]);
150   cnetDDC.SetBasePressure(nodesDDC[0], 0);
151 }
```

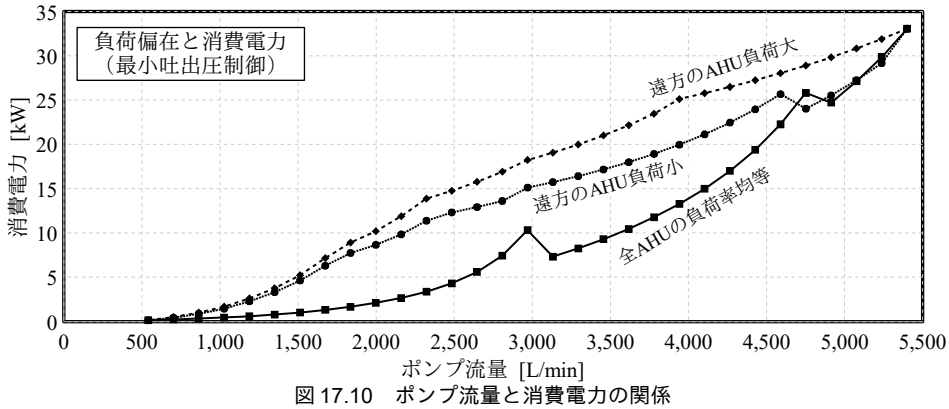


図 17.10　ポンプ流量と消費電力の関係

【第 17 章 記号表】

A_f	：流路断面積 [m²]	L_p	：周長 [m]
A_{ovl}	：オーバルダクトの断面積 [m²]	P_{ovl}	：オーバルダクトの周長 [m]
d_e	：等価直径 [m]	ΔP	：差圧 [kPa]
d_i	：流路の内径 [m]	Q	：体積流量 [m³/s]
d_{th}	：配管厚 [m]	R_a	：レンジアビリティ [-]
K	：流動抵抗係数 [kPa/(m³/s)²]	v	：平均流速 [m/s]
K_0	：全開抵抗 [kPa/(m³/s)²]	W_L	：制御弁のリニア特性の重み係数 [-]
l	：管長 [m]	ε	：絶対粗度 [m]
l_{mj}	：楕円の長径 [m]	λ	：管摩擦係数 [-]
l_{mn}	：楕円の短径 [m]	ρ	：流体密度 [kg/m³]
L_f	：バルブ・ダンパの開度	ζ	：局部抵抗係数 [-]

【第 17 章 参考文献】

17.1） Hardy Cross: Analysis of flow in networks of conduits or conductors, Bulletin No.286, Engineering experiment station, University of Illinois, Nov. 1936

17.2） ASHRAE Handbook Fundamentals 1997, Chapter 32 Duct Design

17.3） Huebscher, R.G., Friction equivalents for round, square and rectangular ducts, ASHVE Transactions, Vol.54, pp101-118, 1948

17.4） Heyt, J.W. and M.J. Diaz., Pressure drop in flat-oval spiral air duct, ASHRAE Transactions, Vol.81, pp.221-232, 1975

17.5） 坂東修: Excel で解く配管とポンプの流れ, 株式会社工業調査会, 2008

17.6） 井上宇市: ダクト設計施工便覧, 丸善, 1980

17.7） 建築設備設計基準, 国土交通省大臣官房官庁営繕部設備・環境課監修, 第 4 編 空気調和設備 第 5 章 ダクト設備

17.8） 奥山博康, 建築物の熱回路網モデルに関する理論的研究, 早稲田大学 博士論文, 1987

第18章　太陽エネルギー利用設備
(Solar Collector)

18.1　概要

太陽エネルギーを熱エネルギーに変換する集熱器や電気エネルギーに変換する太陽電池を総称して太陽エネルギー利用設備と呼ぶ。太陽エネルギーは海水熱や河川水熱などの自然エネルギー、あるいは下水熱、地下鉄排熱などの未利用エネルギーとは異なり、立地に殆ど影響を受けない。従って、太陽エネルギー利用設備はほとんどすべての建築で活用可能性があり、その特性を把握して導入の是非について検討を行う必要がある。

太陽熱集熱器の太陽エネルギー変換効率は温水や屋外の条件に大きく左右されるが、一般的に太陽電池よりも高い。一方、太陽電池で得られる電気エネルギーは汎用性が高く、熱エネルギーに比較すると質が高い。従って、エネルギーの質を重視して電気エネルギーに変換するのか、エネルギーの量を重視して熱エネルギーに変換するのか、検討が求められる。設計としては、熱としての直接的な使い道がある限りは変換効率の高い太陽熱集熱器を優先し、残余について太陽電池パネルを導入するという方針になろう。

本章では、太陽熱集熱器と太陽電池のそれぞれの計算方法を示す。空調や給湯に使われることの多い集熱器は、屋根などに固定して使用することを前提とする非集光固定型集熱器であり、平板型集熱器および真空ガラス管型集熱器が代表的である。本書では両者に対応可能な特性モデルを示し、平板型集熱器に関しては積み上げ方式による物理モデルの計算方法も開設する。太陽電池に関しては JIS 8907 に従った計算方法を示す。

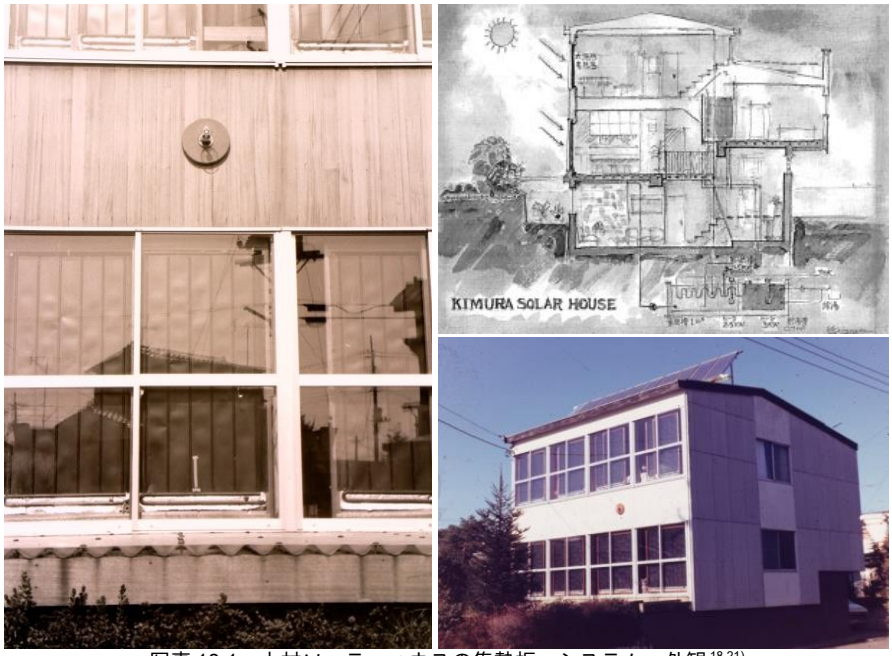


写真 18.1　木村ソーラーハウスの集熱板・システム・外観[18.21)]

18.2　理論

18.2.1　太陽熱集熱器の特性

集熱器の集熱効率特性を図 18.1 に示す。縦軸は集熱効率 η [-]であり、傾斜面入射 [W/m^2]の内、熱に転化できた割合である。横軸 L [K/(W/m^2)]は平均集熱温度 $T_{f,m}$ [°C]と周囲温度 T_a [°C]との差を傾斜面入射 I_{SC} [W/m^2]（太陽エネルギー利用設備について話す場合には日射強度とも呼ぶ）で除した値である。入射を一定とすると、周辺環境との温度差が大きい程に周辺環境への熱損失が拡大するため、このような観点から特性式が作られている。

代表的な集熱器として平板型集熱器と真空ガラス管型集熱器があり、図 18.1 には両集熱器の特性が描かれている。平板型集熱器は真空ガラス管型と比較して断熱性能は低いが、日射を受ける面積を大きく取ることができる。従って周辺環境との温度差が小さい場合には効率が高くなる。逆に真空ガラス管型は受熱面積は小さいが断熱性能に優れるため、高温で温水を製造した場合にも効率低下が小さい。図 18.1 の特性式はこのような特徴を反映して交点を持つことがわかる。

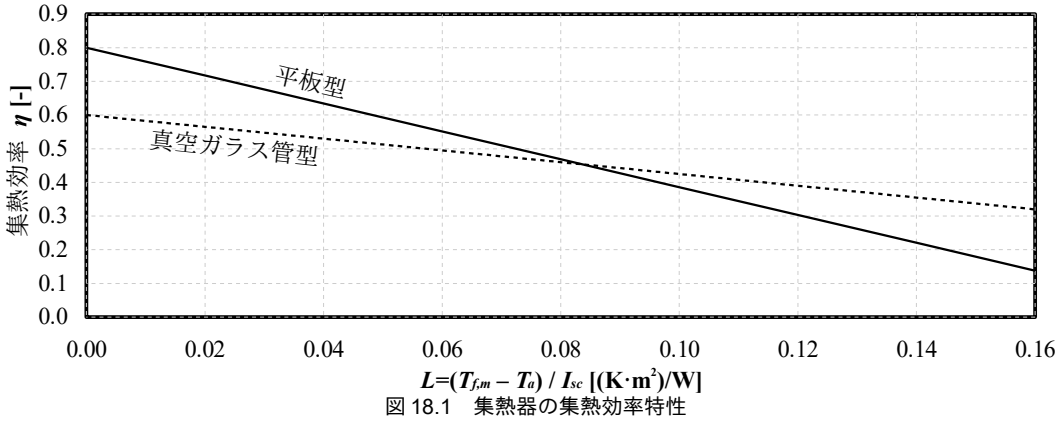


図 18.1　集熱器の集熱効率特性

図 18.1 の特性式は一次式（$\eta = a\,L+b$）で表現することができる。メーカー技術資料等から求めた特性係数の例を表 18.1 に示す。特に真空ガラス管型は多くのメーカーがあり構造もそれぞれ異なるため、検討対象の機器メーカーから都度、特性係数を入手することが望ましい。

表 18.1　集熱器の機器効率特性係数

係数	a	b
平板型集熱器	-4.14	0.8
真空ガラス管型	-1.75	0.6

集熱器の基礎式を 18.1~18.3に示す。

式 18.1 は集熱器内部での流体の昇温幅 ΔT_f [K]を示しており、Q_u [W]は集熱量、c_p [kJ/(kg·K)]は流体の定圧比熱、m_f [kg/s]は流体の質量流量である。式 18.2 は特性式に基づく集熱効率 η [-]の計算であり、T_{fi} [°C]は流体入口温度、T_a [°C]は周囲温度である。式 18.3 は集熱量 Q_u の計算であり、A_c [m^2]は集熱面積、I_{sc} [W/m^2]は傾斜面入射量である。

$$\Delta T_f = 0.001 \frac{Q_u}{c_p m_f} \tag{18.1}$$

$$\eta = a\left(T_{fi} + \Delta T_f / 2 - T_a\right) + b \tag{18.2}$$

$$Q_u = A_c \eta\, I_{sc} \tag{18.3}$$

計算に必要な値の内、傾斜面入射量 I_{sc}、集熱面積 A_c、周囲温度 T_a に関しては境界条件として与え

られることが通常である。なお、傾斜面入射量の計算については第 6 章で解説した。

集熱器の検討にあたっては、出入口温度差 ΔT_f が与えられた場合の循環可能流量 m_f の計算、あるいは流体質量流量 m_f が与えられた場合の成り行きでの流体出口温度の計算が必要となる。前者に関しては式 18.2 を用いて集熱効率 η を求めた後に式 18.3 を用いて集熱量を求め、これを式 18.1 に代入すれば m_f が計算できる。また、式 18.1~18.3 を出入口温度差について整理すると式 18.4 が得られる。後者に関しては式 18.4 によって求めた温度差 ΔT_f を入口流体温度 T_{fi} に加算することで計算できる。

$$\Delta T_f = \frac{0.001 A_c \left[a (T_{fi} - T_a) + b I_{sc} \right]}{c_p m_f - 0.0005 a A_c} \tag{18.4}$$

18.2.2　平板型集熱器の理論

平板型集熱器の理論に関しては Beckman と Duffie による図書[18.1]および種村による解説[18.5) 18.6)]が詳しい。以下は同書にもとづく計算法を基礎としている。

1）断面構造

図 18.2 に平板型集熱器の一般的な断面構造を示す。日射熱を取得するために日射吸収率の高い素材の集熱板が敷かれ、集熱板の熱を熱媒体に移動させるために、適当なピッチで集熱媒体流管が設けられる。熱媒体は通常は水であるが、冬期の凍結を防ぐために不凍液を循環させる例もある。外部への熱損失を防ぐという目的のため、集熱板の裏側には断熱材が設置される。一方、表側は日射透過率の高い素材（透過ガラス）で覆うことで、日射熱取得を阻害せずに対流による熱損失を抑制する。

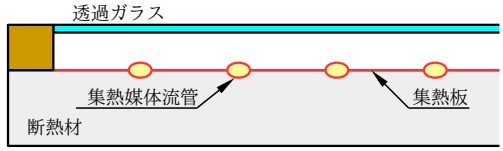


図 18.2　平板型集熱器の一般的な断面構造

2）集熱量の基礎式

太陽熱集熱器のパネル面へ入射する熱は、その全てが熱媒体へ移動するのではなく、いくらかは外部へ漏洩する。パネル面が吸収する日射量を S_p [W/m²]とすると、熱媒体へ有効に伝達できる熱量 Q_u[W]は式 18.5 で表現できる。T_p [°C]と T_a [°C]はパネルと外気の温度である。

$$Q_u = A_c \left(S_p - U_L (T_p - T_a) \right) \tag{18.5}$$

U_L [W/(m²·K)]は外界への熱損失係数であり、その具体的な導出方法が問題であるが、図 18.3 に示す熱回路網として捉えれば、式 18.6 で計算できる。

$$U_L = \frac{U_{p-g} \cdot U_{g-a}}{U_{p-g} + U_{g-a}} + h_{p-b} \tag{18.6}$$

$$U_{g-a} = h_w + h_{r,g-a} (T_g - T_{sky}) / (T_g - T_a) \tag{18.7}$$

$$U_{p-g} = h_{c,p-g} + h_{r,p-g} \tag{18.8}$$

式 18.5 はガラス面での日射吸収 S_g [W/m²]を無視して 0 とした場合の結果であるが、これを考慮するとガラス面の温度が上昇する。このガラス面温度の上昇に伴う集熱量 Q_u [W]のずれを評価すると式 18.5 は式 18.9 となる。

$$Q_u = A_c \left(S_p + S_g \frac{U_{p-g}}{U_{p-g} + U_{g-a}} - U_L (T_p - T_a) \right) \tag{18.9}$$

実際に計算を行う際にはパネルの平均的な温度である T_p よりもチューブ内の流体の入口温度 $T_{f,i}$ [K] で集熱量 Q_u を表現できた方が便利である。Beckman によれば、集熱器熱除去因子 F_R を用いることで式 18.9 を式 18.10 と書き改めることができる。

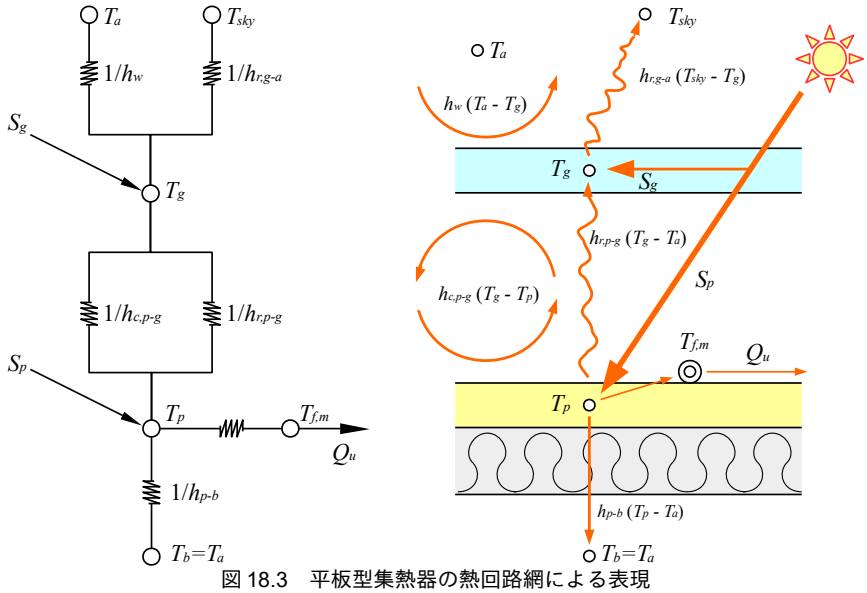


図 18.3　平板型集熱器の熱回路網による表現

$$Q_u = A_c F_R \left(S_p + S_g \frac{U_{p-g}}{U_{p-g} + U_{g-a}} - U_L (T_{f,i} - T_a) \right) \tag{18.10}$$

集熱器熱除去因子 F_R は次の 2 つの温度分布の影響をモデルに表現するための係数である。1）集熱板の熱伝導の制約により熱媒体の流れと直交する方向に生じる温度分布（熱媒体流管部分で温度は最低となり、管から離れるに連れて上昇する）。2）熱取得による熱媒体が集熱器内で昇温し、流れと並行する方向に生じる温度分布（熱媒入口側で温度は最低となり、出口に向かうに連れて上昇する）。集熱器熱除去因子 F_R [-]はフィン効率を用いて式 18.11 で計算できる。

$$F_R = \frac{m_f c_p}{A_c U_L} \left[1 - e^{-\left(A_c U_L F' / m_f c_p \right)} \right] \tag{18.11}$$

F' [-]は円管と平板で構成される集熱器のフィン効率であり、式 18.12 で計算する[†1]。

$$F' = \frac{1/U_L}{W \left[\dfrac{1}{U_L [d_o + (W - d_o) F]} + \dfrac{1}{C_B} + \dfrac{1}{\pi d_i h_{f,i}} \right]} \tag{18.12}$$

d_i [m], d_o [m], W [m]はそれぞれ集熱媒体流管の内径、外径、設置間隔であり、図 18.4 に示す通りである。C_B は流管とパネルの間の接着剤部分の熱通過率であるが、流管とパネルが一体化した伝熱性の高い製品[†2]に関しては本項を無視（$1/C_B = 0$）しても大きな誤差は生まれないだろう。$h_{f,i}$ は熱媒体流管の内部の対流熱伝達率である。F [-]は四角フィンのフィン効率であり、式 18.13、式 18.14 で計算する。ここで k と δ はそれぞれパネルの熱伝導率 [W/(m·K)]と厚み [m]である。

$$F = \frac{\tanh \left[m (W - d_i)/2 \right]}{m (W - d_i)/2} \tag{18.13}$$

†1　Beckman の著書には平板と円管の構成に応じたフィン効率の計算方法がいくつか提示されている。

†2　ロールボンド法と呼ばれる製造方法である。流管部分に圧着防止材を設置した上で二枚のプレートを重ねて圧延して一体化し、その後、圧着防止材部分に高圧空気を吹き込んで膨らませることで流路を作る。

$$m=\sqrt{U_L/k\delta} \tag{18.14}$$

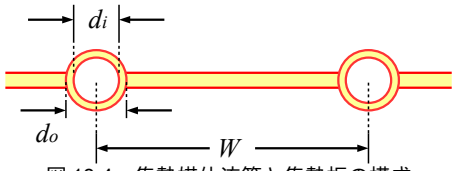


図 18.4　集熱媒体流管と集熱板の構成

3）係数の算出方法

上記の基礎式内の各種の係数の具体的な算出方法を記す。

・ガラスの外側表面の対流熱伝達率 h_w [W/(m²·K)]

外界に面しているため、外部風速の影響により値が変わる。木村らによる実験式[18.2) 18.3)]を式 18.15 に、McAdams による実験式[18.4)]を式 18.16 に示す。v は風速[m/s]である。

$$h_w=4.7+7.6v \tag{18.15}$$

$$h_w=5.7+3.8v \tag{18.16}$$

・ガラスの外側表面の放射熱伝達率 $h_{r,g\text{-}a}$ [W/(m²·K)]

天空を完全黒体（放射定数=1.0）とし、放射温度を T_{sky} [K]とすると式 18.17 で計算できる。ただし σ は黒体の放射定数で 5.67×10⁻⁸ W/(m²·K⁴)である。また、ε_g [-]はガラス面の長波長放射率である。近年の製品は、パネルからの熱損失量を低減することを目的に金属表面処理により放射率を下げており、0.05~0.2 程度の数値をとる[18.7)]。

$$h_{r,g-a}=4\sigma\varepsilon_g\left(\frac{T_{sky}+T_g}{2}\right)^3 \tag{18.17}$$

・パネルとガラスの内側表面の放射熱伝達率 $h_{r,p\text{-}g}$ [W/(m²·K)]

式 18.18 で計算する。ただし、σ_{pg} は有効放射定数であり、パネルとガラスは平行平板であるとして式 18.19 で計算する。また、ε_p はパネルの長波長放射率[-]であるが、素材にブラックステンレスやブラッククロームなどの選択吸収面[†1)]を用いることで 0.1 程度に抑えている製品がある。

$$h_{r,p-g}=4\sigma_{pg}\left(\frac{T_p+T_g}{2}\right)^3 \tag{18.18}$$

$$\sigma_{pg}=\frac{\sigma}{1/\varepsilon_p+1/\varepsilon_g-1} \tag{18.19}$$

・パネルとガラスの内側表面の対流熱伝達率 $h_{c,p\text{-}g}$ [W/(m²·K)]

式 18.20 で計算する。ただし、λ_a [W/(m·K)]は空気の熱伝導率、L_p [m]は空気層の厚みである。空気の熱伝導率 λ_a の計算方法は第 4 章で示した。Nu はヌセルト数であるが、CSD により自然対流を抑制し Nu=1.0 として運用可能な製品がある。

$$h_{c,p-g}=(\lambda_a/L_p)Nu \tag{18.20}$$

・パネルから集熱器背面への熱通過率 $h_{p\text{-}b}$ [W/(m²·K)]

断熱材の厚みを l [m]、熱伝導率を λ_i [W/(m·K)]として式 18.21 で計算する。

$$h_{p-b}=\lambda_i/l \tag{18.21}$$

†1　2μm 未満の短波長の吸収率が高く日射の吸収性能に優れる一方で、2μm 以上の中・長波長の吸収率が低く、吸収した熱を逃がしづらい素材を言う。

・熱媒体流管の内部の対流熱伝達率 h_{fi} [W/(m²·K)]

管内流速や無次元数を用いて導出する。詳細に関しては第 3 章 例題 3.1 を参照のこと。

・ガラス面での日射吸収 S_g [W/m²]とパネル面での日射吸収 S_p [W/m²]

図 18.5 に示す通り、ガラス面に入射した日射エネルギーは反射と透過を繰り返して一部は外界に逃げ、一部はガラスに吸収され、一部はパネル面に吸収される。無限回の反射と吸収を繰り返した結果、ガラス面に吸収される日射量とパネル面に吸収される日射量は式 18.22~18.25で計算できる。ここで、$\alpha_{T,g}$ と $\alpha_{T,p}$ は総合吸収率[-]、τ_g は日射透過率[-]、α_g と α_p は日射吸収率[-]、ρ_g と ρ_p は日射反射率[-]である。添字の g と p はそれぞれガラスとパネルを表している。

$$S_g = I_{DN} \cdot \alpha_{T,g} \tag{18.22}$$

$$S_p = I_{DN} \cdot \alpha_{T,p} \tag{18.23}$$

$$\alpha_{T,g} = \alpha_g + \tau_g \rho_p \alpha_g + \tau_g \rho_g \rho_p^2 \alpha_g + \cdots = \alpha_g \left(1 + \frac{\tau_g \rho_p}{1 - \rho_g \rho_p} \right) \tag{18.24}$$

$$\alpha_{T,p} = \tau_g \alpha_p + \tau_g \rho_g \rho_p \alpha_p + \tau_g \rho_g^2 \rho_p^2 \alpha_p + \cdots = \frac{\tau_g \alpha_p}{1 - \rho_g \rho_p} \tag{18.25}$$

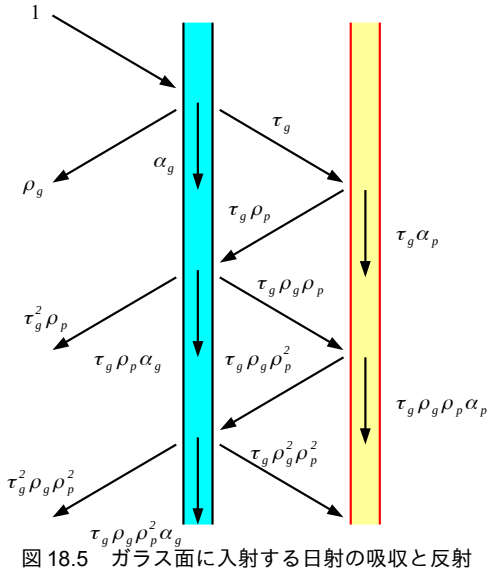


図 18.5　ガラス面に入射する日射の吸収と反射

ガラスの日射透過率、日射反射率、日射吸収率は入射角度への依存性があり、入射角が θ [°]の場合の日射透過率 $\tau_g(\theta)$、日射反射率 $\rho_g(\theta)$、日射吸収率 $\alpha_g(\theta)$はそれぞれ式 18.26~式 18.28 で計算できる。ただし、$\tau(0)$と $\rho(0)$は垂直入射に対する日射透過率と日射反射率である。また、$\tau_n(\theta)$と $\rho_n(\theta)$は入射角 θ における規準化透過率と規準化反射率であり、入射角の余弦 $\cos\theta$ の関数として式 18.29 と式 18.30 で計算できる。ここで、m_i は近似係数であり、透明単板ガラスの場合の値を表 18.2 に示す[18.11) 18.10)]。

$$\tau_g(\theta) = \tau_n(\theta)\tau(0) \tag{18.26}$$

$$\rho_g(\theta) = 1 - \rho_n(\theta)(1 - \rho(0)) \tag{18.27}$$

$$\alpha_g(\theta) = 1 - (\tau_g(\theta) + \rho_g(\theta)) \tag{18.28}$$

$$\tau_n(\theta) = \sum_{i=1}^{5} m_{i,\tau} \cdot \cos^i\theta \tag{18.29}$$

$$\rho_d=\left(2\sum_{n=1}^{5}\frac{m_{n,\rho}}{n+2}\right)\rho(0)=0.939\,\rho(0) \tag{18.30}$$

表 18.2　透明単板ガラスの入射角特性近似係数

係数	m_1	m_2	m_3	m_4	m_5
日射透過率	2.552	1.364	-11.388	13.617	-5.146
日射反射率	5.189	-12.392	16.593	-11.851	3.461

天空日射に関する日射透過率と日射反射率も、垂直日射に対する日射透過率および日射反射率との比率で式 18.31 と式 18.32 で計算することができる[18.15)]。

$$\tau_d=\left(2\sum_{n=1}^{5}\frac{m_{n,\tau}}{n+2}\right)\tau(0)=0.897\,\tau(0) \tag{18.31}$$

$$\rho_d=\left(2\sum_{n=1}^{5}\frac{m_{n,\rho}}{n+2}\right)\rho(0)=0.939\,\rho(0) \tag{18.32}$$

パネルの日射反射率 $\rho_p(\theta)$と日射吸収率 $\alpha_p(\theta)$は入射角 θ [°]の余弦 cos(θ)と線形関係にあると仮定して式 18.33 と式 18.34 で計算する。

$$\alpha_p(\theta)=\alpha_p(0)\cos\theta \tag{18.33}$$

$$\rho_p(\theta)=1-\alpha_p(\theta) \tag{18.34}$$

【例題 18.1】

例題 6.1 と例題 6.2 の条件で設置された太陽熱集熱器（設置角度=35°）についてパネル面での日射吸収 S_p とガラス面での日射吸収 S_g を計算せよ。ただし、垂直入射時のガラスの日射透過率 $\tau_g(0) = 0.90$、日射反射率 $\rho_g(0) = 0.08$、垂直入射時のパネルの日射吸収率 $\alpha_p(0) = 0.93$ とする。なお、法線面直達日射量は 1,000W/m^{2}とする。

【解】

例題 6.2 の結果により $\cos\theta = 0.916$ である。式 18.26~18.30を用いて、ガラスの日射透過率 τ_g、日射反射率 ρ_g、日射吸収率 α_gはそれぞれ、

τ_g = {0 + 0.916 × 2.552 + 0.916^2 × 1.364 + 0.916^3 × (-11.388) + 0.916^4 × 13.617
+ 0.916^5 × (-5.146)} × 0.90
= {2.338 + 1.144 – 8.753 + 9.587 – 3.319} × 0.90 = 0.898

ρ_g = {1.000 + 0.916 × (-5.189) + 0.916^2 × 12.392 + 0.916^3 × (-16.593) + 0.916^4 × 11.851
+ 0.916^5 × (-3.461)} × (1 – 0.08) + 0.08
= {1.000 - 4.753 + 10.398 – 12.753 + 8.343 – 2.232} × 0.92 + 0.08 = 0.083

α_g = 1 – (0.898 + 0.083) = 0.019

となる。

式 18.33 と式 18.34 により、

α_p = 0.93 × 0.916 = 0.852

ρ_p = 1 – 0.852 = 0.148

となる。

式 18.22~式 18.25 により、ガラス面とパネル面それぞれでの総合吸収率は、

$\alpha_{T,p}$ = (0.898 × 0.852) ÷ (1 – 0.083 × 0.148) = 0.765 ÷ 0.988 = 0.774

$\alpha_{T,g}$ = 0.019 × (1 + (0.898 × 0.148) ÷ (1 – 0.083 × 0.148)) = 0.019 × (1 + 0.134) = 0.022

となる。総合吸収率に法線面直達日射量を乗じ、

S_p = 0.774 × 1,000 = 774 W/m^2

S_g = 0.022 × 1,000 = 22 W/m^2

が得られる。

4）天空の仮想温度 T_{sky}の計算方法

パネル面と天空との放射熱移動を計算するためには、天空の仮想温度 T_{sky} を与える必要があるが、これは下記の概念を持った数値である。

宇宙の温度はほぼ 0 K であるため、宇宙からパネル面への放射は無視できるが、大気中にある水蒸気からの放射熱移動は考慮する必要がある。即ち、天空からの放射熱移動量 q_a [W/m^2]は大気中の水蒸気量に依存し、式 18.35 と式 18.36 で計算できる。ここで Br はブレントによる係数であり、外気温と等温の黒体からの放射量（σT_a^4）と、実際の天空からの放射量との比率である。Br は式 18.36 に示す通り、水蒸気分圧の平方根の 1 次関数として表すことができる[18.12)]。

$$q_a = Br \sigma T_a^4 \tag{18.35}$$

$$Br = 0.51 + 0.209\sqrt{P_w} \tag{18.36}$$

ここで係数 Br の効果を温度に表現し、仮想的な天空の温度を式 18.37 で定義すると、天空からの放射熱移動量は式 18.38 となり、取り扱いが容易になる。

$$T_{sky} = \sqrt[4]{Br}\, T_a \tag{18.37}$$

$$q_a = \sigma T_{sky}^4 \tag{18.38}$$

【例題 18.2】

外気温度: 5 °C、水蒸気分圧: 0.5kPa の時の天空の仮想温度 T_{sky} を計算せよ。

【解】

式 18.36 により

$Br = 0.51 + 0.209 \times 0.5^{0.5} = 0.658$

となる。式 18.37 により、

$T_{sky} = 0.658^{0.25} \times (273.15 + 5) = 0.9 \times 278.15 = 250$ K = -22.8 °C

となる。

5）集熱効率

太陽熱集熱器の集熱効率を式 18.39 で定義する。I_{SC} [W/m^2]は傾斜面へ入射する全天日射量である。

$$\eta = \frac{Q_u}{I_{SC}} \tag{18.39}$$

18.2.3　太陽電池パネルの特性

以下の計算は JIS 8907:「太陽光発電システムの発電電力量推定方法」に従ったものである[18.17)]。用語は JIS 8960:「太陽光発電用語」に従う[18.16)]。太陽電池の出力 P は式 18.40 で計算する[†1)]。

$$P = \frac{I_{sc}}{1000} \times K_{PT} \times P_M \times \eta_{INO} \tag{18.40}$$

通常、太陽光発電パネルの傾斜角は年間での総発電量を最大化するように計画されるが、この角度は設置地点の緯度にほぼ一致する。例えば東京であれば 35 度程度となり、日本においては太陽熱集熱器に比較するとやや浅い角度とすることが多い。P_M は基準条件における太陽電池の出力である。基準条件は JIS[18.17)]で定められており、温度は 25 °C、傾斜面日射強度は 1,000 W/m^2 である。

太陽電池から得られる電流は直流であるため、インバータを用いて交流に変換する必要がある。η_{INO} はインバータによる交直変換効率である。値が入手できない場合には 0.9 としておく。

K_{PT} はパネル温度 T_{PA} による補正係数であり、基準温度（=25 °C）からの温度上昇幅の関数として式 18.41 で表現される。α_{PT} は太陽電池の種別に応じた係数であり、アモルファス系の場合には 0.0020、結晶系の場合には 0.0041 とする。アニール現象によりアモルファス系の方が温度上昇に伴う出力低

†1　この式には太陽熱集熱器で考慮したような入射角特性が表現されていないが、西川らの実測調査[18.19)]によれば入射角が 60 度を超えるあたりから発電効率が急速に低下する。集熱器と同様に、90 度で反射率が 100%となるような入射角特性を仮定してこれを乗じるモデルとしても良いように思う。

下幅が小さい傾向となる。

$$K_{PT}=1-\alpha_{PT}(T_{PA}-25) \tag{18.41}$$

パネル温度 T_{PA} は、周囲の温度 T_A、傾斜面日射強度 I_{sc}、風速 v によって計算でき、式 18.42 で表される[18.17) 18.18)]。A および B は係数であり、太陽電池の設置方式によって表 18.3 の値を取る。

$$T_{PA}=T_A+\left(\frac{A}{Bv^{0.8}}+2\right)I_{sc}-2 \tag{18.42}$$

表 18.3　太陽電池設置方式と係数

設置方式	A	B
架台設置形	46	0.41
屋根置き形	50	0.38
屋根材形（裏面通風構造があるタイプ）	57	0.33

18.3　計算法

18.3.1　特性式にもとづく集熱器の計算

式 18.1~18.4を用いて特性式にもとづく計算を行う。プログラム 18.1 に定数・列挙型・インスタンス変数・プロパティの定義を示す。1~8 行に示す通り、真空ガラス管型と平板型を判別する列挙型を定義する。また、傾斜面の情報を計算するため、第 6 章で解説した Incline クラスのインスタンス（14 行）を用意する。

プログラム 18.1　定数・列挙型・インスタンス変数・プロパティの定義

Popolo.HVAC.SolarEnergy.SimpleSolarCollector class

```
 1 /// <summary>受熱面タイプ</summary>
 2 public enum HeatReceiver
 3 {
 4   /// <summary>真空ガラス管型</summary>
 5   VacuumTube,
 6   /// <summary>平板型</summary>
 7   FlatPlate
 8 }
 9
10 /// <summary>傾斜面情報</summary>
11 private Incline incline;
12
14 /// <summary>傾斜面情報を取得する</summary>
15 public ImmutableIncline Incline { get { return incline; } }
16
17 /// <summary>特性式の係数 A[W/K]を取得する</summary>
18 public double CoefficientA { get; private set; }
19
20 /// <summary>特性式の係数 B[-]を取得する</summary>
21 public double CoefficientB { get; private set; }
22
23 /// <summary>集熱効率[-]を取得する</summary>
24 public double Efficiency { get; private set; }
25
26 /// <summary>パネル表面積[m2]を取得する</summary>
27 public double SurfaceArea { get; private set; }
28
29 /// <summary>入口水温[C]を取得する</summary>
30 public double InletWaterTemperature { get; private set; }
31
32 /// <summary>出口水温[C]を取得する</summary>
33 public double OutletWaterTemperature { get; private set; }
34
35 /// <summary>水量[kg/s]を取得する</summary>
36 public double WaterFlowRate { get; private set; }
37
38 /// <summary>集熱量[kW]を取得する</summary>
39 public double HeatCollection
13 { get { return (OutletWaterTemperature - InletWaterTemperature) * WaterFlowRate * WATER_SPECIFIC_HEAT; } }
```

プログラム 18.2 にコンストラクタおよび初期化処理を示す。29~41 行が初期化処理であり、傾斜面情報、特性係数、表面積をプロパティに保存する。1~10 行がコンストラクタであり、特性係数を直

接指定する。また、12~27 行でオーバーロードを行い、表 18.1 の値を用いて平板型か真空ガラス管型かに応じて標準の値で初期化を行う。

プログラム 18.2　コンストラクタおよび初期化処理

Popolo.HVAC.SolarEnergy.SimpleSolarCollector class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="incline">傾斜面</param>
 3 /// <param name="surfaceArea">集熱面積[m2]</param>
 4 /// <param name="coefficientA">特性係数 A</param>
 5 /// <param name="coefficientB">特性係数 B</param>
 6 public SimpleSolarCollector
 7   (ImmutableIncline incline, double surfaceArea, double coefficientA, double coefficientB)
 8 {
 9   initialize(incline, surfaceArea, coefficientA, coefficientB);
10 }
11
12 /// <summary>インスタンスを初期化する</summary>
13 /// <param name="incline">傾斜面</param>
14 /// <param name="surfaceArea">集熱面積[m2]</param>
15 /// <param name="heatReceiver">受熱面タイプ</param>
16 public SimpleSolarCollector(ImmutableIncline incline, double surfaceArea, HeatReceiver heatReceiver)
17 {
18   switch (heatReceiver)
19   {
20     case HeatReceiver.FlatPlate:
21       initialize(incline, surfaceArea, -4.14, 0.8);
22       return;
23     default:
24       initialize(incline, surfaceArea, -1.75, 0.6);
25       return;
26   }
27 }
28
29 /// <summary>パラメータを初期化する</summary>
30 /// <param name="incline">傾斜面</param>
31 /// <param name="surfaceArea">集熱面積[m2]</param>
32 /// <param name="coefficientA">特性係数 A</param>
33 /// <param name="coefficientB">特性係数 B</param>
34 private void initialize
35   (ImmutableIncline incline, double surfaceArea, double coefficientA, double coefficientB)
36 {
37   this.incline = new Incline(incline);
38   this.CoefficientA = coefficientA;
39   this.CoefficientB = coefficientB;
40   this.SurfaceArea = surfaceArea;
41 }
```

プログラム 18.3 に循環可能水量の計算処理を示す。0 割りを防ぐために温度差が 0 以下の場合には停止状態とする（13~20 行）。22~32 行は式 18.2 にもとづく集熱効率の計算である。式 18.1 および式 18.3 にもとづき、35, 36 行で循環可能水量を計算する。40~52 行は傾斜面入射のかわりに第 6 章で解説した太陽のインスタンスを受け取るオーバーロードである。49 行で傾斜面入射を計算し、1~38 行のメソッドを呼び出す。

プログラム 18.3　循環可能水量の計算

Popolo.HVAC.SolarEnergy.SimpleSolarCollector class

```
 1 /// <summary>水量[kg/s]を計算する</summary>
 2 /// <param name="totalIrradiance">傾斜面入射量[W/m2]</param>
 3 /// <param name="waterTemperature">入口水温[C]</param>
 4 /// <param name="temperatureDifferrence">出入口温度差[K]</param>
 5 /// <param name="ambientTemperature">周囲温度[C]</param>
 6 /// <returns>水量[kg/s]</returns>
 7 public double GetWaterFlowRate
 8   (double totalIrradiance, double waterTemperature, double temperatureDifferrence, double ambientTemperature)
 9 {
10   //入口水温保存
11   InletWaterTemperature = waterTemperature;
12
13   //温度差が負または日射 0 の場合には停止
14   if (temperatureDifferrence <= 0 || totalIrradiance <= 0)
15   {
16     OutletWaterTemperature = InletWaterTemperature;
17     Efficiency = 0;
18     WaterFlowRate = 0;
```

```
19     return 0;
20   }
21
22   //出口水温[C]
23   OutletWaterTemperature = waterTemperature + temperatureDifferrence;
24
25   //平均集熱温度[C]
26   double tM = 0.5 * (InletWaterTemperature + OutletWaterTemperature);
27
28   //外気温度差[K]
29   double dT = tM - ambientTemperature;
30
31   //集熱効率[-]
32   Efficiency = Math.Max(0, CoefficientA * (dT / totalIrradiance) + CoefficientB);
33
34   //水量[kg/s]
35   WaterFlowRate = 0.001 * totalIrradiance * SurfaceArea
36     * Efficiency / (temperatureDifferrence * WATER_SPECIFIC_HEAT);
37   return WaterFlowRate;
38 }
39
40 /// <summary>水量[kg/s]を計算する</summary>
41 /// <param name="sun">太陽</param>
42 /// <param name="waterTemperature">入口水温[C]</param>
43 /// <param name="temperatureDifferrence">出入口温度差[K]</param>
44 /// <param name="ambientTemperature">周囲温度[C]</param>
45 /// <returns>水量[kg/s]</returns>
46 public double GetWaterFlowRate
47   (ImmutableSun sun, double waterTemperature, double temperatureDifferrence, double ambientTemperature)
48 {
49   double totalIrradiance = Incline.GetDirectSolarRadiationRate(sun) * sun.DirectNormalRadiation
50     + Incline.ConfigurationFactorToSky * sun.DiffuseHorizontalRadiation;
51   return GetWaterFlowRate(totalIrradiance, waterTemperature, temperatureDifferrence, ambientTemperature);
52 }
```

プログラム 18.4 に成り行きでの出口温度の計算処理を示す。流量が 0 の場合には運転を停止させる（10~17 行）。式 18.4 を用いて 19~22 行で出入口温度差を計算し、24 行でプログラム 18.3 を呼び出す。28~40 行は太陽のインスタンスを受け取るオーバーロードである。

プログラム 18.4　成り行きでの出口温度の計算

Popolo.HVAC.SolarEnergy.SimpleSolarCollector class

```
 1 /// <summary>出口温度[C]を計算する</summary>
 2 /// <param name="totalIrradiance">傾斜面入射量[W/m2]</param>
 3 /// <param name="waterTemperature">入口水温[C]</param>
 4 /// <param name="waterFlowRate">水量[kg/s]</param>
 5 /// <param name="ambientTemperature">周囲温度[C]</param>
 6 /// <returns>出口温度[C]</returns>
 7 public double GetOutletTemperature
 8   (double totalIrradiance, double waterTemperature, double waterFlowRate, double ambientTemperature)
 9 {
10   //流量が 0 の場合には停止
11   if (waterFlowRate <= 0)
12   {
13     OutletWaterTemperature = InletWaterTemperature;
14     Efficiency = 0;
15     WaterFlowRate = 0;
16     return 0;
17   }
18
19   //出入口温度差の計算
20   double dT = 0.001 * SurfaceArea *
21     (CoefficientA * (waterTemperature - ambientTemperature) + CoefficientB * totalIrradiance)
22     / (WATER_SPECIFIC_HEAT * waterFlowRate - 0.0005 * SurfaceArea * CoefficientA);
23
24   GetWaterFlowRate(totalIrradiance, waterTemperature, dT, ambientTemperature);
25   return OutletWaterTemperature;
26 }
27
28 /// <summary>出口温度[C]を計算する</summary>
29 /// <param name="sun">太陽</param>
30 /// <param name="waterTemperature">入口水温[C]</param>
31 /// <param name="waterFlowRate">水量[kg/s]</param>
32 /// <param name="ambientTemperature">周囲温度[C]</param>
33 /// <returns>出口温度[C]</returns>
34 public double GetOutletTemperature
35   (ImmutableSun sun, double waterTemperature, double waterFlowRate, double ambientTemperature)
36 {
```

```
37   double totalIrradiance = Incline.GetDirectSolarRadiationRate(sun) * sun.DirectNormalRadiation
38     + Incline.ConfigurationFactorToSky * sun.DiffuseHorizontalRadiation;
39   return GetOutletTemperature(totalIrradiance, waterTemperature, waterFlowRate, ambientTemperature);
40 }
```

【例題 18.3】

平板型集熱器と真空ガラス管型集熱器について、温度別（25°C, 55°C, 90°C）に各月の集熱効率を計算せよ。ただし設置角度は 30 度、地点は東京、集熱温度差は 3 °C とする。

【解】

プログラム 18.5 に計算処理を示す。3~7 行で平板型と真空ガラス管型の集熱器インスタンスを生成する。設置角度は 4 行で示す通り 30 度である。第 7 章で解説した気象クラスを用いて 9~12 行で 20 年間の気象データを作成する。16 行は集熱温度配列である。18~64 行で具体的な計算処理を行う。20 年間の平均値を求めるため、30 行以降で繰り返し処理を行う。32 行以降は年間 8760 時間の繰り返し処理である。34~36 行で直散分離処理を行い、Sun オブジェクトに設定する。また、38~40 行で傾斜面入射を計算する。42~50 行が集熱量の計算であり、温度差 3 °C を前提に循環可能水量を計算し、0 ではない場合に集熱量を積算する。55~62 行は書き出し処理であり、集熱量を傾斜面入射量で除することで各月の集熱効率を計算する。

計算結果をグラフ化した結果を図 18.6 に示す。集熱器タイプの如何によらず、集熱温度が低いほど効率が高いことが確認できる。また集熱温度が 25 °C の場合には平板型が真空ガラス管型よりも効率が高く、逆に 90 °C で高温集熱を行う場合には真空ガラス管型が平板型よりも効率が高いことがわかる。集熱温度 55°C の場合にはほぼ同等である。25 °C、55 °C、90 °C はそれぞれ、1）低温で集熱を行いヒートポンプで温水供給を行うソーラーヒートポンプ、2）直接に二次側へ温水搬送を行う直接利用システム、3）吸収冷凍機の再生器に供給するソーラークーリングを意図した温度帯であり、温水の使用目的に応じた集熱器のタイプの使い分けが必要であることがわかる。

プログラム 18.5　月別・集熱温度別・集熱器タイプ別の集熱効率の計算

```
 1 private static void SimpleSolarCollectorTest()
 2 {
 3   //集熱器インスタンスの作成
 4   Incline inc = new Incline(Incline.Orientation.S, 30d / 180d * Math.PI);
 5   SimpleSolarCollector[] scs = new SimpleSolarCollector[2];
 6   scs[0] = new SimpleSolarCollector(inc, 1, SimpleSolarCollector.HeatReceiver.FlatPlate);
 7   scs[1] = new SimpleSolarCollector(inc, 1, SimpleSolarCollector.HeatReceiver.VacuumTube);
 8
 9   //気象データの作成
10   double[] dbt, hrt, rad;
11   bool[] fair;
12   RandomWeather.MakeWeather(1, RandomWeather.Location.Tokyo, 20, out dbt, out hrt, out rad, out fair);
13   Sun sun = new Sun(35, 41.2, 0, 139, 45.9, 0, 135, 0, 0);
14
15   //集熱温度
16   double[] cTemp = new double[] { 25, 55, 90 };
17
18   //計算,書き出し処理
19   using (StreamWriter sWriter = new StreamWriter
20     ("SimpleSolarCollector.csv", false, Encoding.GetEncoding("Shift_JIS")))
21   {
22     //タイトル行
23     sWriter.Write("月, 傾斜面日射量[MJ/月]");
24     for (int i = 0; i < 3; i++) sWriter.Write(",平板型" + cTemp[i] + "C");
25     for (int i = 0; i < 3; i++) sWriter.Write(",真空ガラス管型" + cTemp[i] + "C");
26     sWriter.WriteLine();
27
28     DateTime dt = new DateTime(1999, 1, 1, 0, 0, 0);
29     double[,] sum = new double[12, scs.Length * cTemp.Length + 1];
30     for (int i = 0; i < 20; i++)
31     {
32       for (int j = 0; j < 8760; j++)
33       {
34         //太陽位置を更新して直散分離
35         sun.Update(dt);
36         sun.SeparateGlobalHorizontalRadiation(rad[8760 * i + j], Sun.SeparationMethod.Erbs);
37
38         //傾斜面日射量を積算
39         sum[dt.Month - 1, 0] += (inc.GetDirectSolarRadiationRate(sun) * sun.DirectNormalRadiation
40           + inc.ConfigurationFactorToSky * sun.DiffuseHorizontalRadiation) / 1000d;
41
42         //集熱量[Wh]を計算
43         for (int k = 0; k < scs.Length; k++)
44         {
45           for (int m = 0; m < cTemp.Length; m++)
```

```
46          {
47            if (0 != scs[k].GetWaterFlowRate(sun, cTemp[m], 3, dbt[8760 * i + j]))
48              sum[dt.Month - 1, k * cTemp.Length + m + 1] += scs[k].HeatCollection;
49          }
50        }
51        dt = dt.AddHours(1);
52      }
53      dt = dt.AddYears(-1);
54    }
55    //書き出し処理
56    for (int i = 0; i < 12; i++)
57    {
58      sWriter.Write(i + 1);
59      sWriter.Write("," + sum[i, 0] / 20d * 3.6); // kWh/20年→MJ/月に変換
60      for (int j = 1; j < sum.GetLength(1); j++) sWriter.Write("," + sum[i, j] / sum[i, 0]);
61      sWriter.WriteLine();
62    }
63  }
64 }
```

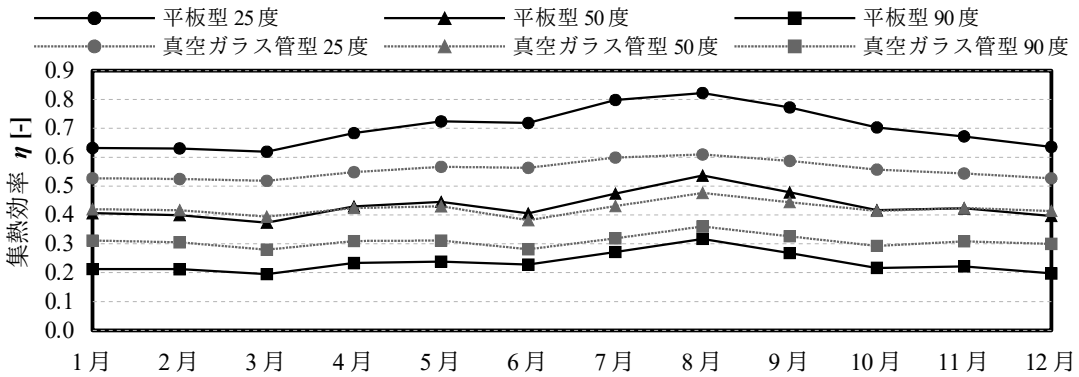


図 18.6　月別・集熱温度別・集熱器タイプ別の集熱効率

18.3.2　理論式にもとづく平板型集熱器の計算

式 18.10 を解くことで集熱量 Q_u が求まれば式 18.43 と式 18.44 によりパネルの平均温度 T_p と熱媒の平均温度 $T_{f,m}$ が計算できる[18.14]。また、パネル温度 T_p が得られれば式 18.45 によってガラス温度 T_g が計算できる。しかし、集熱量 Q_u を計算する過程で必要となる対流熱伝達率や放射熱伝達率を把握するためにはそもそもパネル温度 T_p やガラス温度 T_g が必要であり、計算が循環する。そこで、適当なパネル温度とガラス温度の初期値を設定し、反復計算によって平衡状態を求める。

$$T_p = T_{f,i} + \frac{Q_u / A_c}{U_L F_R}(1 - F_R) \tag{18.43}$$

$$T_{f,m} = T_{f,i} + \frac{Q_u / A_c}{U_L F_R}(1 - F_R / F') \tag{18.44}$$

$$T_g = \frac{S_g + U_{p-g} T_p + h_{r,g-a} T_{sky} + h_w T_a}{U_{p-g} + h_{r,g-a} + h_w} \tag{18.45}$$

図 18.7 に平板型集熱器の反復計算フローを示す。

未知変数はガラスの温度とパネルの温度である。まず、パネル温度を仮定してパネル温度のみに依存する各種の係数を計算する。次にガラス温度を仮定して式 18.17 や式 18.18 などのガラス温度に依存する係数を更新する。式 18.45 を用いてガラス温度を求め、当初仮定したガラス温度との差分を誤差として評価する。許容誤差未満であればガラス温度を確定し、許容誤差以上であればニュートン法でガラス温度を更新する。パネル温度とガラス温度がわかれば集熱量 Q_u を求めることができるため、式 18.43 を用いてパネル温度を計算する。仮定したパネル温度との差分を求め、許容誤差未満で

あれば計算を終了する。許容誤差以上であれば求められた新しいパネル温度を用いて最初の計算に戻る。以上の処理を繰り返し、パネル温度を収束させる。

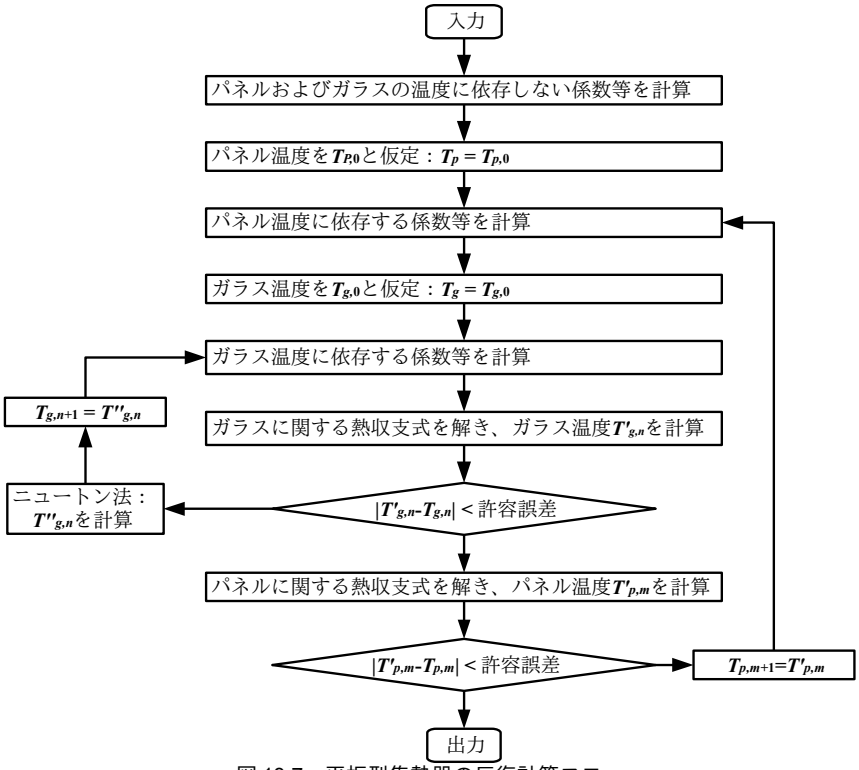


図 18.7　平板型集熱器の反復計算フロー

理論式にもとづく平板型太陽熱集熱器クラスの定数宣言をプログラムを 18.6 に示す。19~29 行の static コンストラクタを用いて式 18.31 と式 18.32 の初期化処理を行う。

プログラム 18.6　平板型集熱器の集熱量の計算

Popolo.HVAC.SolarEnergy.FlatPlateSolarCollector class

```
 1 /// <summary>黒体の放射定数[W/(m2K4)]</summary>
 2 private const double BLACK_CONSTANT = 5.67e-8;
 3
 4 /// <summary>絶対温度と摂氏との変換定数</summary>
 5 private const double CONVERT_C_TO_K = 273.15;
 6
 7 /// <summary>ガラスの透過率の入射角特性近似係数</summary>
 8 private static readonly double[] MI_TAU = new double[] { 2.552, 1.364, -11.388, 13.617, -5.146 };
 9
10 /// <summary>ガラスの反射率の入射角特性近似係数</summary>
11 private static readonly double[] MI_RHO = new double[] { 5.189, -12.392, 16.593, -11.851, 3.461 };
12
13 /// <summary>天空日射に対する透過率の係数</summary>
14 private static readonly double DIFFUSE_TRANSMITTANCE_COEF;
15
16 /// <summary>天空日射に対する反射率の係数</summary>
17 private static readonly double DIFFUSE_REFLECTANCE_COEF;
18
19 /// <summary>static コンストラクタ</summary>
20 static FlatPlateSolarCollector()
21 {
22   for (int i = 0; i < 5; i++)
23   {
24     DIFFUSE_TRANSMITTANCE_COEF += MI_TAU[i] / (i + 3);
25     DIFFUSE_REFLECTANCE_COEF += MI_RHO[i] / (i + 3);
26   }
27   DIFFUSE_TRANSMITTANCE_COEF *= 2;
28   DIFFUSE_REFLECTANCE_COEF *= 2;
29 }
```

平板型太陽熱集熱器の集熱量計算処理をプログラム 18.7 に示す。142~186 行は式 18.22~18.34を用

いてガラス面およびパネル面での吸収日射取得量を計算するメソッドである。1~140 行が集熱量計算プログラムの本体である。69~131 行でパネル温度に関する反復計算、79~96 行でガラス温度に関する反復計算処理をそれぞれ行う。

プログラム 18.7　平板型集熱器の集熱量の計算

Popolo.HVAC.SolarEnergy.FlatPlateSolarCollector class

```
  1 /// <summary>太陽熱集熱器の集熱量を計算する</summary>
  2 /// <param name="skyTemperature">天空の温度[C]</param>
  3 /// <param name="airTemperature">空気温度[C]</param>
  4 /// <param name="directNormalRadiation">法線面直達日射量[W/m2]</param>
  5 /// <param name="diffuseRadiation">天空日射量[W/m2]</param>
  6 /// <param name="waterFlowRate">水量[kg/s]</param>
  7 /// <param name="waterInletTemperature">入口水温[C]</param>
  8 /// <param name="airThickness">空気層の厚み[m]</param>
  9 /// <param name="glassEmissivity">ガラスの放射率[-]</param>
 10 /// <param name="panelEmissivity">パネルの放射率[-]</param>
 11 /// <param name="windSpeed">外部風速[m/s]</param>
 12 /// <param name="insulatorThickness">断熱材厚み[m]</param>
 13 /// <param name="insulatorThermalConductivity">断熱材熱伝導率[W/(mK)]</param>
 14 /// <param name="surfaceArea">パネル面積[m2]</param>
 15 /// <param name="tubePitch">集熱媒体管の設置間隔[m]</param>
 16 /// <param name="innerDiameter">集熱媒体管の内径[m]</param>
 17 /// <param name="outerDiameter">集熱媒体管の外径[m]</param>
 18 /// <param name="panelThickness">パネル厚み[m]</param>
 19 /// <param name="panelThermalConductivity">パネルの熱伝導率[W/(mK)]</param>
 20 /// <param name="cosTheta">入射角の余弦</param>
 21 /// <param name="glassTransmittance">ガラスの日射透過率[-]</param>
 22 /// <param name="glassReflectance">ガラスの日射反射率[-]</param>
 23 /// <param name="panelAbsorptance">パネルの日射吸収率[-]</param>
 24 /// <param name="panelTemperature">パネルの温度[C]</param>
 25 /// <param name="glassTemperature">ガラスの温度[C]</param>
 26 /// <param name="waterOutletTemperature">出口水温[C]</param>
 27 /// <param name="meanWaterTemperature">平均水温[C]</param>
 28 /// <param name="efficiency">集熱効率[-]</param>
 29 /// <returns>集熱量[W]</returns>
 30 public static double GetHeatTransfer
 31   (double skyTemperature, double airTemperature, double directNormalRadiation, double diffuseRadiation,
 32   double waterFlowRate, double waterInletTemperature, double airThickness, double glassEmissivity,
 33   double panelEmissivity, double windSpeed, double insulatorThickness, double insulatorThermalConductivity,
 34   double surfaceArea, double tubePitch, double innerDiameter, double outerDiameter, double panelThickness,
 35   double panelThermalConductivity, double cosTheta, double glassTransmittance, double glassReflectance,
 36   double panelAbsorptance, out double panelTemperature, out double glassTemperature,
 37   out double waterOutletTemperature, out double meanWaterTemperature, out double efficiency)
 38 {
 39   double heatTransfer = 0; //集熱量[W]
 40   glassTemperature = 0;
 41   meanWaterTemperature = 0;
 42
 43   //収束計算に無関係な数値を確定する////////////////////////////
 44   //ガラスとパネルの日射吸収量[W/m2]の計算
 45   double sagDN, sapDN, sagDF, sapDF;
 46   getSolarAbsorptanceOfGlassAndPanel(cosTheta, glassTransmittance,
 47     glassReflectance, panelAbsorptance, out sagDN, out sapDN, out sagDF, out sapDF);
 48   double glassSolarAbsorption = sagDN * directNormalRadiation + sagDF * diffuseRadiation;
 49   double panelSolarAbsorption = sapDN * directNormalRadiation + sapDF * diffuseRadiation;
 50
 51   //ガラスと外気との間の対流熱伝達率[W/m2K]の計算
 52   double convectiveHeatLoss_ga = windSpeed * 7.6 + 4.7;
 53
 54   //ガラスとパネルとの間の熱通過率[W/m2K]の計算
 55   double airThermalConductivity = 0.0241 + 0.000077 * airTemperature;
 56   double convectiveHeatLoss_pg = airThermalConductivity / airThickness;   //Nu=1.0とする
 57   //ガラスとパネルとの間の有効放射定数の計算
 58   double effectiveEmissivity_pg = BLACK_CONSTANT / (1 / panelEmissivity + 1 / glassEmissivity - 1);
 59
 60   //パネルと背面との間の熱通過率[W/m2K]の計算
 61   double heatLoss_pb = insulatorThermalConductivity / insulatorThickness;
 62
 63   //パネル温度の初期値設定
 64   panelTemperature = airTemperature + 5;
 65   //1タイムステップ前のパネル温度
 66   double ptL = panelTemperature + 1;
 67   //パネル温度の収束計算
 68   int iterNumP = 0;
 69   while (0.001 < Math.Abs(panelTemperature - ptL))
 70   {
 71     if (100 < iterNumP) throw new Exception("FlatPlateSolarCollector iteration error");
```

```
72
73      double heatLoss_pg = 0; //ガラスとパネルとの間の熱通過率[W/m2K]
74      double radiativeHeatLoss_ga = 0; //ガラスと天空との間の放射熱伝達率[W/m2K]
75
76      //1 ステップ前のパネル温度を保存
77      ptL = panelTemperature;
78
79      //誤差関数を定義
80      Roots.ErrorFunction efncG = delegate (double gTemp)
81      {
82        //ガラスと天空との放射熱伝達率[W/m2K]の計算
83        radiativeHeatLoss_ga = 4 * BLACK_CONSTANT * glassEmissivity
84        * Math.Pow((gTemp + skyTemperature) / 2 + CONVERT_C_TO_K, 3);
85
86        //ガラスとパネルとの間の熱通過率[W/m2K]の計算
87        double radiativeHeatLoss_pg = 4 * effectiveEmissivity_pg * Math.Pow((ptL + gTemp)/ 2 + CONVERT_C_TO_K,3);
88        heatLoss_pg = convectiveHeatLoss_pg + radiativeHeatLoss_pg;
89
90        //更新されたガラス温度と仮定値との間の誤差
91        return gTemp - (glassSolarAbsorption + heatLoss_pg * ptL + radiativeHeatLoss_ga * skyTemperature
92        + convectiveHeatLoss_ga * airTemperature) / (heatLoss_pg + radiativeHeatLoss_ga + convectiveHeatLoss_ga);
93      };
94
95      //ニュートン・ラプソン法でガラス温度を収束計算
96      glassTemperature = Roots.Newton(efncG, 0.5 * (panelTemperature + airTemperature), 0.0001, 0.01, 0.01, 10);
97
98      //熱損失係数 UL[W/m2K]の計算
99      double heatLoss_ga = convectiveHeatLoss_ga
100       + radiativeHeatLoss_ga * (glassTemperature - skyTemperature) / (glassTemperature - airTemperature);
101     double heatLossCoefficient = (heatLoss_pg * heatLoss_ga) / (heatLoss_pg + heatLoss_ga) + heatLoss_pb;
102
103     //四角フィンのフィン効率[-]の計算
104     double wd2 = Math.Sqrt(heatLossCoefficient / (panelThermalConductivity * panelThickness))
105       * (tubePitch - innerDiameter) / 2;
106     double rectFinEfficiency = Math.Tanh(wd2) / wd2;
107
108     //集熱器のフィン効率[-]の計算
109     double tubeHeatTransfer =
110       getConvectiveHeatTransferCoefficientOfTube(waterInletTemperature, innerDiameter, waterFlowRate);
111     double fe1 = 1 / (heatLossCoefficient * (outerDiameter + (tubePitch - outerDiameter) * rectFinEfficiency));
112     double fe2 = 1 / (Math.PI * innerDiameter * tubeHeatTransfer);
113     double finEfficiency = (1 / heatLossCoefficient) / (tubePitch * (fe1 + fe2));
114
115     //集熱器除去因子 FR[-]の計算
116     double bf = waterFlowRate * Water.GetLiquidIsobaricSpecificHeat(waterInletTemperature)
117       * 1000 / (surfaceArea * heatLossCoefficient);
118     double heatRemovalFactor = bf * (1 - Math.Exp(-finEfficiency / bf));
119
120     //熱移動量[W]の計算
121     heatTransfer = panelSolarAbsorption + glassSolarAbsorption * heatLoss_pg / (heatLoss_pg + heatLoss_ga);
122     heatTransfer -= heatLossCoefficient * (waterInletTemperature - airTemperature);
123     heatTransfer *= surfaceArea * heatRemovalFactor;
124
125     //パネルと熱媒の平均温度を更新
126     double buff = (heatTransfer / surfaceArea) / (heatLossCoefficient * heatRemovalFactor);
127     panelTemperature = waterInletTemperature + buff * (1 - heatRemovalFactor);
128     meanWaterTemperature = waterInletTemperature + buff * (1 - heatRemovalFactor / finEfficiency);
129
130     iterNumP++;
131   }
132   //出口水温の計算
133   waterOutletTemperature = 0.001 * waterInletTemperature + heatTransfer
134     / (waterFlowRate * Water.GetLiquidIsobaricSpecificHeat(waterInletTemperature));
135   //集熱効率の計算
136   efficiency = heatTransfer / (cosTheta * directNormalRadiation + diffuseRadiation) / surfaceArea;
137
138   //集熱量を出力
139   return heatTransfer;
140 }
141
142 /// <summary>ガラスとパネルの総合日射吸収率[-]を計算する</summary>
143 /// <param name="cosineTheta">日射入射角の余弦 cosθ</param>
144 /// <param name="glassTransmittance">ガラスの日射透過率[-]</param>
145 /// <param name="glassReflectance">ガラスの日射反射率[-]</param>
146 /// <param name="panelAbsorptance">パネルの日射吸収率[-]</param>
147 /// <param name="glassDNSolarAbsorptance">法線面直達日射に対するパネルの総合日射吸収率[-]</param>
148 /// <param name="panelDNSolarAbsorptance">法線面直達日射に対するガラスの総合日射吸収率[-]</param>
149 /// <param name="glassDFSolarAbsorptance">天空日射に対するパネルの総合日射吸収率[-]</param>
150 /// <param name="panelDFSolarAbsorptance">天空日射に対するガラスの総合日射吸収率[-]</param>
151 private static void getSolarAbsorptanceOfGlassAndPanel
```

```
152   (double cosineTheta, double glassTransmittance, double glassReflectance, double panelAbsorptance,
153   out double glassDNSolarAbsorptance, out double panelDNSolarAbsorptance,
154   out double glassDFSolarAbsorptance, out double panelDFSolarAbsorptance)
155 {
156   //ガラスの規準化透過率と規準化反射率の計算
157   double tn = 0;
158   double rn = 0;
159   for (int i = 4; 0 <= i; i--)
160   {
161     tn = cosineTheta * (tn + MI_TAU[i]);
162     rn = cosineTheta * (rn + MI_RHO[i]);
163   }
164
165   //法線面直達日射に対するガラス単体の透過率・反射率・吸収率の計算
166   double tauThetaG_DN = glassTransmittance * tn;
167   double rhoThetaG_DN = 1 - (1 - glassReflectance) * rn;
168   double alphaThetaG_DN = 1 - (tauThetaG_DN + rhoThetaG_DN);
169
170   //天空日射に対するガラス単体の透過率・反射率・吸収率の計算
171   double tauThetaG_DF = glassTransmittance * DIFFUSE_TRANSMITTANCE_COEF;
172   double rhoThetaG_DF = glassReflectance * DIFFUSE_REFLECTANCE_COEF;
173   double alphaThetaG_DF = 1 - (tauThetaG_DF + rhoThetaG_DF);
174
175   //パネル単体の吸収率・反射率の計算
176   double alphaThetaP = cosineTheta * panelAbsorptance;
177   double rhoThetaP = 1 - alphaThetaP;
178
179   //ガラスの総合吸収率の計算
180   glassDNSolarAbsorptance = alphaThetaG_DN * (1 + tauThetaG_DN * rhoThetaP / (1 - rhoThetaG_DN * rhoThetaP));
181   glassDFSolarAbsorptance = alphaThetaG_DF * (1 + tauThetaG_DF * rhoThetaP / (1 - rhoThetaG_DF * rhoThetaP));
182
183   //パネルの総合吸収率の計算
184   panelDNSolarAbsorptance = tauThetaG_DN * alphaThetaP / (1 - rhoThetaG_DN * rhoThetaP);
185   panelDFSolarAbsorptance = tauThetaG_DF * alphaThetaP / (1 - rhoThetaG_DF * rhoThetaP);
186 }
187
188 /// <summary>水温・直径・水量から配管内対流熱伝達率を計算する</summary>
189 /// <param name="waterTemperature">水温[C]</param>
190 /// <param name="diameter">直径[m]</param>
191 /// <param name="waterFlowRate">水量[L/min]</param>
192 /// <returns>対流熱伝達率[W/(m2･K)]</returns>
193 private static double getConvectiveHeatTransferCoefficientOfTube
194   (double waterTemperature, double diameter, double waterFlowRate)
195 {
196   //動粘性係数[m2/s]・熱拡散率[m2/s]・熱伝導率[W/(m･K)]を計算
197   double v = Water.GetLiquidDynamicViscosity(waterTemperature);
198   double a = Water.GetLiquidThermalDiffusivity(waterTemperature);
199   double lambda = Water.GetLiquidThermalConductivity(waterTemperature);
201
202   //配管内流速[m/s]を計算
203   double u = (waterFlowRate / 60 / 1000) / (Math.Pow(diameter / 2, 2) * Math.PI);
204
205   //ヌセルト数を計算
206   double reNumber = u * diameter / v;
207   double prNumber = v / a;
208   double nuNumber = 0.023 * Math.Pow(reNumber, 0.8) * Math.Pow(prNumber, 0.4);
209
210   //ヌセルト数から対流熱伝達率を計算
211   return nuNumber * lambda / diameter;
212 }
```

【例題 18.4】

下記仕様の太陽熱集熱器について、集熱効率特性線図を作成せよ。ただし、グラフの横軸は平均熱媒温度 $T_{f,m}$ と外気温度 T_a の差である($T_{f,m}$-T_a)を傾斜面への日射の入射量 I_{SC} [W/m^2]で除した値($T_{f,m}$-T_a)/I_{SC} とし、グラフの縦軸は集熱効率 η [-]とする。

ガラス面放射率 ε_g: 0.1、パネル面放射率 ε_p: 0.1、空気層の厚み L_p: 0.01 m

断熱材熱伝導率 λ_i: 0.05 W/(m·K)（グラスウール 10K 相当）

断熱材厚み l: 0.04 m、集熱面積 A_c: 2.0 m^2、集熱媒体流管の内径 d_i: 0.013 m

集熱媒体流管の外径 d_o: 0.015 m、集熱媒体流管の敷設ピッチ W :0.02 m

パネルの厚さ δ: 0.001 m、パネルの熱伝導率 k: 20 W/(m·K)

ガラスの日射透過率 τ_g (0): 0.90、日射反射率 ρ_g (0) = 0.08

パネルの日射吸収率 α_p (0) = 0.93、通水量 m_w: 0.01 kg/s、外部風速 v: 0 m/s

【解】

日射は垂直に入射するものとし、日射量は 500 W/m^2 で固定する。入口水温を 0 ℃ から 80 ℃ まで変化させて集熱効率の変化を計算する。基準ケースに加えて外部風速を 10m/s、通水量を 0.005kg/s に変化させた場合に集熱効率を計算するプログラムを 18.8 に示す。また、出力をグラフ化した結果を図 18.8 に示す。風速が強くなるとパネル外表面の対流熱伝達率が上昇して集熱効率が低下する。日射量に対して相対的に温度差が大きい場合に影響が強い。通水量の減少によっても集熱効率が低下するが、この場合には逆に温度差に対して相対的に日射量が大きい場合に影響が強い。

プログラム 18.8　太陽熱集熱器の集熱効率の計算

```
 1 private static void FlatPlateSolarCollectorTest()
 2 {
 3   double isc = 500;
 4
 5   Console.WriteLine("平均水温[C], 平均水温/日射量[Cm2K/W], 集熱効率[-]");
 6   Console.WriteLine("*****基準ケース*****");
 7   for (int twi = 0; twi < 80; twi++)
 8   {
 9     double pt, gt, wt, mt, eta;
10     double ht = FlatPlateSolarCollector.GetHeatTransfer
11       (0, 0, isc, 0, 0.01, twi, 0.01, 0.1, 0.1, 0, 0.02, 0.05, 2.0, 0.02, 0.013, 0.015, 0.001, 20,
12       1.0, 0.90, 0.08, 0.93, out pt, out gt, out wt, out mt, out eta);
13     Console.WriteLine((mt / isc).ToString("F3") + ", " + eta.ToString("F2"));
14   }
15   Console.WriteLine("*****外部風速 10m/s*****");
16   for (int twi = 0; twi < 80; twi++)
17   {
18     double pt, gt, wt, mt, eta;
19     double ht = FlatPlateSolarCollector.GetHeatTransfer
20       (0, 0, isc, 0, 0.01, twi, 0.01, 0.1, 0.1, 10, 0.02, 0.05, 2.0, 0.02, 0.013, 0.015, 0.001, 20,
21       1.0, 0.90, 0.08, 0.93, out pt, out gt, out wt, out mt, out eta);
22     Console.WriteLine((mt / isc).ToString("F3") + ", " + eta.ToString("F2"));
23   }
24   Console.WriteLine("*****水量 0.005m/s*****");
25   for (int twi = 0; twi < 80; twi++)
26   {
27     double pt, gt, wt, mt, eta;
28     double ht = FlatPlateSolarCollector.GetHeatTransfer
29       (0, 0, isc, 0, 0.005, twi, 0.01, 0.1, 0.1, 0, 0.02, 0.05, 2.0, 0.02, 0.013, 0.015, 0.001, 20,
30       1.0, 0.90, 0.08, 0.93, out pt, out gt, out wt, out mt, out eta);
31     Console.WriteLine((mt / isc).ToString("F3") + ", " + eta.ToString("F2"));
32   }
33 }
```

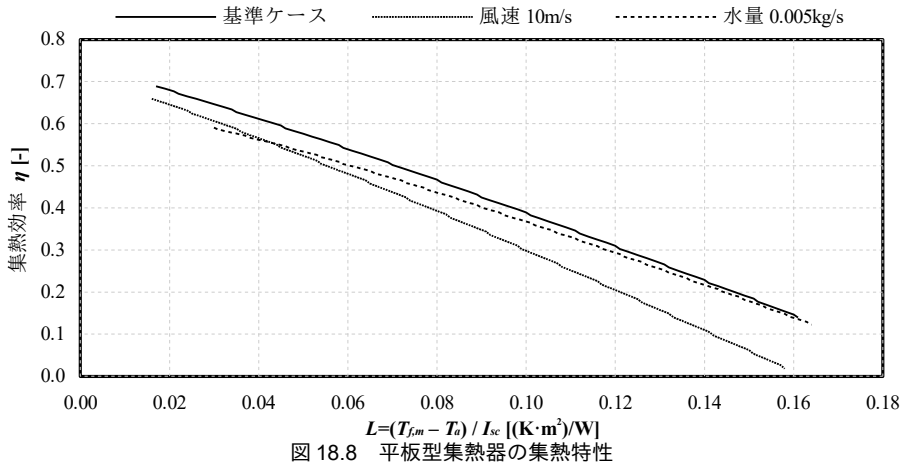


図 18.8　平板型集熱器の集熱特性

18.3.3　太陽電池の計算

列挙型の定義をプログラム 18.9 に示す。パネルの材料種別と設置タイプに関する列挙型である。

プログラム 18.9　定数および列挙型の定義

Popolo.Electric.PhotovoltaicPanel class

```
1 /// <summary>材料</summary>
2 public enum MaterialType
3 {
```

```
  /// <summary>アモルファス</summary>
  Amorphous,
  /// <summary>結晶</summary>
  Crystal
}

/// <summary>設置タイプ</summary>
public enum MountType
{
  /// <summary>屋根置き形</summary>
  RoofMount,
  /// <summary>屋根材一体形</summary>
  RoofIntegrated,
  /// <summary>架台形（ラック形）</summary>
  MountMode
}
```

プログラム 18.10 に太陽電池の出力の計算を示す。

1~33 行はパネル温度による補正係数の計算処理である。12~27 行で設置方式に応じた係数を求め、29 行で式 18.42 を適用してパネル温度を計算する。その後、31, 32 行で式 18.41 を適用して材料に応じた補正係数を算出する。

35~51 行は出力の計算処理である。48, 49 行で温度補正係数を求め、50 行で式 18.40 を適用する。

プログラム 18.10　太陽電池の出力の計算

Popolo.Electric.PhotovoltaicPanel class

```
/// <summary>パネル面の温度上昇による効率変化係数を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[C]</param>
/// <param name="velocity">風速[m/s]</param>
/// <param name="totalIrradiance">
/// 傾斜面日射強度[W/m2]（直達日射+拡散日射）</param>
/// <param name="mount">設置タイプ</param>
/// <param name="material">材料</param>
/// <returns>パネル面の温度上昇による効率変化係数</returns>
private static double getTemperatureRiseCorrectionFactor
  (double drybulbTemperature, double velocity, double totalIrradiance, MountType mount, MaterialType material)
{
  double a, b;
  switch (mount)
  {
    case MountType.MountMode:
      a = 46;
      b = 0.41;
      break;
    case MountType.RoofIntegrated:
      a = 50;
      b = 0.33;
      break;
    default:
      a = 50;
      b = 0.38;
      break;
  }
  //パネル温度の計算
  double tp = drybulbTemperature + (a / (b * Math.Pow(velocity, 0.8) + 1) + 2) * totalIrradiance / 1000d - 2d;

  if (material == MaterialType.Amorphous) return 1 - 0.002 * (tp - 25d);
  else return 1 - 0.004 * (tp - 25d);
}

/// <summary>出力[kW]を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[C]</param>
/// <param name="velocity">風速[m/s]</param>
/// <param name="totalIrradiance">傾斜面日射強度[W/m2]（直達日射+拡散日射）</param>
/// <param name="peakPower">最大出力[kW]</param>
/// <param name="inverterEfficiency">インバータ効率[-]</param>
/// <param name="mount">設置タイプ</param>
/// <param name="material">材料</param>
/// <returns>出力[kW]</returns>
public static double GetPower
  (double drybulbTemperature, double velocity, double totalIrradiance,
  double peakPower, double inverterEfficiency, MountType mount, MaterialType material)
{
  double kpt = getTemperatureRiseCorrectionFactor
    (drybulbTemperature, velocity, totalIrradiance, mount, material);
  return totalIrradiance / 1000d * kpt * peakPower * inverterEfficiency;
```

```
51 }
```

プログラム 18.11 にインスタンス変数およびプロパティの定義を示す。

プログラム 18.11　インスタンス変数およびプロパティの定義

Popolo.Electric.PhotovoltaicPanel class

```
 1 /// <summary>インバータ効率[-]</summary>
 2 private double inverterEfficiency = 0.9;
 3
 4 /// <summary>インバータ効率[-]を設定・取得する</summary>
 5 public double InverterEfficiency
 6 {
 7   get { return inverterEfficiency; }
 8   set { inverterEfficiency = Math.Max(0, Math.Min(1, value)); }
 9 }
10
11 /// <summary>最大出力[W]を取得する</summary>
12 public double PeakPower { get; private set; }
13
14 /// <summary>設置タイプを取得する</summary>
15 public MountType Mount { get; private set; }
16
17 /// <summary>材料を取得する</summary>
18 public MaterialType Material { get; private set; }
19
20 /// <summary>傾斜面を取得する</summary>
21 public ImmutableIncline Incline { get; private set; }
```

プログラム 18.12 にコンストラクタを示す。1~15 行が基準となるコンストラクタであり、最大出力、設置方式、材料種別、傾斜面情報を用いてプロパティを初期化する。傾斜面情報は、第 6 章で定義した Incline クラスを用いる。太陽電池パネルは真南（南半球においては真北）に向けて設置することが通常であり、東西方向に傾けることは少ない。そこで傾斜角度のみを用いて初期化するコンストラクタも 18~27 行で定義する。

プログラム 18.12　コンストラクタ

Popolo.Electric.PhotovoltaicPanel class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="peakPower">最大出力[W]</param>
 3 /// <param name="mount">設置タイプ</param>
 4 /// <param name="material">材料</param>
 5 /// <param name="incline">設置傾斜面</param>
 6 /// <remarks>
 7 /// 最大出力は傾斜面日射強度 1000W/m2、周囲温度 25C における性能値。
 8 /// </remarks>
 9 public PhotovoltaicPanel
10   (double peakPower, MountType mount, MaterialType material, Incline incline)
11 {
12   this.PeakPower = peakPower;
13   this.Mount = mount;
14   this.Material = material;
15   this.Incline = incline;
16 }
17
18 /// <summary>インスタンスを初期化する</summary>
19 /// <param name="peakPower">最大出力[W]</param>
20 /// <param name="mount">設置タイプ</param>
21 /// <param name="material">材料</param>
22 /// <param name="tiltAngle">設置角[degree]（水平面=0 度,垂直面=90 度）</param>
23 /// <remarks>最大出力は傾斜面日射強度 1000W/m2、周囲温度 25C における性能値。</remarks>
24 public PhotovoltaicPanel
25   (double peakPower, MountType mount, MaterialType material, double tiltAngle)
26   : this(peakPower, mount, material, new Incline(Weather.Incline.Orientation.S, tiltAngle * Math.PI / 180d))
27 { }
```

プログラム 18.13 に出力計算処理を示す。PhotovoltaicPanel クラス側で傾斜面日射強度の計算が行えるように、引数として第 6 章で定義した ImmutableSun インスタンスを受けるメソッドも定義する（13~23 行）。式 6.11~6.15 を用いて傾斜面へ入射する直達日射と拡散日射を計算する。

プログラム 18.13　出力計算処理

Popolo.Electric.PhotovoltaicPanel class

```
 1 /// <summary>出力[W]を計算する</summary>
 2 /// <param name="drybulbTemperature">外気乾球温度[C]</param>
 3 /// <param name="velocity">風速[m/s]</param>
 4 /// <param name="totalIrradiance">
 5 /// 傾斜面日射強度[W/m2] (直達日射+拡散日射) </param>
 6 /// <returns>出力[W]</returns>
 7 public double GetPower(double drybulbTemperature, double velocity, double totalIrradiance)
 8 {
 9   return GetPower
10     (drybulbTemperature, velocity, totalIrradiance, PeakPower, InverterEfficiency, Mount, Material);
11 }
12
13 /// <summary>出力[W]を計算する</summary>
14 /// <param name="drybulbTemperature">外気乾球温度[C]</param>
15 /// <param name="velocity">風速[m/s]</param>
16 /// <param name="sun">太陽</param>
17 /// <returns>出力[W]</returns>
18 public double GetPower(double drybulbTemperature, double velocity, ImmutableSun sun)
19 {
20   double totalIrradiance = Incline.GetDirectSolarRadiationRate(sun) * sun.DirectNormalRadiation
21     + Incline.ConfigurationFactorToSky * sun.DiffuseHorizontalRadiation;
22   return GetPower(drybulbTemperature, velocity, totalIrradiance);
23 }
```

【例題 18.5】

札幌、東京、那覇の 3 都市について太陽電池パネルの年間の発電量を角度別に求めよ。設置方式は架台設置、材料はアモルファスとする。

【解】

計算処理をプログラム 18.14 に示す。3~9 行で角度別の太陽電池のインスタンスを作成する。11~16 行は各都市の気象データ作成処理であり、具体的な処理は第 7 章で解説した。太陽電池の耐用年数に合わせて 20 年間分のデータを作成する。18~21 行は各都市の太陽インスタンスである。23 行以下で書き出し処理を行う。38 行以降が 20 年間のループ、41 行以降が年間のループである。43~49 行で各時刻の水平面全天日射の直散分離処理を行い、これを用いて 51~57 行で発電量を計算して足し合わせる。60~65 行は年間発電量の書き出し処理である。

計算された 20 年間の平均値を都市別、角度別にグラフ化すると図 18.9 が得られる。東京では 30 度前後、那覇では 20 度前後にピークが表れており、緯度と同程度の角度とすると発電量が大きくなることが確認できる。一方で、札幌に関しては 30 度程度にピークがあり、緯度よりもやや小さい。気象庁などのデータから過去の札幌の日射量を季節別に確認すると、冬季の日射量がその他の季節に比較して非常に小さいことがわかる。札幌においては冬季よりも夏季ないしは中間期を狙った角度で計画することが有利であり、結果として 30 度程度で年間発電量が最大化する。

プログラム 18.14　札幌、東京、那覇の太陽電池年間発電量（角度別）の計算

```
 1 private static void PhotovoltaicPanelTest()
 2 {
 3   //太陽光発電パネルを作成//定格出力 1W,架台設置,アモルファス,設置角 10 度刻み
 4   PhotovoltaicPanel[] pvs = new PhotovoltaicPanel[10];
 5   for (int i = 0; i < pvs.Length; i++)
 6   {
 7     pvs[i] = new PhotovoltaicPanel
 8       (1, PhotovoltaicPanel.MountType.MountMode, PhotovoltaicPanel.MaterialType.Amorphous, i * 10);
 9   }
10
11   //年間日射データを計算（東京・那覇・札幌）
12   double[] dbtT, dbtN, dbtS, hrt, radS, radT, radN;
13   bool[] fair;
14   RandomWeather.MakeWeather(1, RandomWeather.Location.Tokyo, 20, out dbtT, out hrt, out radT, out fair);
15   RandomWeather.MakeWeather(1, RandomWeather.Location.Sapporo, 20, out dbtS, out hrt, out radS, out fair);
16   RandomWeather.MakeWeather(1, RandomWeather.Location.Naha, 20, out dbtN, out hrt, out radN, out fair);
17
18   //太陽
19   Sun sunT = new Sun(35, 41.2, 0, 139, 45.9, 0, 135, 0, 0);
20   Sun sunS = new Sun(43, 3.5, 0, 141, 19.9, 0, 135, 0, 0);
21   Sun sunN = new Sun(26, 12.2, 0, 127, 41.3, 0, 135, 0, 0);
22
23   //計算,書き出し処理
24   using (StreamWriter sWriter = new StreamWriter
25     ("PhotovoltaicPanelTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
26   {
27     //タイトル行
28     sWriter.Write("年");
29     for (int i = 0; i < 10; i++) sWriter.Write(",東京" + (i * 10) + "度");
30     for (int i = 0; i < 10; i++) sWriter.Write(",札幌" + (i * 10) + "度");
```

```
31      for (int i = 0; i < 10; i++) sWriter.Write(",那覇" + (i * 10) + "度");
32      sWriter.WriteLine();
33
34      DateTime dt = new DateTime(1999, 1, 1, 0, 0, 0);
35      double[] pSumT = new double[10];
36      double[] pSumS = new double[10];
37      double[] pSumN = new double[10];
38      for (int i = 0; i < 20; i++)
39      {
40        for (int j = 0; j < pSumT.Length; j++) pSumT[j] = pSumS[j] = pSumN[j] = 0;
41        for (int j = 0; j < 8760; j++)
42        {
43          //太陽位置を更新して直散分離
44          sunT.Update(dt);
45          sunS.Update(dt);
46          sunN.Update(dt);
47          sunT.SeparateGlobalHorizontalRadiation(radT[8760 * i + j], Sun.SeparationMethod.Erbs);
48          sunS.SeparateGlobalHorizontalRadiation(radS[8760 * i + j], Sun.SeparationMethod.Erbs);
49          sunN.SeparateGlobalHorizontalRadiation(radN[8760 * i + j], Sun.SeparationMethod.Erbs);
50
51          //発電量[W]を計算
52          for (int k = 0; k < pvs.Length; k++)
53          {
54            pSumT[k] += pvs[k].GetPower(dbtT[8760 * i + j], 0.1, sunT);
55            pSumS[k] += pvs[k].GetPower(dbtS[8760 * i + j], 0.1, sunS);
56            pSumN[k] += pvs[k].GetPower(dbtN[8760 * i + j], 0.1, sunN);
57          }
58          dt = dt.AddHours(1);
59        }
60        //書き出し処理
61        sWriter.Write(i + 1);
62        for (int j = 0; j < pvs.Length; j++) sWriter.Write("," + pSumT[j]);
63        for (int j = 0; j < pvs.Length; j++) sWriter.Write("," + pSumS[j]);
64        for (int j = 0; j < pvs.Length; j++) sWriter.Write("," + pSumN[j]);
65        sWriter.WriteLine();
66        dt = dt.AddYears(-1);
67      }
68    }
69 }
```

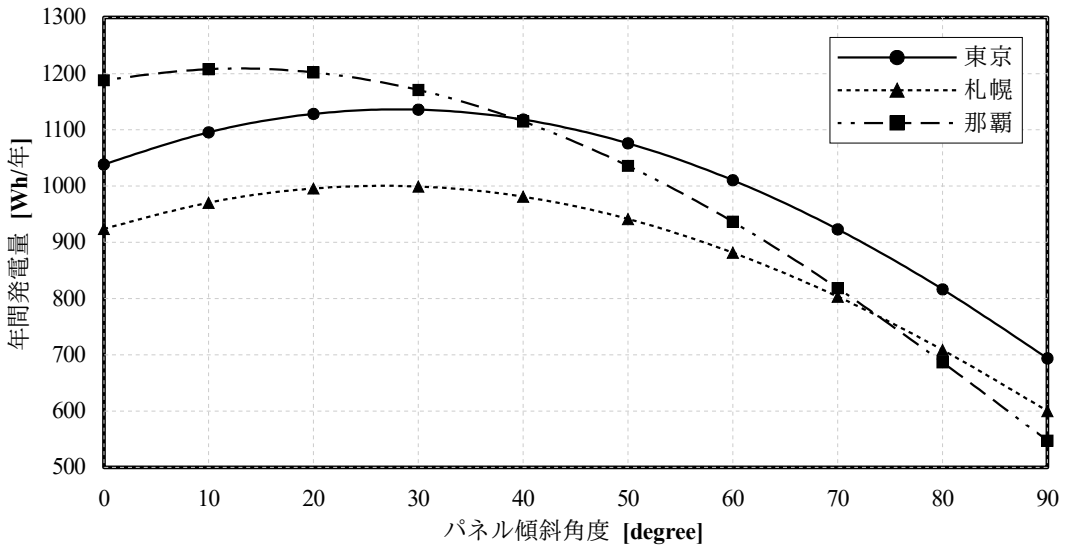


図 18.9　各都市のパネル傾斜角度別の年間発電量

【第 18 章 記号表】

a	：特性式係数（$y=a\,x+b$）	q_a	：天空からの放射熱移動量 [W/m²]
$A,\ B$	：係数 [-]	Q_u	：集熱量 [W]
A_c	：集熱面積 [m²]	S_g	：ガラスの日射吸収量 [W/m²]
b	：特性式係数（$y=a\,x+b$）	S_p	：パネルの日射吸収量 [W/m²]
C_B	：円管の接着剤熱通過率 [W(m²·K)]	T	：温度 [K]
c_p	：流体の定圧比熱 [kJ/(kg·K)]	T_A	：周囲温度 [ºC]
d_i	：内径 [m]	T_{PA}	：パネル温度 [ºC]
d_o	：外径 [m]	U	：熱通過率 [W/K]
F	：四角フィンのフィン効率 [-]	U_L	：熱損失係数 [W/K]
F_R	：集熱器熱除去因子 [-]	W	：集熱管の設置間隔 [m]
F'	：円管と平板のフィン効率 [-]	v	：外部風速 [m/s]
h_c	：対流熱伝達率 [W/(m²·K)]	α	：日射吸収率 [-]
h_r	：放射熱伝達率 [W/(m²·K)]	α_T	：総合日射吸収率 [-]
h_w	：外表面対流熱伝達率 [W(m²·K)]	α_{PT}	：温度補正係数 [ºC⁻¹]
I_{DN}	：法線面直達日射量 [W/m²]	ε	：長波長放射率 [-]
I_D	：傾斜面に入射する直達日射 [W/m²]	δ	：パネルの厚み [m]
I_{sc}	：傾斜面へ入射する全天日射量（日射強度） [W/m²]	η	：集熱効率 [-]
I_{SR}	：傾斜面に入射する拡散日射 [W/m²]	η_{INO}	：インバータ効率 [-]
K_{PT}	：パネル温度による補正係数 [-]	λ_a	：空気の熱伝導率 [W/(m·K)]
k	：パネルの熱伝導率 [W/(m·K)]	λ_i	：断熱材熱伝導率 [W/(m·K)]
l	：断熱材の厚み [m]	ρ	：日射反射率 [-]
L_p	：空気層の厚み [m]	σ	：黒体の放射定数 [W/(m²·K⁴)]
m_f	：流体質量流量 [kg/s]	σ_{pg}	：有効放射定数 [W/(m²·K⁴)]
P	：出力 [kW]	τ	：日射透過率 [-]
P_M	：最大出力 [kW]		
sub scripts			
p	：パネル	g	：ガラス
a	：外気	$p\text{-}g$	：パネルとガラスの間
$g\text{-}a$	：ガラスと外気の間	sky	：天空
f	：管内流体	m	：平均

【第 18 章 参考文献】

18.1） John A. Duffie, William A. Beckman : Solar Engineering of Thermal Processes, 1950

18.2） Ito, N.,Kimura, K. and Oka, J. : A field experiment study on the converctive heat transfer coefficient on exterior surface of a building, ASHRAE Transactions, Vol. 78, part 1, 1972, pp.154-191

18.3） 伊藤直明, 佐藤鑑, 岡樹生 ：自然風下における建築物外壁面の対流熱伝達について (II), 日本建築学会大会学術講演梗概集. 計画系 43, 1968, pp.239-240

18.4） W. H. McAdams : Heat Transmission (3rd ed.), 1954, McGraw-Hill

18.5） 種村栄 ：平板形太陽熱集熱器の高性能化の理論的背景と現状, 空気調和・衛生工学 57(1), 1983, pp.9-17

18.6） 新太陽エネルギー利用ハンドブック, 日本太陽エネルギー学会, pp.141-151

18.7） 矢崎エナジーシステム株式会社, スーパーブルーパネル技術資料

18.8） 宇田川光弘 ：パソコンによる空気調和計算法, オーム社, 1986

18.9） 郡公子, 石野久彌, 郡司智昭, 小林信裕: 直達日射に対する一般窓日射遮蔽性能値の実用的推定法, 空気調和・衛生工学会大会学術講演論文集, 2007, pp.369-372

18.10） JIS R 3106:1998, ガラス類の透過率・反射率・放射率・日射熱取得率の試験方法

18.11） 平成 24 年度国際標準開発事業 窓の遮熱性能計算・試験方法の JIS・ISO 化成果報告書, 平成 25 年 3 月, 一般社団法人 日本建材・住宅設備産業協会

18.12） 山本義一: 大気輻射学, 岩波書店, 1954

18.13） Philipps, H.: Zur Theorie der Warmestrahlung in Bodennahe, Grerl. Beitr. Z. Geophys. 56, 1940, p.229

18.14） Klein, S. A., J. A. Duffie, and W. A. Beckman: Transient Considerations of Flat-Plate Solar Collectors, Journal of Engineering for Power, 96A, 109, 1974

18.15） 宿谷昌則 ：数値計算で学ぶ光と熱の建築環境学, 第 6 章 窓と自然室温, pp.170, , 丸善株式会社, 1993

18.16） JIS8960:太陽光発電用語

18.17） JIS8907:太陽光発電システムの発電電力量推定方法

18.18） 湯川元信 他:太陽電池モジュール温度上昇の推定, 電気学会論文誌.B, Vol. 116 (1996) No. 9

18.19） 西川豊宏, 飯嶋航平: 太陽光発電システムが設置された中規模事務所建物の防災拠点化に関する調査研究, 第 1 報 年間の太陽光発電量と電力使用量に関する基礎調査, 空気調和・衛生工学会論文集, No.226, pp.13~20, 2016 January

18.20） 田中俊六: 太陽熱冷暖房システム, オーム社, 1977

18.21） 新建築 住宅特集, 太陽熱の家 KIMURA SOLAR HOUSE, pp.287-290, 新建築社, 1974.02

第19章　蓄熱槽
(Heat Storage tank)

19.1　概要

熱容量の大きい熱媒を用いて水槽内に熱エネルギーを蓄える装置を蓄熱槽と呼ぶ。熱を蓄える目的は、システムにより様々であるが、主には下記がある。

1）負荷を平準化することで必要な熱源能力を抑える

2）負荷が発生する時間帯を移動させ、相対的に安価な料金時間帯に熱源を運転させる

3）蓄熱槽を介して熱を供給することで、100 %負荷で熱源を運転させる

蓄熱を行う熱媒としては、水または氷を用いることが多く、前者を水蓄熱システム、後者を氷蓄熱システムと呼ぶ。この他にも流動性をもった氷（リキッドアイス）を用いる方式などもあるが、実績としては水または氷に比較して非常に少ない。

氷蓄熱は液体の状態変化に伴う熱も蓄熱に用いるシステムであり、同体積の水蓄熱に比較するとはるかに大きい熱を蓄える事ができる。一方で、水蓄熱に比較して熱源の蒸発温度を下げる必要があるため、熱源単体の効率は低くなる。また熱源を通過する熱媒は凝固点の低い不凍液であり、製氷はこの不凍液と熱交換を行うことで間接的に行われる。

水蓄熱は水の温度差を用いて熱を蓄えるシステムである。原理は単純であるが、運用の良否によっては水槽内の温度が乱れ、期待する性能が得られないことがある。数値計算によって水温の挙動を予測することは、蓄熱槽の運用技術を向上させるという意味においても有益である。

本書では導入実績の多い水蓄熱システムについて、代表的な連結完全混合型と温度成層型を取り上げ、各々について計算方法を示す。

19.2　理論

水蓄熱槽は槽内温度の分布のあり方によって、連結完全混合型と温度成層型に分けられる。連結完全混合型蓄熱槽は小規模な水槽を多数連結して高温の水槽と低温の水槽で全体を構成するものであり、各水槽内での温度分布は期待しない†1)。一方、温度成層型蓄熱槽は水温度による密度差を利用するものであり、相対的に温度の低い冷水を下部に蓄えることで混合ロスを防ぐというものである。蓄熱効率としては温度成層型の方が高いが、連結混合型は平面的に地下ピットを広く使用することで大容量を確保しやすいという利点がある。

19.2.1　連結完全混合型蓄熱槽

図 19.1 に連結完全混合型蓄熱槽モデルの概念を示す。熱源と二次側への水流 Q_p [m³/s]と Q_s [m³/s]を蓄熱槽ヘッダで受け、その差分の流量 Q_{st} [m³/s]が蓄熱槽内を流れる。また、各々の水槽では外界との温度差に応じて貫流熱損失が生じる。従って、第 m 槽での熱収支は式 19.1 で表現できる。ただし θ、ρ_W、c_{pW}はそれぞれ水の温度 [°C]、比重 [kg/m³]、比熱 [kJ/(kg·K)]であり、K および V は水槽の熱損失係数 [kW/K]と体積 [m³]である。図 19.1 は二次側流量に比較して熱源流量が大きい場合の流れを示しているが、二次側の方が大きい場合には流れが逆転し、第 M 槽から第 0 槽へと水が流れることになる。この場合には第 m 槽の水温は m+1 槽に影響を受け、水槽の熱収支は式 19.2 で表現される。

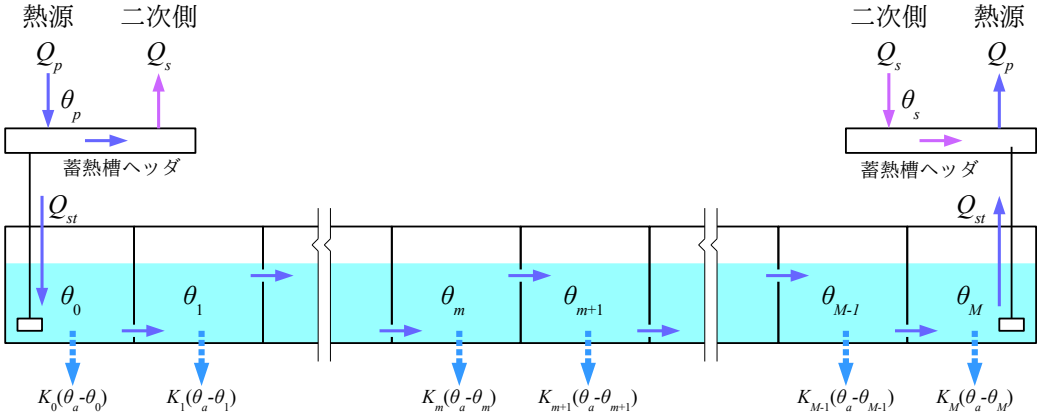


図 19.1　連結完全混合型蓄熱槽モデルの概念

$$\rho c_{pW} V_m \frac{d\theta_m}{dt} = \rho c_{pW} Q_{st}(\theta_{m-1}-\theta_m) + K_m(\theta_a-\theta_m) \tag{19.1}$$

$$\rho c_{pW} V_m \frac{d\theta_m}{dt} = \rho c_{pW} Q_{st}(\theta_{m+1}-\theta_m) + K_m(\theta_a-\theta_m) \tag{19.2}$$

式 19.1 と 19.2 を差分化して行列表記するとそれぞれ式 19.3 と 19.4 が得られる。

$$\begin{bmatrix} 1+s_0+r_0 & 0 & \cdots & \cdots & 0 \\ -s_1 & 1+s_1+r_1 & 0 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -s_M & 1+s_M+r_M \end{bmatrix} \begin{bmatrix} \theta_0^{n+1} \\ \theta_1^{n+1} \\ \vdots \\ \theta_M^{n+1} \end{bmatrix} = \begin{bmatrix} \theta_0^n+\theta_a r_0+s_0\theta_p \\ \theta_1^n+\theta_a r_1 \\ \vdots \\ \theta_M^n+\theta_a r_M \end{bmatrix} \tag{19.3}$$

$$\begin{bmatrix} 1+s_0+r_0 & -s_0 & \cdots & \cdots & 0 \\ 0 & 1+s_1+r_1 & -s_1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -s_{M-1} \\ 0 & \cdots & \cdots & 0 & 1+s_M+r_M \end{bmatrix} \begin{bmatrix} \theta_0^{n+1} \\ \theta_1^{n+1} \\ \vdots \\ \theta_M^{n+1} \end{bmatrix} = \begin{bmatrix} \theta_0^n+\theta_a r_0 \\ \theta_1^n+\theta_a r_1 \\ \vdots \\ \theta_M^n+\theta_a r_M+s_M\theta_s \end{bmatrix} \tag{19.4}$$

†1　日本独特の方式であり、1953 年に柳町政之助が日活多摩撮影所で初めて採用した[19.10]。

ただし、　$s_m=\dfrac{Q_{st}}{V_m}\Delta t$　，　$r_m=\dfrac{K_m}{\rho_W c_{pW} V_m}\Delta t$

19.2.2　温度成層型蓄熱槽

第 3 章で記したとおり、水の密度は温度によって変化し、約 4°C で最大となる。この性質を利用して垂直方向に水の温度分布を形成させることで混合ロスを抑制するシステムが温度成層型の蓄熱槽である。温水蓄熱を行う際には水槽下部から採水し、加温した温水を水槽上部へ戻す。逆に冷水蓄熱を行う際には水槽上部から採水し、冷却した冷水を水槽下部へ戻す。放熱時には逆の流れとなる。この水槽に流入させる冷水あるいは温水の影響を直接に受ける領域を「混合域」と呼ぶ。逆に流入水の影響を直接的に受けず、水の熱拡散および移流のみで温度変化が生じる領域を「一次拡散域」と呼ぶ。

以下に記す計算法は相良らによるモデルに熱損失項を追加したモデルである[19.2)]。水槽内の温度分布は、移流項と拡散項を含む移流拡散方程式として式 19.5 で表現される。

$$\frac{\partial\theta}{\partial t}=\kappa_0\frac{\partial^2\theta}{\partial z^2}-U\frac{\partial\theta}{\partial z}+\frac{\Phi}{A}(\theta_{wi}-\theta)+\frac{K}{\rho c_{pW} V}(\theta_a-\theta) \tag{19.5}$$

θ は水槽内の温度[°C]、t は時間[sec]、z は水深[m]、κ_0 は温度拡散係数[m²/s]、U は槽内垂直方向流速[m/s]、Φ は鉛直方向単位長さあたりの流入流量[m²/s]、θ_{wi} は流入温度[°C]、A は水平方向切断面積[m²]である。Φ は流入流量 q_{wi} [m³/s]と混合域の深さ l_{mix} [m]の関数として式 19.6 で表現される。流入口近傍で最も大きく、流入口から離れるにつれて小さくなり、混合域端部（$z=l_{mix}$）で 0 になる。

$$\Phi=\begin{cases}\dfrac{3}{2}\dfrac{q_{wi}}{l_{mix}}\left(1-\left(\dfrac{z}{l_m}\right)^2\right) & (z\le l_{mix})\\ 0 & (l_{mix}<z)\end{cases} \tag{19.6}$$

以上の前提のもと、水槽内の温度分布と流速分布を図示すると図 19.2 となる。流入量は混合域内で分配され、上部ほど流入量が大きい。この影響を受けて、垂直方向の流速は混合域内下部に向かうにつれて大きくなり、一次拡散域では一様となる。

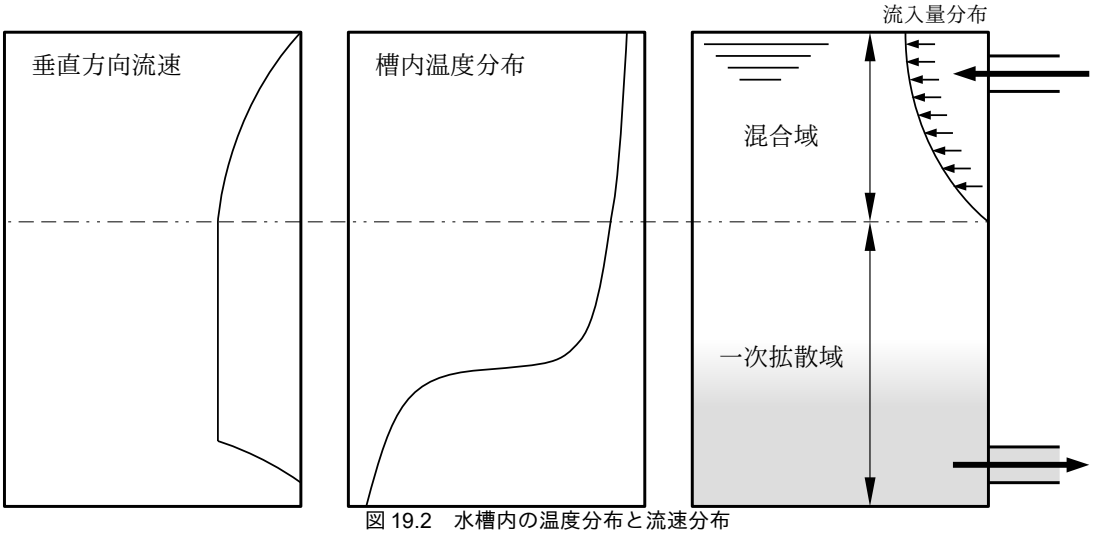


図 19.2　水槽内の温度分布と流速分布

混合域の深さ l_{mix} は槽水深 L [m]に無次元深さ R_m [-]を乗じて式 19.7 で計算する。相良によれば、無次元深さ R_m は実験により式 19.8 で表される[19.2)]。ここで、Ar は流入水のアルキメデス数[-]、d_0 は流入口円管の直径[m]、t^* は無次元時間[-]である。流入口が円管ではない場合には面積の等しい等価直径

とする。流入口の速度を正しく掴むことが大切である。

$$l_{mix}=R_m\cdot L \tag{19.7}$$

$$R_m=0.8\,Ar^{-0.5}\,d_0/L+0.5\,t^* \tag{19.8}$$

アルキメデス数 Ar は式 19.9 で計算する。g は重力加速度[m/s^2]、ρ_0 は槽内基準密度[kg/m^3]、u_{wi} は流入管入口での水速[m/s]、$\Delta\rho$ は式 19.10 で表される流入水の密度と槽内基準密度との差分である。槽内基準密度は、水面から下向き垂直方向への噴流が到達する水深 l_{jet} [m]における水密度であり、式 19.11 で計算する。なお、この水深における水温 θ_0 を槽内基準温度と定義し、後述する無次元時間 t^* の計算の際に用いる。

蓄熱槽の温度成層を崩さないためには無次元深さ R_m が小さいことが望ましい。式 19.8 からわかるように無次元深さ R_m はアルキメデス数 Ar の平方根に反比例するため、アルキメデス数が小さくなると急速に値が大きくなる。従って、設計上はアルキメデス数が 1.0 を上回るようにする。このためには式 19.9 からわかるように流入口の水速 u_{wi} を小さくする必要があり、ディストリビュータと呼ばれる面積を広く取った流入口を用いることもある[19.8) 19.9)]。結果としては流入口の水速 u_{wi} は 0.1 m/s を下回る程度の値となる。

$$Ar=d_0\,g\,\frac{\Delta\rho}{\rho_0\,u_{wi}^2} \tag{19.9}$$

$$\Delta\rho=\begin{cases}\rho_{wi}-\rho_0 & \text{（槽下部から流入の場合）}\\ \rho_0-\rho_{wi} & \text{（槽上部から流入の場合）}\end{cases} \tag{19.10}$$

$$\rho_{wi}u_{wi}^2=g\int_0^{l_{jet}}(\rho-\rho_{wi}) \tag{19.11}$$

式 19.8 の無次元時間 t^*は式 19.12 で計算する。V_{wi} [m^3]は、基準温度 θ_0 との温度差にもとづいて計算された混合域における蓄熱量 $Q_{st,0}$ [kJ]と等しい熱量を得るために必要な積算流入流量であり、式 19.13 で計算する。ただし、密度 ρ は基準温度 θ_0 と入口水温 θ_{wi} の平均値から計算する。蓄熱量 $Q_{st,0}$ は、混合域について水深別の温度差を積分することで式 19.14 で計算する。

$$t^*=V_{wi}/V \tag{19.12}$$

$$V_{wi}=\frac{Q_{st,0}}{c_{pW}\,\rho\,(\theta_{wi}-\theta_0)}=\frac{A}{\theta_{wi}-\theta_0}\left(\int\theta\,dz-\theta_0 L\right) \tag{19.13}$$

$$Q_{st,0}=c_{pW}\,\rho\,A\int_0^{l_{jet}}(\theta-\theta_0)\,dz \tag{19.14}$$

19.3 計算法

19.3.1 連結完全混合型蓄熱槽の計算法

連結完全混合型蓄熱槽の水槽温度の更新処理をプログラム 19.1 に示す。第 1 引数が各水槽の温度であり、この値を更新する。第 2 引数の wMat は 3 重対角行列を計算するための記憶領域である。計算の度に記憶領域を確保すると処理が遅くなるため、メソッドの外部から与える仕様とする。21~41 行で式 19.3 および式 19.4 に従って 3 重対角行列を作成する。順流の場合の処理が 29~34 行、逆流の場合の処理が 36~40 行である。42 行で行列を解き、各水槽の温度を更新する。

プログラム **19.1**　連結完全混合型蓄熱槽の水槽温度の更新処理

Popolo.HVAC.ThermalStorage.MultiConnectedWaterTank class

```
/// <summary>水槽温度を計算する</summary>
/// <param name="temperatures">水槽温度[C]</param>
/// <param name="wMat">計算領域 (3×m) </param>
/// <param name="timeStep">計算間隔[s]</param>
/// <param name="waterInletTemperature">流入温度[C]</param>
/// <param name="waterFlowRate">水量[m3/s]</param>
/// <param name="heatLossCoefficients">熱損失係数[kW/K]</param>
/// <param name="ambientTemperature">周囲の温度[C]</param>
/// <param name="volumes">水槽容量[m3]</param>
/// <param name="isForwardFlow">順流か</param>
public static void UpdateTemperature
  (ref IVector temperatures, ref IMatrix wMat, double timeStep, double waterInletTemperature,
  double waterFlowRate, double[] heatLossCoefficients, double ambientTemperature, double[] volumes,
  bool isForwardFlow)
{
  double tRhoc = timeStep / (WATER_SPECIFIC_HEAT * 1000);
  int num = temperatures.Length;
  double wft = waterFlowRate * timeStep;
  wMat.Initialize(0);

  //対角行列を作成
  for (int i = 0; i < num; i++)
  {
    double s = wft / volumes[i];
    double r = heatLossCoefficients[i] / volumes[i] * tRhoc;

    wMat[1, i] = 1 + s + r;
    temperatures[i] += ambientTemperature * r;
    if (isForwardFlow)
    {
      if (i == 0) temperatures[i] += s * waterInletTemperature;
      else wMat[0, i] = -s;
      wMat[2, i] = 0;
    }
    else
    {
      if (i == num - 1) temperatures[i] += s * waterInletTemperature;
      else wMat[2, i] = -s;
      wMat[0, i] = 0;
    }
  }
  LinearAlgebra.SolveTridiagonalMatrix(wMat, ref temperatures);
}
```

連結完全混合型蓄熱槽のクラスを作成する。インスタンス変数、プロパティ、コンストラクタの定義をプログラム 19.2 に示す。1 行はプログラム 19.1 で用いる計算用の記憶領域であり、57 行に示すようにインスタンス生成の際に用意する。5 行は水槽温度を保持する配列であり、過去の温度を保存するために 2 つの変数配列を用意する。

プログラム **19.2**　連結完全混合型蓄熱槽のインスタンス変数、プロパティ、コンストラクタ

Popolo.HVAC.ThermalStorage.MultiConnectedWaterTank class

```
/// <summary>計算領域</summary>
private IMatrix wMat;

/// <summary>水槽温度[C]</summary>
private IVector temperatures, temperatures_Back;

/// <summary>予測計算中か否か</summary>
private bool isForecasting = false;

/// <summary>水槽容量[m3]</summary>
private double[] volumes;

/// <summary>熱損失係数[kW/K]</summary>
private double[] heatLossCoefficients;

/// <summary>タイムステップ[sec]</summary>
private double timeStep = 60;

/// <summary>タイムステップを設定[sec]・取得する</summary>
public double TimeStep
{
  get { return timeStep; }
```

```
23   set { if (0 < value) timeStep = value; }
24 }
25
26 /// <summary>流入水温[C]を取得する</summary>
27 public double WaterInletTemperature { get; private set; }
28
29 /// <summary>流出水温[C]を取得する</summary>
30 public double WaterOutletTemperarture
31 {
32   get
33   {
34     if (IsForwardFlow) return temperatures[temperatures.Length - 1];
35     else return temperatures[0];
36   }
37 }
38
39 /// <summary>流入量[m3/s]を取得する</summary>
40 public double WaterFlowRate { get; private set; }
41
42 /// <summary>周囲の温度[C]を設定・取得する</summary>
43 public double AmbientTemperature { get; set; } = 20;
44
45 /// <summary>順流か否か</summary>
46 public bool IsForwardFlow { get; private set; }
47
48 /// <summary>水槽の数を取得する</summary>
49 public int TankNumber { get { return temperatures.Length; } }
50
51 /// <summary>インスタンスを初期化する</summary>
52 /// <param name="volumes">水槽容量[m3]</param>
53 public MultiConnectedWaterTank(double[] volumes)
54 {
55   int num = volumes.Length;
56
57   wMat = new Matrix(3, num);
58   temperatures = new Vector(num);
59   temperatures_Back = new Vector(num);
60   temperatures.Initialize(20);
61   temperatures_Back.Initialize(20);
62
63   this.volumes = new double[num];
64   volumes.CopyTo(this.volumes, 0);
65   this.heatLossCoefficients = new double[num];
66 }
```

水槽温度の更新関連の処理をプログラム 19.3 に示す。プログラム 19.1 を使うと将来の水槽温度が得られるが、得られた結果によっては別の入力条件による計算結果と比較したい場合がある。そこで、計算処理を「将来予測」と「状態確定」の 2 つに分ける。1~23 行が将来の水槽温度予測処理である。初回の予測の際には 14~18 行で現在の水温を保存する。2 回目以降の計算の際には 12 行で RestoreState メソッドを呼び出すことで、現在の水温を復元してから計算を行う。このような構成とすることで流入水量や流入水温を変化させた場合の温度分布の変化を試行錯誤することができるようになる。21, 22 行でプログラム 19.1 を呼び出して水温を計算する。状態を確定させる場合には 34 行の FixState メソッドを用いる。

プログラム 19.3　水槽温度の更新関連の処理

Popolo.HVAC.ThermalStorage.MultiConnectedWaterTank class

```
 1 /// <summary>将来の水槽温度を予測する</summary>
 2 /// <param name="waterInletTemperature">流入水温[C]</param>
 3 /// <param name="waterFlowRate">流入流量[m3/s]</param>
 4 /// <param name="isForwardFlow">水流の向きは順流か否か</param>
 5 public void ForecastState(double waterInletTemperature, double waterFlowRate, bool isForwardFlow)
 6 {
 7   this.WaterInletTemperature = waterInletTemperature;
 8   this.WaterFlowRate = waterFlowRate;
 9   this.IsForwardFlow = isForwardFlow;
10
11   //既に予測計算中の場合には状態を復元
12   if (isForecasting) RestoreState();
13   else
14   {
15     //現在の温度を保存
```

```
    isForecasting = true;
    for (int i = 0; i < temperatures.Length; i++) temperatures_Back[i] = temperatures[i];
  }

  //水温更新
  UpdateTemperature(ref temperatures, ref wMat, timeStep, WaterInletTemperature, WaterFlowRate,
    heatLossCoefficients, AmbientTemperature, volumes, IsForwardFlow);
}

/// <summary>水槽温度を復元する</summary>
public void RestoreState()
{
  if (isForecasting)
    for (int i = 0; i < temperatures.Length; i++)
      temperatures[i] = temperatures_Back[i];
}

/// <summary>将来の状態を確定する</summary>
public void FixState() { isForecasting = false; }
```

連結完全混合型蓄熱槽の水温および熱損失率の設定処理をプログラム 19.4 に示す。

プログラム 19.4　水温および熱損失率の設定処理

Popolo.HVAC.ThermalStorage.MultiConnectedWaterTank class

```
/// <summary>水槽温度[C]を初期化する</summary>
/// <param name="temperature">初期化する温度[C]</param>
public void InitializeTemperature(double temperature)
{
  RestoreState();
  this.temperatures.Initialize(temperature);
}

/// <summary>水槽温度[C]を初期化する</summary>
/// <param name="temperature">初期化する温度[C]</param>
/// <param name="tankIndex">水槽番号</param>
public void InitializeTemperature(int tankIndex, double temperature)
{
  RestoreState();
  this.temperatures[tankIndex] = temperature;
}

/// <summary>水槽温度[C]を取得する</summary>
/// <param name="tankIndex">水槽番号</param>
/// <returns>水槽温度[C]</returns>
public double GetTemperature(int tankIndex) { return temperatures[tankIndex]; }

/// <summary>熱損失係数[kW/K]を設定する</summary>
/// <param name="tankIndex">水槽番号</param>
/// <param name="heatLossCoefficient">熱損失係数[kW/K]</param>
public void SetHeatLossCoefficient(int tankIndex, double heatLossCoefficient)
{ heatLossCoefficients[tankIndex] = heatLossCoefficient; }

/// <summary>熱損失係数[kW/K]を取得する</summary>
/// <param name="tankIndex">水槽番号</param>
/// <returns>熱損失係数[kW/K]</returns>
public double GetHeatLossCoefficient(int tankIndex)
{ return heatLossCoefficients[tankIndex]; }
```

【例題 19.1】

20m^3×10 槽の連結完全混合型蓄熱槽について、水槽内温度の時系列変化を計算せよ。初期の水槽温度は 17 °C で一定とする。流入水量は 600 L/min で一定とし、水温は最初の 8 時間（蓄熱運転時）が 7 °C、次の 8 時間（放熱運転時）が 17 °C とする。外界の温度は 15 °C で一定とし、熱損失係数は各水槽ともに 0.02 kW/K とする。

【解】

プログラム 19.5 に計算処理を示す。結果をグラフ化すると図 19.3 が得られる。まずは始端槽から温度変化が始まり、時間の経過とともに終端槽にその影響が波及していく様子が確認できる。

プログラム 19.5　連結完全混合型蓄熱槽の温度の時系列変動

```
private static void WaterHeatStorageTest1()
{
  int tNum = 10;
  double wf = 0.6 / 60d;

  double[] volumes = new double[tNum];
  for (int i = 0; i < tNum; i++) volumes[i] = 15;
  MultiConnectedWaterTank wTank = new MultiConnectedWaterTank(volumes);
```

```
 9   wTank.InitializeTemperature(17);
10   wTank.TimeStep = 600;
11   for (int i = 0; i < tNum; i++) wTank.SetHeatLossCoefficient(i, 0.02);
12   wTank.AmbientTemperature = 15;
13
14   using (StreamWriter sw = new StreamWriter("WaterHeatStorageTest1.csv"))
15   {
16     for (int i = 0; i < 48; i++)
17     {
18       sw.Write(i);
19       for (int j = 0; j < tNum; j++) sw.Write(", " + wTank.GetTemperature(j));
20       sw.WriteLine();
21
22       wTank.ForecastState(7, wf, true);
23       wTank.FixState();
24     }
25     for (int i = 0; i < 48; i++)
26     {
27       sw.Write(i);
28       for (int j = 0; j < tNum; j++) sw.Write(", " + wTank.GetTemperature(j));
29       sw.WriteLine();
30
31       wTank.ForecastState(17, wf, false);
32       wTank.FixState();
33     }
34   }
35 }
```

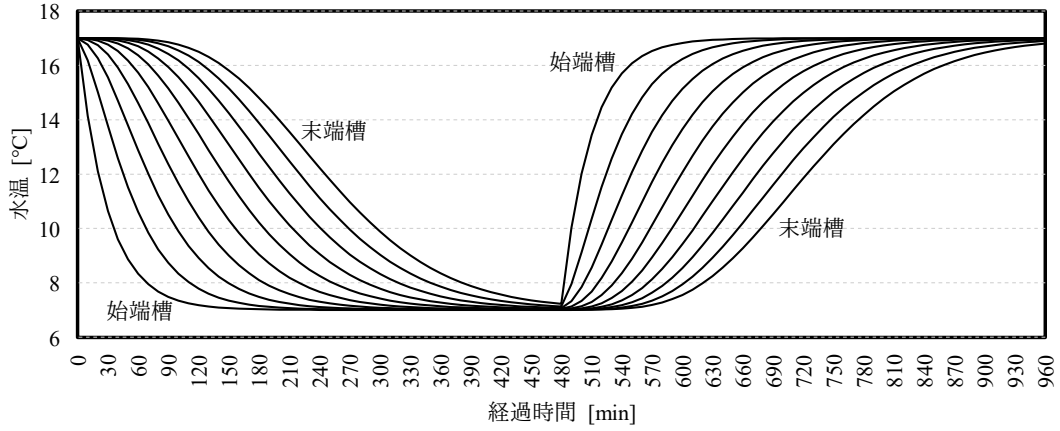


図 19.3　連結完全混合型蓄熱槽の温度の時系列変動

19.3.2　温度成層型蓄熱槽の計算法

式 19.5 を時間（n=0, 1, 2, 3...N）と空間（m=0, 1, 2, 3...M）について離散化して表現すると式 19.15 が得られる[†1]。ただし、空間分割は水槽の最上部を 0、最下部を M とする。

$$\frac{\theta_m^{n+1}-\theta_m^n}{\Delta t}=\begin{cases}\kappa_0\dfrac{2\left(\theta_1^{n+1}-\theta_0^{n+1}\right)}{\Delta z^2}+\dfrac{\Phi_0}{A}\left(\theta_{wi}-\theta_0^{n+1}\right)+\dfrac{K}{\rho\, c_{pW} V}\left(\theta_a-\theta_m^{n+1}\right) & (m=0)\\ \kappa_0\dfrac{2\left(\theta_M^{n+1}-\theta_{M-1}^{n+1}\right)}{\Delta z^2}+\dfrac{\Phi_M}{A}\left(\theta_{wi}-\theta_M^{n+1}\right)+\dfrac{K}{\rho\, c_{pW} V}\left(\theta_a-\theta_m^{n+1}\right) & (m=M)\\ \kappa_0\dfrac{\theta_{m-1}^{n+1}-2\theta_m^{n+1}+\theta_{m+1}^{n+1}}{\Delta z^2}-f(U)+\dfrac{\Phi_m}{A}\left(\theta_{wi}-\theta_m^{n+1}\right)+\dfrac{K}{\rho\, c_{pW} V}\left(\theta_a-\theta_m^{n+1}\right) & (0<m<M)\end{cases} \tag{19.15}$$

式 19.15 第 3 行の右辺第 2 項は移流による項である。連結完全混合の場合の計算と同じように、水流が上向きか下向きかによって異なるため、式 19.16 で切り替える。

†1　第 0 層と第 M 層は断熱境界（$\partial\theta/\partial t=0$）とし、断熱境界の条件である $\theta_{-1}=\theta_1$ と $\theta_{M+1}=\theta_{M-1}$ を用いることで第 1 行と第 2 行の右辺第 1 項は導いている。詳細は参考文献 19.3 を参照。

$$f(U)=\begin{cases} U_{m-1}\dfrac{\theta_m^{n+1}-\theta_{m-1}^{n+1}}{\Delta z} & \text{（下向き）} \\ U_{m+1}\dfrac{\theta_m^{n+1}-\theta_{m+1}^{n+1}}{\Delta z} & \text{（上向き）} \end{cases} \tag{19.16}$$

第 m 層の鉛直方向単位長さあたりの平均的な流入水量 Φ_m [m²/s]は式 19.17 で計算する（各分割領域で式 19.6 を定積分して Δz で除した式）。m_{lm} は混合深さ l_{min} が含まれる分割領域の番号である。

$$\Phi_m=\begin{cases} q_{wi}\dfrac{1}{2l_{mix}^3}\left[3\,l_{mix}^2-\Delta z^2\{3\,m(m+1)+1\}\right] & (m<m_{lm}) \\ q_{wi}\dfrac{1}{2l_{mix}^3}(m\cdot\Delta z-l_{mix})^2\left(m+\dfrac{2l_{mix}}{\Delta z}\right) & (m=m_{lm}) \\ 0 & (m_{lm}<m) \end{cases} \tag{19.17}$$

第 m 層の垂直方向の水速 U_m [m/s]は積算流入水量を用いて式 19.18 で計算する。ただし、流出口がある層以降の層では U_m=0 である。

$$U_m=\frac{\Delta z}{A}\sum_{i=0}^{m}\Phi_i \tag{19.18}$$

式 19.15 に式 19.17 を代入して行列表記すると式 19.19~19.22となる。3 重対角連立一次方程式であるため、基礎編で解説した Thomas algorithm を用いて解くことができる。

$$E_{rhs}=\left(\gamma_{rhs}\right)\left(W_{rhs}-W_{ini}\right) \tag{19.19}$$

$$D=\begin{bmatrix} 2s+r_2+1+q+p & -(2s+r_2) & 0 & 0 \\ -(s+r_1) & 2s+r_1+r_2+1+q+p & -(s+r_2) & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & -(s+r_1) & 2s+r_1+r_2+1+q+p & -(s+r_2) \\ 0 & 0 & -(2s+r_1) & 2s+r_1+1+q+p \end{bmatrix} \tag{19.20}$$

$$\Theta=\left[\theta_0^{n+1},\theta_1^{n+1},\theta_2^{n+1}\ldots\theta_{M-1}^{n+1},\theta_M^{n+1}\right]^T \tag{19.21}$$

$$E=\left[\theta_0^n+q\theta_{wi}+p\theta_a,\theta_1^n+q\theta_{wi}+p\theta_a\ldots\theta_{M-1}^n+q\theta_{wi}+p\theta_a,\theta_M^n+q\theta_{wi}+p\theta_a\right]^T \tag{19.22}$$

ただし、　$s=\dfrac{\Delta t\kappa_0}{\Delta z^2}$　，　$q=\dfrac{\Delta t\Phi}{A}$　，　$p=\dfrac{K\Delta t}{\rho c_{pW}V}$

$$r_1=\begin{cases} U_{m-1}\dfrac{\Delta t}{\Delta z} & \text{（下向き）} \\ 0 & \text{（上向き）} \end{cases}\quad,\quad r_2=\begin{cases} 0 & \text{（下向き）} \\ U_{m+1}\dfrac{\Delta t}{\Delta z} & \text{（上向き）} \end{cases}$$

温度成層型蓄熱槽の水槽内温度分布更新処理をプログラム 19.6 に示す。第 2~4 引数は計算に用いる記憶領域である。温度成層型の場合は領域分割数が多いため、毎回の記憶領域確保は計算速度にかなり影響を与える。領域分割数に一般的な値は無いが、HVACSIM+(J)と LCEM の温度成層型蓄熱槽モデルでは標準値として 100 分割が採用されている。

32~36 行は水流が無い場合の処理であり、熱拡散と外部への熱損失のみが生じる。

37~95 行は上部から温水が流入した場合、あるいは下部から冷水が流入した場合の処理である。水の密度が逆転しているため、逆転の続く層までが完全混合になるとし、平均水温を計算する。この場合には噴流は各層に等配分されるものとする（74 行）。

96~147 行は通常の流入の場合である。102~112 行で式 19.11 を適用して噴流の到達する層を判定する。114~129 行で式 19.7 と式 19.8 を使って混合深さを計算する。131~146 行で式 19.17 と式 19.18 を

使って各層での流入水量 Φ_m と水速 U_m を計算する。

149~178 行で式 19.19~19.22の 3 重対角行列とベクトルを作成し、179 行でこれを解く。

プログラム **19.6**　温度成層型蓄熱槽の水槽内温度分布更新処理

Popolo.HVAC.ThermalStorage.MultipleStratifiedWaterTank class

```
/// <summary>槽内温度を計算する</summary>
/// <param name="temperature">槽内温度分布[C]</param>
/// <param name="wMat">計算領域 (3×層の数) </param>
/// <param name="wVec1">計算領域(層の数)</param>
/// <param name="wVec2">計算領域(層の数)</param>
/// <param name="timeStep">タイムステップ[sec]</param>
/// <param name="waterInletTemperature">流入水温[C]</param>
/// <param name="waterFlowRate">流入量[m3/s]</param>
/// <param name="heatLossCoefficient">熱損失率[kW/K]</param>
/// <param name="ambientTemperature">周囲の温度[C]</param>
/// <param name="waterDepth">水槽の水深[m]</param>
/// <param name="pipeDiameter">流出入口直径[m]</param>
/// <param name="sectionalArea">断面積[m2]</param>
/// <param name="pipeInstallationLayer">流入出口設置層番号</param>
/// <param name="isDownFlow">下向きか否か</param>
public static void UpdateTemperature
  (ref IVector temperature, ref IMatrix wMat, ref IVector wVec1, ref IVector wVec2, double timeStep,
  double waterInletTemperature, double waterFlowRate, double heatLossCoefficient, double ambientTemperature,
  double waterDepth, double pipeDiameter, double sectionalArea, int pipeInstallationLayer, bool isDownFlow)
{
  int layerNum = temperature.Length;  //分割数
  double dz = waterDepth / layerNum;  //分割幅

  //平均水温を計算
  double aveTemp = 0;
  for (int i = 0; i < layerNum; i++) aveTemp += temperature[i];
  aveTemp /= layerNum;

  //混合域の噴流配分の計算
  IVector phiN = wVec1;
  IVector uN = wVec2;
  if (waterFlowRate == 0)
  {
    phiN.Initialize(0);
    uN.Initialize(0);
  }
  //温度逆転の場合は逆転が続く層まで完全混合とし、噴流は等分割
  else if ((isDownFlow && waterInletTemperature <= temperature[0])
    || (!isDownFlow && temperature[layerNum - 1] <= waterInletTemperature))
  {
    //温度逆転範囲を求めて平均温度を計算
    double mixTemp, mixedAve;
    int mixedNum;
    if (isDownFlow)
    {
      mixedAve = temperature[0];
      mixTemp = temperature[0]
        + timeStep / (sectionalArea * dz) * (waterInletTemperature - temperature[0]) * waterFlowRate;
    }
    else
    {
      mixedAve = temperature[temperature.Length - 1];
      mixTemp = temperature[temperature.Length - 1]
        + timeStep * (waterInletTemperature - temperature[temperature.Length - 1]) / (sectionalArea * dz);
    }

    for (mixedNum = 1; mixedNum < layerNum; mixedNum++)
    {
      if (isDownFlow)
      {
        if (temperature[mixedNum] < mixTemp) break;
        mixTemp = (mixTemp * mixedNum + temperature[mixedNum]) / (mixedNum + 1);
        mixedAve += temperature[mixedNum];
      }
      else
      {
        int tgtLayer = layerNum - (mixedNum + 1);
        if (mixTemp < temperature[tgtLayer]) break;
        mixTemp = 0.5 * (mixTemp + temperature[tgtLayer]);
        mixedAve += temperature[tgtLayer];
      }
    }
    mixedAve /= mixedNum;
```

```
74    double bf = waterFlowRate / (mixedNum * dz);
75
76    //平均温度と噴流配分を設定
77    for (int i = 0; i < layerNum; i++)
78    {
79      int tgtLayer = i;
80      if (!isDownFlow) tgtLayer = layerNum - (i + 1);
81
82      if (i < mixedNum)
83      {
84        phiN[tgtLayer] = bf;
85        temperature[tgtLayer] = mixTemp;
86        uN[tgtLayer] = dz / sectionalArea * phiN[tgtLayer];
87      }
88      else phiN[tgtLayer] = uN[tgtLayer] = 0;
89      if (i != 0)
90      {
91        if (isDownFlow) uN[tgtLayer] += uN[tgtLayer - 1];
92        else uN[tgtLayer] += uN[tgtLayer + 1];
93      }
94    }
95  }
96  else
97  {
98    double rhowi = Water.GetLiquidDensity(waterInletTemperature);
99    double pipeSA = Math.Pow(pipeDiameter / 2, 2) * Math.PI; //流入口断面積[m2]
100   double uwi2 = Math.Pow(waterFlowRate / pipeSA, 2);//流入速度２乗[(m/s)^2]
101
102   //噴流が到達する層を求める
103   int lmax;
104   double tgt = rhowi * uwi2 / (G_FORCES * dz);
105   double rr = 0;
106   for (lmax = 0; lmax < layerNum - 1; lmax++)
107   {
108     if (isDownFlow) rr += (Water.GetLiquidDensity(temperature[lmax]) - rhowi);
109     else rr += (rhowi - Water.GetLiquidDensity(temperature[layerNum - lmax - 1]));
110     if (tgt < rr) break;
111   }
112   if (!isDownFlow) lmax = layerNum - lmax - 1;
113
114   //混合域深さの計算
115   double tlMax = temperature[lmax];
116   double lm;
117   //等温の層まで到達する場合には混合域100%とする
118   if (tlMax == waterInletTemperature) lm = waterDepth;
119   else
120   {
121     double rho0 = Water.GetLiquidDensity(tlMax);
122     double ari = pipeDiameter * G_FORCES * Math.Abs(rho0 - rhowi) / (rho0 * uwi2);  //アルキメデス数
123     double ndt = 0;
124     if (isDownFlow)
125       for (int i = 0; i < lmax; i++) ndt += (temperature[i] - tlMax);
126     else for (int i = lmax; i < layerNum; i++) ndt += (temperature[i] - tlMax);
127     ndt = (ndt * dz) / (waterDepth * (waterInletTemperature - tlMax));  //無次元時間
128     lm = waterDepth * 0.8 * Math.Pow(ari, -0.5) * pipeDiameter / waterDepth + 0.5 * ndt;
129   }
130
131   double z1 = 0;
132   double bf = waterFlowRate / (2 * lm * lm * lm);
133   for (int i = 0; i < layerNum; i++)
134   {
135     int ln = i;
136     if (!isDownFlow) ln = layerNum - (i + 1);
137     double z2 = z1 + dz;
138     if (z2 < lm) phiN[ln] = bf * (3 * lm * lm - dz * dz * (3 * i * (i + 1) + 1));
139     else if (lm < z1) phiN[ln] = 0;
140     else phiN[ln] = bf * Math.Pow(i * dz - lm, 2) * (i + 2 * lm / dz);
141
142     if (i == 0) uN[ln] =  dz / sectionalArea * phiN[ln];
143     else if (isDownFlow) uN[ln] = uN[ln - 1] + dz / sectionalArea * phiN[ln];
144     else uN[ln] = uN[ln + 1] + dz / sectionalArea * phiN[ln];
145     z1 = z2;
146   }
147 }
148
149 //3重対角行列を解く
150 wMat.Initialize(0);
151 int outletLayer = pipeInstallationLayer;
152 if (!isDownFlow) outletLayer = layerNum - pipeInstallationLayer - 1;
153 double s = timeStep * Water.GetLiquidThermalDiffusivity(aveTemp) / (dz * dz);
```

```
154   double p = heatLossCoefficient * timeStep /
155     (Water.GetLiquidDensity(aveTemp) * WATER_SPECIFIC_HEAT * waterDepth * sectionalArea);
156   for (int i = 0; i < layerNum; i++)
157   {
158
159     double r1 = 0;
160     double r2 = 0;
161     if (isDownFlow && i != 0 && i <= outletLayer) r1 = uN[i - 1] * timeStep / dz;
162     if (!isDownFlow && i != layerNum - 1 && outletLayer <= i) r2 = uN[i + 1] * timeStep / dz;
163
164     if (i == layerNum - 1) wMat[0, i] = -(2 * s + r1);
165     else if (i != 0) wMat[0, i] = -(s + r1);
166
167     if (i == 0) wMat[2, i] = -(2 * s + r2);
168     else if (i != layerNum - 1) wMat[2, i] = -(s + r2);
169
170     wMat[1, i] = 2 * s + r1 + r2 + 1 + p;
171     temperature[i] += p * ambientTemperature;
172     if (phiN[i] != 0)
173     {
174       double q = timeStep * phiN[i] / sectionalArea;
175       wMat[1, i] += q;
176       temperature[i] += q * waterInletTemperature;
177     }
178   }
179   LinearAlgebra.SolveTridiagonalMatrix(wMat, ref temperature);
180 }
```

温度成層型蓄熱槽のクラスを作成する。インスタンス変数、プロパティ、コンストラクタの定義をプログラム 19.7 に示す。構成は連結完全混合型とほぼ同様である。

プログラム 19.7　温度成層型蓄熱槽のインスタンス変数、プロパティ、コンストラクタ

Popolo.HVAC.ThermalStorage. MultipleStratifiedWaterTank class

```
 1 /// <summary>計算領域</summary>
 2 private IMatrix wMat;
 3 private IVector wVec1, wVec2;
 4
 5 /// <summary>水槽内温度分布[C]</summary>
 6 private IVector temperatures, temperatures_Back;
 7
 8 /// <summary>予測計算中か否か</summary>
 9 private bool isForecasting = false;
10
11 /// <summary>タイムステップ[sec]</summary>
12 private double timeStep = 60;
13
14 /// <summary>タイムステップ[sec]を設定・取得する</summary>
15 public double TimeStep
16 {
17   get { return timeStep; }
18   set { if (0 < value) timeStep = value; }
19 }
20
21 /// <summary>水深[m]を取得する</summary>
22 public double WaterDepth { get; private set; }
23
24 /// <summary>水平断面積[m2]を取得する</summary>
25 public double SectionalArea { get; private set; }
26
27 /// <summary>水量[m3]を取得する</summary>
28 public double WaterVolume { get { return WaterDepth * SectionalArea; } }
29
30 /// <summary>流入水温[C]を取得する</summary>
31 public double WaterInletTemperature { get; private set; }
32
33 /// <summary>上部流出口水温[C]を取得する</summary>
34 public double UpperOutletTemperarture
35 { get { return temperatures[temperatures.Length - PipeInstallationLayer - 1]; } }
36
37 /// <summary>下部流出口水温[C]を取得する</summary>
38 public double LowerOutletTemperarture
39 { get { return temperatures[PipeInstallationLayer]; } }
40
41 /// <summary>流入量[m3/s]を取得する</summary>
42 public double WaterFlowRate { get; private set; }
43
44 /// <summary>流出入口直径[m]を取得する</summary>
45 public double PipeDiameter { get; private set; }
```

```
46
47 /// <summary>流出入口が設置された層番号を取得する</summary>
48 public int PipeInstallationLayer { get; private set; }
49
50 /// <summary>熱損失率[kW/K]を設定・取得する</summary>
51 public double HeatLossCoefficient { get; set; }
52
53 /// <summary>周囲の温度[C]を設定・取得する</summary>
54 public double AmbientTemperature { get; set; } = 20;
55
56 /// <summary>下向き水流か否か</summary>
57 public bool IsDownFlow { get; private set; }
58
59 /// <summary>水槽の分割数を取得する</summary>
60 public int LayerNumber { get { return temperatures.Length; } }
61
62 /// <summary>インスタンスを初期化する</summary>
63 /// <param name="waterDepth">水深[m]</param>
64 /// <param name="sectionalArea">水平断面積[m2]</param>
65 /// <param name="pipeDiameter">流出入口直径[m]</param>
66 /// <param name="pipeInstallationHeight">流出入口設置高さ[m]</param>
67 /// <param name="layerNumber">分割数</param>
68 public MultipleStratifiedWaterTank
69  (double waterDepth, double sectionalArea, double pipeDiameter, double pipeInstallationHeight,int layerNumber)
70 {
71   this.WaterDepth = waterDepth;
72   this.SectionalArea = sectionalArea;
73   this.PipeDiameter = pipeDiameter;
74   wMat = new Matrix(3, layerNumber);
75   wVec1 = new Vector(layerNumber);
76   wVec2 = new Vector(layerNumber);
77   temperatures = new Vector(layerNumber);
78   temperatures_Back = new Vector(layerNumber);
79   temperatures.Initialize(20);
80   temperatures_Back.Initialize(20);
81
82   double dz = WaterDepth / LayerNumber;
83   PipeInstallationLayer = (int)Math.Floor(pipeInstallationHeight / dz);
84 }
```

槽内温度分布の更新関連の処理をプログラム 19.8 に示す。連結完全混合型と同様に将来予測と状態確定に処理を分割する。

プログラム 19.8　水槽温度の更新関連の処理

Popolo.HVAC.ThermalStorage. MultipleStratifiedWaterTank class

```
 1 /// <summary>将来の水温分布を予測する</summary>
 2 /// <param name="waterInletTemperature">流入水温[C]</param>
 3 /// <param name="waterFlowRate">流入流量[m3/s]</param>
 4 /// <param name="isDownFlow">水流の向きは下向きか否か</param>
 5 public void ForecastState(double waterInletTemperature, double waterFlowRate, bool isDownFlow)
 6 {
 7   this.WaterInletTemperature = waterInletTemperature;
 8   this.WaterFlowRate = waterFlowRate;
 9   this.IsDownFlow = isDownFlow;
10
11   //既に予測計算中の場合には状態を復元
12   if (isForecasting) RestoreState();
13   else
14   {
15     //現在の温度を保存
16     isForecasting = true;
17     for (int i = 0; i < temperatures.Length; i++) temperatures_Back[i] = temperatures[i];
18   }
19
20   //水温更新
21   UpdateTemperature
22     (ref temperatures, ref wMat, ref wVec1, ref wVec2, timeStep, WaterInletTemperature, WaterFlowRate,
23     HeatLossCoefficient, AmbientTemperature, WaterDepth, PipeDiameter, SectionalArea, PipeInstallationLayer,
24     IsDownFlow);
25 }
26
27 /// <summary>水温分布を復元する</summary>
28 public void RestoreState()
29 {
30   if(isForecasting)
31     for (int i = 0; i < temperatures.Length; i++)
32       temperatures[i] = temperatures_Back[i];
33 }
34
```

```
35 /// <summary>将来の状態を確定する</summary>
36 public void FixState() { isForecasting = false; }
```

温度成層型蓄熱槽の水槽内温度設定処理をプログラム 19.9 に示す。

プログラム 19.9　水槽内温度設定処理

Popolo.HVAC.ThermalStorage. MultipleStratifiedWaterTank class

```
 1 /// <summary>水槽内温度[C]を初期化する</summary>
 2 /// <param name="temperature">初期化する温度[C]</param>
 3 public void InitializeTemperature(double temperature)
 4 {
 5   RestoreState();
 6   this.temperatures.Initialize(temperature);
 7 }
 8
 9 /// <summary>水槽内温度[C]を初期化する</summary>
10 /// <param name="temperature">初期化する温度[C]</param>
11 /// <param name="layerNumber">分割番号</param>
12 public void InitializeTemperature(int layerNumber, double temperature)
13 {
14   RestoreState();
15   this.temperatures[layerNumber] = temperature;
16 }
17
18 /// <summary>水槽内温度[C]を取得する</summary>
19 /// <param name="layerNumber">分割番号</param>
20 /// <returns>水槽内温度[C]</returns>
21 public double GetTemperature(int layerNumber) { return temperatures[layerNumber]; }
```

【例題 19.2】

0.8m×0.8m の正方形断面で水深が 0.8m の温度成層型蓄熱槽について下記のステップ変化を与えた場合の応答を予測せよ。ただし、流出入口の直径は 0.05m とし、設置高さは 0.72m とする。周囲の温度は 20 ºC とし、蓄熱槽は熱伝導率 0.004 W/(m·K)の断熱材 100mm で囲うものとする。

1）初期温度 15 ºC、流入量 0.25 m³/h。流入温度は換水回数 0.5 回まで 21 ºC、それ以降は 25 ºC。

2）初期温度 15 ºC、流入量 0.25 m³/h。流入温度は換水回数 0.5 回まで 24 ºC、それ以降は 21 ºC。

3）初期温度 6 ºC、流入温度 12 ºC。流入流量は換水回数 0.3~0.8 回では 0.1 m³/h、それ以外は 0.5 m³/h。

【解】

プログラム 19.10 に計算処理を示す。結果をグラフ化すると図 19.4 が得られる。それぞれの実線は、ある換水回数における水槽内の温度分布を表しており、このように表現された図を蓄熱槽の温度プロフィールと呼ぶ。効率的に熱を蓄えるためには水槽内で水が混合しない方が良く、温度プロフィールが変曲する部分が横に長く縦に短い方が良い。例えば図 19.4 中の換水回数 0.5 回の温度プロフィールを図 19.4 右の換水回数 0.9 回の温度プロフィールと比較すると後者の方が縦に広がっている。これは流入流量が大きいために内部の水が混合されてしまった結果である。

本問は相良による実験[19.2)]と条件を合わせており、図 19.4 に記載のプロットは相良による実測結果である。実験では完全なステップ温度変化を与えることができないため、計算値の方が全体的にやや反応が早い結果となっているが、傾向は概ね再現できていることがわかる。

プログラム 19.10　温度成層型蓄熱槽のステップ応答

```
 1 private static void WaterHeatStorageTest2()
 2 {
 3   MultipleStratifiedWaterTank tank = new MultipleStratifiedWaterTank(0.8, 0.8 * 0.8, 0.05, 0.72, 100);
 4   tank.HeatLossCoefficient = 0.001 * 6 * 0.04 / 0.1;
 5   tank.AmbientTemperature = 20;
 6   tank.TimeStep = 10;
 7
 8   using (StreamWriter sWriter = new StreamWriter
 9     ("WaterHeatStorageTest2.csv", false, Encoding.GetEncoding("Shift_JIS")))
10   {
11     sWriter.WriteLine("ステップ温度上昇");
12     double waterChangeRate = 0.0;
13     double trig = 0.0;
14     tank.InitializeTemperature(15);
15     while (waterChangeRate < 1.0)
16     {
17       double inletTemp = 21;
18       if (0.5 < waterChangeRate) inletTemp = 25;
19       tank.ForecastState(inletTemp, 0.25 / 3600, true);
20       tank.FixState();
21       waterChangeRate += (0.25 / 3600 * tank.TimeStep) / tank.WaterVolume;
22
```

```
23        if (trig < waterChangeRate)
24        {
25          sWriter.Write(waterChangeRate.ToString("F1"));
26          for (int j = 0; j < tank.LayerNumber; j++) sWriter.Write(", " + tank.GetTemperature(j));
27          sWriter.WriteLine();
28          trig += 0.1;
29        }
30      }
31
32      sWriter.WriteLine("ステップ温度低下");
33      waterChangeRate = 0.0;
34      trig = 0.0;
35      tank.InitializeTemperature(15);
36      while (waterChangeRate < 1.0)
37      {
38        double inletTemp = 24;
39        if (0.5 < waterChangeRate) inletTemp = 21;
40        tank.ForecastState(inletTemp, 0.25 / 3600, true);
41        tank.FixState();
42        waterChangeRate += (0.25 / 3600 * tank.TimeStep) / tank.WaterVolume;
43
44        if (trig < waterChangeRate)
45        {
46          sWriter.Write(waterChangeRate.ToString("F1"));
47          for (int j = 0; j < tank.LayerNumber; j++) sWriter.Write(", " + tank.GetTemperature(j));
48          sWriter.WriteLine();
49          trig += 0.1;
50        }
51      }
52
53      sWriter.WriteLine("ステップ流量変化");
54      waterChangeRate = 0.0;
55      trig = 0.0;
56      tank.InitializeTemperature(6);
57      while (waterChangeRate < 1.0)
58      {
59        double waterFlowRate = 0.5 / 3600;
60        if (0.3 < waterChangeRate && waterChangeRate < 0.8) waterFlowRate = 0.1 / 3600;
61        tank.ForecastState(12, waterFlowRate, true);
62        tank.FixState();
63        waterChangeRate += (waterFlowRate * tank.TimeStep) / tank.WaterVolume;
64
65        if (trig < waterChangeRate)
66        {
67          sWriter.Write(waterChangeRate.ToString("F1"));
68          for (int j = 0; j < tank.LayerNumber; j++) sWriter.Write(", " + tank.GetTemperature(j));
69          sWriter.WriteLine();
70          trig += 0.1;
71        }
72      }
73    }
74 }
```

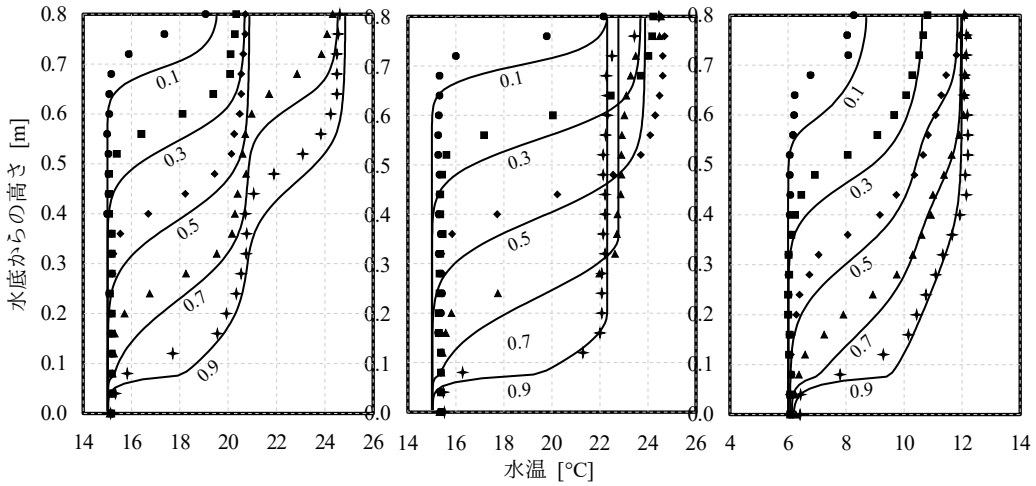


図 19.4　温度成層型蓄熱槽のステップ応答（換水回数別）

（左：ステップ温度上昇、中：ステップ温度低下、右：ステップ流量変化）

【第 19 章 記号表】

A	：水槽の水平方向切断面積 **[m²]**	t^*	：無次元時間 **[-]**
Ar	：アルキメデス数	U	：槽内垂直方向流速 **[m/s]**
c_{pW}	：水の定圧比熱 **[kJ/(kg·K)]**	u_{wi}	：流入口での水速 **[m/s]**
d_0	：流入口円管の直径**[m]**	V	：水槽容量**,** 水の容積 **[m³]**
g	：重力加速度 **[m²/s]**	z	：水面からの深さ方向距離 **[m]**
K	：熱損失率 **[kW/K]**	Δt	：タイムステップ **[sec]**
L	：水深 **[m]**	κ	：水の拡散係数 **[m²/s]**
l_{jet}	：噴流到達深さ **[m]**	θ	：水温 **[ºC]**
l_{mix}	：混合深さ **[m]**	θ_0	：槽内基準温度 **[ºC]**
Q_{st}	：蓄熱量 **[kJ]**	Φ	：鉛直方向単位長さあたりの流入流量 **[m³/s]**
q_{wi}	：流入流量 **[m³/s]**	ρ	：水の密度 **[kg/m³]**
R_m	：無次元深さ **[-]**	ρ_0	：槽内基準密度 **[kg/m³]**

【第 19 章 参考文献】

19.1） 辻本誠, 相良和伸, 中原信生: 蓄熱槽に関する研究 第 1 報 成層型蓄熱槽の槽内混合機構に関する実験的研究, 空気調和・衛生工学会論文集, No.16, 1981.6, pp.23-35

19.2） 北原博亮, 岩田剛, 相良和伸: 温度成層型蓄熱槽の変動入力条件に対応した槽内混合モデルに関する研究, 空気調和・衛生工学会論文集, No.96, 2005.1, pp.31-40

19.3） 平瀬創也: C#で学ぶ偏微分方程式の数値解法, 東京電機大学出版局, 2009

19.4） 棚橋隆彦: はじめての CFD -移流拡散方程式-, コロナ社, 2005

19.5） 中原信生, 山羽基: 氷蓄熱槽の熱特性に関する研究 第 3 報 アイスオンコイル型氷蓄熱槽のシミュレーションモデルと蓄熱槽効率推定表の作成, 空気調和・衛生工学会論文集, Vol.56, pp.13-24, 1994

19.6） S. C. Silver, J. W. Jones, J. L. Peterson and B. D. Hunn: CBS / ICE; A Computer Program for Simulation of Ice Storage System, ASHRAE Transactions, CH-89-22, 1989

19.7） 井上宇一: 冷凍空調史, 第 8 章 わが国の空調の発達, pp.287, 日本冷凍空調設備工業連合会, 1993

19.8） 倉賀野武士, 小嶋稔, 高橋紀行, 木邨繁久, 山崎慶太, 石黒武: 温度成層型蓄熱槽のディストリビュータ形状による性能に関する模型実験について, 空気調和・衛生工学会 学術講演会論文集, pp.881-884, 2001

19.9） 三原一伸, 高須彰, 鈴木隆, 中谷直一: 大規模蓄熱槽を有する DHC の導入効果 : (第 2 報)ディストリビュータの形状選定と実測結果, 空気調和・衛生工学会 学術講演会論文集, pp.913-916, 1988

19.10） 蓄熱技術の歴史・現在・未来, 建築設備と配管工事, Vol.50, No.10, 2012.8

第20章　エネルギー消費と経済性
(Energy Consumption and Economic Efficiency)

20.1　概要

前章までは、各種の設備機器の物理的な挙動を計算する方法について解説してきた。モデルの入出力は基本的には物理量であり、自然科学による方法論である。一方で、現実の建設活動は社会性の大きい行為であり、人間と無関係には成立しない。この意味では、計算によって得られた物理量を人間にとっての価値に結びつける社会科学的な方法論が求められる。本章ではその代表的な手法である経済性について解説する。経済学は比較的最近の学問であり、経済学の父と呼ばれるアダム・スミスによる国富論をその本格的な成立の時期とみなせば、二百数十年の歴史があるに過ぎない。しかし一方で、その取り扱う範囲は非常に広範であり、1 つの章でその全貌をうかがうことはできない。本章では特に建築のエネルギー性能の評価にあたって応用可能性の高い「資産価格決定」と「外部効果」について解説する。

建築設備へ投資する目的は、将来にわたって、より少ない投入エネルギーで快適な活動が確保できるという正の価値が生じるからである。資産価格決定の理論とは、このような、将来において連続して発生する価値を根拠に現在価値を推定する理論である。本章では設備のイニシャルコストとエネルギーコストの計算法に加え、現代ポートフォリオ理論を用いた最適省エネ投資配分と不動産鑑定理論におけるエネルギー消費の取り扱いについても解説する。

経済活動が取引当事者以外の者に影響を与える場合には、市場原理だけでは最適資源配分を達成できない。エネルギー価格は基本的には現在そのエネルギーを生み出すために必要な費用にもとづいて定まり、将来におけるエネルギー枯渇や過剰消費に伴う環境影響を織り込めていない可能性がある。この意味では、最適なエネルギー消費は現在のエネルギーコストのみでは決定できない外部効果を伴った問題である。本章では外部効果がある場合に最適資源配分が達成できない理由と、これを実現するための方法について解説する。

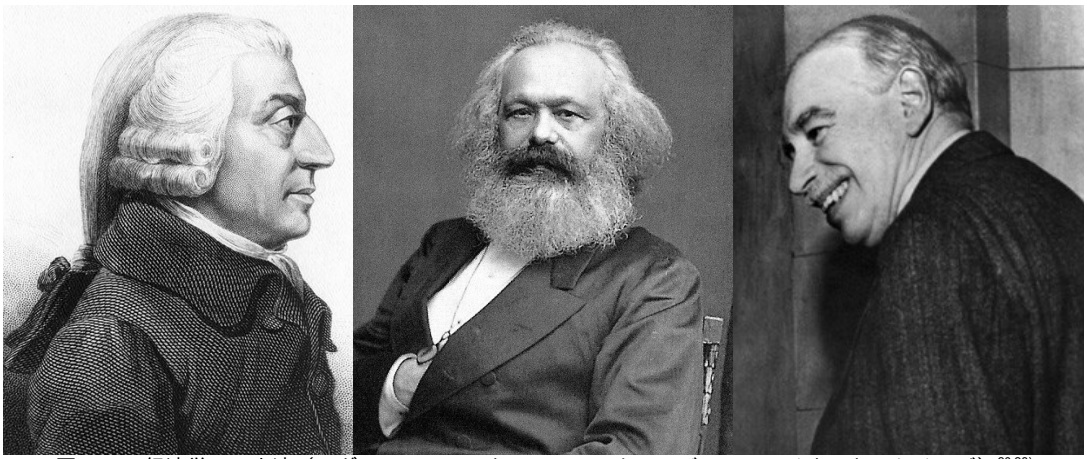


図 20.1　経済学の巨人達（アダム・スミス、カール・マルクス、ジョン・メイナード・ケインズ）[20.28)]

20.2 資産価格の決定

20.2.1 現在価値

資産の価格はその資産から将来得られる利益の合計であり、数式で表現すると式 20.1 となる。

$$NPV = \sum_{n=0}^{N} \frac{CF_n}{(1+Y)^n} \tag{20.1}$$

NPV は割引現在価値（NPV : Net Present Value）と呼ばれる概念であり、資産価値に等しい。右辺の CF_n は将来の第 n 期において生じる利益である。このような将来時点の一連の利益をキャッシュフロー（CF: Cash Flow）と呼ぶ。NPV も CF も単位は貨幣数量であり、式 20.1 を適用する際には設備システムのもたらす各期の価値を金銭換算する必要がある。この方法については次節で解説する。

Y は割引率と呼ばれる概念であり、将来において生じる金銭価値を現在時点の金銭価値に換算するための数値である。例えば、現在手元にある 1 万円は 1 年後においても 1 万円の価値ではない。現在時点において国債への投資を行えば金利 r %によって、1 年後には 1×(1+r %) 万円の価値が得られるからである。逆に捉えれば、1 年後の 1 万円の現在価値は、1÷(1+r %) 万円でしかない。このように時間効果のみを捉えれば割引率 $Y = r$ となるが、実際には Y には金利以外の概念が含まれる。例えば、現在時点において 1 万円で LED 照明器具を購入して蛍光灯と交換した結果、1 年間で投資額に対して r %の光熱費の削減が期待できるとする。一見、上記の国債への投資と LED 照明器具導入への投資の価値は等価であるようにみえるが、これは誤りである。何故ならば国債への投資は実現性の極めて高い安定した投資である一方で、LED 照明器具による光熱費の削減は建物の運用の如何などによって効果が大きく変動する不確実性（リスク）を持つからである。世の中には揺らぎの大きな投資を好む人が多いことも事実であるが、経済学においては人間は安定的でより確実な投資を好むという仮定を置くことが多い。即ち人間は、期待値として同じ 50 万円であっても、半々の確率で 0 円か 100 万円かが得られるギャンブルよりは、100 %の確率で確実に手に入る 50 万円を好むという仮定である。このような仮定を「リスク回避的[†1)]」と言う。この仮定にたてば、リスクのある投資に対してはその価値を割り引いて捉える必要がある。リスクの大きさを g とすると、LED 照明器具への投資による 1 年後の光熱費削減は$(1 + r + g)$で現在価値に割り戻す必要があることがわかる。実際の投資には様々なリスクが折り重なっており、これらのリスクを積み上げた概念が割引率 Y である（式 20.2）。

$$Y = r + \sum_{n=0}^{\infty} g_n \tag{20.2}$$

理論としては、他の代替可能な投資商品と比較した場合の、当該投資固有のリスクを評価することで割引率 Y を計算することはできる。例えば省エネルギー化のための設備投資であれば、水道光熱費の削減に影響を与える各種の条件を確率変数として捉え、将来時点の水道光熱費削減額の実現値を確率分布として捉えることが必要である。この評価方法の理論に関しては 20.2.4 節で解説する。一方で、現在のところは建築設備分野においてこのような研究は未成熟であり、現実には市場において観察される指標等を参考に設定する他に方法は無いだろう。例えば、当該設備を導入する建物の用途や規模に着目し、不動産賃貸事業全体としての割引率の設定値を参考とするという方法が考えられる。

†1 逆に危険を好む仮定を「リスク愛好的」と呼ぶ。魅惑的な響きである。また、不確実性に影響を受けず、期待値のみで行動するという仮定を「リスク中立的」と呼ぶ。あまり人間的でなく、つまらない。

近年では REIT（Real Estate Investment Trast：不動産投資信託）の活発化の影響もあり、用途や規模に応じた不動産賃貸事業の割引率の情報は入手が容易になりつつある†1)。また、各種の不動産マーケット情報も整備が進んでいる†2)。参考に、日本不動産研究所によるアンケート調査[20.11]の結果にもとづいて作成した 2004 年~2014 年の期待利回り†3)の推移を図 20.2 に示す。以下の傾向を読み取れる。1）オフィスに比較すると住宅や商業の利回りは大きい傾向にある、2）景気の変動の影響を受けて期間的に変化する、3）郊外よりも都心の方が利回りが小さい傾向にある。この他、規模や建物の築年・グレードなど、多くの要素に影響を受けるため、計画する建物に応じて適切な利回りを把握して経済性を検討する必要がある。

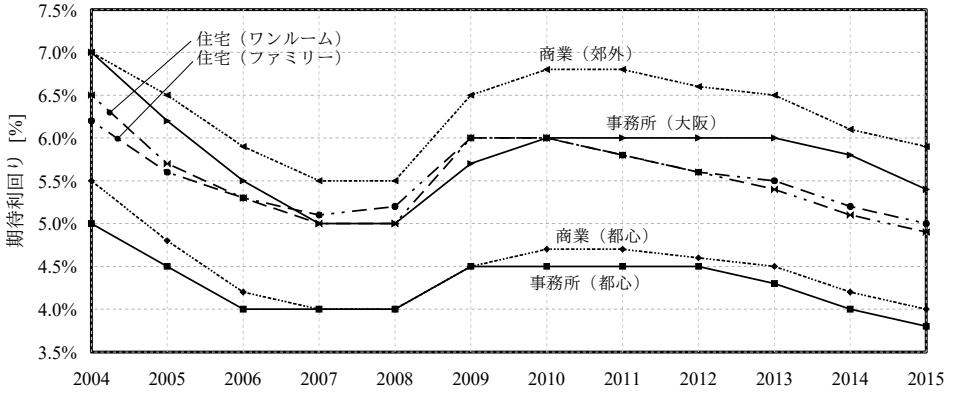


図 20.2　不動産の用途別期待利回り（アンケート調査）

20.2.2　エネルギーコスト

エネルギーの種類は様々であるが、建築設備において一般的に使われるエネルギーは電気およびガスである。本節ではこれらの両エネルギーのコストについて説明する。

エネルギーコストは当該エネルギーを供給する事業主体が定めた料金体系[20.1) 20.2)]にもとづくため、一概には計算できないが、通常は瞬間的なエネルギー使用の大きさ（ピーク負荷）と、期間を通じてのエネルギー使用量と、2 つの面から捉えられる。多くの場合、前者にもとづくコストを基本料金、後者にもとづくコストを従量料金と呼ぶ。一般的なエネルギーコストの計算式を式 20.3 に示す。C はエネルギーコスト[円]、R_{pk} [円/kW または円/(Nm^3/h)]は基本料金率、E_{pk} [kW または Nm^3/h]はピーク負荷、R_{cs} は従量料金率[円/kWh または円/Nm^3] 、E_{cs} [kWh または Nm^3]はエネルギー使用量である。なお、通常はピーク負荷の値としては、年間を通して現実に発生した最大の電力（ガス）需要ではなく、契約上の最大電力（ガス）の値を用いることに注意する†4)。

$$C = R_{pk} E_{pk} + R_{cs} E_{cs} \tag{20.3}$$

正確なコストを知るためには約款にもとづいて基本料金と従量料金を計算する必要があるが、煩雑であり普遍性のある評価とはなりづらい。そこで研究として経済性を検討する場合には基本料金と従量料金の合計額をエネルギー使用量で除したフラットレートと呼ばれる概念を用いることが多い。フラットレートの計算式を式 20.4 に示す。

†1　REIT の有価証券報告書には、各社のポートフォリオに組み込んだ不動産の鑑定評価を行った際に用いた還元利回りや割引率などの数値が掲載されている。

†2　例えば一般財団法人日本不動産研究所の「不動産投資家調査」や CBRE による「事業用不動産レポート」等。

†3　期待利回りは厳密には割引率とは異なる概念であるが、時系列や用途による傾向を把握するという目的においては同様のものと扱って問題ない。詳細は参考文献を参照[20.7) 20.8)]。

†4　現実には過大な契約値によりエネルギーコストが押し上げられているケースがある。設備的な詳細検討の結果を簡単に逆転させるほどに影響力が大きい場合があるため、経済性の検討に際しては契約自体の良否をまず検討すべきである。

$$R_{flt}=\frac{R_{pk}E_{pk}+R_{cs}E_{cs}}{E_{cs}}=R_{pk}\frac{E_{pk}}{E_{cs}}+R_{cs} \tag{20.4}$$

参考に日本エネルギー経済研究所のエネルギー源別価格を参照すると、電力総合単価は 19.20 円/kWh、大口電力は 12.48 円/kWh、都市ガス（大手 3 社）は 9.18 円/千 kcal ≈ 99 円/Nm³ である[20.3]。一方で、個別の契約にもとづく具体的なエネルギーコストは一般的なフラットレートから大きく離れる場合も多いため、実務として経済性検討を行う場合には対象建物の過去の光熱費の資料などを入手し、これにもとづいて計算を進めることが望ましい。

【例題 20.1】

過去 1 年間の電力消費量実績が下記の通りであった。基本料金率を 1,680 円/kW、従量料金率を夏季（7/1~9/30）が 17 円/kWh、その他季が 16 円/kWh として[†1]、年間の電力料金とフラットレートを求めよ。ただし契約電力は 300 kW とする。

4 月　33.5 MWh　5 月　41.9 MWh　6 月　50.1 MWh　7 月　63.8 MWh　8 月　66.1 MWh　9 月　47.2 MWh
10 月　46.5 MWh　11 月　53.1 MWh　12 月　72.7 MWh　1 月　63.2 MWh　2 月　48.5 MWh　3 月　46.5 MWh

【解】

従量料金は、

$(33.5+41.9+50.1+46.5+53.1+72.7+63.2+48.5)\times10^3\times16+(63.8+66.1+47.2)\times10^3\times17 = 10{,}129{,}600$ 円

である。これに基本料金である 300 kW× 1,680 円/kW×12 ヶ月=6,048,000 円を加算し、年間の電力料金は

10,129,600 + 6,048,000 = 16,177,600 円

となる。一方、年間の電力消費量は

$(33.5+41.9+50.1+46.5+53.1+72.7+63.2+48.5+63.8+66.1+47.2)\times10^3 = 633{,}100$ kWh

である。年間電力料金を年間電力消費量で除してフラットレートは

16,177,600 円 ÷ 633,100 kWh ≈ 25.6 円/kWh

となる。

20.2.3　イニシャルコスト

イニシャルコストの試算の前提として理解しておくべきことは、このような試算は現実の契約金額とかなりの幅をもって乖離しうるということである。その予測精度はこれまでの章で検討してきた各種の物理モデルが持つ予測精度に比較すると著しく低い。現実の工事請負契約はその時代の景気動向や、施主と施工者との社会的関係の影響も受ける。不景気であれば、抱えている設備や人材を遊ばせておくくらいならばということで赤字でも契約が行われることがある。逆に好景気であれば、人材が不足するためにそもそも受注しないという意思で提示されたような高額で渋々契約に至ることもある。また、将来における別工事の受注を目的に営業目的で安値で引き受けることや、以前に自社で施工した物件（元施工と呼ぶ）であるために継続性という観点から安値を提示して無理にでも受注することもある。以下に説明するイニシャルコスト算定の手順は一応の定まった形式はあるものの、実態としては上記のような個別的事情も反映される 1 つのフィクション[†2]として捉えた方が良い。従って、理性的に積み重ねた計算結果が物理モデルと同等の精度を持つと期待すると大きな失敗を招く。

工事費の構成を図 20.3 に示す。

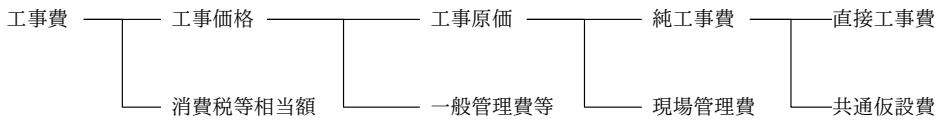


図 20.3　工事費の構成

直接工事費は、機器や材料自体の費用とこれを据え付けるために必要となる費用である。

†1　電力の場合にはガスに比べてエネルギーを貯蔵することが難しい。従って、本問のように相対的な電力需要の多寡に応じて季節や時間帯で従量料金率を変動させる契約形態が多い。

†2　悪意を持ったデマゴーグという意味では無く、内訳書の読み手側との間の不文律にもとづき、一定の世界観と妥当性を保った上での虚構である。歴史書と歴史小説の違いのように捉えれば良いと思う。

機器や材料の値段は製造者のカタログ等にも記載されているが、実際に工事費の内訳書に記載される数値との間には大きな乖離がある。内訳書の金額は施工者が製造者から調達した金額であり†1)、定価に対して掛け率を乗じた値である。俗に「半値八掛二割引†2)」という表現が行われ、定価よりも遥かに低い金額となる。2016 年現在の筆者の感覚としては冷却塔や空調機などであれば 10~30 %、熱源機器であっても 20~40 %という程度であり、そのばらつきは物件ごとに、あるいは契約の時期ごとに非常に大きい。また、製造者によっては機器のラインナップを変更するに際して定価と掛け率をすべて刷新することもあり、研究的には掛け率の絶対値を追求しても埒が明かず、不毛である。市場で現に取引がされている実勢価格に関しては、建設物価調査会が「建設物価」を、一般財団法人経済調査会が「積算資料」を毎月発行している。これらは公表資料であり、また、掛け率も反映された数値であるため、研究目的としてはこれらを引用して値段を査定すると良い。

直接工事費には上記の機器自体の値段に加えて、これらを据え付けるための労務費が含まれる。これを正確に計算するためにはそれぞれの工事に必要となる人工を積算する [20,21]必要があるが、概算であれば、機器の費用に 10~20 %程度を乗じた値を、共通仮設費も含めた金額として計上すれば良い。

現場管理費は工事現場を管理運営するために必要な費用、一般管理費は受注者の継続運営に必要な費用である。これらも積み上げにより計算することはほぼ不可能であるため、両者を合わせた金額として純工事費の 10~20 %を計上する。

【例題 20.2】

定格冷房能力 500RT (=6,330MJ/h)の直焚吸収冷温水機について、COP=1.07（一般型）と 1.35（高効率型）のいずれの機種が経済的に有利かを検討せよ。用途は事務所とし、全負荷相当運転時間は 670 時間、割引率は 7.0%とする。都市ガスのフラットレートは 100 円/Nm^3、設備寿命は 15 年とする。一般型と高効率型の実勢価格はそれぞれ 15,000 千円と 20,000 千円とする。COP は高位発熱量基準とする。エネルギーコスト以外のランニングコスト（保守費用、占有空間固定費、保険料など）は同等とし、税と減価償却費は考慮外とする。

【解】

まず、毎年のエネルギーコストの削減量を求める。全負荷相当運転時間と冷房能力を乗じることで、年間の冷却熱量は 6,330 MJ/h × 670 h/yr = 4,241 GJ/yr である。COP で割り戻すことで投入熱量を求めることができ、一般型は 4,241 ÷ 1.07 = 3,963 GJ/yr、高効率型は 4,241 ÷ 1.35 = 3,141 GJ/yr となる。従ってガス消費量は、一般型が 3,963 GJ/yr ÷ 45GJ/km^3= 88.1 kNm^3、高効率型が 3,141 ÷45 GJ/km^3 = 69.8 kNm^3 となる。これらにフラットレートを乗じ、年間のエネルギーコストは、一般型が 88.1×1,000×100=8,810 千円、高効率型が 69.8×1,000×100=6,980 千円である。一年あたりのエネルギーコスト削減量は、 8,810-6,980=1,830 千円となる。一方、熱源機のイニシャルコストは据付費用ならびに現場管理費および一般管理費をそれぞれ 15 %とすると、

一般型　：15,000 × 1.15 × 1.15 ≈ 19,800 千円

高効率型：20,000 × 1.15 × 1.15 ≈ 26,500 千円

となり、両者の差は 26,500-19,800=6,700 千円である。

高校率機を導入したと仮定し、初年度に発生するイニシャルコストの差分と 2 年目以降に発生するランニングコストの差分を式 20.1 に代入すると *NPV* は、

$$NPV = -6{,}700{,}000 + \sum_{n=1}^{15} \frac{1{,}830{,}000}{(1+0.07)^n} = 9{,}970{,}000 > 0$$

となる。0 < *NPV* であるため、高校率機を導入してランニングコストを削減した方が経済的に有利であることがわかる。別の言い方をすれば、670 万円のイニシャルコストをかけて省エネルギー機器を導入することの経済的な価値は、現在価値で約 997 万円であるということになる。

†1　実際には施工者は内訳書に記載された金額よりも更に低い金額で製造者から仕入れている可能性が高く、実態としてはその差額も施工者の主要な利益の一部となる。

†2　定価の半額に 80 %をかけ、更に 2 割まけるということであるから、結局のところ 32 %の予算で買えるということである。

20.2.4　現代ポートフォリオ理論

1）平均・分散モデル[20.17)]

割引率 Y には時間的な効果以外に投資資産がもたらすキャッシュフローのリスクが織り込まれていると説明した。当然、このリスクは投資資産の特徴を反映した固有のものである。現代ポートフォリオ理論は、それぞれのリスクの特徴を考慮した上で、いかなる資産にどの程度の投資配分を行うかという問題を解くための 1 つの方法である。

まず、具体的な省エネ投資を例に取り、リスクの考え方を定性的に解説する。今、あるテナントビルに対してインバータターボ冷凍機への熱源更新、全熱交換器の設置、太陽電池の設置、という 3 つの省エネ投資の選択肢があるとする。それぞれ投資に必要な金額は 100 万円であり、これによってもたらされる水道光熱費削減額はいずれも 5 万円/年である。期待値は等しいため、リスクを考慮しないのであればいずれの投資も等価であるが、リスクを考慮すると例えば次のような発想が浮かぶ。

不動産賃貸事業においてその収益を大きく左右する要因は建物の空室率であるため、空室率とそれぞれの省エネ投資の効果との相関を考えてみる。空室率が高ければ空調の負荷率が低下し、高ければ負荷率が上昇する。インバータターボ冷凍機は低負荷においてその真価を発揮するため、空室率が上昇して賃貸事業の収益が低下する場合には、インバータターボ冷凍機という省エネ投資は効果が大きくなり、逆に空室率が低下して賃貸事業収益が上昇する場合には効果が小さくなる。従って、不動産賃貸事業とインバータターボ冷凍機への投資はやや逆相関の関係を持つと推測できる。一方、全熱交換器（換気）はテナント不在の場合には停止させるだけであり、テナントがいなければ効果を持たない。この意味では全熱交換器という投資は不動産賃貸事業と正相関の関係を持つであろう。太陽電池は空室率に無関係に年間ではほぼ一定の発電を行うであろうから、無相関と考えて良い。もしも以上のような関係性が成立しているのであれば、現代ポートフォリオ理論を用いることで、インバータターボ冷凍機、太陽電池、全熱交換器という投資の優先順位を判断できる。

不動産賃貸事業および省エネ化投資による期待収益率（リターン）を縦軸、その分散（リスク）を横軸にとったグラフで解説する。このようなモデルを 平均・分散モデルと呼ぶ。図 20.4 左の中央に不動産賃貸事業という投資、右上方に省エネ化投資をプロットする。両投資を組み合わせた複合的な投資のリスクリターンを考えると、両者が完全に正相関（相関係数 ρ=1.0）の場合には、図 20.4 左に破線で示すように両者を単純に結んだ線上に位置することになる。逆に両者が完全に逆相関（ρ=-1.0）の場合には、図 20.4 左に二点鎖線で示すように、適当な投資配分を与えることでリスクが打ち消し合い、完全にリスクを 0 に抑えた投資が可能になる。現実には完全に正相関あるいは逆相関となる投資は存在しないため、図 20.4 左に実線で示すような曲線を描くことになる。市場参加者がリスク回避的であると仮定すれば。図 20.4 において、より左上方に位置する投資が価値が高い（リスクが少なくリターンが大きい）ことになるため、複数の逆相関に近い投資を組み合わせることで、より価値の高い投資商品を生み出すことができることがわかる。これが、上の例でインバータターボ冷凍機の投資優先順位を高めた理由である。

2）最適投資配分

上記は同一のリスク・リターン構造を持つ省エネ化投資の中から 1 つを選択する問題であったが、現実にはそれぞれの省エネ化投資でリスクとリターンが異なる。平均・分散モデルはこのような複数の投資商品に対する最適配分についても示唆を与える。図 20.4 左に示したように、2 つの投資を組み

合わせると左に凸の曲線が得られるが、これは複数の投資であっても同様であり、図 20.4 右に示すように左に凸の曲線群が得られる。傘に似ていることから、これをポートフォリオの傘と呼ぶ。選択可能な最も左上方にある曲線（図では太線表記）を有効フロンティアと呼ぶ。現実にはリスクが無いとみなせる資産（国債などの低リスク低リターン商品）である無リスク資産が存在する。無リスク資産と有効フロンティア上の投資を組み合わせれば、有効フロンティアの接線を描くことができる。市場参加者がリスク回避的であることを前提にすれば、リスクあたりのリターンが最も大きくなる点の集合はこの接線上にあるため、リスクを持つ 3 つの省エネ投資の最適投資配分は接点における配分であることがわかる†1)。この接点のポートフォリオ（投資の組み合わせ）を接点ポートフォリオと呼ぶ。

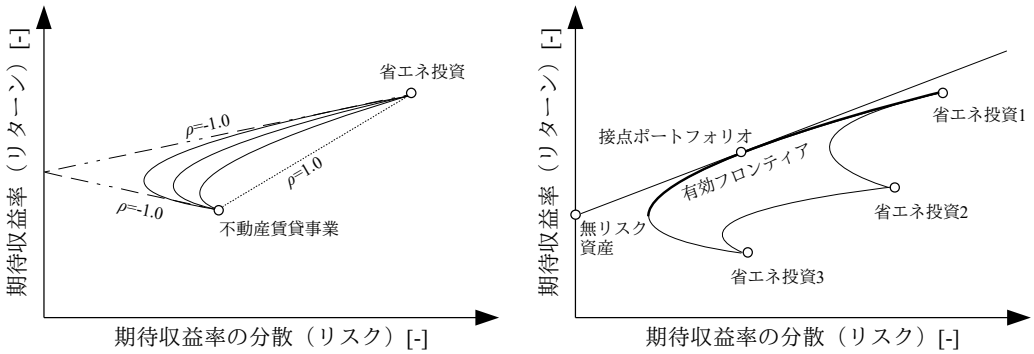


図 20.4　省エネ化投資のポートフォリオ

3）課題

実際に平均・分散モデルを適用するにあたって問題となるのはそれぞれの省エネ化投資のリスク（分散と他の投資との間の相関係数）の定量化である。このリスクを解析的に導くことはできないため、1 つの有効な方法は設備システムのモデルを作成した上で、入力条件自体を確率的に発生させ、数万回の計算を繰り返して数値計算的にリスクを捉えるという方法である [20.15)]。このような方法をモンテカルロ法と呼ぶ。しかし、現状では空調設備分野の殆どのエネルギーシミュレーションはこのような観点と目的意識をもって開発されておらず、入力条件はスケジュールにそって固定的に与えられる†2)。また、入力条件を確率的に発生させるために必要となる基礎的なデータ収集も十分だとは言えない。設備の省エネルギー化投資を他の一般金融商品と共通の軸で評価するための基礎を与える重要な技術であると筆者は考えており、既にいくつかの研究を開始しているが [20.5) 20.6)]、いまだ平均・分散モデルを適用するに十分なデータは揃えることができていない。今後の課題である。

20.2.5　省エネルギー性能と不動産価値

省エネルギー性能が高い熱源空調設備を持つ建物の不動産価値は相対的に高く、そうではない建物の不動産価値は相対的に低い。本節では、不動産価値という観点から建物の省エネルギー性能を定量評価するための方法を解説する。

†1　一昔前に聞いた話であるが、ESCO 事業において新規参入業者が Hf 蛍光灯への交換だけなど、低リスクで確実に回収が可能な投資部分のみを刈り取る危険性があるという話があった。これは俗にクリームスキミング（美味しいクリームだけをかすめ取る）と呼ばれる行為であり、公共性の高い事業において投資採算性の高い対象だけに限定して新規参入が続くことにより、ユニバーサルサービスが提供されなくなるという結果をうむ。仮に ESCO 事業においてローリスク・ローリターンの投資が「クリーム」だとすれば、ハイリスク・ハイリターンな省エネ化投資を組み合わせることにより、さらに価値のある投資商品が開発できるという視点は、このようなクリームスキミングへの対応にもなりえるのではないかと筆者は考えている。

†2　平均・分散モデルによる計算を目的にしたものではないが、不確実性を考慮した計算としては、住宅設備分野においては谷本らの研究があり [20.16)]、衛生設備分野に関しては空気調和・衛生工学会で研究報告がある [20.18)]。

1）価格の 3 面性

不動産の価格は一般に、その不動産を再調達するためにはいくらかかるか（費用性）、その不動産はどの程度の価格で現に取引がなされているか（市場性）、その不動産は将来にどの程度の利益を生み出すか（収益性）、の 3 つの観点から定まり、これを不動産価格の 3 面性と呼ぶ[20.24]。また、これらの 3 つの観点から不動産価値に迫る方法として、原価方式、比較方式、収益方式が存在する†1)。

不動産の環境性能の価値を評価するにあたっても上記の 3 方式に対応した方法が考えられ、山形、吉田、Miller らは市場性の面から研究報告を行っている[20.25) 20.26) 20.27)]。一方で、物理量であるエネルギー消費量を明示的に不動産価値に反映するためには、将来における水道光熱費にもとづいて収益方式により不動産価値を評価することが有効である。そこで、以下では収益方式にもとづく不動産鑑定評価手法である収益還元法を用いて、省エネルギー性能を不動産価値に反映させる方法について解説する[20.4)]。

2）収益還元法

収益還元法は不動産の収益性に着目して不動産価値を推定する手法であり、その試算価格を収益価格と呼ぶ。収益還元法には直接還元法と DCF 法（Discounted Cash Flow 法）がある。

DCF 法は連続する複数の期間に発生する純収益 a_k [円]と復帰価格 P_R [円]を割引率 Y [-]で除して収益価格 P [円]を求める方法であり、式 20.5 で表される。復帰価格 P_R は将来の N 時点において不動産の売却価格である†2)。将来時点の純収益 a_k と復帰価格 P_R は式 20.2 における将来時点のキャッシュフローであるため、式 20.2 の割引現在価値 NPV=0 として右辺の n=0 時点の項を左辺に移行すれば式 20.5 が得られる。つまり DCF 法は、将来時点（0<n）におけるキャッシュフローの割引現在価値と現在（n=0）の不動産価値が一致するということを前提に導出された手法である。

$$P=\sum_{n=1}^{N}\frac{a_k}{(1+Y)^n}+\frac{P_R}{(1+Y)^N} \tag{20.5}$$

式 20.5 における将来時点のキャッシュフローである a_k と P_R を平均し、さらには必要に応じて N 時点で建物を除却・新築するような費用も見込み、1 期間に発生する純収益の期待値 a [円]を推定する。この純収益が永遠につづくとすればその現在価値は式 20.6 で表現することができ、この式で試算価格を求める方法を直接還元法と呼ぶ。注意すべき点は、キャッシュフローを割り戻すための率として Y ではなく R を用いる点である。既に解説したように市場参加者がリスク回避的であるとすれば、将来時点のキャッシュフローの揺らぎ（リスク）は小さい方が価値が高い。a は将来時点のキャッシュフローの期待値であり、その期間的な変動は 0 とみなされている。従って、Y に対してこの期間的な変動リスクを上乗せした率を R として新たに設けたのである。この R を還元利回りと呼ぶ。

†1 実際にはこれらの 3 方式は独立して別個に存在するのではなく、それぞれの中にも 3 つの観点が輻輳して存在する。例えば収益性に着目した収益還元法であっても、利回りの査定にあたっては市場観察により査定を行っていたり、費用性に着目した原価法であっても、土地の再調達原価の査定にあたって造成費用が得られない場合に比準価格で代替することがある。

†2 では、将来の売却価格をどのように求めるのかと言えば、通常は N 時点の純収益を N 時点の還元利回りで還元して直接還元法で求める。収益価格を求める過程で収益価格が必要になってしまっては循環論法になり計算が破綻するように思えるが、不動産鑑定評価の多くの手法にこのような循環がみられる。自然科学とは異なり、揺れ動く人間活動を相手にする社会科学であり、演繹的に唯一の解が求められるという立場ではなく、多くの人間の活動の結果として表れた均衡点を探求するという立場にたてば、このような計算体系になるのかもしれない。慣れてしまえばこれはこれで面白い。

$$P=\sum_{n=1}^{\infty}\frac{a}{(1+R)^n}=\frac{a}{R} \tag{20.6}$$

3）純収益の査定

DCF 法であれ、直接還元法であれ、各期の純収益を査定する必要がある。また、その純収益の中でエネルギー消費に依存する部分が分かれば、エネルギー消費が不動産価値に及ぼす影響の度合いを明示的に示すことができる。

表 20.1 に純収益の算定表の例を示す[20.4]。純収益は運営収益から運営費用を控除して運営純収益を求め、これに一時金の運用益と資本的支出を加減して求める。表 20.1 に記載のそれぞれの区分は不動産鑑定評価基準に明確に定義されている。また、DCF 法を適用するための必須記載事項とされているため、不動産投資信託（REIT）が公開する有価証券報告書などを読むことで市場における具体的な値を知ることができる。

表 20.1　純収益の算定表 [千円/年]

					収支
(a)	運営収益		貸室賃貸収入		255,130
(b)			共益費収入		40,821
(c)			（共益費込み貸室賃料収入）[(a)+(b)]		295,951
(d)			水道光熱費収入		30,616
(e)			駐車場収入		4,320
(f)			その他収入		0
①		(c)+(d)+(e)+(f)			330,887
			(c)(d)空室等損失		16,981
			(e)(f)空室等損失（駐車場とその他）		225
(g)		空室等損失合計			17,206
(h)		貸倒損失			0
②	運営収益 [①-(g)-(h)]				313,681
(i)	運営費用	維持管理費			26,471
(j)		水道光熱費			30,411
(k)		修繕費			14,082
(l)		プロパティーマネジメントフィー			0
(m)		テナント募集費用等			295
(n)		公租公課		土地	20,889
				建物	18,654
				償却資産	0
(o)		損害保険料			914
(p)		その他費用			0
③		運営費用 [(i)+(j)+(k)+(l)+(m)+(n)+(o)+(p)]			111,716
④	運営純収益 [②-③]				201,965
(q)		一時金の運用益・償却額			9,335
(r)		資本的支出			0
⑤	純収益 [④+(q)-(r)]				211,300

表 20.1 に記載の区分の内、設備の性能に特に関連が深い区分は、水道光熱費収入および水道光熱費である[†1)]。前者はテナントから水道光熱費として徴収する金額であり、後者はオーナーがエネルギー会社等に支払う金額である。高効率設備を導入すれば水道光熱費が低下するが、エネルギー会社に支払った額以上の水道光熱費をテナントから徴収して差分をオーナーが得ることは難しい。従って、現実には一般の建物に比較してテナントの水道光熱費が少なくて済む結果、賃貸物件としての競争力が向上し、空室等損失の低下あるいは貸室賃貸収入の向上によって純収益が向上するという構造になる。しかし計算上は単純に、減少した水道光熱費と水道光熱費収入との差によって純収益が向上するという考え方で良いだろう。

†1　設備仕様が変わるのであるから厳密には維持管理費、修繕費、設備部分の公租公課、損害保険料にも影響がある。

【例題 20.3】

表 20.1 の純収益算定表にもとづき、直接還元法により試算価格を求めよ。ただし、還元利回りは 5 %とする。また、高効率設備の導入により水道光熱費を 70 %に減少させた場合、不動産価値はどの程度向上するか計算せよ。

【解】

収益価格は式 20.6 により、純収益を還元利回りで除して

211,300 ÷ 5 % = 4,226,000 千円

である。

高校率設備を導入すると水道光熱費が 30,411 千円×(1 - 70 %)= 9,123 千円、低下するため、純収益は 211,300 + 9,123 = 220,423 千円となる。収益価格を求めると、

220,423 ÷ 5 % = 4,408,000 千円

となるため、不動産価値は 4,408 – 4,226 = 182 百万、向上することになる。

20.3 外部効果

20.3.1 外部経済性と外部不経済性

市場における自由取引は価格競争を通じて資源配分を最適化させるか否かは、経済学における最も大きな命題の 1 つである。しかし、このような価格調整が機能しないと推測される場合がいくつかあり、これらを「市場の失敗」と呼ぶ。本節で取り上げる「外部効果」は市場の失敗の典型例であり、建物のエネルギー性能の評価とも関係が深い。

図 20.5 左は外部効果が無い場合の市場の価格調整を表している。価格が高くなれば供給者はより多くの供給を行おうとするため、供給曲線は右上がりになる。逆に需要者は価格が安いほど多くの需要を表わすため、需要曲線は右下がりになる。結果として両者が均衡する c 点で財の価格と需要供給量が決定される。この時、図 20.5 左でハッチングされた面積 *abc* は、需要する価格が供給価格を下回る面積であり、取引によりこの面積に相当するだけの価値が生まれたことになる。この価値を余剰と呼び、このような分析法を余剰分析と呼ぶ。

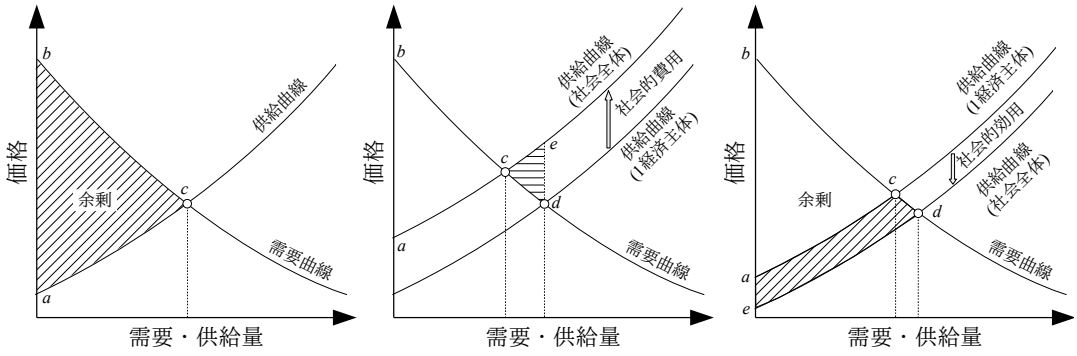


図 20.5 外部効果による余剰の減少

図 20.5 中は外部不経済がある場合の余剰分析である。例えば、現在、非常にエネルギー性能の悪いボイラを安価に販売して *d* 点で需要供給量と価格が均衡しているとする。しかしエネルギー枯渇や環境への影響を考慮すると、実際には将来において取引当事者以外の経済主体が間接的に費用を負担する可能性が高い。このような社会的費用を考慮すると、図 20.5 中に示すように社会全体の供給曲線は取引当事者にとっての供給曲線よりも上方に移動する。従って、社会全体としての余剰を最大化するためには *c* 点で均衡させて面積 *abc* の余剰を手に入れるべきであるが、取引当事者は社会的費用を直接に負担しないため、1 経済主体としての供給曲線に従って行動を行い、*d* 点で均衡する結果と

なる。この時、面積 *cde* は供給価格が需要価格を上回り、負の余剰が発生している。

図 20.5 右は外部経済がある場合の余剰分析である。上記とは逆に、エネルギー効率が高いボイラが販売され、*c* 点で均衡していたとする。1 経済主体にとっては向こう数十年の水道光熱費が削減されるという効果しかないが、環境悪化を防ぐという意味では将来世代に広く良い影響を与える投資であり、社会的な効用がある。図 20.5 右に示すように社会全体の供給曲線は取引当事者にとっての供給曲線よりも下方に移動する。もしも *d* 点で均衡させることができれば、社会全体としては面積 *ebd* の余剰を手に入れることができるため、現状、面積 *acde* に相当する余剰の逸失が生じている。

上記のような外部効果の影響を考慮した上で社会全体の余剰を最大化させるために税と補助金を用いる方法がある。図 20.5 中に示したように外部不経済が生じている場合には、社会的費用に相当するだけの税を供給者に課せば、均衡点は *d* から *c* に移る。これをピグー税と呼ぶ。図 20.5 右に示したように外部経済が生じている場合には、社会的効用に相当するだけの補助金を供給者に提供すれば、均衡点は *c* から *d* に移る。これは環境建築に対する助成金制度の 1 つの理論的根拠である。このように社会的に負担していた費用や便益を受けていた効用を、何らかの方法で取引当事者に還元することで、社会的な効用の最大化と当事者にとっての効用の最大化が同一の均衡点で実現するようにする仕組みを外部効果の内部化と呼ぶ。

20.3.2　排出権取引制度

二酸化炭素（CO_2）の排出権取引制度も外部効果の内部化の 1 つの方法である。CO_2 を尺度としてエネルギー消費に伴う環境負荷を評価し、社会的に余剰が最大化する総排出量を予め定める。この排出量を市場参加者に権利として割り当て、市場参加者間で自由に排出権を売買させる。結果として、相対的に安価に排出量削減が可能な市場参加者から順に排出の削減が行われ、最小限の投資で当初定めた総排出量を満足することになる。

建物のライフサイクル CO_2 排出量（$LCCO_2$）の過半は運用時に発生するとされている[20.23]。運用時に発生する CO_2 の値を把握するためには電気やガスの消費量を計算し、これに排出係数を乗じて CO_2 排出量に換算する必要がある。ガスに関しては組成が変わらなければ排出係数も変わらないが、電力に関してはどのような燃料を用いて電力を発生させたかによって排出係数が変化する。2016 年現在、東京ガスの都市ガス 13A の排出係数は 2.21 $kgCO_2/Nm^3$ である。また、東京電力の 2014 年の排出係数の実績値は 0.496 $kgCO_2/kWh$ である。電力の排出係数はもともと 0.37 $kgCO_2/kWh$ 程度であったが、原子力発電所の稼働停止を受けて排出係数が大きく増加した。

【例題 20.4】

例題 20.2 の熱源比較において、5 年後から排出権取引制度の実施が予想されたとする。この時の熱源機の経済性を再検討せよ。ただし、排出権の取引価格は 4,000 円/tCO_2 とする。

【解】

エネルギーコストとイニシャルコストは例題 20.2 と同じである。両熱源のガス消費量の差は 88.1 – 69.8 = 18.3 kNm3 であり、これに排出係数 2.21 kgCO2/Nm^3 と排出権取引価格 4,000 円/tCO_2 を乗じて、

$(18.3 \times 10^3) \times 2.21 \times (4{,}000 \times 10^{-3}) = 162{,}000$ 円/年

が排出権取引による利益である。これを現在価値の式 20.1 に代入すると、

$$NPV = -6{,}700{,}000 + \sum_{n=1}^{15} \frac{1{,}830{,}000}{(1+0.07)^n} + \sum_{n=5}^{15} \frac{162{,}000}{(1+0.07)^n} = 10{,}890{,}000 > 0$$

となる。排出権取引が無い場合の価値は 997 万円であるため、投資の価値が 100 万円程度向上する。

【第 20 章 記号表】

a	：純収益 [円]	r	：金利 [-]
C	：エネルギーコスト [円]	R	：還元利回り [-]
CF	：キャッシュフロー [円]	R_{cs}	：従量料金率 [円/kWh または円/Nm^3]
E_{cs}	：エネルギー使用量 [kWh または Nm^3]	R_{flt}	：フラットレート [円/kWh または円/Nm^3]
E_{pk}	：ピーク負荷 [kW または Nm^3/h]	R_{pk}	：基本料金率 [円/kW または円/(Nm^3/h)]
g	：リスク [-]	P_R	：復帰価格 [円]
NPV	：割引現在価値 [円]	Y	：割引率 [-]

【第 20 章 参考文献】

20.1） 電気供給約款, 東京電力

20.2） 一般ガス供給約款, 東京ガス

20.3） 日本エネルギー経済研究所 計量分析ユニット: EDMC エネルギー・経済統計要覧 2014

20.4） 富樫英介: 設備システムの省エネルギー化が不動産価値に与える影響の定量的評価方法に関する研究 第 1 報-直接還元法に基づく省エネルギー化投資の評価方法とケーススタディ , 空気調和・衛生工学会論文集, No.196, pp.29-37, July, 2013

20.5） 富樫英介: 設備システムの省エネルギー化が不動産価値に与える影響の定量的評価方法に関する研究 第 2 報-省エネ投資リスク評価のための確率的気象モデルの開発 , 空気調和・衛生工学会論文集, No.221, pp.21-29, Aug., 2015

20.6） 富樫英介: 建築設備システムの省エネ化投資のリスク評価を可能にする確率モデルの開発 , 平成 28 年度科学研究費補助金 若手研究(B), 課題番号 16K18198

20.7） 新・要説不動産鑑定評価基準, 公益社団法人 日本不動産鑑定士協会連合会 監修, 住宅新報社, 2010

20.8） 前川俊一: 不動産投資分析論, 明海大学不動産学部不動産学叢書, 1999

20.9） 薮下史郎: 非対称情報の経済学 スティグリッツと新しい経済学, 光文社新書, 2002

20.10） 中村達也, 八木紀一郎, 新村聡, 井上義朗: 経済学の歴史, 有斐閣アルマ, 2001

20.11） 日本不動産研究所, 不動産投資家調査 第 10~32 回

20.12） キース・カットバートソン, ダーク・ニッチェ（吉野直行 監訳）: ファイナンスの基礎理論 株式・債権・外国為替、慶応義塾大学出版会, 2013

20.13） マイケル・エアハルト（真壁昭夫, 鈴木毅彦 訳）: 資本コストの理論と実務、東洋経済新報社, 2001

20.14） 刈谷武昭: 信用リスク分析の基礎、東洋経済新報社, 1999

20.15） 刈谷武昭: 不動産収益還元価値評価モデルと賃料キャッシュフローのリスク分析法 商業用不動産リアル・オプション価値評価法, ジャレフ・ジャーナル 2003, pp.143-162, 東京経済新報社, 2003

20.16） 萩島理, 谷本潤 他: 生活スケジュールの多様性を考慮した負荷計算に基づく集合住宅の熱負荷の確率密度関数, 空気調和・衛生工学会論文集, pp.11-18, No.184, 2012

20.17） Harry Markowitz: Portfolio Selection, The Journal of Finance, Vol. 7, No.1. Mar., p.77-91, 1952

20.18） モンテカルロ・シミュレーション手法を用いた新しい給水給湯負荷算定法, 空気調和・衛生工学会, 給排水衛生設備委員会 給水給湯負荷算定法小委員会, H16.3.31

20.19） 電力空調研究会: ヒートポンプ空調システム, 第 10 章 経済計算, 1992

20.20） 日本ガス協会: 新編 都市ガス空調システム, 第 6 章 ガス空調システムの経済的評価手法, 1990

20.21） 建築コスト管理システム研究所: 国土交通省大臣官房官庁営繕部監修 公共建築工事積算基準 H19

20.22） 浅子和美, 落合勝昭, 落合由紀子: グラフィック 環境経済学, 新世社, 2015

20.23） 石福昭：大気環境の問題と対策（4） CO_2 削減対策、空気調和・衛生工学会誌, Vol 68, No.9, 1994

20.24） 新・要説不動産鑑定評価基準, 公益社団法人 日本不動産鑑定士協会連合会 監修, 住宅新報社, 2010

20.25） 山形与志樹, 村上大輔, 瀬谷創, 堤盛人, 川口有一郎 ：環境性能評価が不動産価格に与える影響の時空間波及分析, 日本不動産金融工学学会, ジャレフ・ジャーナル, No.5, 2011

20.26） 吉田二郎, 清水千弘 ：環境配慮型建築物が不動産価格に与える影響：日本の新築マンションのケース , 東京大学空間情報科学研究センター Discussion Paper, No. 106

20.27） Miller, Norm, Jay Spivey and Andy Florance: Does Green Pay Off? <http://www.costar.com/JOSRE > (accessed December 20, 2012).

20.28） Public domain data
<https://www.imf.org/external/np/adm/pictures/captions.htm>,
<http://www.library.hbs.edu/hc/collections/kress/kress_img/adam_smith2.htm>,
<https://commons.wikimedia.org/wiki/File:Karl_Marx_001.jpg> (accessed August 11, 2016).

第21章　熱源設備システムのエネルギー評価
(Evaluation of Heat Source Systems)

21.1　概要

熱源設備システムを構成する要素機器（熱源機器、冷却塔、冷却水ポンプ、冷水・温水ポンプなど）の計算法については第 19 章までに解説を行った。本章ではこれらの機器を組み合わせて熱源設備システムモデルを作成して連成計算する方法について解説する。

機器単体の計算であれば、入力条件が与えられるため、モデルが複雑でなければ反復計算を行わずに出力を求めることができることも多い。しかし、システムの計算を行う場合には、ある機器の出力が別の機器の入力となり、さらにその別の機器の出力が最初に計算を行った機器の入力となり、状態量が循環する場合がある。典型的な問題はターボ冷凍機や吸収式冷凍機の冷却水温度である。熱源機の出口冷却水は冷却塔に運ばれ、冷却塔の出口冷却水はポンプを通り再び冷凍機に戻る。このような場合には機器の間を行き来する状態量が最終的にどの値で落ち着くのかを反復計算によって求める必要が出てくる。また、本章では一次側の熱源システムをモデル化するが、第 III 編で取り扱う二次側との連成計算も見据える必要がある。具体的には、冷水と温水の温度を二次側のモデルに受け渡した後に、二次側から戻る冷水と温水の状態値を反復計算することに備える必要がある。

現実の熱源設備システムは単体の熱源装置によって構成されることは少なく、エネルギー源の分散による堅牢性の向上や、エネルギー効率の向上などを目的に、複数の熱源装置で構成されることが多い。熱源設備システムの一部を構成する、冷水や温水の製造が可能な機器群を熱源設備サブシステムと呼ぶ。熱源設備サブシステムの構成方法は多様であるため、本書のモデルではこれを表現する interface を定義することで将来の拡張性を担保する。本章では具体的な実装例として、ボイラによるサブシステム、ターボ冷凍機によるサブシステム、直焚吸収冷温水機によるサブシステム、空気熱源ヒートポンプによるサブシステム、蓄熱槽によるサブシステムを作成する。

21.2 理論

21.2.1 熱源設備システム

本章で開発する熱源設備システムモデルの例を図 21.1 に示す。冷水および温水の還条件（流量と温度）にもとづいて熱源、ポンプ、バイパス流量などの計算を行い、冷温水の供給条件を求めるモデルである[†1)]。冷水および温水ヘッダ以降の計算は第 III 編で解説する熱負荷計算および空調機モデルで取り扱うこととし、二次側空調設備システムモデルを統合した熱源空調設備システム全体の計算法については第 29 章で解説する。

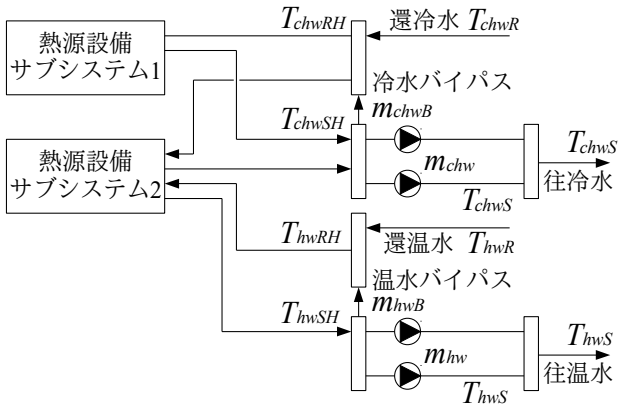


図 21.1 熱源設備システムモデル

図 21.1 に示すように、熱源設備システムには温水あるいは冷水を製造する熱源設備サブシステムが含まれている。サブシステムは、冷水のみを製造するもの、蓄熱槽を有するもの、冷却塔を用いるものなど、機器の構成は様々であるが、いずれも与えられた流量の冷水または温水を設定値まで冷却または加熱するという機能を持つ。図 21.1 の熱源設備システムの例は 2 つのサブシステムを持ち、サブシステム 1 は冷水製造のみ、サブシステム 2 は冷水に加えて温水も製造するシステムである。

1）熱源設備サブシステムの増減段

熱源設備システムは冷水および温水の供給条件を設定値通りに維持できるように熱源設備サブシステムを起動させる。このためには、二次側から要求される流量よりも全サブシステムの合計流量が大きい必要があると同時に、製造熱量も要求量を上回る必要がある。流量を基準とする負荷を負荷流量、熱量を基準とする負荷を負荷熱量と呼ぶ。負荷流量と負荷熱量を満足できない場合には、サブシステムを追加起動させる。このような処置を熱源の増段と呼ぶ。逆に、起動するサブシステムが過剰であり、運転を停止させることを熱源の減段と呼ぶ。現実のシステムでは、熱源の増減段は負荷熱量と負荷流量に加えて、熱源が安定稼働するまでの効果待ち時間を考慮しながら決定される。また、厳密には熱源サブシステムが製造可能な冷温熱量は外界条件や還温度などの条件によって変化するが、このような能力変化を制御に組み込むことは難しいため、定格能力で増減段判定が行われることが多い。計算上は負荷流量によって最低限必要な運転系統数を仮定した後、過負荷状態となる系統が無くなるまで 1 段ずつ増段を繰り返すことで必要な運転系統数を決定する方針とする。

†1 冷水や温水が機器から出るのか入るのかを区別するために「往」あるいは「還」という言葉を用いて形容することが多い。厳密な表現のためには「何から見て」往なのか還なのかを知る必要があるが、文脈の中で暗黙に主体を捉えるしかないことも多く、結局のところ出入口のいずれを指しているのかがわからないことが少なくない。例えば、冷却塔の冷却水出口温度は冷却塔からみれば往温度であり、冷凍機からみれば還温度なのである。設備システム全体の中で熱源を中心に捉えて、熱源から遠ざかる方向を「往」と表現することもある。表現が一貫するため、本書では、以降、この見方で統一することにする。例えば冷却塔の出口冷却水温度は冷却水還温度であり、2 次側への温水供給温度は温水往温度となる。

2）冷温水のバイパスと二次側往還温度差

現実の熱源は冷水や温水の流量を無限小に絞ることはできず、下限流量が定められている。従って、二次側の負荷流量が小さい場合に、一次側（サブシステム側）の循環水量をこれよりも大きく維持しなければならないことがある。この場合には余剰の流量は二次側まで送らず、往還ヘッダ間でバイパスさせる。還ヘッダでは往冷温水が流れこむことになるため、冷水であれば温度が下がり、温水であれば温度が上がる。冷水バイパス流量を m_{chwB} [kg/s]とし、二次側循環流量を m_{chw} [kg/s]とすれば、還ヘッダでの冷水温度は式 21.1 で計算できる。ただし、T_{chwR} [K]は二次側からの冷水還温度、T_{chwSH} [K]は往ヘッダでの冷水温度である。同様に、温水の場合には式 21.2 が成立する。

ヘッダ間バイパスが生じるとき、2 ポンプ方式の場合には二次側循環流量を下げることができるため、水搬送に必要なエネルギーを削減することができる。1 ポンプ方式の場合には 1 次側の下限流量未満には水量を下げられないという点に注意する。

$$T_{chwRH}=\frac{m_{chw}T_{chwR}+m_{chwB}T_{chwSH}}{m_{chw}+m_{chwB}} \tag{21.1}$$

$$T_{hwRH}=\frac{m_{hw}T_{hwR}+m_{hwB}T_{hwSH}}{m_{hw}+m_{hwB}} \tag{21.2}$$

21.2.2　熱源設備サブシステム

熱源設備サブシステムの構成は様々にある。以下に典型的な構成例を示す。

1）ボイラ

ボイラによるサブシステムは温水製造のみを行うサブシステムであり、ボイラと温水ポンプで構成される。図 21.2 にシステムの構成を示す。ボイラの入口温水温度 T_{hwRp} [K]は還温度 T_{hwR} [K]にポンプによる昇温の影響を加えて式 21.3 で計算する。E_{phw} [kW]はポンプの消費電力、c_{pw} は水の定圧比熱(=4.186 kJ/(kg·K))、m_{hw} [kg/s]は温水の質量流量である。現実には消費電力の一部のみが温水に与えられるが、ここではモデルを簡単にするためにすべてが熱に転化するとしている。

$$T_{hwRp}=T_{hwR}+\frac{E_{phw}}{c_{pw}m_{hw}} \tag{21.3}$$

2）ターボ冷凍機

ターボ冷凍機によるサブシステムは冷水製造のみを行うシステムであり、ターボ冷凍機と冷水ポンプに加え、冷却塔と冷却水ポンプで構成される。図 21.3 にシステムの構成を示す。ボイラによるサブシステムと同様、冷水ポンプと冷却水ポンプによる昇温の影響を式 21.4 と式 21.5 で計算する。

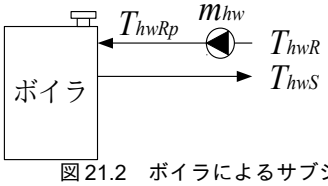


図 21.2　ボイラによるサブシステム

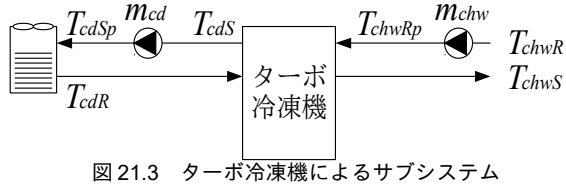


図 21.3　ターボ冷凍機によるサブシステム

$$T_{chwRp}=T_{chwR}+\frac{E_{pchw}}{c_{pw}m_{chw}} \tag{21.4}$$

$$T_{cdSp}=T_{cdS}+\frac{E_{pcd}}{c_{pw}m_{cd}} \tag{21.5}$$

第 12 章で解説したように、冷却塔の冷却水出口温度は制御する場合と制御しないで成り行きとする場合があるが、通常はターボ冷凍機の効率を最大化させるために、最大風量で成り行き運転とする。冷却塔が過負荷となり冷却水温度が設定値に維持できない場合や、成り行き運転とする場合には、冷却水出口温度 T_{cdR} [K]が変動するため、ターボ冷凍機モデル、冷却水ポンプモデル、冷却塔モデルを連成させて収束計算を行う必要がある。図 21.4 に冷却水還温度と冷凍機および冷却塔能力の関係を示す。冷却水還温度が低い場合には冷凍機の能力が上昇するため、冷凍機の廃熱も増加する。一方、外気湿球温度との温度差は縮小するため、冷却塔の能力は低下する。最終的に外気湿球温度と冷却水還温度が一致する点で冷却塔能力は 0 になる。従って、図 21.4 に示すように冷凍機と冷却塔の能力は冷却水還温度に対して逆の傾向を示し、適当な還温度において両者の能力は一致する。

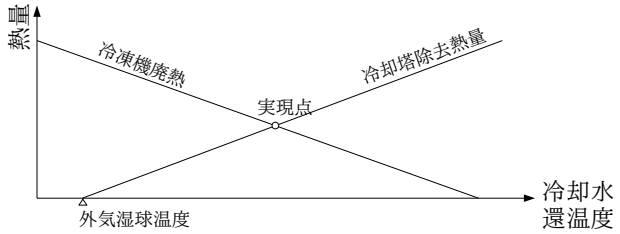


図 21.4　冷却水還温度と冷凍機および冷却塔能力の関係

3）空気熱源ヒートポンプ

空気熱源ヒートポンプによるサブシステムは冷水と温水の両方を製造するサブシステムであり、空気熱源ヒートポンプと冷水ポンプ、温水ポンプによって構成される。冷水ポンプと温水ポンプを兼用させて冷温水ポンプとする場合も多い。図 21.5 にシステムの構成を示す。ポンプの昇温の計算に関してはボイラおよびターボ冷凍機のシステムと同じである。

4）直焚吸収冷温水機

直焚吸収冷温水機によるサブシステムは冷水と温水の両方を製造するサブシステムであり、直焚吸収冷温水機、冷水ポンプ、温水ポンプ、冷却塔、冷却水ポンプによって構成される。図 21.6 にシステムの構成を示す。温水製造時には冷却塔と冷却水ポンプは停止させる。ターボ冷凍機によるシステムと同様に冷却水出口温度が変動する場合には収束計算が必要となる。

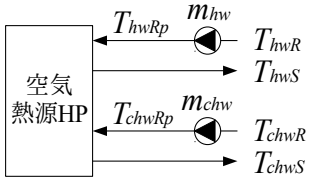


図 21.5　空気熱源 HP によるサブシステム

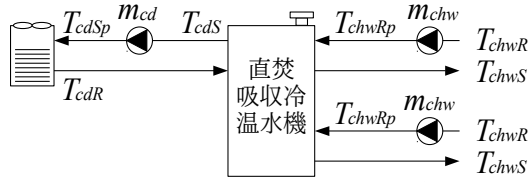


図 21.6　直焚吸収冷温水機によるサブシステム

5）蓄熱システム

蓄熱槽を用いたサブシステムは、蓄熱槽に蓄えた熱を放熱することで熱供給を行うシステムである。図 21.7 にシステムの構成を示す。図 21.7 はターボ冷凍機のみを用いた冷専のシステムであるが、空気熱源ヒートポンプなどを用いることで温水蓄熱にも対応させたシステムもある。蓄熱槽は開放系統であるため、水搬送動力を抑制するために熱交換器を介して放熱を行う。従ってシステムはターボ冷凍機、冷却塔、冷却水ポンプ、蓄熱槽、放熱ポンプ、蓄熱ポンプ、熱交換器によって構成される。

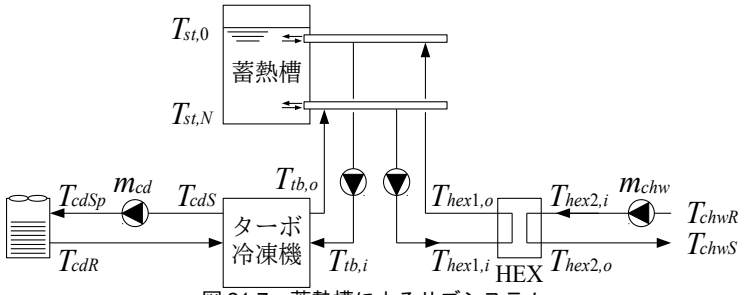


図 21.7　蓄熱槽によるサブシステム

21.2.3　冷温水負荷の設定

熱源システムのエネルギー消費量を計算するためには、入力条件である冷温水負荷が必要である。第 III 編で解説するように建物の熱負荷計算と空調機の計算を行えば、検討対象としたい建物に即応する冷温水負荷を得ることはできる。しかし設計の初期段階などにおいては建築の仕様は未確定であり、このような段階で積み上げる方法で冷温水負荷を求めても、ほぼ確実に手戻りが発生することになるであろうし、また、設計に費やせる時間的にも困難であろう。以下に、建物熱負荷計算を必要としない、幾つかの簡略計算法を示す。

1）全負荷相当運転時間法

全負荷相当運転時間法は最も簡単な期間エネルギー計算法である。それぞれの設備機器の定格条件におけるエネルギー消費量に対して全負荷相当運転時間を乗じることで期間的なエネルギー消費量を計算する。例えば 100 kW の熱源機器で全負荷相当運転時間が 800 h であれば、エネルギー消費量は 100×800=80,000 kWh となる。従って、そもそも設備システムのモデルすら不要な手法であるため[†1]、本書ではこれ以上は触れない。手法の詳細については参考文献を参照されたい[21.1]。

2）熱負荷原単位の利用

過去に観測された統計値を用いて、用途や規模などから冷温水負荷を推定する手法である。用途と床面積から年間の冷温水負荷を求め、これを各月に割り振り、さらに各時刻に割り振ることで、各月の平均日における時系列データを作成する。このような熱負荷に関する統計値を熱負荷原単位と呼び、多くの研究報告がある[21.2）21.3）21.4）21.5）21.6）21.7）]。

【例題 21.1】

事務所の熱負荷原単位として以下の数値が得られたとする。これらの数値を用いて 10,000 m² の事務所建物の各月平均日の時系列冷温水負荷を作成せよ。

年間冷房負荷原単位：255 MJ/m²　　年間暖房負荷原単位：126 MJ/m²

表 21.1　月別の冷暖房負荷比（事務所）[%]

	1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月
冷房負荷比	-	-	-	-	3.92	15.67	27.63	30.72	19.79	2.27	-	-
暖房負荷比	25.93	22.79	17.66	4.27	-	-	-	-	-	-	7.98	21.37

表 21.2　時刻別の冷暖房負荷比（事務所）[%]

	6:00	7:00	8:00	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00
冷房	-	1.79	9.57	9.17	8.97	9.27	9.37	9.27	8.97	10.69	8.97	9.27	3.89	0.40	0.40
暖房	0.30	16.99	12.29	8.09	10.29	10.49	10.29	8.39	8.19	9.09	5.59	-	-	-	-

†1　ただし、全負荷相当運転時間は BECS/CEC/AC プログラムを用いて数多くの感度解析を行ってまとめられた数値であるから、その導出過程においては熱負荷および設備システムのシミュレーションが活用されている。

【解】

年間の冷暖房負荷は、255×10,000 = 2,550,000 MJ/yr と 126×10,000 = 1,260,000 MJ/yr であり、表 21.1 の比率を用いてこの値を各月に割り振ると表 21.3 が得られる。ただし、1 日あたりの負荷は各月の営業日数を 20 日とした場合の値である。

表 21.3　月別の冷暖房負荷（事務所）

		1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月
冷房	MJ/月	0	0	0	0	99,960	399,585	704,565	783,360	504,645	57,885	0	0
	MJ/日	0	0	0	0	4,998	19,979	35,228	39,168	25,232	2,894	0	0
暖房	MJ/月	326,718	287,154	222,516	53,802	0	0	0	0	0	0	100,548	269,262
	MJ/日	16,336	14,358	11,126	2,690	0	0	0	0	0	0	5,027	13,463

表 21.2 を用いて表 21.3 の平均日日積算負荷を各時刻に割り振ると表 21.4 が得られる。

表 21.4　月別の代表日時系列冷暖房負荷（事務所）[MJ/h]

	1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月
	暖房				冷房						暖房	
0:00	0	0	0	0	0	0	0	0	0	0	0	0
1:00	0	0	0	0	0	0	0	0	0	0	0	0
2:00	0	0	0	0	0	0	0	0	0	0	0	0
3:00	0	0	0	0	0	0	0	0	0	0	0	0
4:00	0	0	0	0	0	0	0	0	0	0	0	0
5:00	0	0	0	0	0	0	0	0	0	0	0	0
6:00	49	43	33	8	0	0	0	0	0	0	15	40
7:00	2,775	2,439	1,890	457	89	358	631	701	452	52	854	2,287
8:00	2,008	1,765	1,367	331	478	1,912	3,371	3,748	2,415	277	618	1,655
9:00	1,322	1,162	900	218	458	1,832	3,230	3,592	2,314	265	407	1,089
10:00	1,681	1,477	1,145	277	448	1,792	3,160	3,513	2,263	260	517	1,385
11:00	1,714	1,506	1,167	282	463	1,852	3,266	3,631	2,339	268	527	1,412
12:00	1,681	1,477	1,145	277	468	1,872	3,301	3,670	2,364	271	517	1,385
13:00	1,371	1,205	933	226	463	1,852	3,266	3,631	2,339	268	422	1,130
14:00	1,338	1,176	911	220	448	1,792	3,160	3,513	2,263	260	412	1,103
15:00	1,485	1,305	1,011	245	534	2,136	3,766	4,187	2,697	309	457	1,224
16:00	913	803	622	150	448	1,792	3,160	3,513	2,263	260	281	753
17:00	0	0	0	0	463	1,852	3,266	3,631	2,339	268	0	0
18:00	0	0	0	0	194	777	1,370	1,524	982	113	0	0
19:00	0	0	0	0	20	80	141	157	101	12	0	0
20:00	0	0	0	0	20	80	141	157	101	12	0	0
21:00	0	0	0	0	0	0	0	0	0	0	0	0
22:00	0	0	0	0	0	0	0	0	0	0	0	0
23:00	0	0	0	0	0	0	0	0	0	0	0	0
日積算	16,336	14,358	11,126	2,690	4,998	19,979	35,228	39,168	25,232	2,894	5,027	13,463

3）熱負荷モードの利用

本手法は「LCEM のための空調システムシミュレーションツール」において採用されている手法である。予め代表的な地域について熱負荷計算を実行した上で、冷温水負荷を降順で並べて幾つかのグループに分け、グループごとの平均的な負荷率と外気温湿度条件を用意する。これらの各グループについてエネルギーシミューレーションを実行して積和をとることで年間のエネルギー消費量を予測する方法である。表 21.5 に東京都の事務所の熱負荷モードの例を示す[21.8)]。各モードはそれぞれ 100 時間の平均値であるため、8 ないしは 9 の運転モードについてエネルギーシミュレーションを実行した後、100 時間を乗じて合算することで年間のエネルギー消費量が得られる。

表 21.5　東京都事務所の熱負荷モード（冷暖房各々 4 ヶ月パターン）

		モード 1	モード 2	モード 3	モード 4	モード 5	モード 6	モード 7	モード 8	モード 9
冷房	乾球温度 [℃]	30	29.2	28.6	27.5	26.3	24.7	22.9	21.2	-
	湿球温度 [℃]	24	23.7	23.2	22.6	21.5	20.2	19	18	-
	負荷率 [%]	76%	67%	60%	53%	42%	30%	19%	7%	-
暖房	乾球温度 [℃]	4.4	6.4	7.5	8.5	8.3	9.5	10.6	12.6	15.5
	湿球温度 [℃]	1.2	2.5	2.8	3.3	3.5	4.4	5.4	7.3	9.1
	負荷率 [%]	68%	46%	39%	34%	31%	27%	23%	16%	8%

21.3　計算法

熱源設備サブシステムの種類は無数にあるため、特定の構成のみに特化したモデルを開発してプログラムの汎用性を失うことは好ましくない。そこで、本書のモデルではまず、熱源設備サブシステム一般を表わす interface を定義し、熱源設備システムモデルではこの interface を用いて計算を行う仕様とする。個別に開発する具体的なサブシステムにはこの interface を実装させる。

21.3.1　熱源設備サブシステム interface の作成

熱源設備サブシステムの interface の定義をプログラム 21.1 に示す。

5~9 行は運転モード関連のプロパティであり、後述する熱源設備システムクラスの列挙型を用いて設定する。「温水製造のみ」「冷水製造のみ」「両方」「停止」を選択可能とする。

12 行は外気条件である。空気熱源ヒートポンプや冷却塔を用いるシステムなどで必要となる情報である。乾球温度と絶対湿度ではなく、湿球温度を必要とする機器もあるため、湿り空気に関する情報を一括して湿り空気クラスのインスタンス（第 4 章）を用いて設定する。

第 III 編では建物熱負荷および二次側空調システムについて解説するが、最終的に建物全体で問題を解く際には、本章の熱源システムと二次側空調システムとを連成計算する必要があり、この場合には反復収束計算が発生する。サブシステムに含まれる機器が全て熱容量を持たない機器であるならば、何も考えずに反復計算を繰り返せば良いが、蓄熱槽のように熱容量を持って状態が変化する場合には注意が必要である。反復計算の過程を時間の経過と混同し、水槽温度を変化させてはならないためである。そこで、連成のための反復計算の場合には 23 行の ForecastSupplyWaterTemperature メソッドを用いて冷温水往温度を予測し、反復が完了した場合のみ、26 行の FixState メソッドを用いて状態を確定するという方法をとる。

サブシステムが過負荷の場合には増段処理が必要となる。32 行と 35 行はこの判定を行うためのプロパティである。

37~71 行は冷水および温水の温度と流量に関する情報を設定するためのプロパティである。

プログラム 21.1　熱源設備サブシステム interface

Popolo.HVAC.SubSystem.IHeatSourceSubSystem interface

```
 1 /// <summary>熱源設備サブシステムインターフェース</summary>
 2 public interface IHeatSourceSubSystem
 3 {
 4
 5   /// <summary>運転可能モードを取得する</summary>
 6   HeatSourceSystemModel.OperatingMode SelectableMode { get; }
 7
 8   /// <summary>運転モードを設定・取得する</summary>
 9   HeatSourceSystemModel.OperatingMode Mode { get; set; }
10
11   /// <summary>外気条件を設定・取得する</summary>
12   ImmutableMoistAir OutdoorAir { get; set; }
13
14 /// <summary>現在の日時を設定・取得する</summary>
15   DateTime CurrentDateTime { get; set; }
16
17   /// <summary>タイムステップを設定・取得する</summary>
18   double TimeStep { get; set; }
19
20   /// <summary>冷温水往温度を予測する</summary>
21   /// <param name="chilledWaterFlowRate">冷水流量[kg/s]</param>
22   /// <param name="hotWaterFlowRate">温水流量[kg/s]</param>
23   void ForecastSupplyWaterTemperature(double chilledWaterFlowRate, double hotWaterFlowRate);
24
25   /// <summary>状態を確定する</summary>
26   void FixState();
```

```
27
28   /// <summary>熱源サブシステムを停止させる</summary>
29   void ShutOff();
30
31   /// <summary>冷水供給が過負荷か否か</summary>
32   bool IsOverLoad_C { get; }
33
34   /// <summary>温水供給が過負荷か否か</summary>
35   bool IsOverLoad_H { get; }
36
37   /// <summary>温水還温度[C]を設定・取得する</summary>
38   double HotWaterReturnTemperature { get; set; }
39
40   /// <summary>温水往温度設定値[C]を設定・取得する</summary>
41   double HotWaterSupplyTemperatureSetpoint { get; set; }
42
43   /// <summary>温水往温度[C]を取得する</summary>
44   double HotWaterSupplyTemperature { get; }
45
46   /// <summary>温水流量[kg/s]を取得する</summary>
47   double HotWaterFlowRate { get; }
48
49   /// <summary>温水上限流量[kg/s]を取得する</summary>
50   double MaxHotWaterFlowRate { get; }
51
52   /// <summary>温水下限流量比[-]を取得する</summary>
53   double MinHotWaterFlowRatio { get; }
54
55   /// <summary>冷水還温度[C]を設定・取得する</summary>
56   double ChilledWaterReturnTemperature { get; set; }
57
58   /// <summary>冷水往温度設定値[C]を設定・取得する</summary>
59   double ChilledWaterSupplyTemperatureSetpoint { get; set; }
60
61   /// <summary>冷水往温度[C]を取得する</summary>
62   double ChilledWaterSupplyTemperature { get; }
63
64   /// <summary>冷水流量[kg/s]を設定する</summary>
65   double ChilledWaterFlowRate { get; }
66
67   /// <summary>冷水上限流量[kg/s]を取得する</summary>
68   double MaxChilledWaterFlowRate { get; }
69
70   /// <summary>冷水下限流量比[-]を取得する</summary>
71   double MinChilledWaterFlowRatio { get; }
72
73 }
```

21.3.2　熱源設備サブシステムクラスの作成

1）温水ボイラによる熱源設備サブシステム

温水ボイラによる熱源設備サブシステムクラスをプログラム 21.2 に示す。

5~24 行は本サブシステム固有の変数である。複数台の温水ボイラ設置に対応するため、運転台数に関する情報を保有する。

26~130 行は IHeatSourceSubSystem interface のプロパティの実装であり、95~127 行が主たる処理である温水往温度予測である。108, 109 行で負荷流量から最低限必要な運転台数を求める。113~123 行で増段を繰り返し、過負荷にならないように台数を決定する。ただし 121 行に示すように、最大台数に到達した場合には過負荷であっても増段を打ち切る。この場合には 126 行に示すように、温水往温度が設定値を満足できず、成り行きの値となる。本モデルは熱容量を持たず静的な処理のみを行うため、FixState メソッドに処理は無い（131 行）。

プログラム 21.2　温水ボイラによる熱源設備サブシステムクラス

Popolo.HVAC.SubSystem.HotWaterBoilerSystem class

```
1 /// <summary>温水ボイラによる熱源サブシステム</summary>
2 public class HotWaterBoilerSystem: IHeatSourceSubSystem
3 {
4
5   /// <summary>水の定圧比熱[kJ/(kgK)]</summary>
6   private const double WATER_SPECIFIC_HEAT = 4.186;
```

```
 7
 8  /// <summary>温水ボイラ</summary>
 9  private HotWaterBoiler boiler;
10
11  /// <summary>温水ポンプ</summary>
12  private CentrifugalPump hwPump;
13
14  /// <summary>温水ボイラを取得する</summary>
15  public ImmutableHotWaterBoiler Boiler { get { return boiler; } }
16
17  /// <summary>温水ポンプを取得する</summary>
18  public ImmutableCentrifugalPump HotWaterPump { get { return hwPump; } }
19
20  /// <summary>ボイラの台数を取得する</summary>
21  public int BoilerNumber { get; private set; }
22
23  /// <summary>運転台数を取得する</summary>
24  public int OperatingNumber { get; private set; }
25
26  /// <summary>冷水供給が過負荷か否か</summary>
27  public bool IsOverLoad_C { get { return false; } }
28
29  /// <summary>温水供給が過負荷か否か</summary>
30  public bool IsOverLoad_H { get; private set; }
31
32  /// <summary>運転可能モードを取得する</summary>
33  public HeatSourceSystemModel.OperatingMode SelectableMode
34  { get { return HeatSourceSystemModel.OperatingMode.Heating; } }
35
36  /// <summary>運転モードを設定・取得する</summary>
37  public HeatSourceSystemModel.OperatingMode Mode { get; set; }
38
39  /// <summary>現在の日時を設定・取得する</summary>
40  public DateTime CurrentDateTime { get; set; }
41
42  /// <summary>タイムステップを設定・取得する</summary>
43  public double TimeStep { get; set; }
44
45  /// <summary>温水還温度[C]を設定・取得する</summary>
46  public double HotWaterReturnTemperature { get; set; } = 40;
47
48  /// <summary>温水往温度設定値[C]を設定・取得する</summary>
49  public double HotWaterSupplyTemperatureSetpoint { get; set; } = 45;
50
51  /// <summary>温水往温度[C]を取得する</summary>
52  public double HotWaterSupplyTemperature { get; private set; } = 45;
53
54  /// <summary>温水流量[kg/s]を設定・取得する</summary>
55  public double HotWaterFlowRate { get; set; }
56
57  /// <summary>温水上限流量[kg/s]を取得する</summary>
58  public double MaxHotWaterFlowRate
59  { get { return boiler.MaxWaterFlowRate * BoilerNumber; } }
60
61  /// <summary>温水下限流量比[-]を取得する</summary>
62  public double MinHotWaterFlowRatio
63  { get { return boiler.MinWaterFlowRatio / BoilerNumber; } }
64
65  /// <summary>冷水還温度[C]を設定・取得する</summary>
66  public double ChilledWaterReturnTemperature { get; set; }
67
68  /// <summary>冷水往温度設定値[C]を設定・取得する</summary>
69  public double ChilledWaterSupplyTemperatureSetpoint { get; set; }
70
71  /// <summary>冷水往温度[C]を取得する</summary>
72  public double ChilledWaterSupplyTemperature { get; private set; }
73
74  /// <summary>冷水流量[kg/s]を設定・取得する</summary>
75  public double ChilledWaterFlowRate { get; set; }
76
77  /// <summary>冷水上限流量[kg/s]を取得する</summary>
78  public double MaxChilledWaterFlowRate { get { return 0; } }
79
80  /// <summary>冷水下限流量比[-]を取得する</summary>
81  public double MinChilledWaterFlowRatio { get { return 0; } }
82
83  /// <summary>外気条件を設定・取得する</summary>
84  public ImmutableMoistAir OutdoorAir { get; set; }
85
86  /// <summary>熱源サブシステムを停止させる</summary>
```

```
 87   public void ShutOff()
 88   {
 89     IsOverLoad_H = false;
 90     BoilerNumber = 0;
 91     boiler.ShutOff();
 92     hwPump.ShutOff();
 93   }
 94
 95   /// <summary>冷温水往温度を予測する</summary>
 96   /// <param name="chilledWaterFlowRate">冷水流量[kg/s]</param>
 97   /// <param name="hotWaterFlowRate">温水流量[kg/s]</param>
 98   public void ForecastSupplyWaterTemperature(double chilledWaterFlowRate, double hotWaterFlowRate)
 99   {
100     ChilledWaterFlowRate = chilledWaterFlowRate;
101
102     if (Mode != HeatSourceSystemModel.OperatingMode.Heating || hotWaterFlowRate == 0)
103     {
104       ShutOff();
105       return;
106     }
107
108     OperatingNumber = (int)Math.Ceiling(hotWaterFlowRate / boiler.MaxWaterFlowRate);
109     OperatingNumber = Math.Min(BoilerNumber, OperatingNumber);
110     boiler.AmbientTemperature = OutdoorAir.DrybulbTemperature;
111     boiler.OutletWaterSetPointTemperature = HotWaterSupplyTemperatureSetpoint;
112
113     while (true)
114     {
115       //ポンプによる昇温を評価
116       double hwFlow = hotWaterFlowRate / OperatingNumber;
117       hwPump.UpdateState(0.001 * hwFlow);
118       double twi = HotWaterReturnTemperature
119         + hwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * hwFlow);
120       boiler.Update(twi, hwFlow);
121       if (OperatingNumber == BoilerNumber || !boiler.IsOverLoad) break;
122       else OperatingNumber++;
123     }
124
125     IsOverLoad_H = boiler.IsOverLoad;
126     if (IsOverLoad_H) HotWaterSupplyTemperature = boiler.OutletWaterTemperature;
127     else HotWaterSupplyTemperature = HotWaterSupplyTemperatureSetpoint;
128   }
129
130   /// <summary>状態を確定する</summary>
131   public void FixState() { }
132
133   /// <summary>インスタンスを初期化する</summary>
134   /// <param name="boiler">温水ボイラ</param>
135   /// <param name="hwPump">温水ポンプ</param>
136   /// <param name="number">ボイラ台数</param>
137   public HotWaterBoilerSystem(HotWaterBoiler boiler, CentrifugalPump hwPump, int number)
138   {
139     this.boiler = boiler;
140     this.hwPump = hwPump;
141     BoilerNumber = number;
142     Mode = HeatSourceSystemModel.OperatingMode.ShutOff;
143   }
144 }
```

2）ターボ冷凍機による熱源設備サブシステム

ターボ冷凍機による熱源設備サブシステムクラスをプログラム 21.3 に示す。

8~39 行は本サブシステム固有のプロパティである。ボイラによるサブシステムと同様に複数の熱源機の設置に対応させているが、注意すべきは冷却塔とターボ冷凍機との対応関係である。既に第 14 章で解説したように、一般に、冷却水温度を低下させると圧縮冷凍サイクルの効率は向上する。この特徴を利用して、複数のターボ冷凍機系統がある場合に各系統の冷却塔を集合させるシステムがある。低負荷時にターボ冷凍機よりも多い冷却塔を稼働させることで冷却水温度を下げて効率を向上させることが狙いである。これを集合式冷却塔あるいは集合型冷却塔などと呼ぶ。図 21.8 に一般の冷却塔と集合式冷却塔の違いを示す。図 21.8 左は通常の構成であり、1 系統のターボ冷凍機に 1 系統の冷却塔（本図では 2 セル）が接続されている。図 21.8 右が集合式冷却塔であり、3 系統の冷却塔を

1 つにまとめている。従って、低負荷時に 1 台のターボ冷凍機が稼働している場合にも、3 系統（6 セル）の冷却塔をまとめて運転させ、低い冷却水温度を保つことができる。市川らの調査によれば 35 %程度の熱源システムが集合式冷却塔を採用している[21.10]。33 行はターボ冷凍機 1 台あたりのセル数であり、冷却塔を集合させて運転するか否かは 39 行のプロパティで設定する。

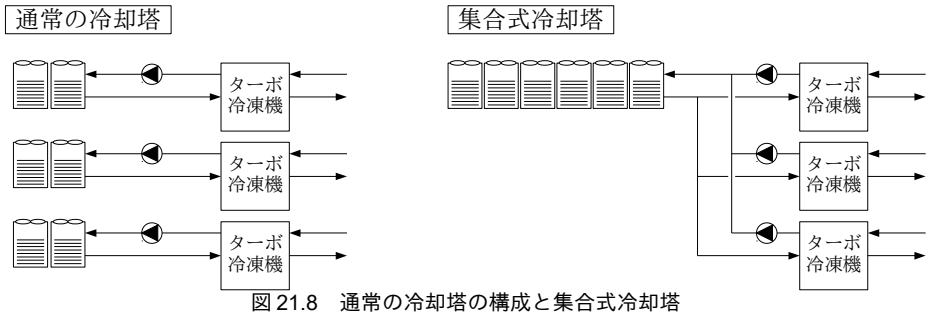


図 21.8　通常の冷却塔の構成と集合式冷却塔

冷却水に関しては冷却塔出口温度を制御するか否かに加え、流量を制御するか否かという判断がある。冷却水ポンプの搬送動力の削減を狙うのであれば冷凍機の負荷率に比例的に冷却水を減少させる。一方で、冷却水流量が大きければ凝縮温度を下げることができ、ターボ冷凍機の効率は向上する。冷却水流量を負荷比例とするか定格値で固定とするかは 45 行で設定する。

129~198 行が冷水往温度予測をする主たるメソッドである。134~148 行はボイラによるサブシステムと同様であり、定格流量と負荷流量から最低限必要なターボ冷凍機の台数を求めた後、149~193 行で 1 台ずつ増段する。151~159 行は先に記した冷却塔および冷却水の制御に関する処理であり、1 台のターボ冷凍機あたりでどれだけの冷却水が流れるのかを計算する。168~176 行は冷却水還温度を制御する場合であり、この場合には基本的には収束計算は不要である。冷却塔が過負荷となる場合、あるいはそもそも冷却水還温度を制御しない場合には 178~189 行で冷却水還温度を収束計算する。本モデルでは 10~32 ºC の範囲で二分法を用いて求根する。

204~223 行はコンストラクタである。第 14 章で開発したターボ冷凍機のインターフェースを用いることで、物理式による詳細モデルと特性式による簡易モデルの両方に対応させる。

プログラム 21.3　ターボ冷凍機による熱源設備サブシステムクラス

Popolo.HVAC.SubSystem.CentrifugalInverterChillerSystem class

```
 1 /// <summary>ターボ冷凍機による熱源サブシステム</summary>
 2 public class CentrifugalChillerSystem: IHeatSourceSubSystem
 3 {
 4
 5   /// <summary>水の定圧比熱[kJ/(kgK)]</summary>
 6   private const double WATER_SPECIFIC_HEAT = 4.186;
 7
 8   /// <summary>ターボ冷凍機</summary>
 9   private ICentrifugalChiller chiller;
10
11   /// <summary>ポンプ</summary>
12   private CentrifugalPump cdwPump, chwPump;
13
14   /// <summary>冷却塔</summary>
15   private CoolingTower cTower;
16
17   /// <summary>ターボ冷凍機を取得する</summary>
18   public ImmutableCentrifugalChiller Chiller { get { return chiller; } }
19
20   /// <summary>冷水ポンプを取得する</summary>
21   public ImmutableCentrifugalPump ChilledWaterPump { get { return chwPump; } }
22
23   /// <summary>冷却水ポンプを取得する</summary>
24   public ImmutableCentrifugalPump CoolingWaterPump { get { return cdwPump; } }
25
```

```
26  /// <summary>冷却塔を取得する</summary>
27  public ImmutableCoolingTower CoolingTower { get { return cTower; } }
28
29  /// <summary>冷凍機の台数を取得する</summary>
30  public int ChillerNumber { get; private set; }
31
32  /// <summary>冷却塔のセル数（1台の冷凍機あたり）を取得する</summary>
33  public int CoolingTowerNumber { get; private set; }
34
35  /// <summary>運転台数を取得する</summary>
36  public int OperatingChillerNumber { get; private set; }
37
38  /// <summary>冷却塔を1対1で稼働させるか否か</summary>
39  public bool OperateCoolingTowerOneOnOne { get; set; } = true;
40
41  /// <summary>冷却水温度を制御するか否か</summary>
42  public bool ControlCoolingWaterTemperature { get; set; }
43
44  /// <summary>冷却水流量を制御するか否か（負荷率比例）</summary>
45  public bool ControlCoolingWaterFlowRate { get; set; }
46
47  /// <summary>最小冷却水流量比[-]を設定・取得する</summary>
48  public double MinimumCoolingWaterFlowRatio { get; set; } = 0.5;
49
50  /// <summary>冷却水温度設定値[C]を設定・取得する</summary>
51  public double CoolingWaterTemperatureSetpoint { get; set; } = 32;
52
53  /// <summary>冷却水流量[kg/s]を設定・取得する</summary>
54  public double CoolingWaterFlowSetpoint { get; set; }
55
56  /// <summary>冷水供給が過負荷か否か</summary>
57  public bool IsOverLoad_C { get; private set; }
58
59  /// <summary>温水供給が過負荷か否か</summary>
60  public bool IsOverLoad_H { get; private set; }
61
62  /// <summary>運転可能モードを取得する</summary>
63  public HeatSourceSystemModel.OperatingMode SelectableMode
64  { get { return HeatSourceSystemModel.OperatingMode.Cooling; } }
65
66  /// <summary>運転モードを設定・取得する</summary>
67  public HeatSourceSystemModel.OperatingMode Mode { get; set; }
68
69  /// <summary>現在の日時を設定・取得する</summary>
70  public DateTime CurrentDateTime { get; set; }
71
72  /// <summary>タイムステップを設定・取得する</summary>
73  public double TimeStep { get; set; }
74
75  /// <summary>温水還温度[C]を設定・取得する</summary>
76  public double HotWaterReturnTemperature { get; set; } = 40;
77
78  /// <summary>温水往温度設定値[C]を設定・取得する</summary>
79  public double HotWaterSupplyTemperatureSetpoint { get; set; } = 45;
80
81  /// <summary>温水往温度[C]を取得する</summary>
82  public double HotWaterSupplyTemperature { get; private set; } = 45;
83
84  /// <summary>温水流量[kg/s]を設定・取得する</summary>
85  public double HotWaterFlowRate { get; set; }
86
87  /// <summary>温水上限流量[kg/s]を取得する</summary>
88  public double MaxHotWaterFlowRate
89  { get { return 0; } }
90
91  /// <summary>温水下限流量比[-]を取得する</summary>
92  public double MinHotWaterFlowRatio
93  { get { return 0; } }
94
95  /// <summary>冷水還温度[C]を設定・取得する</summary>
96  public double ChilledWaterReturnTemperature { get; set; } = 12;
97
98  /// <summary>冷水往温度設定値[C]を設定・取得する</summary>
99  public double ChilledWaterSupplyTemperatureSetpoint { get; set; } = 7;
100
101 /// <summary>冷水往温度[C]を取得する</summary>
102 public double ChilledWaterSupplyTemperature { get; private set; } = 7;
103
104 /// <summary>冷水流量[kg/s]を設定・取得する</summary>
105 public double ChilledWaterFlowRate { get; set; }
```

```
106
107   /// <summary>冷水上限流量[kg/s]を取得する</summary>
108   public double MaxChilledWaterFlowRate
109   { get { return chiller.MaxChilledWaterFlowRate * ChillerNumber; } }
110
111   /// <summary>冷水下限流量比[-]を取得する</summary>
112   public double MinChilledWaterFlowRatio
113   { get { return chiller.MinChilledWaterFlowRatio / ChillerNumber; } }
114
115   /// <summary>外気条件を設定・取得する</summary>
116   public ImmutableMoistAir OutdoorAir { get; set; }
117
118   /// <summary>熱源サブシステムを停止させる</summary>
119   public void ShutOff()
120   {
121     IsOverLoad_C = IsOverLoad_H = false;
122     OperatingChillerNumber = 0;
123     chiller.ShutOff();
124     chwPump.ShutOff();
125     cdwPump.ShutOff();
126     cTower.ShutOff();
127   }
128
129   /// <summary>冷温水往温度を予測する</summary>
130   /// <param name="chilledWaterFlowRate">冷水流量[kg/s]</param>
131   /// <param name="hotWaterFlowRate">温水流量[kg/s]</param>
132   public void ForecastSupplyWaterTemperature(double chilledWaterFlowRate, double hotWaterFlowRate)
133   {
134     ChilledWaterFlowRate = chilledWaterFlowRate;
135     HotWaterFlowRate = hotWaterFlowRate;
136
137     if (Mode != HeatSourceSystemModel.OperatingMode.Cooling || chilledWaterFlowRate == 0)
138     {
139       ShutOff();
140       return;
141     }
142
143     chiller.IsOperating = true;
144     cTower.SetOutdoorAirState(OutdoorAir.WetbulbTemperature, OutdoorAir.HumidityRatio);
145
146     OperatingChillerNumber = (int)Math.Ceiling(chilledWaterFlowRate / chiller.MaxChilledWaterFlowRate);
147     OperatingChillerNumber = Math.Min(ChillerNumber, OperatingChillerNumber);
148     chiller.ChilledWaterOutletSetPointTemperature = ChilledWaterSupplyTemperatureSetpoint;
149     while (true)
150     {
151       //冷水・冷却水流量を計算
152       double chwFlow = chilledWaterFlowRate / OperatingChillerNumber;
153       double pLoad = chwFlow / chiller.MaxChilledWaterFlowRate;
154       double cdwFlow;
155       if (ControlCoolingWaterFlowRate) cdwFlow = cTower.WaterFlowRate
156         = cTower.MaxWaterFlowRate * CoolingTowerNumber * Math.Max(pLoad, MinimumCoolingWaterFlowRatio);
157       else cdwFlow = cTower.WaterFlowRate = cTower.MaxWaterFlowRate;
158       if (!OperateCoolingTowerOneOnOne) cdwFlow /= ChillerNumber;
159       cTower.WaterFlowRate = cdwFlow / CoolingTowerNumber;
160
161       //ポンプによる昇温を評価
162       cdwPump.UpdateState(0.001 * cdwFlow);
163       double dCDT = cdwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * cdwFlow);
164       chwPump.UpdateState(0.001 * chwFlow);
165       double twi = ChilledWaterReturnTemperature
166         + chwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * chwFlow);
167
168       //冷却塔と冷凍機の連成計算（冷却水温度の計算）
169       bool needIteration = !ControlCoolingWaterTemperature;
170       if (ControlCoolingWaterTemperature)
171       {
172         chiller.Update(CoolingWaterTemperatureSetpoint + dCDT, twi, cdwFlow, chwFlow);
173         cTower.OutletWaterSetPointTemperature = CoolingWaterTemperatureSetpoint;
174         cTower.Update(chiller.CoolingWaterOutletTemperature, true);
175         if (cTower.IsOverLoad) needIteration = true;
176       }
177       //過負荷または冷却水温度成行の場合には収束計算
178       if (needIteration)
179       {
180         Roots.ErrorFunction eFnc = delegate (double cdt)
181         {
182           chiller.Update(cdt + dCDT, twi, cdwFlow, chwFlow);
183           cTower.Update(chiller.CoolingWaterOutletTemperature, cTower.NominalAirFlowRate);
184           return cTower.OutletWaterTemperature - cdt;
185         };
```

```
186         if (eFnc(10) < 0) break;
187         else if (0 < eFnc(32)) break;
188         else Roots.Bisection(eFnc, 10, 32, 0.01, 0.001, 10);
189       }
190
191       if (OperatingChillerNumber == ChillerNumber || !chiller.IsOverLoad) break;
192       else OperatingChillerNumber++;
193     }
194
195     IsOverLoad_C = chiller.IsOverLoad;
196     if (IsOverLoad_C) ChilledWaterSupplyTemperature = chiller.ChilledWaterOutletTemperature;
197     else ChilledWaterSupplyTemperature = ChilledWaterSupplyTemperatureSetpoint;
198   }
199
201   /// <summary>状態を確定する</summary>
202   public void FixState() { }
203
204   /// <summary>インスタンスを初期化する</summary>
205   /// <param name="chiller">ターボ冷凍機</param>
206   /// <param name="chwPump">冷水ポンプ</param>
207   /// <param name="cdwPump">冷却水ポンプ</param>
208   /// <param name="cTower">冷却塔</param>
209   /// <param name="chillerNumber">冷凍機台数</param>
210   /// <param name="coolingTowerNumber">冷却塔台数（1台の冷凍機あたりの台数）</param>
211   public CentrifugalChillerSystem
212     (ICentrifugalChiller chiller, CentrifugalPump chwPump,
213     CentrifugalPump cdwPump, CoolingTower cTower, int chillerNumber, int coolingTowerNumber)
214   {
215     this.chiller = chiller;
216     this.chwPump = chwPump;
217     this.cdwPump = cdwPump;
218     this.cTower = cTower;
219     this.ChillerNumber = chillerNumber;
220     this.CoolingTowerNumber = coolingTowerNumber;
221
222     Mode = HeatSourceSystemModel.OperatingMode.ShutOff;
223   }
224 }
```

3）空気熱源ヒートポンプによる熱源設備サブシステム

空気熱源ヒートポンプによる熱源設備サブシステムクラスをプログラム 21.4 に示す。

プロパティの定義や運転台数判定処理などに関してはボイラやターボ冷凍機によるサブシステムと同様である。ただし、空気熱源ヒートポンプの場合には冷水製造と温水製造の両者に対応しているため、108, 109 行に示すように運転モードに応じて分岐をかける。

プログラム 21.4　空気熱源ヒートポンプによる熱源設備サブシステムクラス

Popolo.HVAC.SubSystem.AirHeatSourceModularChillersSystem class

```
 1 /// <summary>空気熱源モジュールヒートポンプによる熱源サブシステム</summary>
 2 public class AirHeatSourceModularChillersSystem : IHeatSourceSubSystem
 3 {
 4
 5   /// <summary>水の定圧比熱[kJ/(kgK)]</summary>
 6   private const double WATER_SPECIFIC_HEAT = 4.186;
 7
 8   /// <summary>空気熱源ヒートポンプ</summary>
 9   private AirHeatSourceModularChillers mChiller;
10
11   /// <summary>冷水・温水ポンプ</summary>
12   private CentrifugalPump chwPump, hwPump;
13
14   /// <summary>空気熱源ヒートポンプを取得する</summary>
15   public ImmutableAirHeatSourceModularChillers AirHeatSourceModularChillers { get { return mChiller; } }
16
17   /// <summary>冷水ポンプを取得する</summary>
18   public ImmutableCentrifugalPump ChilledWaterPump { get { return chwPump; } }
19
20   /// <summary>温水ポンプを取得する</summary>
21   public ImmutableCentrifugalPump HotWaterPump { get { return hwPump; } }
22
23   /// <summary>モジュールチラーのセット数を取得する</summary>
24   public int ChillerNumber { get; private set; }
25
26   /// <summary>運転台数を取得する</summary>
27   public int OperatingChillerNumber { get; private set; }
28
```

```
29  /// <summary>冷水供給が過負荷か否か</summary>
30  public bool IsOverLoad_C { get; private set; }
31
32  /// <summary>温水供給が過負荷か否か</summary>
33  public bool IsOverLoad_H { get; private set; }
34
35  /// <summary>運転可能モードを取得する</summary>
36  public HeatSourceSystemModel.OperatingMode SelectableMode { get; }
37
38  /// <summary>運転モードを設定・取得する</summary>
39  public HeatSourceSystemModel.OperatingMode Mode { get; set; }
40
41  /// <summary>現在の日時を設定・取得する</summary>
42  public DateTime CurrentDateTime { get; set; }
43
44  /// <summary>タイムステップを設定・取得する</summary>
45  public double TimeStep { get; set; }
46
47  /// <summary>温水還温度[C]を設定・取得する</summary>
48  public double HotWaterReturnTemperature { get; set; } = 40;
49
50  /// <summary>温水往温度設定値[C]を設定・取得する</summary>
51  public double HotWaterSupplyTemperatureSetpoint { get; set; } = 45;
52
53  /// <summary>温水往温度[C]を取得する</summary>
54  public double HotWaterSupplyTemperature { get; private set; } = 45;
55
56  /// <summary>温水流量[kg/s]を設定・取得する</summary>
57  public double HotWaterFlowRate { get; set; }
58
59  /// <summary>温水上限流量[kg/s]を取得する</summary>
60  public double MaxHotWaterFlowRate
61  { get { return mChiller.MaxHotWaterFlowRate * ChillerNumber; } }
62
63  /// <summary>温水下限流量比[-]を取得する</summary>
64  public double MinHotWaterFlowRatio
65  { get { return mChiller.MinHotWaterFlowRate / MaxHotWaterFlowRate; } }
66
67  /// <summary>冷水還温度[C]を設定・取得する</summary>
68  public double ChilledWaterReturnTemperature { get; set; } = 12;
69
70  /// <summary>冷水往温度設定値[C]を設定・取得する</summary>
71  public double ChilledWaterSupplyTemperatureSetpoint { get; set; } = 7;
72
73  /// <summary>冷水往温度[C]を取得する</summary>
74  public double ChilledWaterSupplyTemperature { get; private set; } = 7;
75
76  /// <summary>冷水流量[kg/s]を設定・取得する</summary>
77  public double ChilledWaterFlowRate { get; set; }
78
79  /// <summary>冷水上限流量[kg/s]を取得する</summary>
80  public double MaxChilledWaterFlowRate
81  { get { return mChiller.MaxChilledWaterFlowRate * ChillerNumber; } }
82
83  /// <summary>冷水下限流量比[-]を取得する</summary>
84  public double MinChilledWaterFlowRatio
85  { get { return mChiller.MinChilledWaterFlowRate / MaxChilledWaterFlowRate; } }
86
87  /// <summary>外気条件を設定・取得する</summary>
88  public ImmutableMoistAir OutdoorAir { get; set; }
89
90  /// <summary>熱源サブシステムを停止させる</summary>
91  public void ShutOff()
92  {
93    IsOverLoad_C = IsOverLoad_H = false;
94    OperatingChillerNumber = 0;
95    mChiller.ShutOff();
96    chwPump.ShutOff();
97    hwPump.ShutOff();
98  }
99
100 /// <summary>冷温水往温度を予測する</summary>
101 /// <param name="chilledWaterFlowRate">冷水流量[kg/s]</param>
102 /// <param name="hotWaterFlowRate">温水流量[kg/s]</param>
103 public void ForecastSupplyWaterTemperature(double chilledWaterFlowRate, double hotWaterFlowRate)
104 {
105   ChilledWaterFlowRate = chilledWaterFlowRate;
106   HotWaterFlowRate = hotWaterFlowRate;
107
108   if (Mode == HeatSourceSystemModel.OperatingMode.Cooling) forecastCooling(chilledWaterFlowRate);
```

```
109     else if (Mode == HeatSourceSystemModel.OperatingMode.Heating) forecastHeating(hotWaterFlowRate);
110     else ShutOff();
111   }
112
113   /// <summary>将来状態を予測する（暖房）</summary>
114   /// <param name="hotWaterFlowRate">温水流量[kg/s]</param>
115   private void forecastHeating(double hotWaterFlowRate)
116   {
117     if (hotWaterFlowRate == 0)
118     {
119       ShutOff();
120       return;
121     }
122     mChiller.Mode = HeatSource.AirHeatSourceModularChillers.OperatingMode.Heating;
123     ChilledWaterSupplyTemperature = ChilledWaterReturnTemperature;
124
125     OperatingChillerNumber = (int)Math.Ceiling(hotWaterFlowRate / mChiller.MaxHotWaterFlowRate);
126     OperatingChillerNumber = Math.Min(ChillerNumber, OperatingChillerNumber);
127     mChiller.WaterOutletSetPointTemperature = HotWaterSupplyTemperatureSetpoint;
128     while (true)
129     {
130       //ポンプによる昇温を評価
131       double hwFlow = hotWaterFlowRate / OperatingChillerNumber;
132       hwPump.UpdateState(0.001 * hwFlow);
133       double twi = HotWaterReturnTemperature
134         + hwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * hwFlow);
135
136       mChiller.Update(twi, hwFlow, OutdoorAir.DrybulbTemperature);
137       IsOverLoad_H = mChiller.IsOverLoad;
138       if (OperatingChillerNumber == ChillerNumber || !IsOverLoad_H) break;
139       else OperatingChillerNumber++;
140     }
141
142     if (mChiller.IsOverLoad) HotWaterSupplyTemperature = mChiller.WaterOutletTemperature;
143     else HotWaterSupplyTemperature = HotWaterSupplyTemperatureSetpoint;
144   }
145
146   /// <summary>将来状態を予測する（冷房）</summary>
147   /// <param name="chilledWaterFlowRate">冷水流量[kg/s]</param>
148   private void forecastCooling(double chilledWaterFlowRate)
149   {
150     if (chilledWaterFlowRate == 0)
151     {
152       ShutOff();
153       return;
154     }
155     mChiller.Mode = HeatSource.AirHeatSourceModularChillers.OperatingMode.Cooling;
156     HotWaterSupplyTemperature = HotWaterReturnTemperature;
157
158     OperatingChillerNumber = (int)Math.Ceiling(chilledWaterFlowRate / mChiller.MaxChilledWaterFlowRate);
159     OperatingChillerNumber = Math.Min(ChillerNumber, OperatingChillerNumber);
160     mChiller.WaterOutletSetPointTemperature = ChilledWaterSupplyTemperatureSetpoint;
161     while (true)
162     {
163       //ポンプによる昇温を評価
164       double chwFlow = chilledWaterFlowRate / OperatingChillerNumber;
165       chwPump.UpdateState(0.001 * chwFlow);
166       double twi = ChilledWaterReturnTemperature
167         + chwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * chwFlow);
168
169       mChiller.Update(twi, chwFlow, OutdoorAir.DrybulbTemperature);
170       IsOverLoad_C = mChiller.IsOverLoad;
171       if (OperatingChillerNumber == ChillerNumber || !IsOverLoad_C) break;
172       else OperatingChillerNumber++;
173     }
174
175     if (mChiller.IsOverLoad) ChilledWaterSupplyTemperature = mChiller.WaterOutletTemperature;
176     else ChilledWaterSupplyTemperature = ChilledWaterSupplyTemperatureSetpoint;
177   }
178
179   /// <summary>状態を確定する</summary>
180   public void FixState() { }
181
182   /// <summary>インスタンスを初期化する</summary>
183   /// <param name="mChiller">モジュールチラー</param>
184   /// <param name="chwPump">冷水ポンプ</param>
185   /// <param name="hwPump">温水ポンプ</param>
186   /// <param name="number">モジュールチラーの数</param>
187   public AirHeatSourceModularChillersSystem
188     (AirHeatSourceModularChillers mChiller, CentrifugalPump chwPump, CentrifugalPump hwPump, int number)
```

```
189   {
190     this.mChiller = mChiller;
191     this.ChillerNumber = number;
192     this.chwPump = chwPump;
193     this.hwPump = hwPump;
194
195     //加熱運転・冷却運転対応
196     SelectableMode =
197       HeatSourceSystemModel.OperatingMode.Cooling | HeatSourceSystemModel.OperatingMode.Heating;
198     Mode = HeatSourceSystemModel.OperatingMode.ShutOff;
199   }
200 }
```

4）直焚吸収冷温水機による熱源設備サブシステム

直焚吸収冷温水機による熱源設備サブシステムクラスをプログラム 21.5 に示す。

空気熱源ヒートポンプによるサブシステムと同様に冷水製造と温水製造の両方に対応する（137, 138 行）ことに加え、ターボ冷凍機によるサブシステムと同様に冷却運転時の冷却水温度計算（215~236 行）に対応させる。

プログラム 21.5　直焚吸収冷温水機による熱源設備サブシステムクラス

Popolo.HVAC.SubSystem.DirectFiredAbsorptionChillerSystem class

```
  1 /// <summary>直焚吸収冷温水機による熱源サブシステム</summary>
  2 public class DirectFiredAbsorptionChillerSystem: IHeatSourceSubSystem
  3 {
  4
  5   /// <summary>水の定圧比熱[kJ/(kgK)]</summary>
  6   private const double WATER_SPECIFIC_HEAT = 4.186;
  7
  8   /// <summary>直焚吸収冷温水機</summary>
  9   private DirectFiredAbsorptionChiller chiller;
 10
 11   /// <summary>ポンプ</summary>
 12   private CentrifugalPump cdwPump, chwPump, hwPump;
 13
 14   /// <summary>冷却塔</summary>
 15   private CoolingTower cTower;
 16
 17   /// <summary>直焚吸収冷温水機を取得する</summary>
 18   public ImmutableDirectFiredAbsorptionChiller DirectFiredAbsorptionChiller { get { return chiller; } }
 19
 20   /// <summary>冷水ポンプを取得する</summary>
 21   public ImmutableCentrifugalPump ChilledWaterPump { get { return chwPump; } }
 22
 23   /// <summary>温水ポンプを取得する</summary>
 24   public ImmutableCentrifugalPump HotWaterPump { get { return hwPump; } }
 25
 26   /// <summary>冷却水ポンプを取得する</summary>
 27   public ImmutableCentrifugalPump CoolingWaterPump { get { return cdwPump; } }
 28
 29   /// <summary>冷却塔を取得する</summary>
 30   public ImmutableCoolingTower CoolingTower { get { return cTower; } }
 31
 32   /// <summary>冷凍機のセル数を取得する</summary>
 33   public int ChillerNumber { get; private set; }
 34
 35   /// <summary>冷却塔の台数（1 台の冷凍機あたり）を取得する</summary>
 36   public int CoolingTowerNumber { get; private set; }
 37
 38   /// <summary>運転台数を取得する</summary>
 39   public int OperatingChillerNumber { get; private set; }
 40
 41   /// <summary>冷却塔を 1 対 1 で稼働させるか否か</summary>
 42   public bool OperateCoolingTowerOneOnOne { get; set; } = true;
 43
 44   /// <summary>冷却水温度を制御するか否か</summary>
 45   public bool ControlCoolingWaterTemperature { get; set; }
 46
 47   /// <summary>冷却水流量を制御するか否か（負荷率比例）</summary>
 48   public bool ControlCoolingWaterFlowRate { get; set; }
 49
 50   /// <summary>最小冷却水流量比[-]を設定・取得する</summary>
 51   public double MinimumCoolingWaterFlowRatio { get; set; } = 0.5;
 52
 53   /// <summary>冷却水温度設定値[C]を設定・取得する</summary>
```

```
54  public double CoolingWaterTemperatureSetpoint { get; set; } = 32;
55
56  /// <summary>冷水供給が過負荷か否か</summary>
57  public bool IsOverLoad_C { get; private set; }
58
59  /// <summary>温水供給が過負荷か否か</summary>
60  public bool IsOverLoad_H { get; private set; }
61
62  /// <summary>運転可能モードを取得する</summary>
63  public HeatSourceSystemModel.OperatingMode SelectableMode { get; }
64
65  /// <summary>運転モードを設定・取得する</summary>
66  public HeatSourceSystemModel.OperatingMode Mode { get; set; }
67
68  /// <summary>現在の日時を設定・取得する</summary>
69  public DateTime CurrentDateTime { get; set; }
70
71  /// <summary>タイムステップを設定・取得する</summary>
72  public double TimeStep { get; set; }
73
74  /// <summary>温水還温度[C]を設定・取得する</summary>
75  public double HotWaterReturnTemperature { get; set; } = 40;
76
77  /// <summary>温水往温度設定値[C]を設定・取得する</summary>
78  public double HotWaterSupplyTemperatureSetpoint { get; set; } = 45;
79
80  /// <summary>温水往温度[C]を取得する</summary>
81  public double HotWaterSupplyTemperature { get; private set; } = 45;
82
83  /// <summary>温水流量[kg/s]を設定・取得する</summary>
84  public double HotWaterFlowRate { get; set; }
85
86  /// <summary>温水上限流量[kg/s]を取得する</summary>
87  public double MaxHotWaterFlowRate
88  { get { return chiller.MaxHotWaterFlowRate * ChillerNumber; } }
89
90  /// <summary>温水下限流量比[-]を取得する</summary>
91  public double MinHotWaterFlowRatio
92  { get { return chiller.MinHotWaterFlowRate / MaxHotWaterFlowRate; } }
93
94  /// <summary>冷水還温度[C]を設定・取得する</summary>
95  public double ChilledWaterReturnTemperature { get; set; } = 12;
96
97  /// <summary>冷水往温度設定値[C]を設定・取得する</summary>
98  public double ChilledWaterSupplyTemperatureSetpoint { get; set; } = 7;
99
100 /// <summary>冷水往温度[C]を取得する</summary>
101 public double ChilledWaterSupplyTemperature { get; private set; } = 7;
102
103 /// <summary>冷水流量[kg/s]を設定・取得する</summary>
104 public double ChilledWaterFlowRate { get; set; }
105
106 /// <summary>冷水上限流量[kg/s]を取得する</summary>
107 public double MaxChilledWaterFlowRate
108 { get { return chiller.MaxChilledWaterFlowRate * ChillerNumber; } }
109
110 /// <summary>冷水下限流量比[-]を取得する</summary>
111 public double MinChilledWaterFlowRatio
112 { get { return chiller.MinChilledWaterFlowRate / MaxChilledWaterFlowRate; } }
113
114 /// <summary>外気条件を設定・取得する</summary>
115 public ImmutableMoistAir OutdoorAir { get; set; }
116
117 /// <summary>熱源サブシステムを停止させる</summary>
118 public void ShutOff()
119 {
120   IsOverLoad_C = IsOverLoad_H = false;
121   OperatingChillerNumber = 0;
122   chiller.ShutOff();
123   chwPump.ShutOff();
124   hwPump.ShutOff();
125   cdwPump.ShutOff();
126   cTower.ShutOff();
127 }
128
129 /// <summary>冷温水往温度を予測する</summary>
130 /// <param name="chilledWaterFlowRate">冷水流量[kg/s]</param>
131 /// <param name="hotWaterFlowRate">温水流量[kg/s]</param>
132 public void ForecastSupplyWaterTemperature(double chilledWaterFlowRate, double hotWaterFlowRate)
133 {
```

```
134     ChilledWaterFlowRate = chilledWaterFlowRate;
135     HotWaterFlowRate = hotWaterFlowRate;
136
137     if (Mode == HeatSourceSystemModel.OperatingMode.Cooling) forecastCooling(chilledWaterFlowRate);
138     else if (Mode == HeatSourceSystemModel.OperatingMode.Heating) forecastHeating(hotWaterFlowRate);
139     else ShutOff();
140   }
141
142   /// <summary>将来状態を予測する（暖房）</summary>
143   /// <param name="hotWaterFlowRate">温水流量[kg/s]</param>
144   private void forecastHeating(double hotWaterFlowRate)
145   {
146     if (hotWaterFlowRate == 0)
147     {
148       ShutOff();
149       return;
150     }
151     chiller.IsCoolingMode = false;
152
153     ChilledWaterSupplyTemperature = ChilledWaterReturnTemperature;
154     cTower.ShutOff();
155     cdwPump.ShutOff();
156     chwPump.ShutOff();
157
158     OperatingChillerNumber = (int)Math.Ceiling(hotWaterFlowRate / chiller.MaxHotWaterFlowRate);
159     OperatingChillerNumber = Math.Min(ChillerNumber, OperatingChillerNumber);
160     chiller.OutletWaterSetPointTemperature = HotWaterSupplyTemperatureSetpoint;
161     while (true)
162     {
163       //ポンプによる昇温を評価
164       double hwFlow = hotWaterFlowRate / OperatingChillerNumber;
165       hwPump.UpdateState(0.001 * hwFlow);
166       double twi = HotWaterReturnTemperature
167         + hwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * hwFlow);
168
169       chiller.Update(32, twi, 0, hwFlow);
170       if (OperatingChillerNumber == ChillerNumber || !chiller.IsOverLoad) break;
171       else OperatingChillerNumber++;
172     }
173
174     IsOverLoad_H = chiller.IsOverLoad;
175     if (IsOverLoad_H) HotWaterSupplyTemperature = chiller.OutletWaterTemperature;
176     else HotWaterSupplyTemperature = HotWaterSupplyTemperatureSetpoint;
177   }
178
179   /// <summary>将来状態を予測する（冷房）</summary>
180   /// <param name="chilledWaterFlowRate">冷水流量[kg/s]</param>
181   private void forecastCooling(double chilledWaterFlowRate)
182   {
183     if (chilledWaterFlowRate == 0)
184     {
185       ShutOff();
186       return;
187     }
188     chiller.IsCoolingMode = true;
189
190     HotWaterSupplyTemperature = HotWaterReturnTemperature;
191     cTower.SetOutdoorAirState(OutdoorAir.WetbulbTemperature, OutdoorAir.HumidityRatio);
192     hwPump.ShutOff();
193
194     OperatingChillerNumber = (int)Math.Ceiling(chilledWaterFlowRate / chiller.MaxChilledWaterFlowRate);
195     OperatingChillerNumber = Math.Min(ChillerNumber, OperatingChillerNumber);
196     chiller.OutletWaterSetPointTemperature = ChilledWaterSupplyTemperatureSetpoint;
197     while (true)
198     {
199       //冷水・冷却水流量を計算
200       double chwFlow = chilledWaterFlowRate / OperatingChillerNumber;
201       double pLoad = chwFlow / chiller.MaxChilledWaterFlowRate;
202       double cdwFlow;
203       if (ControlCoolingWaterFlowRate) cdwFlow = cTower.WaterFlowRate
204           = cTower.MaxWaterFlowRate * Math.Max(pLoad, MinimumCoolingWaterFlowRatio);
205       else cdwFlow = cTower.WaterFlowRate = cTower.MaxWaterFlowRate;
206       if (!OperateCoolingTowerOneOnOne) cdwFlow /= ChillerNumber;
207
208       //ポンプによる昇温を評価
209       cdwPump.UpdateState(0.001 * cdwFlow);
210       double dCDT = cdwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * cdwFlow);
211       chwPump.UpdateState(0.001 * chwFlow);
212       double twi = ChilledWaterReturnTemperature
213         + chwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * chwFlow);
```

```
214
215       //冷却塔と冷凍機の連成計算（冷却水温度の計算）
216       bool needIteration = !ControlCoolingWaterTemperature;
217       if (ControlCoolingWaterTemperature)
218       {
219         chiller.Update(CoolingWaterTemperatureSetpoint + dCDT, twi, cdwFlow, chwFlow);
220         cTower.OutletWaterSetPointTemperature = CoolingWaterTemperatureSetpoint;
221         cTower.Update(chiller.CoolingWaterOutletTemperature, true);
222         if (cTower.IsOverLoad) needIteration = true;
223       }
224       //過負荷または冷却水温度成行の場合には収束計算
225       if (needIteration)
226       {
227         Roots.ErrorFunction eFnc = delegate (double cdt)
228         {
229           chiller.Update(cdt + dCDT, twi, cdwFlow, chwFlow);
230           cTower.Update(chiller.CoolingWaterOutletTemperature, cTower.NominalAirFlowRate);
231           return cTower.OutletWaterTemperature - cdt;
232         };
233         if (eFnc(10) < 0) break;
234         else if (0 < eFnc(32)) break;
235         else Roots.Bisection(eFnc, 10, 32, 0.01, 0.001, 10);
236       }
237
238       if (OperatingChillerNumber == ChillerNumber || !chiller.IsOverLoad) break;
239       else OperatingChillerNumber++;
240     }
241
242     IsOverLoad_C = chiller.IsOverLoad;
243     if (IsOverLoad_C) ChilledWaterSupplyTemperature = chiller.OutletWaterTemperature;
244     else ChilledWaterSupplyTemperature = ChilledWaterSupplyTemperatureSetpoint;
245   }
246
247   /// <summary>状態を確定する</summary>
248   public void FixState() { }
249
250   /// <summary>インスタンスを初期化する</summary>
251   /// <param name="chiller">直焚吸収冷温水発生機</param>
252   /// <param name="chwPump">冷水ポンプ</param>
253   /// <param name="hwPump">温水ポンプ</param>
254   /// <param name="cdwPump">冷却水ポンプ</param>
255   /// <param name="cTower">冷却塔</param>
256   /// <param name="chillerNumber">冷凍機台数</param>
257   /// <param name="coolingTowerNumber">冷却塔台数（1台の冷凍機あたりの台数）</param>
258   public DirectFiredAbsorptionChillerSystem
259     (DirectFiredAbsorptionChiller chiller, CentrifugalPump chwPump, CentrifugalPump hwPump,
260     CentrifugalPump cdwPump, CoolingTower cTower, int chillerNumber, int coolingTowerNumber)
261   {
262     this.chiller = chiller;
263     this.chwPump = chwPump;
264     this.hwPump = hwPump;
265     this.cdwPump = cdwPump;
266     this.cTower = cTower;
267     this.ChillerNumber = chillerNumber;
268     this.CoolingTowerNumber = coolingTowerNumber;
269      //加熱運転・冷却運転対応
270     SelectableMode = HeatSourceSystemModel.OperatingMode.Cooling |
271       HeatSourceSystemModel.OperatingMode.Heating;
272     Mode = HeatSourceSystemModel.OperatingMode.ShutOff;
273   }
274
275 }
```

5）温度成層型蓄熱槽による熱源設備サブシステム

温度成層型蓄熱槽による熱源設備サブシステムクラスをプログラム 21.6 に示す。

既に述べたように、熱容量を持つ動的システムであるため、水槽の温度を更新する際には、その更新処理が反復計算の過程におけるものであるか、あるいは反復計算を終了させる確定的な更新処理かを判定する必要がある。このために 9 行目で反復計算の過程における予測計算であるか否かを判定するフラグを設ける。以降 90 行までは蓄熱槽による熱源設備サブシステム固有のプロパティ定義である。76 行は蓄熱槽の温度更新のためのタイムステップであり、計算の精度を向上させるために、全体のタイムステップよりも小さい数値を設定できるようにする。85~90 行は蓄熱運転フラグであり、

ターボ冷凍機により蓄熱を行う場合には true を設定する。98~100 行の運転モードは二次側へ冷水を供給するか否かの運転モード判定であり、本フラグはこれとは独立に設定可能であることに注意する。

166~283 行が冷水温度の予測処理である。175~181 行で反復計算中か初回の計算かを判定し、初回の計算であれば現在の水温を一時保存する（180 行）。反復計算中であれば、一時保存した水温を復元する（176 行）。362 行の FixState メソッドが呼び出されるまでは、反復計算が継続されているということであるため、水温の復元を繰り返す。

187~276 行は水槽の温度更新の繰り返し計算処理である。既に述べたように蓄熱槽のタイムステップは全体のタイムステップよりも小さい値に設定できるため、189~192 行で両者の大小関係と残余の時間を計算し、全体のタイムステップに到達するまで水槽温度の更新を続ける。

蓄熱槽を用いた熱源システムは、蓄熱と放熱の組み合わせによって大きく 4 つの運転状態に分類できる。1 つめは追掛運転であり、二次側への放熱を行いつつターボ冷凍機を運転させる状態である（194~226 行）。2 つめは放熱運転であり、ターボ冷凍機を停止させて二次側への放熱のみを行う状態である（227~254 行）。3 つめは蓄熱運転であり、二次側放熱を行わず、ターボ冷凍機を運転させる状態である（255~265 行）。4 つめは停止運転であり、この場合には蓄熱槽からの放熱ロスのみが計算対象となる（266~272 行）。放熱運転と蓄熱運転の場合には水流の向きと流量が確定的であるため計算が単純であるが、追掛運転の場合にはターボ冷凍機の出入口温度と水槽温度が混合して計算式が循環するため、反復計算が必要になる。このために二分法で放熱用熱交換器の熱源水流量を求根する。誤差関数は 201~205 行と 283~328 行で示すとおりである。288~290 行で水流の向きを確定する。ターボ冷凍機が出口設定温度を実現可能と仮定し、292~311 行で水流の向きに応じた冷水の混合処理を行い、熱交換器の出口状態を計算する。熱交換器の出口状態にもとづいて 314 行でターボ冷凍機の入口冷水温度を計算する。ターボ冷凍機が過負荷の場合には先の仮定が満たされないため、316~327 行で 1 度だけ再計算を行う（計算速度を高めるため反復計算は行わない）。なお、冷却塔との連成計算法については、既に記したターボ冷凍機による熱源設備サブシステムクラスと同様である。

プログラム 21.6　温度成層型蓄熱槽による熱源設備サブシステムクラス

Popolo.HVAC.SubSystem.MultipleStratifiedWaterTankSystem class

```
/// <summary>温度成層型蓄熱槽による熱源サブシステム</summary>
public class MultipleStratifiedWaterTankSystem : IHeatSourceSubSystem
{

    /// <summary>水の定圧比熱[kJ/(kgK)]</summary>
    private const double WATER_SPECIFIC_HEAT = 4.186;

    /// <summary>予測計算中か否か</summary>
    private bool isForecasting = false;

    /// <summary>水槽内温度分布</summary>
    private double[] oldTemps;

    /// <summary>冷凍機ポンプによる昇温[K]</summary>
    private double dtChillingPump, dtCoolingPump;

    /// <summary>ターボ冷凍機</summary>
    private ICentrifugalChiller chiller;

    /// <summary>ポンプ</summary>
    private CentrifugalPump cdwPump, chwPump, chgPump, disPump;

    /// <summary>冷却塔</summary>
    private CoolingTower cTower;
```

```
25
26      /// <summary>プレート熱交換器</summary>
27      private PlateHeatExchanger pHex;
28
29      /// <summary>温度成層型蓄熱槽</summary>
30      private MultipleStratifiedWaterTank wTank;
31
32      /// <summary>ターボ冷凍機を取得する</summary>
33      public ImmutableCentrifugalChiller Chiller { get { return chiller; } }
34
35      /// <summary>冷水ポンプを取得する</summary>
36      public ImmutableCentrifugalPump ChilledWaterPump { get { return chwPump; } }
37
38      /// <summary>冷却水ポンプを取得する</summary>
39      public ImmutableCentrifugalPump CoolingWaterPump { get { return cdwPump; } }
40
41      /// <summary>蓄熱ポンプを取得する</summary>
42      public ImmutableCentrifugalPump ChargePump { get { return chgPump; } }
43
44      /// <summary>放熱ポンプを取得する</summary>
45      public ImmutableCentrifugalPump DischargePump { get { return disPump; } }
46
47      /// <summary>冷却塔を取得する</summary>
48      public ImmutableCoolingTower CoolingTower { get { return cTower; } }
49
50      /// <summary>プレート熱交換器を取得する</summary>
51      public ImmutablePlateHeatExchanger PlateHeatExchanger { get { return pHex; } }
52
53      /// <summary>温度成層型蓄熱槽</summary>
54      public ImmutableMultipleStratifiedWaterTank WaterTank { get { return wTank; } }
55
56      /// <summary>冷凍機の台数を取得する</summary>
57      public int ChillerNumber { get; private set; }
58
59      /// <summary>冷却塔のセル数（1台の冷凍機あたり）を取得する</summary>
60      public int CoolingTowerNumber { get; private set; }
61
62      /// <summary>冷却水温度を制御するか否か</summary>
63      public bool ControlCoolingWaterTemperature { get; set; }
64
65      /// <summary>冷却水温度設定値[C]を設定・取得する</summary>
66      public double CoolingWaterTemperatureSetpoint { get; set; } = 32;
67
68      /// <summary>冷凍機を運転させるか否か</summary>
69      public bool OperateChiler
70      {
71        get { return chiller.IsOperating; }
72        set { chiller.IsOperating = value; }
73      }
74
75      /// <summary>水蓄熱槽の状態更新タイムステップ[sec]を設定・取得する</summary>
76      public double TankUpdateTimeStep { get; set; } = 600;
77
78      /// <summary>蓄熱温度[C]を設定・取得する</summary>
79      public double StorageTemperature
80      {
81        get { return chiller.ChilledWaterOutletSetPointTemperature; }
82        set { chiller.ChilledWaterOutletSetPointTemperature = value; }
83      }
84
85      /// <summary>蓄熱運転中か否か</summary>
86      public bool Charging
87      {
88        get { return chiller.IsOperating; }
89        set { chiller.IsOperating = value; }
90      }
91
92      /// <summary>冷水供給が過負荷か否か</summary>
93      public bool IsOverLoad_C { get; private set; }
94
95      /// <summary>温水供給が過負荷か否か</summary>
96      public bool IsOverLoad_H { get; private set; }
97
98      /// <summary>運転可能モードを取得する</summary>
99      public HeatSourceSystemModel.OperatingMode SelectableMode
100     { get { return HeatSourceSystemModel.OperatingMode.Cooling; } }
101
102     /// <summary>運転モードを設定・取得する</summary>
103     public HeatSourceSystemModel.OperatingMode Mode { get; set; }
104
```

```
105     /// <summary>現在の日時を設定・取得する</summary>
106     public DateTime CurrentDateTime { get; set; }
107
108     /// <summary>タイムステップを設定・取得する</summary>
109     public double TimeStep { get; set; }
110
111     /// <summary>温水還温度[C]を設定・取得する</summary>
112     public double HotWaterReturnTemperature { get; set; } = 40;
113
114     /// <summary>温水往温度設定値[C]を設定・取得する</summary>
115     public double HotWaterSupplyTemperatureSetpoint { get; set; } = 45;
116
117     /// <summary>温水往温度[C]を取得する</summary>
118     public double HotWaterSupplyTemperature { get; private set; } = 45;
119
120     /// <summary>温水流量[kg/s]を取得する</summary>
121     public double HotWaterFlowRate { get; private set; }
122
123     /// <summary>温水上限流量[kg/s]を取得する</summary>
124     public double MaxHotWaterFlowRate { get { return 0; } }
125
126     /// <summary>温水下限流量比[-]を取得する</summary>
127     public double MinHotWaterFlowRatio { get { return 0; } }
128
129     /// <summary>冷水還温度[C]を設定・取得する</summary>
130     public double ChilledWaterReturnTemperature { get; set; } = 12;
131
132     /// <summary>冷水往温度設定値[C]を設定・取得する</summary>
133     public double ChilledWaterSupplyTemperatureSetpoint
134     {
135       get { return pHex.SupplyTemperatureSetpoint; }
136       set { pHex.SupplyTemperatureSetpoint = value; }
137     }
138
139     /// <summary>冷水往温度[C]を取得する</summary>
140     public double ChilledWaterSupplyTemperature { get; private set; } = 7;
141
142     /// <summary>冷水流量[kg/s]を取得する</summary>
143     public double ChilledWaterFlowRate { get; private set; }
144
145     /// <summary>冷水上限流量[kg/s]を取得する</summary>
146     public double MaxChilledWaterFlowRate { get { return pHex.MaxSupplyFlowRate; } }
147
148     /// <summary>冷水下限流量比[-]を取得する</summary>
149     public double MinChilledWaterFlowRatio { get { return 0; } }
150
151     /// <summary>外気条件を設定・取得する</summary>
152     public ImmutableMoistAir OutdoorAir { get; set; }
153
154     /// <summary>熱源サブシステムを停止させる</summary>
155     public void ShutOff()
156     {
157       IsOverLoad_C = IsOverLoad_H = false;
158       chiller.ShutOff();
159       chwPump.ShutOff();
160       cdwPump.ShutOff();
161       chgPump.ShutOff();
162       disPump.ShutOff();
163       cTower.ShutOff();
164     }
165
166     /// <summary>冷温水往温度を予測する</summary>
167     /// <param name="chilledWaterFlowRate">冷水流量[kg/s]</param>
168     /// <param name="hotWaterFlowRate">温水流量[kg/s]</param>
169     public void ForecastSupplyWaterTemperature(double chilledWaterFlowRate, double hotWaterFlowRate)
170     {
171       ChilledWaterFlowRate = chilledWaterFlowRate;
172       HotWaterFlowRate = hotWaterFlowRate;
173       cTower.SetOutdoorAirState(OutdoorAir.WetbulbTemperature, OutdoorAir.HumidityRatio);
174
175       //水槽内温度分布の一時保存と復元
176       if (isForecasting) wTank.InitializeTemperature(oldTemps);
177       else
178       {
179         isForecasting = true;
180         wTank.GetTemperatures(ref oldTemps);
181       }
182
183       //タイムステップが経過するまで水槽温度更新を続ける
184       double remTime = TimeStep;
```

```
185       double aveTemp = 0;
186       IsOverLoad_C = false;
187       while (true)
188       {
189         bool isLastCalc = remTime < TankUpdateTimeStep;
190         wTank.TimeStep = TankUpdateTimeStep;
191         if (isLastCalc) wTank.TimeStep = remTime;
192         remTime -= wTank.TimeStep;
193
194         //冷凍機稼働かつ二次側負荷有りの場合はHEX所要流量を収束計算（追掛運転）
195         if (Charging && 0 < ChilledWaterFlowRate)
196         {
197           //冷水ポンプの昇温幅を計算
198           chwPump.UpdateState(0.001 * ChilledWaterFlowRate);
199           double dtCHP = chwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * ChilledWaterFlowRate);
200
201           Roots.ErrorFunction eFnc = delegate (double hexFlow)
202           {
203             calcHexHeatTransfer(hexFlow, ChilledWaterReturnTemperature + dtCHP);
204             return pHex.SupplyTemperature - ChilledWaterSupplyTemperatureSetpoint;
205           };
206
207           //最大流量で処理可能か
208           if (eFnc(pHex.MaxHeatSourceFlowRate) < 0)
209           {
210             IsOverLoad_C = false;
211             //最小流量で制御可能か
212             if (0 < eFnc(pHex.MaxHeatSourceFlowRate * 0.001))
213               Roots.Bisection(eFnc, pHex.MaxHeatSourceFlowRate * 0.001,
214             pHex.MaxHeatSourceFlowRate, 0.01, pHex.MaxHeatSourceFlowRate * 0.01, 20);
215           }
216           else IsOverLoad_C = true;
217
218           //水槽内温度更新処理
219           double tTNKin;
220           bool isDownFlow = chiller.ChilledWaterFlowRate < pHex.HeatSourceFlowRate;
221           if (isDownFlow) tTNKin = pHex.HeatSourceOutletTemperature
222             + disPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * pHex.HeatSourceFlowRate);
223           else tTNKin = chiller.ChilledWaterOutletTemperature;
224           wTank.UpdateState
225             (tTNKin, 0.001 * Math.Abs(chiller.ChilledWaterFlowRate - pHex.HeatSourceFlowRate), isDownFlow);
226         }
227         //冷凍機非稼働+二次側負荷有り（放熱運転）
228         else if (0 < ChilledWaterFlowRate)
229         {
230           chiller.ShutOff();
231           chgPump.ShutOff();
232           cdwPump.ShutOff();
233           cTower.ShutOff();
234
235           //冷水ポンプの昇温幅を計算
236           chwPump.UpdateState(0.001 * ChilledWaterFlowRate);
237           double dtCHP = chwPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * ChilledWaterFlowRate);
238
239           //放熱用プレート熱交換器の必要通水量を計算
240           pHex.ControlSupplyTemperature
241             (WaterTank.LowerOutletTemperarture, ChilledWaterReturnTemperature + dtCHP, ChilledWaterFlowRate);
242
243           //水槽内温度を更新
244           double tankInletTemp = 0;
245           if (0 < pHex.HeatSourceFlowRate)
246           {
247             disPump.UpdateState(0.001 * pHex.HeatSourceFlowRate);
248             tankInletTemp = pHex.HeatSourceOutletTemperature
249               + disPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * pHex.HeatSourceFlowRate);
250           }
251           wTank.UpdateState(tankInletTemp, 0.001 * pHex.HeatSourceFlowRate, true);
252
253           IsOverLoad_C = IsOverLoad_C || pHex.IsOverLoad;
254         }
255         //冷凍機稼働+二次側負荷無し（蓄熱運転）
256         else if (Charging)
257         {
258           pHex.ShutOff();
259           disPump.ShutOff();
260           chwPump.ShutOff();
261           calcChillerAndCoolingTower(wTank.UpperOutletTemperarture + dtChillingPump);
262           wTank.UpdateState
263             (chiller.ChilledWaterOutletTemperature, 0.001 * chiller.ChilledWaterFlowRate, false);
264           IsOverLoad_C = false;
```

```
265         }
266         //冷凍機非稼働+二次側負荷無し（蓄熱槽熱損失のみ）
267         else
268         {
269           ShutOff();
270           wTank.UpdateState(0, 0, true);
271           IsOverLoad_C = false;
272         }
273         aveTemp += pHex.SupplyTemperature * wTank.TimeStep;
274
275         if (isLastCalc) break;
276       }
277
278       //過負荷の場合には平均的な供給水温を出力
279       if (IsOverLoad_C) ChilledWaterSupplyTemperature = aveTemp / TimeStep;
280       else ChilledWaterSupplyTemperature = ChilledWaterSupplyTemperatureSetpoint;
281     }
282
283     /// <summary>熱交換器の熱流を更新する</summary>
284     /// <param name="hexFlow">熱交換器流量[kg/s]</param>
285     /// <param name="rtnTmp">還冷水温度[C]（ポンプ昇温込み）</param>
286     private void calcHexHeatTransfer(double hexFlow, double rtnTmp)
287     {
288       //水槽内の流れ方向を確定
289       double ttlChilFlow = chiller.MaxChilledWaterFlowRate * ChillerNumber;
290       bool isDownFlow = ttlChilFlow < hexFlow;
291
292       //ターボ冷凍機供給温度を仮定
293       double chilOut = StorageTemperature;
294       double chilIn = dtChillingPump;
295       if (isDownFlow)
296       {
297         double tHexIn = (WaterTank.LowerOutletTemperarture * (hexFlow - ttlChilFlow)
298           + chilOut * ttlChilFlow) / hexFlow;
299         pHex.Update(tHexIn, rtnTmp, hexFlow, ChilledWaterFlowRate);
300         disPump.UpdateState(pHex.HeatSourceFlowRate);
301         double dtDisP = disPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * pHex.HeatSourceFlowRate);
302         chilIn += pHex.HeatSourceOutletTemperature + dtDisP;
303       }
304       else
305       {
306         pHex.Update(chilOut, rtnTmp, hexFlow, ChilledWaterFlowRate);
307         disPump.UpdateState(pHex.HeatSourceFlowRate);
308         double dtDisP = disPump.GetElectricConsumption() / (WATER_SPECIFIC_HEAT * pHex.HeatSourceFlowRate);
309         chilIn += ((pHex.HeatSourceOutletTemperature + dtDisP) * hexFlow
310           + wTank.UpperOutletTemperarture * (ttlChilFlow - hexFlow)) / ttlChilFlow;
311       }
312
313       //冷却塔と冷凍機の連成計算
314       calcChillerAndCoolingTower(chilIn);
315
316       //ターボ冷凍機過負荷（冷水出口温度が上昇）の場合には 1 回だけ HEX 交換熱量を補正
317       if (chiller.IsOverLoad)
318       {
319         if (isDownFlow)
320         {
321           double tHexIn = (WaterTank.LowerOutletTemperarture * (hexFlow - ttlChilFlow)
322             + chiller.ChilledWaterOutletTemperature * ttlChilFlow) / hexFlow;
323           pHex.Update(tHexIn, rtnTmp, hexFlow, ChilledWaterFlowRate);
324         }
325         else
326           pHex.Update(chiller.ChilledWaterOutletTemperature, rtnTmp, hexFlow, ChilledWaterFlowRate);
327       }
328     }
329
330     /// <summary>ターボ冷凍機と冷却塔を連成計算する</summary>
331     /// <param name="inletChilledWaterTemp">冷水還温度[C]</param>
332     private void calcChillerAndCoolingTower(double inletChilledWaterTemp)
333     {
334       //冷却塔と冷凍機の連成計算（冷却水温度の計算）
335       double mch = 1000 * chgPump.DesignFlowRate;
336       double mcd = 1000 * cdwPump.DesignFlowRate;
337       bool needIteration = !ControlCoolingWaterTemperature;
338       if (ControlCoolingWaterTemperature)
339       {
340         chiller.Update(CoolingWaterTemperatureSetpoint + dtCoolingPump, inletChilledWaterTemp, mcd, mch);
341         cTower.OutletWaterSetPointTemperature = CoolingWaterTemperatureSetpoint;
342         cTower.Update(chiller.CoolingWaterOutletTemperature, true);
343         if (cTower.IsOverLoad) needIteration = true;
344       }
```

```
345       //過負荷または冷却水温度成行の場合には収束計算
346       if (needIteration)
347       {
348         Roots.ErrorFunction eFnc = delegate (double cdt)
349         {
350           chiller.Update(cdt + dtCoolingPump, inletChilledWaterTemp, mcd, mch);
351           cTower.Update(chiller.CoolingWaterOutletTemperature, cTower.MaxAirFlowRate);
352           return cTower.OutletWaterTemperature - cdt;
353         };
354         double fmax = eFnc(OutdoorAir.WetbulbTemperature);
355         double fmin = eFnc(37);
356         if (0 <= fmin && fmax <= 0)
357           Roots.Bisection(eFnc, 37, OutdoorAir.WetbulbTemperature, fmin, fmax, 0.01, 0.01, 10);
358       }
359     }
360
361     /// <summary>状態を確定する</summary>
362     public void FixState() { isForecasting = false; }
363
364     /// <summary>インスタンスを初期化する</summary>
365     /// <param name="waterTank">温度成層型蓄熱槽</param>
366     /// <param name="plateHex">プレート熱交換器</param>
367     /// <param name="chiller">ターボ冷凍機</param>
368     /// <param name="chwPump">冷水ポンプ</param>
369     /// <param name="cdwPump">冷却水ポンプ</param>
370     /// <param name="chargePump">蓄熱ポンプ</param>
371     /// <param name="dischargePump">放熱ポンプ</param>
372     /// <param name="cTower">冷却塔</param>
373     /// <param name="chillerNumber">冷凍機台数</param>
374     /// <param name="coolingTowerNumber">冷却塔台数（1台の冷凍機あたりの台数）</param>
375     public MultipleStratifiedWaterTankSystem
376       (MultipleStratifiedWaterTank waterTank, PlateHeatExchanger plateHex, ICentrifugalChiller chiller,
377       CentrifugalPump chwPump, CentrifugalPump cdwPump, CentrifugalPump chargePump,
378       CentrifugalPump dischargePump, CoolingTower cTower, int chillerNumber, int coolingTowerNumber)
379     {
380       this.chiller = chiller;
381       this.chwPump = chwPump;
382       this.cdwPump = cdwPump;
383       this.chgPump = chargePump;
384       this.disPump = dischargePump;
385       this.cTower = cTower;
386       this.pHex = plateHex;
387       this.wTank = waterTank;
388       this.ChillerNumber = chillerNumber;
389       this.CoolingTowerNumber = coolingTowerNumber;
390       oldTemps = new double[WaterTank.LayerNumber];
391
392       //冷凍機ポンプの昇温幅を計算・保存
393       chgPump.UpdateState(chgPump.DesignFlowRate);
394       dtChillingPump = chgPump.GetElectricConsumption()
395         / (WATER_SPECIFIC_HEAT * 1000 * chgPump.DesignFlowRate);
396       cdwPump.UpdateState(cdwPump.DesignFlowRate);
397       dtCoolingPump = cdwPump.GetElectricConsumption()
398         / (WATER_SPECIFIC_HEAT * 1000 * cdwPump.DesignFlowRate);
399
400       Mode = HeatSourceSystemModel.OperatingMode.ShutOff;
401     }
402 }
```

21.3.3　熱源設備システムクラスの作成

21.3.1 節と 21.3.2 節で開発したサブシステムを組み合わせた問題を解く、熱源設備システムクラスを作成する。熱源設備クラスの列挙型、インスタンス変数、プロパティ、コンストラクタの定義をプログラム 21.7 に示す。1~13 行は運転モードの列挙型であり、既に熱源サブシステムクラスでも使用した。複数の熱源設備サブシステムで構成されたシステムを計算対象とするが、16 行に示すように IHeatSourceSubSystem 型のインスタンスとして取り扱い、個別の熱源設備サブシステムの具体的な実装には依存させない構造にする点に注意する。19 行は二次ポンプであるが、1 ポンプシステムの場合には null にする。95~102 行のコンストラクタに示すように、1 ポンプシステムとするためにポンプを引数に取らないインスタンスを用意する。22 行は運転順位であり、それぞれの熱源設備サブシステムが何段目に起動するのかを保持する。21.2.1 節で記したように二次側の負荷流量が下限流量未満の

場合にはバイパスが生じるため、この水量を 37 行と 55 行で保持する。

プログラム **21.7**　熱源設備システムクラスの列挙型、インスタンス変数、プロパティ、コンストラクタの定義

Popolo.HVAC.SubSystem.HeatSourceSystemModel class

```
/// <summary>運転モード</summary>
[Flags]
public enum OperatingMode
{
  /// <summary>停止</summary>
  ShutOff = 0,
  /// <summary>加熱運転</summary>
  Heating = 1,
  /// <summary>冷却運転</summary>
  Cooling = 2,
  /// <summary>加熱・冷却運転</summary>
  HeatingAndCooling = 4
}

/// <summary>サブシステム</summary>
private IHeatSourceSubSystem[] subSystems;

/// <summary>2 次ポンプ</summary>
private PumpSystem chilledWaterPumps, hotWaterPumps;

/// <summary>運転順位</summary>
private int[] opRankC, opRankH;

/// <summary>冷水往温度設定値を設定・取得する</summary>
public double ChilledWaterSupplyTemperatureSetpoint { get; set; } = 7;

/// <summary>冷水往温度[C]を取得する</summary>
public double ChilledWaterSupplyTemperature { get; private set; } = 7;

/// <summary>冷水還温度[C]を取得する</summary>
public double ChilledWaterReturnTemperature { get; private set; } = 12;

/// <summary>冷水流量[kg/s]を取得する</summary>
public double ChilledWaterFlowRate { get; private set; }

/// <summary>冷水バイパス流量[kg/s]を取得する</summary>
public double ChilledWaterBypassFlowRate { get; private set; }

/// <summary>冷水供給が過負荷か否か</summary>
public bool IsOverLoad_C { get; private set; }

/// <summary>温水往温度設定値を設定・取得する</summary>
public double HotWaterSupplyTemperatureSetpoint { get; set; } = 45;

/// <summary>温水往温度[C]を取得する</summary>
public double HotWaterSupplyTemperature { get; private set; } = 45;

/// <summary>温水還温度[C]を取得する</summary>
public double HotWaterReturnTemperature { get; private set; } = 40;

/// <summary>温水流量[kg/s]を取得する</summary>
public double HotWaterFlowRate { get; private set; }

/// <summary>温水バイパス流量[kg/s]を取得する</summary>
public double HotWaterBypassFlowRate { get; private set; }

/// <summary>温水供給が過負荷か否か</summary>
public bool IsOverLoad_H { get; private set; }

/// <summary>2 次ポンプ方式か否か</summary>
public bool IsSecondaryPumpSystem { get { return chilledWaterPumps != null; } }

/// <summary>冷水二次ポンプシステムを取得する</summary>
public ImmutablePumpSystem ChilledWaterPumpSystem { get { return chilledWaterPumps; } }

/// <summary>温水二次ポンプシステムを取得する</summary>
public ImmutablePumpSystem HotWaterPumpSystem { get { return hotWaterPumps; } }

/// <summary>外気条件を設定・取得する</summary>
public ImmutableMoistAir OutdoorAir { get; set; } = new MoistAir(35, 0.0195);

/// <summary>現在の日時を設定・取得する</summary>
public DateTime CurrentDateTime { get; set; }

/// <summary>計算時間間隔[sec]を設定・取得する</summary>
```

```
 76 public double TimeStep { get; set; } = 3600;
 77
 78 /// <summary>配管熱損失率[-]を設定・取得する</summary>
 79 public double PipeHeatLossRate { get; set; } = 0.08;
 80
 81 /// <summary>インスタンスを初期化する（2ポンプシステム）</summary>
 82 /// <param name="subSystems">熱源サブシステムリスト</param>
 83 /// <param name="chilledWaterPumps">冷水二次ポンプ</param>
 84 /// <param name="hotWaterPumps">温水二次ポンプ</param>
 85 public HeatSourceSystemModel
 86   (IHeatSourceSubSystem[] subSystems, PumpSystem chilledWaterPumps, PumpSystem hotWaterPumps)
 87 {
 88   this.subSystems = subSystems;
 89   this.chilledWaterPumps = chilledWaterPumps;
 90   this.hotWaterPumps = hotWaterPumps;
 91   opRankC = new int[subSystems.Length];
 92   opRankH = new int[subSystems.Length];
 93 }
 94
 95 /// <summary>インスタンスを初期化する（1ポンプシステム）</summary>
 96 /// <param name="subSystems">熱源サブシステムリスト</param>
 97 public HeatSourceSystemModel(IHeatSourceSubSystem[] subSystems)
 98 {
 99   this.subSystems = subSystems;
100   opRankC = new int[subSystems.Length];
101   opRankH = new int[subSystems.Length];
102 }
```

段数と運転モードの設定および確認処理をプログラム 21.8 に示す。1~14 行は段数と運転モードの設定である。16~27 行は運転モードの確認処理であり、熱源設備サブシステムが指定の運転モードに合致するか否かを確認する。

プログラム 21.8　段数と運転モードの設定および確認処理

Popolo.HVAC.SubSystem.HeatSourceSystemModel class

```
 1 /// <summary>冷却運転の段数を設定する</summary>
 2 /// <param name="subSystemIndex">サブシステム番号</param>
 3 /// <param name="stage">段数 (1, 2, 3...) </param>
 4 public void SetChillingOperationSequence(int subSystemIndex, int stage) { opRankC[subSystemIndex] = stage; }
 5
 6 /// <summary>暖房運転の段数を設定する</summary>
 7 /// <param name="subSystemIndex">サブシステム番号</param>
 8 /// <param name="stage">段数 (1, 2, 3...) </param>
 9 public void SetHeatingOperationSequence(int subSystemIndex, int stage) { opRankH[subSystemIndex] = stage; }
10
11 /// <summary>運転モードを設定する</summary>
12 /// <param name="subSystemIndex">サブシステム番号</param>
13 /// <param name="mode">運転モード</param>
14 public void SetOperatingMode(int subSystemIndex, OperatingMode mode){subSystems[subSystemIndex].Mode = mode;}
15
16 /// <summary>運転モードに合致するか否か</summary>
17 /// <param name="subsystemNumber">サブシステム番号</param>
18 /// <param name="cooling">冷却運転か否か</param>
19 /// <returns>運転モードに合致するか否か</returns>
20 private bool isModeAdapts(int subsystemNumber, bool cooling)
21 {
22   IHeatSourceSubSystem ss = subSystems[subsystemNumber];
23   if (ss.Mode == OperatingMode.ShutOff) return false;
24   if (cooling && ss.Mode == OperatingMode.Heating) return false;
25   if (!cooling && ss.Mode == OperatingMode.Cooling) return false;
26   return true;
27 }
```

将来状態の予測・確定処理をプログラム 21.9 に示す。1, 2 行は状態確定処理であり、すべての熱源設備サブシステムインスタンスの FixState メソッドを呼び出す。

4~175 行は将来状態の予測処理である。13~28 行で引数をプロパティなどに設定する。二次ポンプがある場合には 30~49 行でポンプによる昇温を計算する。現実には還温度ではなく往温度に対して温度上昇が生じるが、これを厳密に取り扱うと二次側へ渡す温度が変動するために二次側システムとの間で収束計算が必須になってしまう。そこで、反復計算を避けるために還温度に昇温分を加算して熱量のみを合致させる方針とする。51~74 行は運転段数の初期化処理である。負荷流量を上回るまで増

段を重ねる。76~144 行は負荷熱量による増段処理である。それぞれの熱源設備サブシステムには均等に負荷流量が割り振られる前提とするため、84~93 行で最も大きい最小絞り流量比を求め、この流量比を上回るようにサブシステムの循環流量を設定する。95~114 行はバイパス流量の有無の判定と、バイパスがある場合の還温度の補正処理である。16~129 行でそれぞれの熱源設備サブシステムの状態を更新して過負荷判定を行い、1 系統でも過負荷となる系統がある場合には 116~143 行で増段する。129 行に示すように、過負荷系統が無くなった場合、あるいは最大段数に到達した場合にはループを抜ける。最大段数に達して過負荷の場合には冷水または温水の出口温度が設定値とならないため、148~174 行で補正を行う。

プログラム 21.9　将来状態の予測・確定処理

Popolo.HVAC.SubSystem.HeatSourceSystemModel class

```
 1 /// <summary>状態を確定する</summary>
 2 public void FixState() { foreach (IHeatSourceSubSystem hs in subSystems) hs.FixState(); }
 3
 4 /// <summary>将来の状態を予測する</summary>
 5 /// <param name="chilledWaterFlowRate">冷水流量[kg/s]</param>
 6 /// <param name="chilledWaterReturnTemperature">冷水還温度[C]</param>
 7 /// <param name="hotWaterFlowRate">温水流量[kg/s]</param>
 8 /// <param name="hotWaterReturnTemperature">温水還温度[C]</param>
 9 public void ForecastSupplyWaterTemperature
10   (double chilledWaterFlowRate, double chilledWaterReturnTemperature,
11   double hotWaterFlowRate, double hotWaterReturnTemperature)
12 {
13   ChilledWaterFlowRate = chilledWaterFlowRate;
14   HotWaterFlowRate = hotWaterFlowRate;
15   ChilledWaterReturnTemperature = chilledWaterReturnTemperature
16     - (ChilledWaterSupplyTemperatureSetpoint - chilledWaterReturnTemperature) * PipeHeatLossRate;
17   HotWaterReturnTemperature = hotWaterReturnTemperature
18     - (HotWaterSupplyTemperatureSetpoint - hotWaterReturnTemperature) * PipeHeatLossRate;
19
20   //日時・冷温水往温度・外気条件を設定
21   for (int i = 0; i < subSystems.Length; i++)
22   {
23     subSystems[i].TimeStep = TimeStep;
24     subSystems[i].CurrentDateTime = CurrentDateTime;
25     subSystems[i].OutdoorAir = OutdoorAir;
26     subSystems[i].ChilledWaterSupplyTemperatureSetpoint = ChilledWaterSupplyTemperatureSetpoint;
27     subSystems[i].HotWaterSupplyTemperatureSetpoint = HotWaterSupplyTemperatureSetpoint;
28   }
29
30   //2次ポンプによる昇温を反映
31   double tcwi = chilledWaterReturnTemperature;
32   double thwi = hotWaterReturnTemperature;
33   if (IsSecondaryPumpSystem)
34   {
35     if (chilledWaterFlowRate != 0)
36     {
37       chilledWaterPumps.TotalFlowRate = 0.001 * chilledWaterFlowRate;
38       chilledWaterPumps.UpdateState();
39       tcwi += chilledWaterPumps.GetElectricConsumption() / (chilledWaterFlowRate * WATER_SPECIFIC_HEAT);
40     }
41     else chilledWaterPumps.ShutOff();
42     if (hotWaterFlowRate != 0)
43     {
44       hotWaterPumps.TotalFlowRate = 0.001 * hotWaterFlowRate;
45       hotWaterPumps.UpdateState();
46       thwi += hotWaterPumps.GetElectricConsumption() / (hotWaterFlowRate * WATER_SPECIFIC_HEAT);
47     }
48     else hotWaterPumps.ShutOff();
49   }
50
51   //負荷流量に応じて最低運転台数を計算
52   int cStage, hStage;
53   double fcsum = 0; //冷却運転
54   if (chilledWaterFlowRate <= 0) cStage = 0;
55   else
56   {
57     for (cStage = 1; cStage <= opRankC.Length; cStage++)
58     {
59       for (int j = 0; j < subSystems.Length; j++)
```

```
60          if (opRankC[j] == cStage && isModeAdapts(j, true)) fcsum += subSystems[j].MaxChilledWaterFlowRate;
61        if (chilledWaterFlowRate < fcsum) break;
62      }
63    }
64    double fhsum = 0; //加熱運転
65    if (hotWaterFlowRate <= 0) hStage = 0;
66    else
67    {
68      for (hStage = 1; hStage <= opRankH.Length; hStage++)
69      {
70        for (int j = 0; j < subSystems.Length; j++)
71          if (opRankH[j] == hStage && isModeAdapts(j, false)) fhsum += subSystems[j].MaxHotWaterFlowRate;
72        if (hotWaterFlowRate < fhsum) break;
73      }
74    }
75
76    //過負荷系統が無くなるまで増段
77    double bypsC = 0;
78    double bypsH = 0;
79    while (true)
80    {
81      IsOverLoad_C = false;
82      IsOverLoad_H = false;
83
84      //最大絞り流量比を計算
85      double minMaxC = 0.0;
86      double minMaxH = 0.0;
87      for (int i = 0; i < subSystems.Length; i++)
88      {
89        if (opRankC[i] <= cStage && isModeAdapts(i, true))
90          minMaxC = Math.Max(minMaxC, subSystems[i].MinChilledWaterFlowRatio);
91        if (opRankH[i] <= hStage && isModeAdapts(i, false))
92          minMaxH = Math.Max(minMaxH, subSystems[i].MinHotWaterFlowRatio);
93      }
94
95      //バイパス流量を考慮して熱源入口水温を計算
96      double cpl, hpl;
97      if (chilledWaterFlowRate == 0 || fcsum == 0) cpl = 0;
98      else
99      {
100       cpl = chilledWaterFlowRate / fcsum;
101       bypsC = Math.Max(0, minMaxC - cpl);
102       double tcwi2 = (tcwi * cpl + ChilledWaterSupplyTemperatureSetpoint * bypsC) / (cpl + bypsC);
103       foreach (IHeatSourceSubSystem hss in subSystems) hss.ChilledWaterReturnTemperature = tcwi2;
104       cpl = Math.Max(minMaxC, cpl);
105     }
106     if (hotWaterFlowRate == 0 || fhsum == 0) hpl = 0;
107     else
108     {
109       hpl = hotWaterFlowRate / fhsum;
110       bypsH = Math.Max(0, minMaxH - hpl);
111       double thwi2 = (thwi * hpl + HotWaterSupplyTemperatureSetpoint * bypsH) / (hpl + bypsH);
112       foreach (IHeatSourceSubSystem hss in subSystems) hss.HotWaterReturnTemperature = thwi2;
113       hpl = Math.Max(minMaxH, hpl);
114     }
115
116     //過負荷系統が無いか確認
117     for (int i = 0; i < subSystems.Length; i++)
118     {
119       double cf, hf;
120       cf = hf = 0;
121       if (opRankC[i] <= cStage && isModeAdapts(i, true)) cf = subSystems[i].MaxChilledWaterFlowRate * cpl;
122       if (opRankH[i] <= hStage && isModeAdapts(i, false)) hf = subSystems[i].MaxHotWaterFlowRate * hpl;
123       subSystems[i].ForecastSupplyWaterTemperature(cf, hf);
124
125       if (subSystems[i].IsOverLoad_C) IsOverLoad_C = true;
126       if (subSystems[i].IsOverLoad_H) IsOverLoad_H = true;
127     }
128     int maxStage = opRankC.Length;
129     if ((!IsOverLoad_C || maxStage <= cStage) && (!IsOverLoad_H || maxStage <= hStage)) break;
130
131     //増段処理
132     if (IsOverLoad_C && cStage != maxStage)
133     {
134       cStage++;
135       for (int j = 0; j < subSystems.Length; j++)
136         if (opRankC[j] == cStage && isModeAdapts(j, true)) fcsum += subSystems[j].MaxChilledWaterFlowRate;
137     }
138     if (IsOverLoad_H && hStage != maxStage)
139     {
```

```
140       hStage++;
141       for (int j = 0; j < subSystems.Length; j++)
142         if (opRankH[j] == hStage && isModeAdapts(j, false)) fhsum += subSystems[j].MaxHotWaterFlowRate;
143     }
144   }
145   ChilledWaterBypassFlowRate = bypsC * fcsum;
146   HotWaterBypassFlowRate = bypsH * fhsum;
147
148   //冷水出口温度を計算
149   if (IsOverLoad_C)
150   {
151     ChilledWaterSupplyTemperature = 0;
152     double cwSum = 0;
153     foreach (IHeatSourceSubSystem ss in subSystems)
154     {
155       ChilledWaterSupplyTemperature += ss.ChilledWaterSupplyTemperature * ss.MaxChilledWaterFlowRate;
156       cwSum += ss.MaxChilledWaterFlowRate;
157     }
158     ChilledWaterSupplyTemperature /= cwSum;
159   }
160   else ChilledWaterSupplyTemperature = ChilledWaterSupplyTemperatureSetpoint;
161
162   //温水出口温度を計算
163   if (IsOverLoad_H)
164   {
165     HotWaterSupplyTemperature = 0;
166     double hwSum = 0;
167     foreach (IHeatSourceSubSystem ss in subSystems)
168     {
169       HotWaterSupplyTemperature += ss.HotWaterSupplyTemperature * ss.MaxHotWaterFlowRate;
170       hwSum += ss.MaxHotWaterFlowRate;
171     }
172     HotWaterSupplyTemperature /= hwSum;
173   }
174   else HotWaterSupplyTemperature = HotWaterSupplyTemperatureSetpoint;
175 }
```

【例題 21.2】

図 21.9 に示す熱源設備システムを作成し、負荷率とエネルギー消費量の関係を計算せよ。ただし、システムを構成する機器の仕様は下表の通りである。冷却運転、暖房運転ともに、1 段目に空気熱源ヒートポンプ系統を起動し、2 段目に吸収式系統を起動するものとする。

表 21.6　機器の仕様一覧

名称		仕様			
吸収式系統	直焚吸収冷温水機	冷却能力　：527 kW 冷水温度条件　：12.0→ 7.0 ℃ 冷却水温度条件：32.0→37.0 ℃ 加熱能力　：346 kW 温水温度条件　：51.7→55.0 ℃ 補機消費電力　：5.1 kW （冷暖共通）		ガス消費量：32.4 m³/h 冷水流量　：1,512 L/min 冷却水流量：2,500 L/min ガス消費量：31.8 m³/h 温水流量　：1,512 L/min	
	冷却塔	冷却能力　：939 kW 冷却水温度条件：37.0→32.0 ℃ 外気湿球温度　：27.0 ℃		ファン動力：7.5 kW×1 台 冷却水流量：2,693 L/min	
	冷水 1 次ポンプ	流量：1,512 L/min	定格揚程：150 kPa	設計揚程：140 kPa	
	温水 1 次ポンプ	流量：1,512 L/min	定格揚程：150 kPa	設計揚程：140 kPa	
	冷却水ポンプ	流量　：2,693 L/min 設計揚程：240 kPa		定格揚程：250 kPa 実揚程：50kPa	
空気熱源 HP 系統	空気熱源ヒートポンプ	モジュール数　：2 セット 冷却能力　：150 kW 冷水温度条件　：12.0→ 7.0 ℃ 加熱能力　：150 kW 温水温度条件　：50.0→55.0 ℃ 補機消費電力　：1.9 kW （冷暖共通）		消費電力　：49.8 kW 冷水流量　：430 L/min 消費電力　：50.0 kW 温水流量　：430 L/min 風量　：850 m³/min(冷暖共通)	
	冷水 1 次ポンプ	流量：860 L/min	定格揚程：150 kPa	設計揚程：140 kPa	台数：2 台
	温水 1 次ポンプ	流量：860 L/min	定格揚程：150 kPa	設計揚程：140 kPa	台数：2 台
冷水 2 次ポンプ、温水 2 次ポンプ		流量　：1,077 L/min 実揚程：100kPa	定格揚程：250 kPa 台数　：3 台×2（冷温系統）	設計揚程：240 kPa	

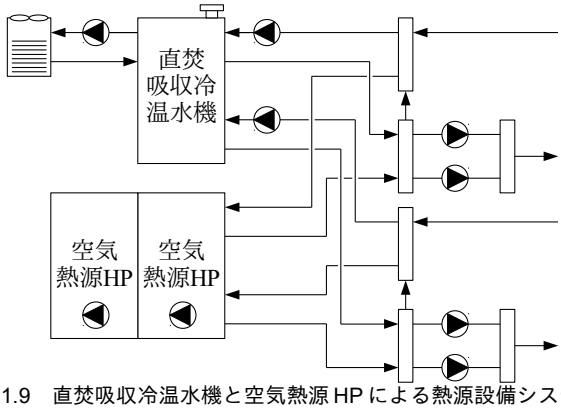


図 21.9　直焚吸収冷温水機と空気熱源 HP による熱源設備システム

【解】

熱源システムの作成処理をプログラム 21.10 に示す。詳細は第 28 章で述べるが、本システムは空気調和・衛生工学会の「建物のエネルギーシミュレーションの評価手法に関するガイドライン[21.9)]」で計算対象となっている建物で採用されているシステムである。本書では第 29 章で二次側システムとの連成計算を行う。このため、本建物のモデル構築関連の処理を SHASE_SimulationTest クラスにまとめておき、再利用する。第 29 章で感度解析を行うことに備えて USE_VWV_SYSTEM と IS_HIGHEFF_HSOURCE という 2 つのフラグを用いてモデルを作成する。前者は VWV 制御の有無（4~6 行）、後者は高効率熱源採用の真偽（11~16 行）である。8~22 行は空気熱源ヒートポンプによるサブシステム、24~37 行は直焚吸収冷温水機によるサブシステムの作成処理である。39~43 行で二次ポンプを追加し、44 行で熱源システムを作成する。45~50 行は冷温水の出口温度と熱源運転順位の設定である。

プログラム 21.10　熱源設備システムの作成処理

SHASE_SimulationTest class

```
 1 public static void MakeHeatSourceSystem(out HeatSourceSystemModel hss,
 2   out AirHeatSourceModularChillersSystem ahSystem, out DirectFiredAbsorptionChillerSystem arSystem)
 3 {
 4   CentrifugalPump.ControlMethod P1_CTRL, P2_CTRL;
 5   if(USE_VWV_SYSTEM) P1_CTRL = P2_CTRL = CentrifugalPump.ControlMethod.ConstantPressureWithInverter;
 6   else P1_CTRL = P2_CTRL = CentrifugalPump.ControlMethod.ConstantPressureWithBypass;
 7
 8   //空気熱源ヒートポンプシステムの作成
 9   double mf = 430d / 60;  //冷温水質量流量[kg/s]
10   double COP_C, COP_H;
11   if (IS_HIGHEFF_HSOURCE)
12   {
13     COP_C = 4.6;
14     COP_H = 3.5;
15   }
16   else COP_C = COP_H = 3.0;
17   AirHeatSourceModularChillers ahpChiler = new AirHeatSourceModularChillers
18     (150, 7, mf, 35, 850d / 60 * 1.2, 150 / COP_C, 150, 45, mf, 7, 850d / 60 * 1.2, 150 / COP_H, 2, 1.9);
19   ahpChiler.MaximizeEfficiency = IS_HIGHEFF_HSOURCE;
20   CentrifugalPump chwPump = new CentrifugalPump(150, 0.001 * mf, 140, 0.001 * mf, P1_CTRL, 1);
21   CentrifugalPump hwPump = new CentrifugalPump(150, 0.001 * mf, 140, 0.001 * mf, P1_CTRL, 1);
22   ahSystem = new AirHeatSourceModularChillersSystem(ahpChiler, chwPump, hwPump, 2);
23
24   //吸収冷温水機システムの作成
25   mf = 1512d / 60;  //冷温水質量流量[kg/s]
26   DirectFiredAbsorptionChiller arChiller = new DirectFiredAbsorptionChiller
27     (32.4d / 3600, 31.8d / 3600, 12, 7, 32, 37, 51.7, 55, mf, 2500d / 60, mf, 5.1, Boiler.Fuel.Gas13A);
28   arChiller.HasSolutionInverterPump = IS_HIGHEFF_HSOURCE;
29   CoolingTower cTower = new CoolingTower
30     (37, 32, 27, 2693d / 60, 1783d / 60 * 1.2, CoolingTower.AirFlowDirection.CrossFlow, 7.5 / 0.92, false);
31   chwPump = new CentrifugalPump(150, 0.001 * mf, 140, 0.001 * mf, P1_CTRL, 1);
32   hwPump = new CentrifugalPump(150, 0.001 * mf, 140, 0.001 * mf, P1_CTRL, 1);
33   CentrifugalPump cdwPump = new CentrifugalPump
34     (250, 2.693 / 60, 240, 2.693 / 60, CentrifugalPump.ControlMethod.MinimumPressure, 50);
35   arSystem = new DirectFiredAbsorptionChillerSystem(arChiller, chwPump, hwPump, cdwPump, cTower, 1, 1);
36   arSystem.ControlCoolingWaterFlowRate = true;
37   arSystem.ControlCoolingWaterTemperature = true;
38
39   //熱源サブシステムの作成
40   CentrifugalPump cp2 = new CentrifugalPump(250, 1.077 / 60, 240, 1.077 / 60, P2_CTRL, 1);
41   PumpSystem cp2system = new PumpSystem(cp2, 240, 1.077 * 3 / 60, 100, 3);
42   CentrifugalPump hp2 = new CentrifugalPump(250, 1.077 / 60, 240, 1.077 / 60, P2_CTRL, 1);
```

```
43   PumpSystem hp2system = new PumpSystem(hp2, 240, 1.077 * 3 / 60, 100, 3);
44   hss = new HeatSourceSystemModel(new IHeatSourceSubSystem[] { ahSystem, arSystem }, cp2system, hp2system);
45   hss.ChilledWaterSupplyTemperatureSetpoint = 7;
46   hss.HotWaterSupplyTemperatureSetpoint = 45;
47   hss.SetChillingOperationSequence(0, 1);
48   hss.SetChillingOperationSequence(1, 2);
49   hss.SetHeatingOperationSequence(0, 1);
50   hss.SetHeatingOperationSequence(1, 2);
51 }
```

負荷率変更による感度解析をプログラム 21.11 に示す。3~8 行でプログラム 21.10 を使って熱源システムを作成する。17~35 行は冷却運転に関する感度解析であり、最大能力に対して 10%刻みで負荷率を低下（23 行）させてエネルギー消費量を計算する。37~56 行は加熱運転に関する感度解析であり、処理の内容は冷却運転と同様である。

計算結果をグラフ化すると図 21.10 が得られる。電力とガスの一次エネルギー換算係数をそれぞれ 9.76 MJ/kWh と 45MJ/Nm³ として換算を行い、一次エネルギー消費量と一次 COP も記載した。直焚吸収冷温水機はインバータ仕様としたため、負荷率の低下にともなって COP が上昇する様子が確認できる。また、負荷率が 50 %を下回ると空気熱源ヒートポンプ単独による運転が可能となり、さらに COP が向上する。

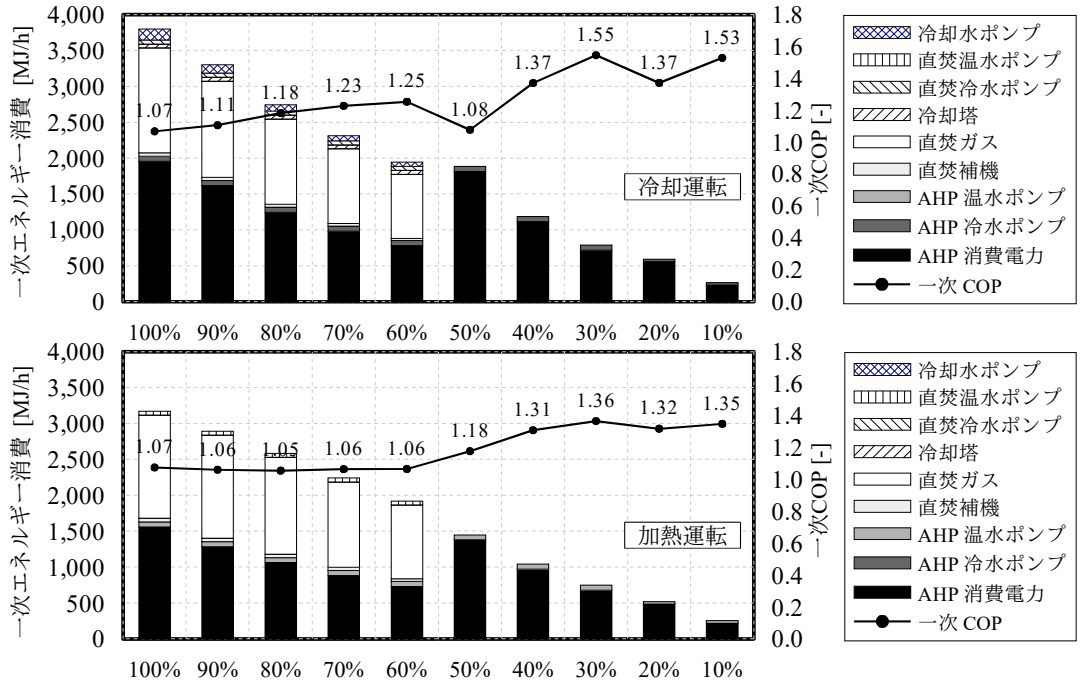


図 21.10　負荷率とエネルギー消費量の関係

プログラム **21.11**　負荷率についての感度解析

```
 1 private static void HeatSourceSubsystemTest1()
 2 {
 3   AirHeatSourceModularChillersSystem ahSystem;
 4   DirectFiredAbsorptionChillerSystem arSystem;
 5   HeatSourceSystemModel hsSystem;
 6   SHASE_SimulationTest.MakeHeatSourceSystem(out hsSystem, out ahSystem, out arSystem);
 7   ImmutableAirHeatSourceModularChillers ahpChiller = ahSystem.AirHeatSourceModularChillers;
 8   ImmutableDirectFiredAbsorptionChiller arChiller = arSystem.DirectFiredAbsorptionChiller;
 9
10   using (StreamWriter sWriter = new StreamWriter
11     ("HeatSourceSubsystemTest1.csv", false, Encoding.GetEncoding("Shift_JIS")))
12   {
13     //タイトル行
14     sWriter.WriteLine("時刻,負荷,AHP 電力,AHP 冷水ポンプ電力,AHP 温水ポンプ電力,直焚電力,直焚ガス,"
15       + "冷却塔電力,直焚冷水ポンプ電力,直焚温水ポンプ電力,冷却水ポンプ電力,冷水往温度,温水往温度");
16
17     sWriter.WriteLine("冷却運転テスト");
18     hsSystem.OutdoorAir = new MoistAir(34.4, 0.0194);
19     hsSystem.SetOperatingMode(0, HeatSourceSystemModel.OperatingMode.Cooling);
20     hsSystem.SetOperatingMode(1, HeatSourceSystemModel.OperatingMode.Cooling);
```

```
21      for (int i = 0; i <= 10; i++)
22      {
23        double load = (527 + 300 + 300) * (0.1 * (10 - i));
24        hsSystem.ForecastSupplyWaterTemperature(load / (5 * 4.186), 12, 0, 40);
25        int ahpNum = ahSystem.OperatingChillerNumber;
26        sWriter.WriteLine(load +
27        ", " + (ahpChiller.ElectricConsumption * ahpChiller.OperatingNumber * ahpNum) +
28        ", " + (ahSystem.ChilledWaterPump.GetElectricConsumption() * ahpNum) +
29        ", " + (ahSystem.HotWaterPump.GetElectricConsumption() * ahpNum) +
30        ", " + arChiller.ElectricConsumption + ", " + (arChiller.FuelConsumption * 3600) +
31        ", " + arSystem.CoolingTower.ElectricConsumption +
32        ", " + arSystem.ChilledWaterPump.GetElectricConsumption() +
33        ", " + arSystem.HotWaterPump.GetElectricConsumption() +
34        ", " + arSystem.CoolingWaterPump.GetElectricConsumption());
35      }
36
37      sWriter.WriteLine("加熱運転テスト");
38      hsSystem.OutdoorAir = new MoistAir(2.0, 0.0014);
39      hsSystem.SetOperatingMode(0, HeatSourceSystemModel.OperatingMode.Heating);
40      hsSystem.SetOperatingMode(1, HeatSourceSystemModel.OperatingMode.Heating);
41      for (int i = 0; i <= 10; i++)
42      {
43        double load = (346 + 300 + 300) * (0.1 * (10 - i));
44        hsSystem.ForecastSupplyWaterTemperature(0, 12, load / (5 * 4.186), 40);
45        int ahpNum = ahSystem.OperatingChillerNumber;
46        sWriter.WriteLine(load +
47        ", " + (ahpChiller.ElectricConsumption * ahpChiller.OperatingNumber * ahpNum) +
48        ", " + (ahSystem.ChilledWaterPump.GetElectricConsumption() * ahpNum) +
49        ", " + (ahSystem.HotWaterPump.GetElectricConsumption() * ahpNum) +
50        ", " + arChiller.ElectricConsumption + ", " + (arChiller.FuelConsumption * 3600) +
51        ", " + arSystem.CoolingTower.ElectricConsumption +
52        ", " + arSystem.ChilledWaterPump.GetElectricConsumption() +
53        ", " + arSystem.HotWaterPump.GetElectricConsumption() +
54        ", " + arSystem.CoolingWaterPump.GetElectricConsumption());
55      }
56    }
57  }
```

【例題 21.3】

例題 21.2 の熱源設備システムに関して、表 21.4 の年間冷暖房負荷を用いて各月各時刻のエネルギー消費量を計算せよ。ただし外気温湿度は月平均の値を用いることにし表 21.7 に示す通りである。

表 21.7　各月の平均温湿度

	1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月
乾球温度 [ºC]	6.3	5.9	10.2	14.8	20.5	21.6	26.1	27.0	21.8	18.2	12.8	8.3
相対湿度 [%]	52.8	48.6	62.0	57.9	67.0	73.4	75.2	72.6	70.3	63.6	58.9	54.1

【解】

計算処理をプログラム 21.12 に示す。

3~10 行は例題 21.1 で用いた熱負荷原単位である。12~17 行でプログラム 21.10 を用いて熱源設備システムを作成する。29~35 行は運転モード設定であり、月に応じて加熱運転と冷却運転を切り替える。44~53 行で熱負荷原単位を用いて各時刻の熱負荷を計算し、熱源設備システムを更新する。55~65 行は書き出し処理である。出力したデータを各月で集計すると表 21.8 が得られる。年間の一次 COP は 1.45 となる。5, 6, 10, 11 月など、外気条件が良く、空気熱源ヒートポンプ単独での運転が主になる時期の COP が高いことがわかる。表 21.8 をグラフにすると図 21.11 が得られる。

プログラム 21.12　熱源設備システムの年間エネルギー消費計算

```
 1 private static void HeatSourceSubsystemTest2()
 2 {
 3   double[] dLoads = new double[]
 4   { 16336, 14358, 11126, 2690, 4998, 19979, 35228, 39168, 25232, 2894, 5027, 13463 };
 5   double[] sumLRates = new double[] { 0, 0, 0, 0, 0, 0, 0,
 6     1.79, 9.57, 9.17, 8.97, 9.27, 9.37, 9.27, 8.97, 10.69, 8.97, 9.27, 3.89, 0.40, 0.40, 0, 0, 0 };
 7   double[] winLRates = new double[] { 0, 0, 0, 0, 0, 0, 0,
 8     0.30, 16.99, 12.29, 8.09, 10.29, 10.49, 10.29, 8.39, 8.19, 9.09, 5.59, 0, 0, 0, 0, 0, 0, 0 };
 9   double[] dbTemp = new double[] { 6.3, 5.9, 10.2, 14.8, 20.5, 21.6, 26.1, 27.0, 21.8, 18.2, 12.8, 8.3 };
10   double[] rHumid = new double[] { 52.8, 48.6, 62.0, 57.9, 67.0, 73.4, 75.2, 72.6, 70.3, 63.6, 58.9, 54.1 };
11
12   AirHeatSourceModularChillersSystem ahSystem;
13   DirectFiredAbsorptionChillerSystem arSystem;
14   HeatSourceSystemModel hsSystem;
15   SHASE_SimulationTest.MakeHeatSourceSystem(out hsSystem, out ahSystem, out arSystem);
16   ImmutableAirHeatSourceModularChillers ahpChiller = ahSystem.AirHeatSourceModularChillers;
17   ImmutableDirectFiredAbsorptionChiller arChiller = arSystem.DirectFiredAbsorptionChiller;
18
```

```
19   using (StreamWriter sWriter = new StreamWriter
20     ("HeatSourceSubsystemTest2.csv", false, Encoding.GetEncoding("Shift_JIS")))
21   {
22     //タイトル行
23     sWriter.WriteLine("時刻,負荷,AHP 電力,AHP 冷水ポンプ電力,AHP 温水ポンプ電力,直焚電力,直焚ガス,"
24       + "冷却塔電力,直焚冷水ポンプ電力,直焚温水ポンプ電力,冷却水ポンプ電力,冷水往温度,温水往温度");
25
26     for (int i = 0; i < 12; i++)
27     {
28       sWriter.WriteLine((i + 1) + "月");
29       //運転モード設定
30       bool isSummer = (4 <= i && i <= 9);
31       HeatSourceSystemModel.OperatingMode mode;
32       if (isSummer) mode = HeatSourceSystemModel.OperatingMode.Cooling;
33       else mode = HeatSourceSystemModel.OperatingMode.Heating;
34       hsSystem.SetOperatingMode(0, mode);
35       hsSystem.SetOperatingMode(1, mode);
36
37       double aHumid =
38         MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity(dbTemp[i], rHumid[i], 101.325);
39       hsSystem.OutdoorAir = new MoistAir(dbTemp[i], aHumid);
40
41       for (int j = 0; j < 24; j++)
42       {
43         double load;
44         if (isSummer)
45         {
46           load = dLoads[i] * sumLRates[j] / 360d;  // MJ/h→kW に変換
47           hsSystem.ForecastSupplyWaterTemperature(load / (5 * 4.186), 12, 0, 40);
48         }
49         else
50         {
51           load = dLoads[i] * winLRates[j] / 360d;  // MJ/h→kW に変換
52           hsSystem.ForecastSupplyWaterTemperature(0, 12, load / (5 * 4.186), 40);
53         }
54
55         int ahpNum = ahSystem.OperatingChillerNumber;
56         sWriter.WriteLine(j + ":00," + load +
57           ", " + (ahpChiller.ElectricConsumption * ahpChiller.OperatingNumber * ahpNum) +
58           ", " + (ahSystem.ChilledWaterPump.GetElectricConsumption() * ahpNum) +
59           ", " + (ahSystem.HotWaterPump.GetElectricConsumption() * ahpNum) +
60           ", " + arChiller.ElectricConsumption + ", " + (arChiller.FuelConsumption * 3600) +
61           ", " + arSystem.CoolingTower.ElectricConsumption +
62           ", " + arSystem.ChilledWaterPump.GetElectricConsumption() +
63           ", " + arSystem.HotWaterPump.GetElectricConsumption() +
64           ", " + arSystem.CoolingWaterPump.GetElectricConsumption() +
65           ", " + hsSystem.ChilledWaterSupplyTemperature + ", " + hsSystem.HotWaterSupplyTemperature);
66       }
67     }
68   }
69 }
```

表 21.8　月別の熱負荷と一次エネルギー消費量ならびに一次 COP

	熱負荷	空気熱源 HP 系統			直焚系統						COP
		熱源電力	冷水ポンプ	温水ポンプ	熱源ガス	熱源電力	冷却塔	冷水ポンプ	温水ポンプ	冷却水ポンプ	
1 月	326.7	202.8	0.0	14.1	1.0	27.4	0.0	0.0	1.2	0.0	1.33
2 月	287.2	170.2	0.0	14.1	0.9	24.2	0.0	0.0	1.2	0.0	1.36
3 月	222.5	133.1	0.0	11.2	0.0	0.0	0.0	0.0	0.0	0.0	1.54
4 月	53.8	24.9	0.0	7.7	0.0	0.0	0.0	0.0	0.0	0.0	1.65
5 月	100.0	40.1	9.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.00
6 月	399.6	190.5	16.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.93
7 月	704.6	208.6	17.6	0.0	8.3	240.7	5.9	11.5	0.0	18.7	1.38
8 月	783.4	263.4	17.6	0.0	9.3	267.0	6.0	11.5	0.0	23.6	1.31
9 月	504.6	116.7	16.9	0.0	6.1	173.3	4.5	11.5	0.0	10.9	1.49
10 月	57.9	23.9	9.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.71
11 月	100.5	51.3	0.0	7.7	0.0	0.0	0.0	0.0	0.0	0.0	1.70
12 月	269.3	146.3	0.0	14.1	0.8	22.9	0.0	0.0	1.2	0.0	1.45
年間	3,810.0	1,571.9	88.5	68.9	26.3	755.4	16.4	34.6	3.5	53.2	1.45

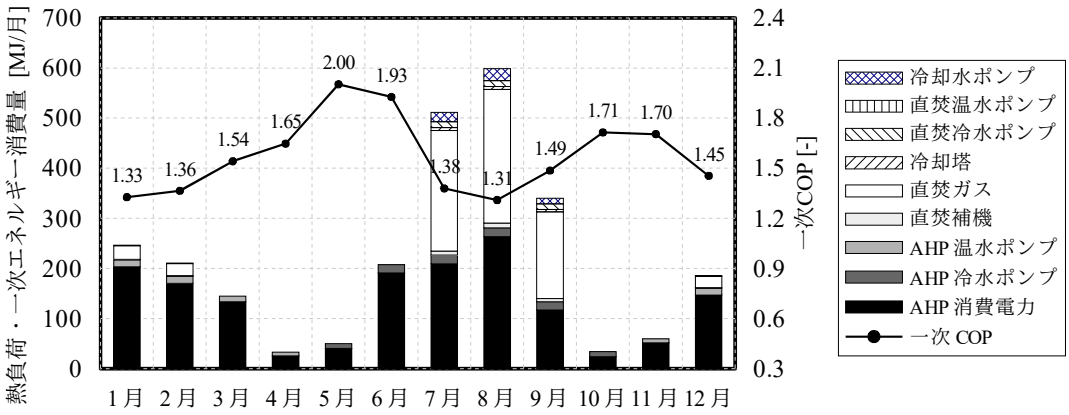


図 21.11　月別の熱負荷と一次エネルギー消費量ならびに一次 COP

【例題 21.4】

蓄熱システムと非蓄熱システムについてエネルギー消費の違いを評価せよ。機器の構成と仕様は表 21.9、時系列負荷は表 21.10 の通りとする。23:00~7:00 を蓄熱運転、7:00~8:00 を追掛運転、8:00~23:00 を放熱運転とする。ポンプの制御方式は、放熱ポンプを最小流量制御、その他のポンプをインバータによる吐出圧一定制御とする。

表 21.9　機器の仕様一覧

名称		仕様	
蓄熱・非蓄熱システム共通	ターボ冷凍機	冷却能力　　：500 kW 冷水温度条件　：12.0→ 7.0 ℃ 冷却水温度条件：32.0→37.0 ℃	COP：6.0 冷水流量　：1,433 L/min 冷却水流量：1,670 L/min
	冷却塔	冷却能力　　：583 kW 冷却水温度条件：37.0→32.0 ℃ 外気湿球温度　：27.0 ℃	ファン動力：7.5 kW×1 台 冷却水流量：1,670 L/min
	冷水ポンプ	流量　　：1,433 L/min 設計揚程：140 kPa	定格揚程：150 kPa 実揚程：50kPa
	冷却水ポンプ	流量　　：1,670 L/min 設計揚程：140 kPa	定格揚程：150 kPa 実揚程：50kPa
蓄熱システム	蓄熱槽	有効容量　　：10m×10m×10m 流出入口直径：0.8 m 断熱仕様　　：ポリスチレンフォーム 50 mm、0.037 W/(m·K) 蓄熱温度　　：5 ℃	計算用分割数　　：20 流出入口設置位置：上下端部から 0.5 m
	熱交換器	熱源水流量：1,433 L/min 冷水流量　：1,433 L/min	熱源水温度条件：6.0→11.0 ℃ 冷水温度条件　：12.0→7.0 ℃
	蓄熱ポンプ 放熱ポンプ	流量　　：1,433 L/min 設計揚程：140 kPa	定格揚程：150 kPa 実揚程：50kPa

表 21.10　時系列負荷

時刻	7:00	8:00	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00
負荷 [kW]	76	407	390	381	394	398	394	381	455	381	394	165	17	17

【解】

計算処理をプログラム 21.13 に示す。

11~38 行はターボ冷凍機、冷却塔、冷水ポンプ、冷却水ポンプによる非蓄熱システムの計算処理である。4, 5 行で与えた熱負荷に対して各時刻のエネルギー消費量を計算して出力する。

40~90 行は上記のシステムに蓄熱槽を加えたシステムの計算処理である。23:00 時点の残蓄量が翌日の蓄熱槽の状態に影響を与えるため、24 時間の計算を繰り返すことで周期定常状態に至らせて出力を行う。

計算結果を集計すると表 21.11 の通りとなる。蓄熱槽の熱ロスがあるため、蓄熱システムの冷熱製造量は非蓄熱システムよりもやや大きくなり、また、蓄熱ポンプと放熱ポンプの消費電力が発生する。一方で、2 次側負荷に依存せずに 100 %の水量で運転が可能であるため、熱源が低負荷状態になりづらく熱源単体の効率は高い。また、そもそもの運転時間が短いため、固定的に消費電力が発生する冷却塔、冷却水ポンプ、冷水ポンプの動力も非蓄熱システムよりも小さくなる。結果としては非蓄熱システムよりもやや大きいシステム COP が得られる。また、昼間と夜間で電力料金が異なる場合には、この観点からの比較も必要

であろう。なお、供給すべき熱の内、夜間時間帯に熱製造を移行でいた比率を夜間移行率と呼ぶ。

蓄放熱時それぞれの水槽内の温度プロフィールを図 21.12 に示す。本問では非蓄熱システムとの比較のために温度差を 12→7 ℃ としたが、水槽容量の節約のため、17→7 ℃ 程度の温度差とすることも多い。また、本問では 7:00~8:00 を追掛運転時間に設定したが、これは追掛運転処理の動作確認のためであり、実際のシステムではこのような運用を行うことは少ない。

プログラム 21.13　蓄熱システムと非蓄熱システムのエネルギー消費比較

```
private static void HeatSourceSubsystemTest3()
{
  //時系列負荷
  double[] hLoads = new double[]
  { 0, 0, 0, 0, 0, 0, 0, 76, 407, 390, 381, 394, 398, 394, 381, 455, 381, 394, 165, 17, 17, 0, 0, 0 };

  //ターボ冷凍機の定格冷水・冷却水量[kg/s]
  const double NCH_FLOW = 500d / (12 - 7) / 4.186;
  const double NCD_FLOW = 1670d / 60;

  //ターボ冷凍機によるサブシステムの計算
  //個別機器のインスタンスを作成
  SimpleCentrifugalChiller chiller =new SimpleCentrifugalChiller(500d / 6d, 0.2, 12, 7, 37, NCH_FLOW, false);
  CentrifugalPump chPmp = new CentrifugalPump
   (150,0.001 * NCH_FLOW,150, 0.001 *NCH_FLOW,CentrifugalPump.ControlMethod.ConstantPressureWithInverter, 50);
  CentrifugalPump cdPmp = new CentrifugalPump
   (150, 0.001 *NCD_FLOW,150, 0.001 *NCD_FLOW,CentrifugalPump.ControlMethod.ConstantPressureWithInverter, 50);
  CoolingTower cTower= new CoolingTower(37, 32, 27, NCD_FLOW, CoolingTower.AirFlowDirection.CrossFlow, false);
  //サブシステムを作成
  CentrifugalChillerSystem crSystem = new CentrifugalChillerSystem(chiller, chPmp, cdPmp, cTower, 1, 1);
  //熱源システムを作成
  HeatSourceSystemModel hsSystem = new HeatSourceSystemModel(new IHeatSourceSubSystem[] { crSystem });
  hsSystem.SetOperatingMode(0, HeatSourceSystemModel.OperatingMode.Cooling);
  hsSystem.SetChillingOperationSequence(0, 1);
  hsSystem.ChilledWaterSupplyTemperatureSetpoint = 7;
  hsSystem.OutdoorAir = new MoistAir(35, 0.0195);
  hsSystem.TimeStep = 3600;

  //24h の計算実行
  Console.WriteLine("冷熱供給,ターボ,冷水ポンプ,冷却水ポンプ,冷却塔");
  for (int i = 0; i < hLoads.Length; i++)
  {
    hsSystem.ForecastSupplyWaterTemperature(hLoads[i] / (4.186 * 5), 12, 0, 40);
    hsSystem.FixState();
    Console.WriteLine(hLoads[i].ToString("F0") + ", " +
     chiller.ElectricConsumption.ToString("F2")+ ", " + chPmp.GetElectricConsumption().ToString("F2") + ", " +
      cdPmp.GetElectricConsumption().ToString("F2") + ", " + cTower.ElectricConsumption.ToString("F2"));
  }

  //蓄熱槽によるサブシステムの計算
  //個別機器のインスタンスを作成（冷凍機・冷水ポンプ・冷却水ポンプ・冷却塔は使い回す）
  PlateHeatExchanger pHex = new PlateHeatExchanger(500, 6, NCH_FLOW, 7, NCH_FLOW);
  MultipleStratifiedWaterTank wTank = new MultipleStratifiedWaterTank(10, 10 * 10, 0.8, 9.5, 20);
  CentrifugalPump chgPmp = new CentrifugalPump
   (150, 0.001 * NCH_FLOW,150, 0.001 *NCH_FLOW, CentrifugalPump.ControlMethod.ConstantPressureWithBypass, 50);
  CentrifugalPump disPmp = new CentrifugalPump
    (150, 0.001 * NCH_FLOW, 150, 0.001 * NCH_FLOW, CentrifugalPump.ControlMethod.MinimumPressure, 50);
  //サブシステムを作成
  MultipleStratifiedWaterTankSystem wtSystem =
    new MultipleStratifiedWaterTankSystem(wTank, pHex, chiller, chPmp, cdPmp, chgPmp, disPmp, cTower, 1, 1);
  wTank.HeatLossCoefficient = 0.001 * (0.037 / 0.05) * (10 * 10 * 6);
  wtSystem.StorageTemperature = 5.0;
  //熱源システムを作成
  hsSystem = new HeatSourceSystemModel(new IHeatSourceSubSystem[] { wtSystem });
  hsSystem.SetOperatingMode(0, HeatSourceSystemModel.OperatingMode.Cooling);
  hsSystem.SetChillingOperationSequence(0, 1);
  hsSystem.ChilledWaterSupplyTemperatureSetpoint = 7;
  hsSystem.OutdoorAir = new MoistAir(35, 0.0195);
  hsSystem.TimeStep = 3600;

  //24h の計算を周期定常になるまで繰り返す
  Console.WriteLine("冷熱供給,ターボ,冷水ポンプ,冷却水ポンプ,冷却塔,蓄熱ポンプ,放熱ポンプ,冷熱製造,水槽温度");
  bool lastCalc = false;
  double lastST = 100;
  while (!lastCalc)
  {
    lastCalc = (Math.Abs(lastST - wTank.GetTemperature(10)) < 0.001);
    lastST = wTank.GetTemperature(10);
    for (int i = 0; i < hLoads.Length; i++)
    {
      if (i < 9 || 22 < i) wtSystem.Charging = true;
```

```
72        else wtSystem.Charging = false;
73        hsSystem.ForecastSupplyWaterTemperature(hLoads[i] / (4.186 * 5), 12, 0, 40);
74        hsSystem.FixState();
75        if (lastCalc)
76        {
77          Console.Write(hLoads[i].ToString("F0") + ", " +
78            chiller.ElectricConsumption.ToString("F2") + ", " +
79            chPmp.GetElectricConsumption().ToString("F2") + ", " +
80            cdPmp.GetElectricConsumption().ToString("F2") + ", " +
81            cTower.ElectricConsumption.ToString("F2") + ", " +
82            chgPmp.GetElectricConsumption().ToString("F2") + ", " +
83            disPmp.GetElectricConsumption().ToString("F2") +  ", " +
84            chiller.CoolingLoad.ToString("F1"));
85          for (int j = 1; j < wTank.LayerNumber; j += 2)
86            Console.Write(", " + wTank.GetTemperature(j).ToString("F2"));
87          Console.WriteLine();
88        }
89      }
90    }
91 }
```

表 21.11　計算結果集計

	熱供給 [kWh/日]	冷熱製造 [kWh/日]	消費電力 [kWh/日]							COP
			ターボ	冷水ポンプ	冷却水ポンプ	冷却塔	蓄熱ポンプ	放熱ポンプ	合計	
非蓄熱	4,250	4,250	845.0	55.6	84.8	105.0	0.0	0.0	1,096	3.90
蓄熱	4,250	4,469	812.4	50.9	60.6	75.0	56.3	25.9	1,085	3.93

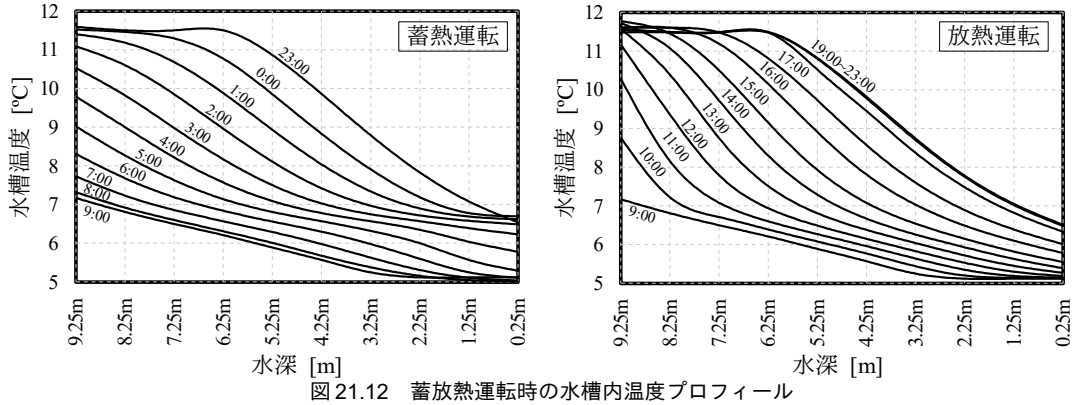


図 21.12　蓄放熱運転時の水槽内温度プロフィール

【例題 21.5】

例題 21.4 の非蓄熱システムについて冷却塔の冷却水出口温度と冷却水流量の最適化を行え。ただし、熱源負荷率は 60 %、外気条件は 24 ºC, 12.5 g/kg とする。

【解】

冷却水温度を下げるとターボ冷凍機のエネルギー消費は減少するが、冷却塔のエネルギー消費は増大する。同様に、冷却水流量を上げるとターボ冷凍機のエネルギー消費は減少するが、冷却水ポンプのエネルギー消費は増大する。これらのトレードオフを考慮しながら最適な温度と流量を推定する問題である。複数の入力条件がある場合の関数の極小値探索であり、第 2 章の準ニュートン法を用いる。

計算処理をプログラム 21.14 に示す。3~21 行は例題 21.4 と同様である。ただし、冷却水温度と流量を入力条件とするため 17, 18 行で両者を制御対象に設定する。

24~38 行は極小化する関数である。冷却水温度と冷却水流量をモデルに設定し、システムの計算を行って消費電力を積算する（27~31 行）。冷却水温度と流量は、数値計算上は任意の実数であるが、現実には物理的に実現しうる範囲が存在する。従って、この範囲から外れないように範囲外に飛び出そうとする場合にはペナルティを与える（34, 35 行）。反復計算の回数が増加するに連れてこのペナルティの値が大きくなるようにすることで（37 行）、最終的にたどり着く解が物理的に実現する範囲内に納まるようにする。

44~58 行は数値計算で極小値を探すのではなく、網羅的に探索する場合の処理である。流量比を 5 %刻

み、温度を 1 ºC 刻みで変化させながらすべての組み合わせを実際に解いてエネルギー消費量を出力する。60~65 行は準ニュートン法で極小値を探索する場合の処理である。計算結果である電力消費量を等高線に表すと図 21.13 が得られる。準ニュートン法の反復計算の各回の状態値もプロットした。32 ºC, 100 % の初期値から開始して、4 回程度の反復計算でほぼ最適な組み合わせ（25.4 ºC, 74.2 %）にたどり着くことがわかる。なお、初期値として 32 ºC, 100 %を選択するということには意味がある。等高線から明らかなように、冷却水温度が低い条件では誤差関数が冷却水温度に対して勾配をもたない。即ち、冷却塔が出口温度設定値を満足できずに過負荷状態となるため、温度設定値に関わらず同じ状態でシステムが均衡するということである。このような状態を初期値として選択しては勾配が 0 となってしまい、準ニュートン法が上手く機能しない。従って、それぞれの入力条件に対して微分値が 0 以外になる適切な初期値を選択する必要がある。

プログラム 21.14　冷却水流量と温度の最適化

```
 1 private static void HeatSourceSubsystemTest4()
 2 {
 3   //ターボ冷凍機の定格冷水・冷却水量[kg/s]
 4   const double NCH_FLOW = 500d / (12 - 7) / 4.186;
 5   const double NCD_FLOW = 1670d / 60;
 6
 7   //ターボ冷凍機によるサブシステムの計算
 8   //個別機器のインスタンスを作成
 9   SimpleCentrifugalChiller chiller = new SimpleCentrifugalChiller(500d / 6d, 0.2, 12, 7, 37, NCH_FLOW, false);
10   CentrifugalPump chPmp = new CentrifugalPump
11   (150, 1e-3 * NCH_FLOW, 140, 1e-3 * NCH_FLOW, CentrifugalPump.ControlMethod.ConstantPressureWithInverter,50);
12   CentrifugalPump cdPmp = new CentrifugalPump
13   (150, 1e-3 * NCD_FLOW, 140, 1e-3 * NCD_FLOW, CentrifugalPump.ControlMethod.MinimumPressure, 50);
14   CoolingTower cTower = new CoolingTower(37, 32, 27, NCD_FLOW, CoolingTower.AirFlowDirection.CrossFlow, true);
15   //サブシステムを作成
16   CentrifugalChillerSystem crSystem = new CentrifugalChillerSystem(chiller, chPmp, cdPmp, cTower, 1, 1);
17   crSystem.ControlCoolingWaterTemperature = true;
18   crSystem.ControlCoolingWaterFlowRate = true;
19   crSystem.Mode = HeatSourceSystemModel.OperatingMode.Cooling;
20   MoistAir oa = new MoistAir(24, 0.0125);
21   crSystem.OutdoorAir = oa;
22
23   double pLoad = 0.6;
24   MultiMinimization.MinimizeFunction mFnc = delegate (IVector vec, int iteration)
25   {
26     //エネルギー消費量の計算
27     crSystem.CoolingWaterTemperatureSetpoint = vec[0];
28     crSystem.CoolingWaterFlowSetpoint = vec[1] * cTower.MaxWaterFlowRate;
29     crSystem.ForecastSupplyWaterTemperature(500 * pLoad / (5 * 4.186), 0);
30     double ecsmp = crSystem.Chiller.ElectricConsumption
31     + crSystem.CoolingTower.ElectricConsumption + cdPmp.GetElectricConsumption();
32
33     //ペナルティの計算
34     double pSum = Math.Max(0, 12 - vec[0]) + Math.Max(0, vec[0] - 32);
35     pSum += 10 * (Math.Max(0, 0.5 - vec[1]) + Math.Max(0, vec[1] - 1.0));
36
37     return ecsmp + pSum * (1 + 0.5 * iteration);
38   };
39
40   IVector vecX = new Vector(2);
41   using (StreamWriter sWriter = new StreamWriter
42     ("HeatSourceSubsystemTest4.csv", false, Encoding.GetEncoding("Shift_JIS")))
43   {
44     //網羅的に探索
45     sWriter.WriteLine(",50%,55%,60%,65%,70%,75%,80%,85%,90%,95%,100%");
46     for (int i = 20; i <= 32; i++)
47     {
48       sWriter.Write(i + "C");
49       for (int j = 0; j < 11; j++)
50       {
51         vecX[0] = i;
52         vecX[1] = 0.5 + 0.05 * j;
53         double ec = mFnc(vecX, 1);
54         if (crSystem.IsOverLoad_C) sWriter.Write(",-");
55         else sWriter.Write("," + ec);
56       }
57       sWriter.WriteLine();
58     }
59
60     //準ニュートン法で探索
61     sWriter.WriteLine("準ニュートン法 探索結果");
62     int iter;
63     vecX[0] = 32; vecX[1] = 1.0;
```

```
64     MultiMinimization.QuasiNewton(ref vecX, mFnc, 50, 1e-4, 1e-4, 1e-4, out iter);
65     sWriter.WriteLine("温度設定," + vecX[0] + ", 流量比, " + vecX[1]);
66   }
67 }
```

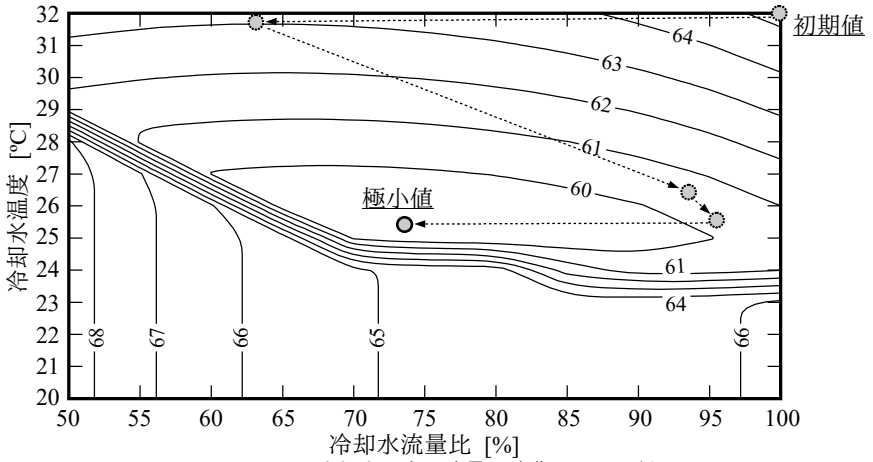


図 21.13　冷却水温度・流量と消費電力の関係

【第 21 章 記号表】

c_{pw}	: 比熱 **[kJ/(kg·K)]**	m	: 質量流量 **[kg/s]**
E	: 消費電力 **[kW]**	T	: 温度 **[K]**
Subscripts:			
B	: バイパス	***R***	: 還
cd	: 冷却水	***RH***	: 還ヘッダ
chw	: 冷水	***S***	: 往
hex	: 熱交換器	***SH***	: 往ヘッダ
hw	: 温水	***st***	: 蓄熱槽
p	: ポンプ	***tb***	: ターボ冷凍機

【第 21 章 参考文献】

21.1） 財団法人 建築環境・省エネルギー機構: 改訂 空調エネルギー消費係数（CEC/AC）計算法 -全負荷相当運転時間法による CEC/AC 計算法-, 2003

21.2） 日本地域冷暖房協会: 地域冷暖房技術手引書<改訂新板>, 2002

21.3） 東京都環境保全局: 地域冷暖房推進に関する指導要綱, pp.70~81, 1991

21.4） 日本ガス協会: 次世代型ガスコージェネレーション技術研究会報告書, pp.348~350, 1990

21.5） 空気調和・衛生工学会: コージェネレーションシステム設計・計画と評価, 1994

21.6） 住宅・建築省エネルギー機構（IBEC）: 都市再開発におけるコージェネレーションシステム導入推進に関する調査, pp.17~18, 1992

21.7） 省エネルギーセンター: 分散型電源システムの最適化に関する調査, pp.44~55, 1985

21.8） 国土交通省大臣官房官庁営繕部設備･環境課: LCEM ツール ver 3.10 操作説明書, 2014

21.9） 空気調和・衛生工学会 SHASE-G 1008-2016 建物エネルギーシミュレーションツールの評価手法に関するガイドライン, 2016

21.10） 市川卓也, 中島正人, 下田吉之, 松尾陽: 空調システムの最適化を目的とした統合的設計と運転に関する研究, 第 10 報 熱源廻りの省エネルギー手法に関する実態調査, 空気調和・衛生工学会大会学術講演論文集, pp.9-12, 2009

第 III 編　室内環境 評価編

第 III 編では、主に居住空間の温熱環境や快適性に影響を与える要素について解説する。人間と建物の接点の計算であり、居住空間の温湿度や放射環境の計算に加え、人体自体の熱流計算が必要となる。まず、第 22, 23, 24 章で壁の熱伝導、窓の日射熱取得、日射遮蔽など、建築躯体の熱流に影響を与えるそれぞれの要素について解説する。第 25 章でこれらの要素を統合して建物全体の熱負荷計算を行う方法を示す。また、温湿度を調整するための空気調和機の計算法を第 26 章で取り扱う。第 27 章では人間が感じる熱的快適性や満足度を予測する方法を示す。第 28 章では 25 章の熱負荷計算と 26 章の空気調和機の計算を連成させた二次側空調システムモデルを作成し、建築内部空間の年間の空気温湿度や放射温度などの熱的環境を予想する。これらを 27 章の熱的快適性モデルに適用することで建築内部空間の年間の熱環境評価が可能となる。さらに、第 29 章では第 II 編で開発した一次側熱源システムモデルと本編の二次側空調システムモデルの連成計算を行う。これにより、経済性、環境性に加え、熱的快適性を同時に評価することが可能となり、これらのトレードオフを考慮しながら最適な空調設備を検討する技術が得られたことになる。最終章では建築熱環境計算の歴史を振り返り、その将来を占う。

第22章　熱負荷計算 1: 壁体の熱流
(Heat Load Calculation 1: Wall heat transfer)

22.1　概要

1）熱負荷計算とは

熱負荷は空調・熱環境設計の根幹である[†1]。建築は学際的な学問分野であり、設備側には機械を専門とする別の分野があり、室内側には人間を専門とする別の分野が広がる。このような中で建築環境工学がその本丸とできる領域の 1 つはおそらくは建築熱負荷計算である。

熱環境設計の主要な目的の 1 つは、空間内の温湿度を目標とする状態に制御することにある。温湿度制御の方法は大きく分けて 2 つあり、1 つは日除けや自然通風などを利用し、自然の特性に沿わせた建築形態とすることで熱環境を制御する方法である。これはパッシブシステムと呼ばれる。もう 1 つは冷温風供給に代表されるように、機械力で強制的に熱を移動させる方法である。一般的には機械力を用いた後者を「空調」と呼ぶことが多く、以下、本書ではこの意味で「空調」という用語を用いる。空調により温湿度制御を行う場合には、空間に対して熱の投入あるいは除去が必要となり、この熱流を「熱負荷」と呼ぶ。もちろん、パッシブシステムであっても各所に熱流は生じるが、この場合の熱流は熱負荷とは呼ばない。また、空調を行わない場合には空間の温度および湿度は、外乱や空間内部で発生する熱の影響を受けて自由に揺れ動く。この場合の温度を「自然室温」と呼ぶ。「熱負荷」と「自然室温」の計算は表裏一体であり、通常は両者を含めて「熱負荷計算」と総称する。

熱負荷計算の全貌を一挙に理解することはできない。本書では第 22 章で壁体の非定常熱伝導、第 23 章で日射遮蔽技術、第 24 章で窓からの熱取得について解説し、熱負荷計算で必要となる個別要素のモデル化を行う。第 25 章でこれらを組み合わせて多数室の連成計算を行う方法に発展させる。

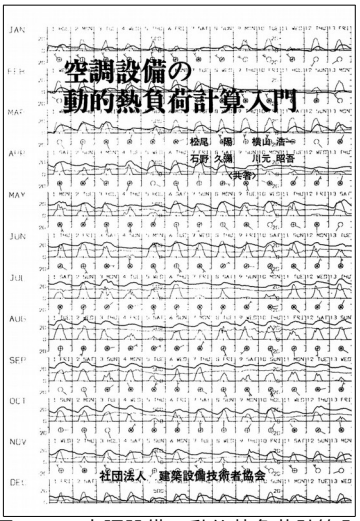


図 22.1　空調設備の動的熱負荷計算入門
（日本における動的熱負荷計算研究の結晶）

†1　石野による言葉である[22.1)]。

2）熱負荷計算モデルの全体像

熱負荷計算には、壁、窓、日射遮蔽物、室内発熱体、ゾーン、室、太陽、など、多数の要素が登場する。これらの全てを表現する巨大な単一のプログラムコードを作成することは可読性の面から問題がある。そこで、本書ではオブジェクト指向言語の特徴を活かし、要素ごとにクラスを作成し、これらを組み合わせることで全体の熱負荷計算モデルを構築する方針とする。図 22.2 に本書で開発する熱負荷計算モデルのクラス図を示す。以下、本クラス図を用いてモデル全体の構成を解説するが、現段階では理解できない用語や概念も含まれると思われる。後章での個別要素の詳細な解説を理解した後、適宜、本図に戻って全体モデルとの関係性を確認してもらいたい。

「BuildingThermalModel」は熱負荷計算モデル全体を統括するクラスであり、複数の「MultiRoom」クラスを持つ。「MultiRoom」は多数室の熱負荷計算を行うクラスであり、このクラスに含まれるゾーン・壁体・窓の温度は連成計算の対象となる。逆に言えば、異なる「MultiRoom」に属するゾーンや壁の温度は連成計算されず、「BuildingThermalModel」によって一定時間間隔で状態値が通信されるのみである。即ち、ある「MultiRoom」で計算する温湿度の値は、他の「MultiRoom」からは境界条件として扱われる。

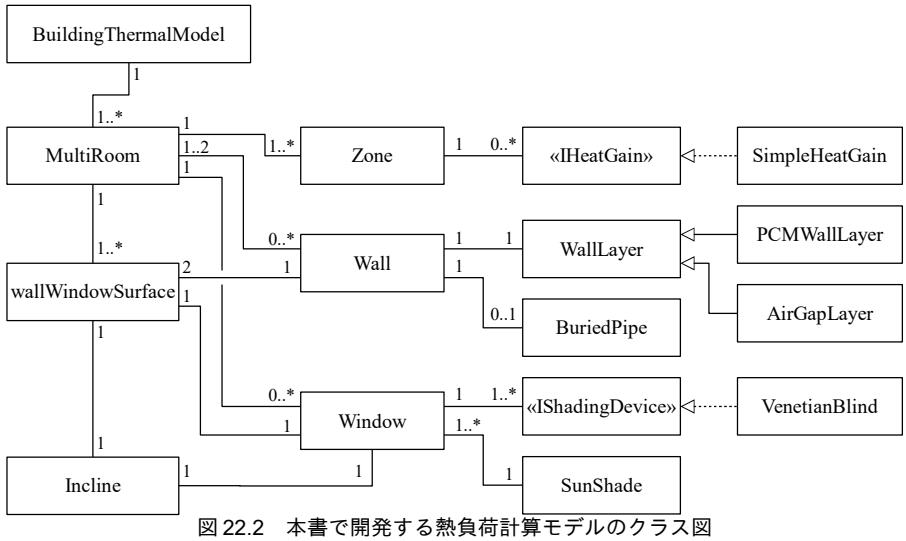


図 22.2　本書で開発する熱負荷計算モデルのクラス図

「MultiRoom」は、ゾーンを表す「Zone」、壁体を表す「Wall」、窓を表す「Window」を持ち、これらの連成計算を行う。室内の多重相互放射の計算のためには壁表面および窓表面の温度が必要となるが、統合的に温度情報の管理を行うために「wallWindowSurface」を用いる（後述）。特に外表面に関しては、日射の影響を計算するために傾斜面の情報が必要となる。このため「wallWindowSurface」は第 6 章で開発した「Incline」を保持する。

「Zone」は 1 つの質点であり、熱容量、水分容量、温湿度などの情報を持つ。また、ゾーン内で発生する熱を管理するために「IHeatGain」を持つ。「IHeatGain」は顕熱と潜熱（水分）の発生の計算機能を持つインターフェースである。顕熱および潜熱の発生は計算対象によって様々であるため、「IHeatGain」インターフェースを実装することで、ユーザーが任意のモデルを実装できる仕様とする。本書では、具体例として固定値を出力する「SimpleHeatGain」を作成する。

「Wall」は多層壁を表し、非定常熱伝導を計算する。「WallLayer」は材料の熱伝導率、熱抵抗、

厚みなど、壁層の情報を持つ。現実の建物では同じ構成の壁が各所で表れるため、壁層にかかわる情報を「WallLayer」で管理し、これに面積や方位の情報を加えることで「Wall」を作成する仕様とする。このような構成とすれば、例えば北側と南側の壁や、1m^{2}の壁と2m^{2}の壁とで同じ「WallLayer」を使い回すことができ、壁層情報を何度も作りなおす必要がなくなる。「PCMWallLayer」および「AirGapLayer」は「WallLayer」の派生クラスであり、前者は潜熱蓄熱材料による壁層、後者は空気層である。いずれも通常の壁材料とは異なる特性を持つため、別にモデル化をする。「BuriedPipe」は壁内に埋め込まれた配管である。埋設配管を持つ放射冷暖房システムを計算する際に用いる。以上の壁の熱流の詳細は第22章で解説する。

「Window」は窓を表しており、ガラスの光学特性にもとづいて透過日射、吸収日射などを計算する。また、ガラスおよび中空層の伝熱特性にもとづいて貫流熱も計算する。これらの計算法は第24章で解説する。「Window」は外部日射遮蔽の影響を計算する「SunShade」と、窓に設けられた日射遮蔽物の影響を計算する「IShadingDevice」を持つ。窓に設ける日射遮蔽物は、ブラインド、ロールスクリーン、障子、緑化、遮光フィルムなど様々であるため、ユーザーが任意の実装ができる仕様としている。本書では、具体例としてベネシャンブラインドを表わす「VenetianBlind」を作成する。日射遮蔽については第23章で解説する。

「Wall」も「Window」も表側と裏側（本書では便宜的にF側とB側と呼ぶ）がある。多数室の連成計算のためには室を囲む「Wall」と「Window」の表面温度が必要になる。従って、ある室の計算にあたっては、室から見えがかりになっている面についての情報が必要であるが、それが「Wall」としてF側なのかB側なのかは興味がない。表面温度などの情報を取得するために、その表面がF側なのかB側なのかを一々判断することは大きな手間である。また、長波長放射の計算のように、表面温度と放射率の情報のみが重要であり「Wall」なのか「Window」なのかは問わないような場面もある。そこで、壁や窓の表面を表す「wallWindowSurface」というクラスを定義し、ある室にとっての表面情報を管理する方針とする。「wallWindowSurface」はクラス間の情報の受け渡しをわかりやすくするためのプログラム開発者側の工夫であり、本モデルを用いるユーザー側にとっては無用の概念である。そこで、アクセス修飾子をinternalに設定し、ユーザーからは見えないようにする。

「Wall」「Window」「SunShade」は単独でも使用可能なクラスとする。各章の例題などを用いてそれぞれで十分に単体動作検証を行うことが重要である。個別のクラスが正常に動かなければ、これらを組み合わせた複合的なモデルは動かない。

22.2　理論

22.2.1　壁体表面の顕熱流

1）相当温度

壁体内部の温度変動を予測するためには、壁体の両側の熱的条件を定める必要がある。壁体表面の熱流に影響をあたえる要素としては、主に表面付近の空気温度と壁体表面を取り巻く物体による平均放射温度があり、両者を合わせて相当温度 T_{SolWal} [K]（SAT: Sol Air Temperature）という概念で表現することができる[†1)]。式 22.1 に屋外表面と屋内表面の相当温度計算式を示す。

$$T_{SolWal}=\begin{cases} T_o+\dfrac{a_s I_W-\varepsilon_{LW} F_S R_N}{\alpha_{o(c+r)}} & (\text{exterior surface}) \\ k_c T_{ZN}+k_r T_S+\dfrac{R_{SLW}}{\alpha_{i(c+r)}} & (\text{interior surface}) \end{cases} \tag{22.1}$$

$$k_c=\frac{\alpha_{i(c)}}{\alpha_{i(c+r)}} \tag{22.2}$$

$$k_r=\frac{\alpha_{i(r)}}{\alpha_{i(c+r)}} \tag{22.3}$$

屋外の場合には外気温度 T_o [K]に対して、傾斜面日射 I_W [W/m^2]と夜間放射 R_N [W/m^2]の影響を加える。a_sは日射吸収率[-]、ε_{LW}は長波長放射率[-]、F_Sは天空への形態係数[-]、$\alpha_{o(c+r)}$は外表面総合熱伝達率[W/(m^2·K)]である。日射吸収率 a_sの値は外壁の色や粗さなどによって影響を受けるが、暗色タイル、コンクリート、石などであれば 0.65~0.80 程度[†2)]、白色あるいはクリーム色の塗装であれば 0.30~0.50 程度であり、よく磨かれた金属面では 0.1 程度の値もとり得る[22.22)]。

屋内の場合にはゾーン[†3)]の空気温度 T_{ZN} [K]と平均放射温度 T_S [ºC]を対流熱伝達率と放射熱伝達率で重み付け平均した温度に対して、その他の放射熱取得 R_{SLW} [W/m^2]の影響を加える。k_c [-]と k_r [-]はそれぞれ式 22.2 と式 22.3 に示すように、室内側の対流熱伝達率 $\alpha_{i(c)}$ [W/(m^2·K)]と放射熱伝達率 $\alpha_{i(r)}$ [W/(m^2·K)]を総合熱伝達率 $\alpha_{i(c+r)}$ [W/(m^2·K)]で除した値である。平均放射温度 T_Sを計算するためには壁を取り囲むすべての物体の表面温度を知る必要があるが、壁体単体の熱流計算においては物体の温度が空気温度 T_{ZN}と等しいと仮定し、$k_c T_{ZN}+k_r T_S=\alpha_{i(c+r)} T_{ZN}$として計算することも多い。室内のそれぞれの壁の表面温度の値を個別に捉え、相互放射を正確に計算する方法については第 25 章で解説する。

2）対流熱伝達率

固体と流体との間の温度差によって熱が移動する現象を対流熱伝達と呼び、この時の温度差に対する比例定数を対流熱伝達率と呼ぶ。壁体表面と周囲の空気との間の熱移動は対流熱伝達である。

流体の流れ方としては自然対流と強制対流がある。自然対流は固体表面近くの流体が加熱あるいは冷却されることで生じる流体の密度差を駆動力とする流体の流れである。例えば垂直に置かれた冷たい壁を暖かい空気が包んでいる場合、壁表面付近の空気は冷やされて密度が大きくなり、下方へ落ちていく。一方、強制対流は送風機や屋外の風など、自然対流以外の原因で生じる流体の流れである。強制対流による流速が大きい場合には、自然対流の影響は相対的に小さく無視できるが、無風に近い

†1　外壁表面において日射の影響を考慮した温度を「相当外気温度」と呼ぶことが多いが、本書では室内側における放射の影響を考慮した温度も含めて「相当温度」と総称する。

†2　最大熱負荷計算に用いられる実効温度差 ETD（Effective Temperature Difference）の表は a_s=0.70 を前提に作成されることが多い。デフォルト値としては 0.70 程度を設定しておけばよいだろう。

†3　本書では室よりも小さなゾーンという単位に分けて空間の温度を計算する。詳細については第 25 章で解説する。

条件では温度差による自然対流の効果を考慮する必要がある。壁体表面の熱流に即して言えば、通常は屋外では風速が平均で数 m/s 程度となるために強制対流の影響が大きく、逆に屋内は自然対流が支配的となることが多い。

対流熱伝達率は固体の形状、流体の物性、流れる速さなどに依存し、各種の図表や計算式が提供されている[22.5)]。ただし、建物の年間熱負荷計算の場合には計算速度向上のために、室内側を 1~4 W/(m²·K)程度、屋外側を 5~20 W/(m²·K)程度の固定値として取り扱うことも多い。

建物外表面の対流熱伝達率に関しては、伊藤らが実建物の実測結果をもとに式 22.4 の推定式を提案している[22.6)]。v_s [m/s]は壁表面近傍の風速であり、外部風速 v [m/s]の大きさやその風向に依存する。特定の建物について結果ではあるが、伊藤らは式 22.5 を報告している[†1)]。

$$\alpha_{o(c)}=4.7+7.6v_s \tag{22.4}$$

$$v_s=\begin{cases} v_s=0.25v & (\text{風上},2<v) \\ v_s=0.5v & (\text{風上},v\le 2) \\ v_s=0.3+0.05v & (\text{風下}) \end{cases} \tag{22.5}$$

屋内表面では自然対流が支配的であるため、その大きさは壁の位置や壁表面と空気の温度関係（熱流の向き）に依存する。例えば天井面が室温に比較して冷えている場合には冷やされた空気が下降して天井付近の空気を撹拌するため対流熱伝達率は大きく（3~5W/(m²·K)）なるが、逆に室温に比較して暖かい場合には軽い空気が天井付近に層を形成するため、対流熱伝達率は小さい（0.5~1.5W/(m²·K)）。床の場合にはこの逆の現象が生じる。放射冷暖房のように壁表面の温度に特徴があるシステムを検討する場合などにはこのような傾向をモデルに表現することが望ましい[†2)]。

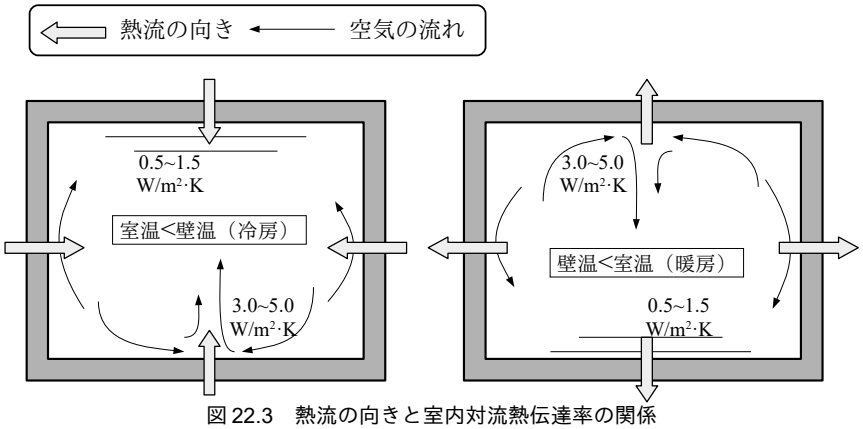


図 22.3　熱流の向きと室内対流熱伝達率の関係

3）放射熱伝達率

放射によって面 1 から面 2 に移動する熱 $Q_{r(1\text{-}2)}$ [W]は式 22.6 で表現される。ε [-]は各々の面の放射率、A_1 [m²]は面 1 の面積、F_{12} [-]は面 1 から面 2 を見た形態係数、σ はステファン・ボルツマン係数（=5.67×10⁻⁸ W/(m²·K⁴)）である。

$$Q_{r(1-2)}=\varepsilon_1\varepsilon_2A_1F_{12}\sigma\left(T_1^4-T_2^4\right) \tag{22.6}$$

式 22.6 は温度の 4 乗を含み扱いづらいため、以下の方法で線形近似する。両表面の平均温度を T_{MID}

†1　この他に屋外表面の熱流に影響を与える要素としては降雨の際の水分蒸発がある。水分蒸発の評価法およびその効果に関しては参考文献 22.17 に詳しい。

†2　同じ放射空調システムであっても、暖房が主となる住宅においては床放射が多く、冷房が主となるオフィスにおいては天井放射が多い。放射パネルを熱流の向きに合わせた配置とすることで対流熱伝達率を向上させていることが一つの理由である。

[K]とおき、式 22.7 と式 22.8 で両表面の温度を表現する。これらを式 22.6 に代入すると式 22.9 の 1 行目が得られる。ΔT は我々を取り巻く物体間の温度差であるため、精々 10~20 K 程度であるが、T_{MID} は 300 K 程度である。従って、ΔT^2 は T_{MID}^2 に比べて非常に小さく、この項を無視して式 22.9 の 2 行目で近似する[†1)]。放射熱伝達率 $\alpha_{(r)}$ を式 22.10 で定義すれば、式 22.9 は式 22.11 となるため、熱流を両表面の温度差の線形式で表現できる。常温（25°C 程度）で両表面の放射率を 0.9 程度とすると、式 22.10 は 5 W/(m²·K)程度の値となり、放射熱伝達率としてはこの程度の値を用いることが多い。

$$T_1 = T_{MID} + \Delta T \tag{22.7}$$

$$T_2 = T_{MID} - \Delta T \tag{22.8}$$

$$\begin{aligned} Q_{r(1-2)} &= 4\varepsilon_1\varepsilon_2 A_1 F_{12}\sigma T_{MID}(T_1 - T_2)(T_{MID}^2 + \Delta T^2) \\ &\approx 4\varepsilon_1\varepsilon_2\sigma T_{MID}^3 A_1 F_{12}(T_1 - T_2) \end{aligned} \tag{22.9}$$

$$\alpha_{(r)} = 4\varepsilon_1\varepsilon_2\sigma T_{MID}^3 \tag{22.10}$$

$$Q_{r(1-2)} \approx \alpha_{(r)} A_1 F_{12}(T_1 - T_2) \tag{22.11}$$

4）総合熱伝達率

対流による熱流と放射による熱流の合算値は対流熱伝達率および放射熱伝達率を用いて式 22.12 で表現できる。壁表面を取り囲む物体の温度が空気の温度 T_a と等しいとすれば、平均放射温度 T_S=空気温度 T_a となるため式 22.12 は式 22.13 に変形でき、対流熱伝達率と放射熱伝達率を合算した値である $\alpha_{(c+r)}$ を総合熱伝達率と呼ぶ。計算速度を向上させるため、年間の熱負荷計算においては屋外の総合熱伝達率を 23 W/(m²·K)程度、屋内の総合熱伝達率を 9 W/(m²·K)程度（ガラス表面の場合には壁体よりもやや熱流が大きいため、12W/(m²·K)程度[22.1)]）で固定値として取り扱うことも多い[†2)]。

$$Q_{(c+r)} = Q_c + Q_r = \alpha_{(c)}(T_{ws} - T_{ZN}) + \alpha_{(r)}(T_{ws} - T_S) \tag{22.12}$$

$$Q_{(c+r)} = (\alpha_{(c)} + \alpha_{(r)})(T_{ws} - T_a) = \alpha_{(c+r)}(T_{ws} - T_a) \tag{22.13}$$

22.2.2　壁体内部の顕熱流

1）熱容量質点系

現実の壁の内部では温度がなめらかに分布しており、正確に熱流を把握するためには、壁体内部のすべての微小点について温度を知る必要がある。しかし、このためには時間方向の温度変化に加えて空間方向の温度分布を計算する必要があり、偏微分方程式を解くことが必要になる。そこで、計算を簡略化させるために、空間（壁体）をいくつかのブロックに分割してこれを一つの点で置き換え、有限の点と点との間の熱移動のみで空間的な熱流を把握するという方法がある。このとき、ブロックの熱容量はすべて一点に集中していると考える。このような考え方を熱容量質点系と呼び、顕熱流だけではなく、水分移動や換気による空気の移動に対しても適用できる考え方である。顕熱流の場合には熱容量を集中させるが、水分の場合には水分容量、電気の場合には電気容量であり、このようなモデルを総称して集中定数系モデルと呼ぶ。

2）非定常計算と差分化

壁を囲む空間の温度を一定に維持した上で無限の時間を計算させれば、壁の内部の温度分布は一定になる。このような状態を定常状態と呼ぶ。逆に、壁を取り囲む空間の熱環境が変動する場合には、壁内部の温度は定まらず、熱流も時々刻々と変化し続ける。このような状態を非定常状態と呼ぶ。現

†1　この近似は非常に多くの場面で登場するため、考え方を理解しておくと良い。

†2　9.3 W/(m²·K)と 23.2W/(m²·K)としていることも多いが、これは工学単位の時代に慣用的に用いていた 8 kcal/(h·m²·K)と 20 kcal/(h·m²·K)を単純に SI 単位に変換した値であり、そもそもの精度が小数点第 1 位まであるとは思われない。

実の建物は非定常状態にあるため、年間の熱負荷・室温計算を行うためには非定常な熱流を計算する必要がある。非定常状態における熱流 Q [W]は一般に熱容量 C [J/K]と温度 T [K]を用いて式 22.14 のような微分方程式で表される。熱流 Q の表現方法は問題によって様々あり、必ずしも解析的に微分方程式が解けるとは限らない。このため、式 22.15 に示すように有限の時間に分割して近似するという方法を取る。この方法を差分法と呼び、式 22.14 のような連続的な式から式 22.15 を導出することを差分化と呼ぶ。Δt [sec]は分割した時間間隔であり、問題の性質によって設定すべき値は異なるが、建築壁体の場合には熱容量が大きいため、1 時間（3600 sec）程度とすることが多い。T^*は現在時点の温度、Tは将来時点の温度である。この式を Tについて解けば、現在の温度 T^*から開始して、式を繰り返し適用することで将来の温度変動を求めることができる。

$$C\frac{dT}{dt}=Q \tag{22.14}$$

$$C\frac{T-T^*}{\Delta t}=Q \tag{22.15}$$

式 22.15 は右辺の熱流が固定値であったが、現実には式 22.16 のように温度 T に依存して熱流が記述されることが多い。KA [W/K]は伝熱係数、T_b [K]は境界条件の温度である。この時、式 22.15 は式 22.17 または式 22.18 で表現できる。現在時点の温度 T^*で熱流を表現した式 22.17 を前進差分式、将来時点の温度 T で表現した式 22.18 を後退差分式と呼ぶ。本例ではいずれも簡単に Tについて解くことができるが、後退差分式は未知変数である T が両辺に含まれるため、熱流の式が複雑になると解析的に解けなくなることがある。一方で、前進差分式の場合には計算時間間隔 Δt が大きくなると計算が発散することがある。本書では計算の安定が保証された後退差分式によるモデルを前提として以降の解説を行う。

$$Q=KA(T_b-T) \tag{22.16}$$

$$C\frac{T-T^*}{\Delta t}=KA(T_b-T^*) \tag{22.17}$$

$$C\frac{T-T^*}{\Delta t}=KA(T_b-T) \tag{22.18}$$

3）多層壁の熱収支式

壁は、コンクリート、断熱材、仕上げ材など、複数の材料から層状に構成されていることが多い。このような層状の壁を多層壁と呼ぶ。層によって形成される温度の分布を表現するために、図 22.4 に示すように層と層との境界および両側の表面に全部で M 個の質点をとるモデルとする†1)。壁平面に垂直な方向の熱流のみを考慮し、平面内の温度分布を考慮しないため、このようなモデルを一次元熱伝導モデルと呼ぶ。便宜上、図の左側を表(Front)、右側を裏(Back)とし、添字の F と B で使い分けることとする。各質点の熱収支は式 22.19 で表現できる。ただし、C_{SWal} [J/(m²·K)]は質点の熱容量であり†2)、質点の左右の材料の容積比熱 $c\rho$ [kJ/(m³·K)]および厚み d [m]を用いて式 22.22 で計算する。また R_{Wal} [(m²·K)/W]は熱抵抗であり、壁材料の熱伝導率 λ [W/(m·K)]あるいは壁表面の総合熱伝達率 $\alpha_{(c+r)}$ を用いて式 22.23 で計算する。H_S [W/m²]は放射冷暖房システムの場合に冷温水配管や電熱線によって特

†1　図は壁の場合だが、床や天井の場合にも 90º 回転させるだけで同じモデルを用いる。
†2　式 22.14 と単位が異なるが、一次元熱伝導のため、単位面積あたりで計算モデルを立てているためである。

定の層に直接に伝えられる熱流であり、詳細は後述する。壁の両端（m=0、m=M）の場合には隣接温度条件が壁ではなく空気の相当温度 T_{SolWal} [K]となるため、式 22.20 と式 22.21 となる。22.2.1 節で解説したとおり、周囲の物体の温度が空気温度と概ね等しく、その他の大きな放射を受けない壁に関しては空気温度としてもよい。

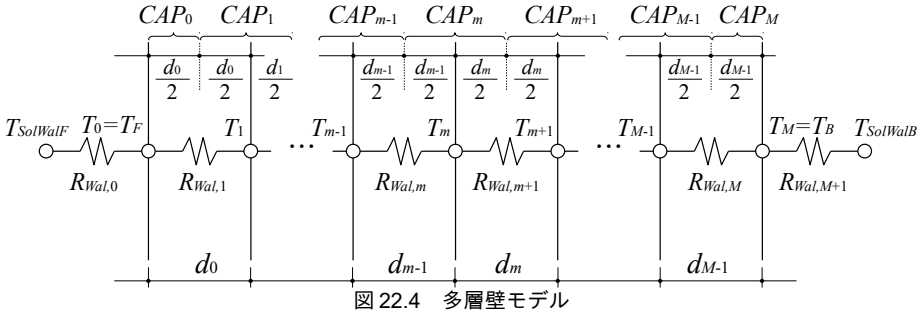


図 22.4　多層壁モデル

$$C_{SWal,m}\frac{dT_m}{dt}=\frac{T_{m-1}-T_m}{R_{Wal,m}}+\frac{T_{m+1}-T_m}{R_{Wal,m+1}}+H_{S,m} \tag{22.19}$$

$$C_{SWal,0}\frac{dT_0}{dt}=\frac{T_{SolWalF}-T_0}{R_{Wal,0}}+\frac{T_1-T_0}{R_{Wal,1}}+H_{S,0} \tag{22.20}$$

$$C_{SWal,M}\frac{dT_M}{dt}=\frac{T_{M-1}-T_M}{R_{Wal,M}}+\frac{T_{SolWalB}-T_M}{R_{Wal,M+1}}+H_{S,M} \tag{22.21}$$

$$C_{SWal,m}=0.5\left(c\rho_m d_{m-1}+c\rho_{m+1}d_m\right)\times 1000 \tag{22.22}$$

$$R_{Wal,m}=\begin{cases}1/\alpha_{(c+r),F} & (m=0)\\ 1/\alpha_{(c+r),B} & (m=M)\\ d_m/\lambda_m & (\text{other})\end{cases} \tag{22.23}$$

建築熱負荷計算で用いることが多い主要な壁材料の熱伝導率 λ と容積比熱 $c\rho$ を表 22.1 に示す†1)。多層壁の場合には設定する値が多いが、熱容量の大きい壁層（コンクリートやブロックなど）と熱抵抗の大きい壁層（断熱材や中空層など）を正しく設定することが大切である。逆に言えば熱容量も熱抵抗も小さい壁層（カーペットやビニールクロスなど）に関しては無視してもあまり影響はない。

表 22.1　主要な壁材料の熱定数

材料名	熱伝導率 λ [W/(m·K)]	容積比熱 $c\rho$ [kJ/(m³·K)]	材料名	熱伝導率 λ [W/(m·K)]	容積比熱 $c\rho$ [kJ/(m³·K)]
PC コンクリート	1.5	1,900	アスファルト	0.11	920
普通コンクリート	1.4	1,900	畳	0.15	290
軽量コンクリート	0.78	1,600	木材（中量）	0.17	650
ALC	0.17	650	合板	0.19	720
モルタル	1.5	1,600	グラスウール（24K）	0.042	20
石膏ボード	0.17	1,000	岩綿吸音板	0.064	250
ガラス	1.0	1,900	スチレン発泡板（押出）	0.037	35
タイル	1.3	2,000	硬質ウレタン発泡板	0.028	47

やや特殊な壁層として中空層がある。空気であるため、中空層を挟む両側の壁層は放射によって直接に熱をやりとりする。また、中空層内部の空気が全く動かなければ空気の熱伝導率を用いて通常の壁層と同じ方法で熱流を計算すればよいが、現実には温度分布による空気循環が発生する。従って厳密には、中空層を挟む壁層表面の放射率と温度、熱流の向き、中空層の傾きなどに影響を受けて熱抵抗は変化するが、実用計算上は非密閉中空層で 0.07 m²·K/W、密閉中空層で 0.15 m²·K/W 程度として一定値とすることが多い。

†1　この他の材料の物性に関しては参考文献 22.10 などを参照。

4）差分化と行列表現

式 22.19 を差分化すると式 22.24 の後退差分式が得られる†1)。u_{Fm} と u_{Bm} を式 22.25 で表現し、式 22.24 を現在の温度 T^* について解くと式 22.26 が得られ、これを行列表現すると式 22.27 となる。ただし[TH^*]は現在時点で得られる条件（壁温、外部からの投入熱量、相当温度など）により式 22.28 で表現されるベクトル、[T]は将来の未知の壁温ベクトルであり、いずれも要素数は M である。[U]は式 22.29 で示される M×M の係数行列である。[U]の逆行列を[UX]とすれば、式 22.30 により現在の条件[TH^*]から将来時点の温度[T]を得ることができる。なお、顕熱容量 C_{SWal} が 0 の場合（空気層など）には式 22.25 が発散してしまうが、式 22.24 に C_{SWal}=0 を代入してみると、式 22.25 において $\Delta t/C_{SWal,m}$=1、式 22.26 において T_m^*=T_m とすれば良いということがわかる†2)。

$$C_{SWal,m}\frac{T_m-T_m^*}{\Delta t}=\frac{T_{m-1}-T_m}{R_{Wal,m}}+\frac{T_{m+1}-T_m}{R_{Wal,m+1}}+H_{S,m} \tag{22.24}$$

$$u_{F,m}=\frac{\Delta t}{C_{SWal,m}R_{Wal,m}} \quad , \quad u_{B,m}=\frac{\Delta t}{C_{SWal,m}R_{Wal,m+1}} \tag{22.25}$$

$$T_m^*=-u_{F,m}T_{m-1}+T_m\left(1+u_{F,m}+u_{B,m}\right)-u_{B,m}T_{m+1}-\frac{H_{S,m}\Delta t}{C_{SWal,m}} \tag{22.26}$$

$$[\boldsymbol{TH^*}]=[\boldsymbol{U}][\boldsymbol{T}] \tag{22.27}$$

$$TH_m^*=\begin{cases} T_0^*+\dfrac{H_{S,0}\Delta t}{C_{SWal,0}}+u_{F,0}T_{SolWalF} & (m=0) \\ T_M^*+\dfrac{H_{S,M}\Delta t}{C_{SWal,M}}+u_{B,M}T_{SolWalB} & (m=M) \\ T_m^*+\dfrac{H_{S,m}\Delta t}{C_{SWal,m}} & (\text{other}) \end{cases} \tag{22.28}$$

$$U_{m,n}=\begin{cases} 1+u_{F,m}+u_{B,m} & (m=n) \\ -u_{F,m} & (n=m-1) \\ -u_{B,m} & (n=m+1) \\ 0 & (\text{other}) \end{cases} \tag{22.29}$$

$$[\boldsymbol{T}]=[\boldsymbol{UX}][\boldsymbol{TH^*}] \tag{22.30}$$

単体の壁体の熱流計算であれば式 22.30 で十分であるが、室温計算を行う場合には問題が生じる。ある壁の熱流を解くためにはその壁の相当温度を知る必要があるが、相当温度は他の壁の表面温度に依存するため、すべての壁の温度を同時に解く必要があるからである。この計算法の詳細は第 25 章で解説することとし、とりあえずここでは壁体単体の壁表面温度を相当温度で表現しておく。式 22.30 から壁表面温度（m=0 と m=M）の行を取り出し、壁両端の相当温度で表現すると式 22.31 と式 22.32 が得られる。ただし IF は壁体内部の現在の温度に依存する項、FF は表側の相当温度に依存する項、BF は裏側の相当温度に依存する項であり、それぞれ式 22.33~22.35である。

$$T_0=T_F=IF_F+FF_FT_{SolF}+BF_FT_{SolB} \tag{22.31}$$

$$T_M=T_B=IF_B+FF_BT_{SolF}+BF_BT_{SolB} \tag{22.32}$$

†1 本書では差分法による集中定数系モデルを解説するが、壁体熱流の計算法としては分布定数系モデルである応答係数法がよく知られている[22.2) 22.3) 22.9)]。計算速度が非常に速いため、HASP/ACLD を始め、多くのシミュレーションプログラムで採用されている。一方で、相変化材料、放射冷暖房、熱水分同時移動などの問題への対応は難しい。

†2 少し迂遠な表現になっているが、T_m^*=T_m とするということは結局のところ、左辺を 0、右辺の$(1+u_{F,m}+u_{B,m})$を$(u_{F,m}+u_{B,m})$にするということである。

$$IF_F=\sum_{n=0}^{M} UX_{0,n}\left(T_n^*+\frac{H_{S,n}\Delta t}{C_{SWal,n}}\right) \quad , \quad IF_B=\sum_{n=0}^{M} UX_{M,n}\left(T_n^*+\frac{H_{S,n}\Delta t}{C_{SWal,n}}\right) \tag{22.33}$$

$$FF_F=UX_{0,0}u_{F,0} \quad , \quad FF_B=UX_{M,0}u_{F,0} \tag{22.34}$$

$$BF_F=UX_{0,M}u_{B,M} \quad , \quad BF_B=UX_{M,M}u_{B,M} \tag{22.35}$$

22.2.3　相変化材料

近年、室内の温熱環境を安定化させる目的から、相変化を利用して蓄熱を行う相変化材料（PCM: Phase Change Material）が用いられることがある。このような場合には材料の物性値が変化するため、上記のモデルに修正を加える必要がある[22.7) 22.8)]。

ある温度を基準状態とした時に、PCM の入熱量と温度との関係を図 22.5 のようにモデル化する。入熱にともなって温度が上昇するが、二相域においては液体から固体への状態変化に熱が費やされるため、温度上昇幅が小さくなる。そこで、二相域においては容積比熱 $c\rho_{tp}$ [kJ/(m³·K)]が大きな値に変化したとみなす。なお、添字の *sl* は固相（solid）、*tp* は二相（two phase）、*lq* は液相（liquid）、*fp* は凝固点（freezing point）、*mt* は融点（melting point）を表す。なお、材料によっては、固体から液体への相変化が生じる温度と、液体から固体への相変化が生じる温度とが異なる場合がある。このような性質をヒステリシスと呼ぶが、本モデルではこの現象は無視している。

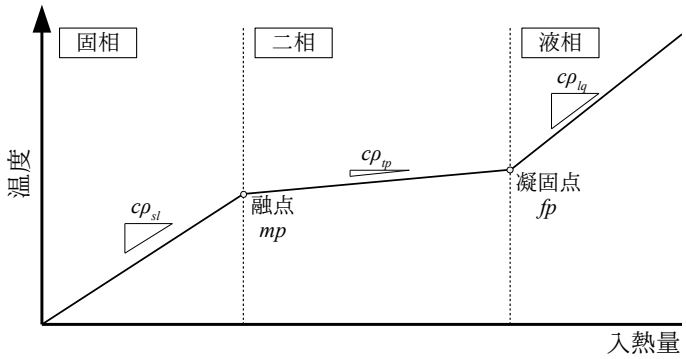


図 22.5　PCM の入熱量と温度の関係

固相・二相・液相が変化すると熱伝導率と温度上昇幅も変化するため、正確に計算を行うためには融点および凝固点で物性を変更する必要がある。しかし、このためにはタイムステップを変化させた反復計算が必要となる。そこで簡易化のため、一旦、タイムステップと物性は固定したまま温度を更新し、状態変化が生じた場合には誤差に相当する熱量を割り戻して補正するという手法を取る。例えば計算の結果、壁材料の温度が T となったとする。この時、相変化温度（凝固点または融点）を超えて変化した場合には、式 22.36 で T_{cor} に補正する。ただし C_{Spcm} [J/(m²·K)]は従前の相における PCM の熱容量、$C_{Spcm\text{-}cor}$ [J/(m²·K)]は補正後の相における PCM の熱容量であり、T_{PC} [K]は相変化が生じる温度である。本書のモデルでは図 22.4 に示したように質点を 2 つの壁材料の境界にとるため、2 種類の壁材料で熱容量が定まる。従って、壁の両端（単一材料）であれば単純に式 22.36 を適用すれば良いが、その他の部位に関しては PCM 以外のもう 1 つの壁材料の影響を表現する必要がある。第 *m* 層の熱容量 $C_{SWal,m}$ が PCM 材料側の熱容量 $C_{Spcm,m}$ と一般壁材料側の熱容量 $C_{SWal,m}$ の和で式 22.37 で表されているとき、補正後の温度 $T_{m,cor}$ は式 22.38 で計算できる。

$$T_{cor}=T_{PC}+\frac{C_{Spcm}}{C_{Spcm-cor}}\left(T-T_{PC}\right) \tag{22.36}$$

$$C_{SWal,m}=C_{Spcm,m}+C_{SWal,m} \tag{22.37}$$

$$T_{m,cor}=\frac{T_m\left(C_{Spcm,m}+C_{SWal,m}\right)+T_{m,PC}\left(C_{Spcm-cor,m}-C_{Spcm,m}\right)}{C_{Spcm-cor,m}+C_{SWal,m}} \tag{22.38}$$

22.2.4 放射冷暖房システム

1）放射冷暖房システムの方式

放射冷暖房システムには、電熱線を埋め込む方式（この場合には加熱しかできないため、放射暖房システムとなる）と冷温水管を埋め込む方式とがある。電熱線の場合には単位長さあたりの発熱量がわかるため、式 22.19 の H_S にはその値を直接に代入することができる。一方、冷温水管の場合には配管と壁内部の温度差によって熱流が変化するため、モデルを修正する必要がある。

2）差分化式への組み込み

冷温水管は通常は適当なピッチで床や天井に敷設されるため、水平方向に温度分布が形成される。この影響をフィン効率 η_P [-]を用いて表現すると、定常状態においては式 22.39 が成立する†1)。ただし T_m は m 層の平均温度、$T_{P,m}$ [K]は m 層に敷設された冷温水管表面の平均温度である。これを $T_{P,m}$ について解くと式 22.40 となる。

$$\eta_{PF,m}\left(\frac{T_{m-1}-T_{P,m}}{R_{Wal,m}}\right)+\eta_{PB,m}\left(\frac{T_{m+1}-T_{P,m}}{R_{Wal,m+1}}\right)=\frac{T_{m-1}-T_m}{R_{Wal,m}}+\frac{T_{m+1}-T_m}{R_{Wal,m+1}} \tag{22.39}$$

$$T_{P,m}=-\frac{R_{Wal,m+1}\left(1-\eta_{PF,m}\right)T_{m-1}-\left(R_{Wal,m}+R_{Wal,m+1}\right)T_m+R_{Wal,m}\left(1-\eta_{PB,m}\right)T_{m+1}}{R_{Wal,m+1}\eta_{PF,m}+R_{Wal,m}\eta_{PB,m}} \tag{22.40}$$

一方、冷温水管と冷温水との間での交換熱量 $H_{S,m}$ は熱通過有効度 ε_P [-]を用いて式 22.41 で表現する。ここで A [m^2]は壁の面積、m_w [kg/s]は冷温水流量、c_{pw} [J/(kg·K)]は冷温水の定圧比熱、T_{wi} [K]は冷温水の入口水温である。熱通過有効度 ε_P は、配管外表面温度を一定とみなせば†2)第 8 章で解説した片側温度一定の熱交換器として式 8.21 と式 8.23 で計算できる。ただし、伝熱係数 KA は配管内流体から配管外表面までの伝熱係数であり、式 22.42 で計算する。L [m]は配管長、d_{Pi} [m]は配管内径、d_{Po} [m]は配管外径、λ_P [W/(m·K)]は配管の熱伝導率、α_w [W/(m^2·K)]は配管内対流熱伝達率である。

$$H_{S,m}=\frac{c_{pw}m_{w,m}\varepsilon_{P,m}}{A}\left(T_{wi,m}-T_{P,m}\right) \tag{22.41}$$

$$\frac{1}{KA}=\frac{1}{L\pi}\left(\frac{1}{d_{Pi}\alpha_w}+\frac{1}{2\lambda_P}\ln\left(d_{Po}/d_{Pi}\right)\right) \tag{22.42}$$

式 22.41 に式 22.40 を代入して冷温水管表面の温度 $T_{P,m}$ を消去すると、式 22.43 に示すように $H_{S,m}$ を壁層の温度 T_m と入口水温 T_{wi} の線形式で表現できる。ただし、壁の両端においては T_{m-1} と T_{m+1} はそれぞれ相当温度 T_{Sol} とする。

$$H_{S,m}=\frac{c_{pw}m_{w,m}\varepsilon_{P,m}}{A}\left(T_{wi,m}+\frac{R_{Wal,m+1}\left(1-\eta_{PF,m}\right)T_{m-1}-\left(R_{Wal,m}+R_{Wal,m+1}\right)T_m+R_{Wal,m}\left(1-\eta_{PB,m}\right)T_{m+1}}{R_{Wal,m+1}\eta_{PF,m}+R_{Wal,m}\eta_{PB,m}}\right) \tag{22.43}$$

式 22.43 を式 22.19~22.21の熱収支式に反映すると、式 22.28 と式 22.29 はそれぞれ式 22.44 と式 22.45 になる。

---

†1 フィン効率の計算に関しては第 9 章でも解説した。ただし、このモデルだとフィン効率が低い場合に不合理な計算結果（冷房時に $T_{P,m}$ が $T_{Wi,m}$ を下回る、暖房時に $T_{P,m}$ が $T_{Wi,m}$ を上回る）が出ることがあることに注意する必要がある。

†2 従って、床や壁内における配管系路にそった温度分布を十分に表現したい場合には、床や壁をいくつかの部位に分割する必要がある。

$$TH_m^* = \begin{cases} T_0^* + u_{P,0} T_{wi,0} + (u_{F,0} + u_{PF,0}) T_{SolWalF} & (m=0) \\ T_M^* + u_{P,M} T_{wi,M} + (u_{B,M} + u_{PB,M}) T_{SolWalB} & (m=M) \\ T_m^* + u_{P,m} T_{wi,m} & (\text{other}) \end{cases} \tag{22.44}$$

$$U_{m,n} = \begin{cases} 1 + u_{F,m} + u_{B,m} + u_{PM,m} & (m=n) \\ -(u_{F,m} + u_{PF,m}) & (n=m-1) \\ -(u_{B,m} + u_{PB,m}) & (n=m+1) \\ 0 & (\text{other}) \end{cases} \tag{22.45}$$

$$u_{P,m} = \frac{c_{pw} m_{w,m} \varepsilon_{P,m} \Delta t}{C_{SWal,m} \cdot A} \tag{22.46}$$

$$u_{PF,m} = u_{P,m} \frac{R_{Wal,m+1}(1-\eta_{PF,m})}{R_{Wal,m+1}\eta_{PF,m} + R_{Wal,m}\eta_{PB,m}} \tag{22.47}$$

$$u_{PM,m} = u_{P,m} \frac{R_{Wal,m} + R_{Wal,m+1}}{R_{Wal,m+1}\eta_{PF,m} + R_{Wal,m}\eta_{PB,m}} \tag{22.48}$$

$$u_{PB,m} = u_{P,m} \frac{R_{Wal,m}(1-\eta_{PB,m})}{R_{Wal,m+1}\eta_{PF,m} + R_{Wal,m}\eta_{PB,m}} \tag{22.49}$$

床冷暖房システムの場合には式 22.31 と式 22.32 の *IF*、*FF*、*BF* はそれぞれ式 22.50~22.52となる。

$$IF_F = \sum_{n=0}^{M} UX_{0,n}\left(T_n^* + u_{P,n} T_{wi,n}\right) \quad , \quad IF_B = \sum_{n=0}^{M} UX_{M,n}\left(T_n^* + u_{P,n} T_{wi,n}\right) \tag{22.50}$$

$$FF_F = UX_{0,0}(u_{F,0} + u_{PF,0}) \quad , \quad FF_B = UX_{M,0}(u_{F,0} + u_{PF,0}) \tag{22.51}$$

$$BF_F = UX_{0,M}(u_{B,M} + u_{PB,M}) \quad , \quad BF_B = UX_{M,M}(u_{B,M} + u_{PB,M}) \tag{22.52}$$

行列計算で各層の温度が得られれば、式 22.53（式 22.40 と式 22.41 がら導出）を用いて冷温水管からの熱流 H_S が求められる。また、熱流 H_s を式 22.54 に代入すれば冷温水出口温度 T_{wo} [K]を計算することができる。

$$H_{S,m} = \frac{C_{SWal,m}}{\Delta t}\left(u_{P,m} T_{wi,m} + u_{PF,m} T_{m-1} - u_{PM,m} T_m + u_{PB,m} T_{m+1}\right) \tag{22.53}$$

$$T_{wo,m} = T_{wi,m} - \frac{H_{S,m} A}{c_{pw} m_{w,m}} \tag{22.54}$$

3）フィン効率の計算法

フィン効率は図 22.6 のようにモデル化する。図 22.6 左に示すような形状の直線フィンのフィン効率は式 22.55、式 22.56 によって表され[22.5)]、フィン材料の熱伝導率 λ とフィン表面からの放熱抵抗 R（熱伝達率の逆数）に依存する。ここで、図 22.6 右に示すように、第 m 層の壁（床）内に埋設された配管をフィンの根本部分ととして捉えると、配管外径 d_{Po} の範囲はフィン効率は 1.0 であり、配管敷設ピッチ w_P [m]から配管外径 d_{Po} を差し引いた範囲では温度分布が生じるためにフィン効率は 1.0 以下となる。m-1 層側と m+1 層側では熱伝導率 λ と抵抗係数 R が異なるため、m-1 層側のフィン効率を η_{PL}、m+1 層側のフィン効率を η_{PR} として別々に計算する†1)。式 22.56 の y_b は配管外径 $d_{Po,m}$ とするが、壁層の厚み d_m が $d_{Po,m}$ を下回る場合には d_m とする†2)。以上により、m 層のフィン効率 η_{PL} と η_{PR} は式 22.57~22.60で表現される。なお、壁の両端表面の場合にはフィンの熱伝導率として隣接する壁材料の熱伝導率を用いることとする（m=0 では λ_0 を、m=M では λ_M を用いる）。また、この場合に壁層の厚

†1　両面を合わせて一体のフィンとして計算する方法については文献 22.4 を参照。

†2　厳密にはフィンの厚みとみなした幅に関しては壁層間の抵抗 R を減じる必要があるが、計算が複雑になるため、この補正は行わない。

み d_m が $d_{Po,m}$ を下回るならば y_b=0.5d_m とする。

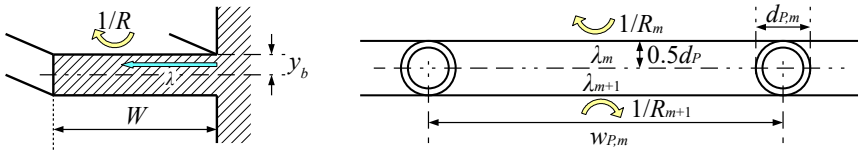


図 22.6　埋設冷温水管のフィン効率のモデル化

$$\eta_P=\frac{\tanh u_b}{u_b} \tag{22.55}$$

$$u_b=W\frac{1}{\sqrt{R\lambda y_b}} \tag{22.56}$$

$$\eta_{PL,m}=\frac{1}{w_{P,m}}\left\{d_{Po,m}+(w_{P,m}-d_{Po,m})\frac{\tanh u_{bL,m}}{u_{bL,m}}\right\} \tag{22.57}$$

$$u_{bL,m}=\frac{w_{P,m}-d_{P,m}}{2}\frac{1}{\sqrt{R_m\lambda_m\min(d_{P,m},d_{m-1})}} \tag{22.58}$$

$$\eta_{PR,m}=\frac{1}{w_{P,m}}\left\{d_{Po,m}+(w_{P,m}-d_{Po,m})\frac{\tanh u_{bR,m}}{u_{bR,m}}\right\} \tag{22.59}$$

$$u_{bR,m}=\frac{w_{P,m}-d_{P,m}}{2}\frac{1}{\sqrt{R_{m+1}\lambda_{m+1}\min(d_{P,m},d_m)}} \tag{22.60}$$

【例題 22.1】

コンクリート内に冷温水配管を埋設した床冷暖房システムについて、熱通過有効度 ε_P と上側のフィン効率 η_{PL} を計算せよ。図 22.7 に示すように冷温水は 100L/min で流入した後、250mm ピッチで並んだ 10 の並列回路に分岐するものとする。コンクリートの厚みは 200mm とし、配管は上側表面から 50mm の位置に埋設しているものとする。配管内径 d_{Pi} は 20mm、配管外径 d_{Po} は 23mm、配管材料は架橋ポリエチレンとする。コンクリートとポリエチレンの熱伝導率 λ はそれぞれ 1.6 W/(m·K)と 0.47 W/(m·K)とする。

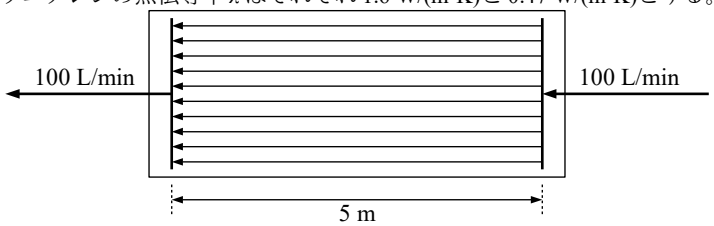


図 22.7　床冷暖房配管敷設平面

【解】

円管内の対流熱伝達の計算については第 3 章の例題 3.1 を参照。図 22.7 から 1 本あたりの水量は 10 L/min であり、水温を 20 ℃ とすると本間の場合には α_w = 4,330 W/(m²·K)となる。式 22.42 により、

伝熱係数 KA = $\left[\frac{1}{50\pi}\left(\frac{1}{0.02\times4330}+\frac{1}{2\times0.47}\ln(23/20)\right)\right]^{-1}$ = 1,000 W/K

となる。これを式 8.21 に代入すると、移動単位数 NTU=1,000÷(4186×10÷60)=1.43 となる。さらに NTU を式 8.23 に代入し、ε_P=1-exp(-1.43)=0.761 となる。

式 22.23 により上側の熱抵抗 R_m=0.05÷1.6=0.031 (m²·K)/W である。これを式 22.58 に代入すると、

$$u_{bL,m}=\frac{0.25-0.023}{2}\frac{1}{\sqrt{0.031\times1.6\times\min(0.023,0.05)}}=3.36$$

となる。これをさらに式 22.57 に代入し、

$$\eta_{PL,m}=\frac{1}{0.25}\left\{0.023+(0.25-0.023)\frac{\tanh 3.36}{3.36}\right\}=0.36$$ となる。

22.2.5　熱・水分同時移動解析

1）熱・水分移動方程式

建築壁体の内部では熱が移動するだけではなく、水分も移動して室に対して吸放湿が行われる。この現象を捉えるためには水分の蒸発・凝縮熱を含めて壁体内の熱流を解く必要があり、これを熱・水分同時移動解析と呼ぶ。計算に必要となる材料物性値の整備が、顕熱移動の計算に比較すると十分ではなく、また計算自体もやや煩雑なため、水分移動を表現しない熱負荷計算ソフトウェアも多い。しかし、木材を多用した建築など、水分容量の大きい建物の調湿機能を評価するなどの場合には必要となる計算である†1)。

熱・水分移動方程式を式 22.61~22.63に示す†2)。式 22.61 は壁層の温度変化に関する式であり、式 22.19 に似ているが、右辺第 4 項に壁層内に存在する空気の絶対湿度変化に伴う潜熱発生の影響が追加されている。式 22.62 は壁層の絶対湿度変化に関する微分方程式であり、W [kg/kg]は絶対湿度、R_{LWal} [(kg/kg)·m²/(kg/s)]は透湿抵抗である。C_{LWal} [kg/m²]は壁層の水分容量であり、壁層の空隙率 C_{ma} [kg/kg]を用いて式 22.63 で計算する。γ は水の蒸発潜熱、ρ_{da} は乾燥空気の密度であり、本モデルでは 2,500×10³ J/kg と 1.2 kg/m³ で固定値とする。なお、添字 m の付け方は図 22.4 と同じである。

$$\left(C_{SWal,m}+\gamma C_{v,m}\right)\frac{dT_m}{dt}=\frac{T_{m-1}-T_m}{R_{Wal,m}}+\frac{T_{m+1}-T_m}{R_{Wal,m+1}}+H_{S,m}+\gamma C_{\kappa,m}\frac{dW_m}{dt} \tag{22.61}$$

$$\left(C_{LWal,m}+C_{\kappa,m}\right)\frac{dW_m}{dt}=\frac{W_{m-1}-W_m}{R_{LWal,m}}+\frac{W_{m+1}-W_m}{R_{LWal,m+1}}+C_{v,m}\frac{dT_m}{dt} \tag{22.62}$$

$$C_{LWal,m}=0.5\rho_{da}\left(C_{ma,m}d_{m-1}+C_{ma,m+1}d_m\right) \tag{22.63}$$

ある温湿度条件で十分な時間を経過させると物体（壁材料）の含水率は平衡状態に達し、この時の含水率を平衡含水率と呼ぶ。式 22.61 と 22.62 における $C_{v,m}$ [kg/(m²·K)]と $C_{\kappa,m}$ [kg/m²]は平衡含水率の温湿度依存性を表現するための項であり、式 22.64 で計算する。平衡含水率を温度 T と絶対湿度 W の関数として式 22.65 で表現すると、式 22.64 の ν と κ は理論的には式 22.66 である。しかしこのまま微分値として取り扱うと式 22.61 と 22.62 が解きづらいため、κ と ν は一定値をとるものと仮定して線形式として解く†3)。それぞれの壁材料の平衡含水率曲線（式 22.65）が得られれば、式 22.66 を計算して ν と κ の値を得ることができるが、これらの物性値の整備は十分ではない。参考に、松本らが文献中で採用した値を表 22.2 に挙げておく†4)。なお、式 22.61 は吸放湿を含めて式 22.19 を一般化させた式であり、平衡含水率が温度や絶対湿度に依存しない場合（κ=ν=0）には、式 22.19 と一致する。

$$C_{v,m}=0.5\left(\nu_m d_{m-1}+\nu_{m+1}d_m\right)\quad,\quad C_{\kappa,m}=0.5\left(\kappa_m d_{m-1}+\kappa_{m+1}d_m\right) \tag{22.64}$$

$$\varphi=f(T,W) \tag{22.65}$$

$$\nu=-\frac{\partial\varphi}{\partial T}\quad,\quad \kappa=\frac{\partial\varphi}{\partial W} \tag{22.66}$$

†1　尾崎らは湿気容量の大きい煉瓦造実験住宅の実測結果とシミュレーションの結果を比較し、水分容量を無視することにより生じる誤差の大きさについて報告している[22.14)]。

†2　本式は内部結露などが無く、水が気体として移動する（蒸気拡散支配領域:Hygroscopic region）ことを仮定して導出された式である。液相水分移動を含めた一般化した表現については松本による参考文献 22.12 を参照。松本によれば相対湿度 95 %程度までの範囲であれば、液相水分移動を含めた正確解からの誤差は小さい[22.16)]。また、鉾井は液体として水が浸透した状態における熱流について研究を行っている[22.11)]。

†3　松本による提案である[22.12)]。松本は木繊維板、木材、発泡軽量コンクリートの計算を行い、このような仮定を設けることによる誤差について報告している。

†4　土壁、スギ集成材、高強度コンクリートの平衡含水率に関してはそれぞれ高田ら、大橋ら、李らの報告[22.19)] [22.20)] [22.21)]がある。

表 22.2　空隙率・ν・κの値

材料	空隙率 C_{ma}	ν	κ
木繊維板（相対湿度 35~75 %）	0.788	1.715	3080
木繊維板（相対湿度 55~85 %）		3.59	4610
木材	0.793	256	4710
発泡軽量コンクリート（相対湿度 57~75 %）	0.780	0.4	700
発泡軽量コンクリート（相対湿度 75~95 %）		7.1	6700

各層の透湿抵抗は式 22.67 で計算する。壁表面の抵抗は湿気伝達率 α_L [(kg/s)/((kg/kg)·m^2)]の逆数であり、ルイスの係数 L_e を用いて対流熱伝達率 $\alpha_{(c)}$から求められる†1)。λ_L [(kg/s) / ((kg/kg)·m)]は湿気伝導率であり、代表的な壁材料の物性値を表 22.3 に示す。

$$R_{LWal,m}=\begin{cases}1/\alpha_{L,F}=c_{pma}L_e/\alpha_{(c),F} & (m=0)\\ 1/\alpha_{L,B}=c_{pma}L_e/\alpha_{(c),B} & (m=M)\\ d_m/\lambda_{L,m} & (\text{other})\end{cases} \tag{22.67}$$

表 22.3　代表的な壁材料の湿気伝導率 λ_L[22.15)]

材料	湿気伝導率 [(kg/s) / ((kg/kg)·m)]	材料	湿気伝導率 [(kg/s) / ((kg/kg)·m)]
コンクリート	0.00044	合板	0.00044
気泡コンクリート	0.013	防湿膜（ビニルなど）	1.5×10^{-6}
モルタル	0.00064	断熱材	0.00064

2）差分化式への組み込み

式 22.61 と 22.62 を差分化すると式 22.68 と式 22.69 が得られる。両式を連立させて現在時点の温度 T^*と絶対湿度 W^*について整理すると式 22.70 と式 22.71 となる。ただし式中の係数は式 22.72、22.73 に示す通りである。$C_{SWal,m}=C_{v,m}=0$ の場合には式 22.70 と式 22.71 の右辺が発散するが、この場合は $C_{SWal,m}=\Delta t=1$ とし、式 22.70 で $T_m^*=T_m$ とすればよい。同様に $C_{LWal,m}=\gamma C_{\kappa,m}=0$ の場合には、$C_{LWal,m}=\Delta t=1$ とし、式 22.71 で $W_m^*=W_m$とする。また、$C_{SWal,m}=C_{LWal,m}=0$ の場合にも式 22.70 と式 22.71 の右辺が発散する。しかし顕熱容量が 0 に近いということはそれだけ空隙率が高く水分容量が大きいということであるから、真空の中空層でないかぎりはこのような組み合わせは発生しない†2)。

$$\left(C_{SWal,m}+\gamma C_{v,m}\right)\frac{T_m-T_m^*}{\Delta t}=\frac{T_{m-1}-T_m}{R_{Wal,m}}+\frac{T_{m+1}-T_m}{R_{Wal,m+1}}+H_{S,m}+\gamma C_{\kappa,m}\frac{W_m-W_m^*}{\Delta t} \tag{22.68}$$

$$\left(C_{LWal,m}+C_{\kappa,m}\right)\frac{W_m-W_m^*}{\Delta t}=\frac{W_{m-1}-W_m}{R_{LWal,m}}+\frac{W_{m+1}-W_m}{R_{LWal,m+1}}+C_{v,m}\frac{T_m-T_m^*}{\Delta t} \tag{22.69}$$

$$\begin{aligned}T_m^*=&-u_{SF2,m}T_{m-1}+T_m\left(1+u_{SF2,m}+u_{SB2,m}\right)-u_{SB2,m}T_{m+1}\\&-u_{LF2,m}W_{m-1}+W_m\left(u_{LF2,m}+u_{LB2,m}\right)-u_{LB2,m}W_{m+1}-\left(C_{LWal,m}+C_{\kappa,m}\right)/C_{SL,m}H_{S,m}\Delta t\end{aligned} \tag{22.70}$$

$$\begin{aligned}W_m^*=&-u_{SF3,m}T_{m-1}+T_m\left(u_{SF3,m}+u_{SB3,m}\right)-u_{SB3,m}T_{m+1}\\&-u_{LF3,m}W_{m-1}+W_m\left(1+u_{LF3,m}+u_{LB3,m}\right)-u_{LB3,m}W_{m+1}\end{aligned} \tag{22.71}$$

$$\begin{aligned}&u_{SF2,m}=\frac{\Delta t}{C_{SL,m}R_{Wal,m}}\left(C_{LWal,m}+C_{\kappa,m}\right)\qquad u_{SB2,m}=\frac{\Delta t}{C_{SL,m}R_{Wal,m+1}}\left(C_{LWal,m}+C_{\kappa,m}\right)\\&u_{LF2,m}=\frac{\Delta t}{C_{SL,m}R_{LWal,m}}\gamma C_{\kappa,m}\qquad u_{LB2,m}=\frac{\Delta t}{C_{SL,m}R_{LWal,m+1}}\gamma C_{\kappa,m}\\&u_{SF3,m}=\frac{\Delta t}{C_{SL,m}R_{Wal,m}}C_{v,m}\qquad u_{SB3,m}=\frac{\Delta t}{C_{SL,m}R_{Wal,m+1}}C_{v,m}\\&u_{LF3,m}=\frac{\Delta t}{C_{SL,m}R_{LWal,m}}\left(C_{SWal,m}+\gamma C_{v,m}\right)\qquad u_{LB3,m}=\frac{\Delta t}{C_{SL,m}R_{LWal,m+1}}\left(C_{SWal,m}+\gamma C_{v,m}\right)\end{aligned} \tag{22.72}$$

†1　ルイスの係数と湿気伝達率については第 12 章で解説した。
†2　机上計算としては対応しておきたい組み合わせであるが、簡単なプログラムコードで実装することが困難であった。

$$C_{SL,m}=C_{SWal,m}C_{LWal,m}+C_{SWal,m}C_{\kappa,m}+\gamma C_{\nu,m}C_{LWal,m} \tag{22.73}$$

式 22.70 と式 22.71 に放射冷暖房システムの際に導出した式 22.53 を代入して行列表示すると式 22.74~22.76が得られる。ただし、u_{P2}は式 22.77 で計算し、これを式 22.47~式 22.49 の u_Pに代入して得られた値が u_{PF2}、u_{PB2}、u_{PM2}である。

[*TWH*]は現在の条件から得られる要素数 2*M* のベクトルであり、*m* 番目の要素は式 22.75 で示される。[*TW*]は将来時点の温度を 0~*M* 番目の要素に、将来時点の絶対湿度を *M*+1~2*M* の要素にとるベクトルである。[*UW*]は 2*M*×2*M* の係数行列であり、*m* 行 *n* 列の要素は式 22.76 で計算する。また変数 *p* と *q* はそれぞれ *M*<*m*、*M*<*n* の範囲で意味を持ち、*p*=*m*-*M*+1、*q*=*n*-*M*+1 である。[*UW*]の逆行列[*UWX*]を求めれば、これを[*TWH*]に対して左側から乗ずることで将来時点の温湿度ベクトル[*TW*]を計算することができる。

$$[\boldsymbol{TWH^*}]=[\boldsymbol{UW}][\boldsymbol{TW}] \tag{22.74}$$

$$TWH_m^*=\begin{cases} T_0^*+u_{P2,0}T_{wi,0}+(u_{SF2,0}+u_{PF2,0})T_{SolWalF}+u_{LF2,0}W_{maF} & (m=0) \\ T_M^*+u_{P2,M}T_{wi,M}+(u_{SB2,M}+u_{PB2,M})T_{SolWalB}+u_{LB2,0}W_{maB} & (m=M) \\ T_m^*+u_{P2,m}T_{wi,m} & (0<m<M) \\ W_0^*+u_{SF3,0}T_{SolWalF}+u_{LF3,0}W_{maF} & (p=0) \\ W_M^*+u_{SB3,M}T_{SolWalB}+u_{LB3,0}W_{maB} & (p=M) \\ W_p^* & (0<p<M) \end{cases} \tag{22.75}$$

$$UW_{m,n}=\begin{cases} 1+u_{SF2,m}+u_{SB2,m}+u_{PM2,m} & (n=m) \\ -(u_{SF2,m}+u_{PF2,m}) & (n=m-1) \\ -(u_{SB2,m}+u_{PB2,m}) & (n=m+1) \\ u_{LF2,m}+u_{LB2,m} & (q=m) \\ -u_{LF2,m} & (q=m-1) \\ -u_{LB2,m} & (q=m+1) \\ u_{SF3,m}+u_{SB3,m} & (n=p) \\ -u_{SF3,m} & (n=p-1) \\ -u_{SB3,m} & (n=p+1) \\ 1+u_{LF3,m}+u_{LB3,m} & (q=p) \\ -u_{LF3,m} & (q=p-1) \\ -u_{LB3,m} & (q=p+1) \\ 0 & (\text{other}) \end{cases} \tag{22.76}$$

$$u_{P2,m}=\frac{C_{LWal,m}+C_{\kappa,m}}{C_{SL,m}}\cdot\frac{c_{pw}m_{w,m}\varepsilon_{P,m}\Delta t}{A} \tag{22.77}$$

顕熱移動のみを考慮するモデルでは、壁体の相互放射を考慮した計算を行うために、壁を囲む空間の相当温度を用いて壁表面温度を表現した（式 22.31、式 22.32）。熱水分同時移動の場合には空間の絶対湿度も関係してくるため、式 22.78~式 22.81 に示すように、空間の相当温度と絶対湿度を用いて壁表面温度と壁表面絶対湿度を表現する。係数 *IF*, *FFS*, *BFS*, *FFL*, *BFL* はそれぞれ式 22.82~22.86で計算する。

$$T_0=T_F=IF2_F+FFS2_FT_{SolF}+BFS2_FT_{SolB}+FFL2_FW_{maF}+BFL2_FW_{maB} \tag{22.78}$$

$$T_M=T_B=IF2_B+FFS2_BT_{SolF}+BFS2_BT_{SolB}+FFL2_BW_{maF}+BFL2_BW_{maB} \tag{22.79}$$

$$W_0=W_F=IF3_F+FFS3_FT_{SolF}+BFS3_FT_{SolB}+FFL3_FW_{maF}+BFL3_FW_{maB} \tag{22.80}$$

$$W_M = T_B = IF3_B + FFS3_B T_{SolF} + BFS3_B T_{SolB} + FFL3_B W_{maF} + BFL3_B W_{maB} \tag{22.81}$$

$$\begin{aligned}
IF2_F &= \sum_{n=0}^{M} UWX_{0,n}\left(T_n^* + u_{P2,n} T_{wi,n}\right) + \sum_{n=M+1}^{2M} UWX_{0,n} W_{(m-M+1)}^* \\
IF2_B &= \sum_{n=0}^{M} UWX_{M,n}\left(T_n^* + u_{P2,n} T_{wi,n}\right) + \sum_{n=M+1}^{2M} UWX_{M,n} W_{(m-M+1)}^* \\
IF3_F &= \sum_{n=0}^{M} UWX_{M+1,n}\left(T_n^* + u_{P2,n} T_{wi,n}\right) + \sum_{n=M+1}^{2M} UWX_{M+1,n} W_{(m-M+1)}^* \\
IF3_B &= \sum_{n=0}^{M} UWX_{2M,n}\left(T_n^* + u_{P2,n} T_{wi,n}\right) + \sum_{n=M+1}^{2M} UWX_{2M,n} W_{(m-M+1)}^*
\end{aligned} \tag{22.82}$$

$$\begin{aligned}
FFS2_F &= UWX_{0,0}\left(u_{SF2,0} + u_{PF2,0}\right) + UWX_{0,M+1} u_{SF3,0} \\
FFS2_B &= UWX_{M,0}\left(u_{SF2,0} + u_{PF2,0}\right) + UWX_{M,M+1} u_{SF3,0} \\
FFS3_F &= UWX_{M+1,0}\left(u_{SF2,0} + u_{PF2,0}\right) + UWX_{M+1,M+1} u_{SF3,0} \\
FFS3_B &= UWX_{2M,0}\left(u_{SF2,0} + u_{PF2,0}\right) + UWX_{2M,M+1} u_{SF3,0}
\end{aligned} \tag{22.83}$$

$$\begin{aligned}
BFS2_F &= UWX_{0,M}\left(u_{SB2,M} + u_{PB2,M}\right) + UWX_{0,2M} u_{SB3,M} \\
BFS2_B &= UWX_{M,M}\left(u_{SB2,M} + u_{PB2,M}\right) + UWX_{M,2M} u_{SB3,M} \\
BFS3_F &= UWX_{M+1,M}\left(u_{SB2,M} + u_{PB2,M}\right) + UWX_{M+1,2M} u_{SB3,M} \\
BFS3_B &= UWX_{2M,M}\left(u_{SB2,M} + u_{PB2,M}\right) + UWX_{2M,2M} u_{SB3,M}
\end{aligned} \tag{22.84}$$

$$\begin{aligned}
FFL2_F &= UWX_{0,0} u_{LF2,0} + UWX_{0,M+1} u_{LF3,0} \\
FFL2_B &= UWX_{M,0} u_{LF2,0} + UWX_{M,M+1} u_{LF3,0} \\
FFL3_F &= UWX_{M+1,0} u_{LF3,0} + UWX_{M+1,M+1} u_{LF3,0} \\
FFL3_B &= UWX_{2M,0} u_{LF3,0} + UWX_{2M,M+1} u_{LF3,0}
\end{aligned} \tag{22.85}$$

$$\begin{aligned}
BFL2_F &= UWX_{0,M} u_{LB2,M} + UWX_{0,2M} u_{LB3,M} \\
BFL2_B &= UWX_{M,M} u_{LB2,M} + UWX_{M,2M} u_{LB3,M} \\
BFL3_F &= UWX_{M+1,M} u_{LB3,M} + UWX_{M+1,2M} u_{LB3,M} \\
BFL3_B &= UWX_{2M,M} u_{LB3,M} + UWX_{2M,2M} u_{LB3,M}
\end{aligned} \tag{22.86}$$

22.3 計算法

22.3.1 壁層クラスの作成

1）一般の壁層クラス

プログラム 22.1 に一般の壁層を表す WallLayer クラスのプロパティの定義を示す。各種の物性値と厚みを保持する。熱負荷計算では壁材料の物性値を一定値とすることが多いが、大きく変化する場合（後述の相変化材料を用いる場合や、熱水分同時移動解析で平衡含水率を相対湿度の関数として解く場合など）にはこれを考慮する必要がある。この判定のため、IsVariableProperties を定義している。顕熱容量を F 側と B 側に分割しているが、これは相変化材料で F 側と B 側が異なる相になり、熱容量が等しくなくなる場合に備えたものである。

プログラム 22.1　プロパティの定義

Popolo.HVAC.ThermalLoad.WallLayer class

```
/// <summary>名称を設定・取得する</summary>
public string Name { get; set; }

/// <summary>物性が変化しうるか</summary>
public bool IsVariableProperties { get; protected set; }

/// <summary>熱伝導率[W/(mK)]を取得する</summary>
public double ThermalConductivity { get; protected set; }

/// <summary>湿気伝導率[(kg/s) / ((kg/kg)･m)]を取得する</summary>
public double MoistureConductivity { get; protected set; }

/// <summary>容積比熱[kJ/(m^3K)]を取得する</summary>
public double VolSpecificHeat { get; protected set; }

/// <summary>熱コンダクタンス[W/(m2K)]を取得する</summary>
```

```
17 public double HeatConductance { get; protected set; }
18
19 /// <summary>顕熱容量（F 側）[J/(m2K)]を取得する</summary>
20 public double HeatCapacity_F { get; protected set; }
21
22 /// <summary>顕熱容量（B 側）[J/(m2K)]を取得する</summary>
23 public double HeatCapacity_B { get; protected set; }
24
25 /// <summary>水分容量[kg/m2]を取得する</summary>
26 public double WaterCapacity { get; protected set; }
27
28 /// <summary>単位絶対湿度差に対する吸湿係数[kg/m2]を取得する</summary>
29 public double KappaC { get; private set; }
30
31 /// <summary>単位温度差に対する放湿係数[kg/(m2K)]を取得する</summary>
32 public double NuC { get; private set; }
33
34 /// <summary>壁層の厚み[m]を取得する</summary>
35 public double Thickness { get; protected set; }
```

コンストラクタと初期化処理をプログラム 22.2 に示す。1~7 行は顕熱移動のみの場合のコンストラクタ、9~26 行は熱水分同時移動解析の場合のコンストラクタである。後者の場合には湿気伝導率や吸放湿の係数を引数に取り、22~25 行で水分移動関連のパラメータの初期化を行う。

プログラム 22.2　コンストラクタと初期化処理

Popolo.HVAC.ThermalLoad.WallLayer class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="name">名称</param>
 3 /// <param name="thermalConductivity">熱伝導率[W/(mK)]</param>
 4 /// <param name="volSpecificHeat">容積比熱[kJ/(m3K)]</param>
 5 /// <param name="thickness">壁層の厚み[m]</param>
 6 public WallLayer(string name, double thermalConductivity, double volSpecificHeat, double thickness)
 7 { initialize(name, thermalConductivity, volSpecificHeat, thickness); }
 8
 9 /// <summary>インスタンスを初期化する</summary>
10 /// <param name="name">名称</param>
11 /// <param name="thermalConductivity">熱伝導率[W/(mK)]</param>
12 /// <param name="volSpecificHeat">容積比熱[kJ/(m3K)]</param>
13 /// <param name="moistureConductivity">湿気伝導率[(kg/s)/((kg/kg)･m)]</param>
14 /// <param name="voidage">空隙率[-]</param>
15 /// <param name="kappa">吸湿係数[kg/m3]</param>
16 /// <param name="nu">放湿係数[kg/(m3K)]</param>
17 /// <param name="thickness">壁層の厚み[m]</param>
18 public WallLayer(string name, double thermalConductivity, double volSpecificHeat,
19   double moistureConductivity, double voidage, double kappa, double nu, double thickness)
20 {
21   initialize(name, thermalConductivity, volSpecificHeat, thickness);
22   MoistureConductivity = moistureConductivity;
23   WaterCapacity = 0.5 * voidage * thickness * MOISTAIR_DENSITY;
24   KappaC = 0.5 * kappa * thickness;
25   NuC = 0.5 * nu * thickness;
26 }
27
28 /// <summary>インスタンスを初期化する</summary>
29 /// <param name="name">名称</param>
30 /// <param name="thermalConductivity">熱伝導率[W/(mK)]</param>
31 /// <param name="volumetricSpecificHeat">容積比熱[kJ/(m3K)]</param>
32 /// <param name="thickness">壁層の厚み[m]</param>
33 protected void initialize
34   (string name, double thermalConductivity, double volumetricSpecificHeat, double thickness)
35 {
36   Name = name;
37   ThermalConductivity = thermalConductivity;
38   VolSpecificHeat = volumetricSpecificHeat;
39   Thickness = thickness;
40   HeatConductance = ThermalConductivity / Thickness;
41   HeatCapacity_F = HeatCapacity_B = 0.5 * VolSpecificHeat * Thickness * 1000d;
42 }
```

壁層を作るたびに材料物性を調べて入力する作業は煩雑である。プログラム 22.3 に示すように予め代表的な壁材料を列挙型で定義しておき（1~13 行）、これを引数とするコンストラクタ（15~34 行）を作成することでモデル作成作業を軽減する。

プログラム **22.3**　材料種別の列挙型定義とコンストラクタ

Popolo.HVAC.ThermalLoad.WallLayer class

```
/// <summary>壁材料</summary>
public enum Materials
{
  /// <summary>セメント・モルタル</summary>
  Mortar,
  /// <summary>鉄筋コンクリート</summary>
  ReinforcedConcrete,
  /// <summary>軽量骨材コンクリート１種</summary>
  LightweightAggregateConcrete1,

    ・・・以下略・・・

}

/// <summary>新しいインスタンスを初期化する</summary>
/// <param name="material">材料</param>
/// <param name="thickness">壁層の厚み[m]</param>
public WallLayer(Materials material, double thickness)
{
  switch (material)
  {
    case Materials.Mortar:
      initialize("Mortar", 1.512, 1591.0, thickness);
      break;
    case Materials.ReinforcedConcrete:
      initialize("Reinforced Concrete", 1.600, 1896.0, thickness);
      break;
    case Materials.LightweightAggregateConcrete1:
      initialize("Lightweight Aggregate Concrete 1", 0.810, 1900.0, thickness);
      break;

    ・・・以下略・・・

}
```

材料の温度や湿度の変化に対して物性値を変化させる処理をプログラム 22.4 に示す。WallLayer クラスでは物性値を一定とするため、本メソッドの中身は空である。virtual メソッドであるため、必要に応じて派生クラスで実装する。物性値に変更があった場合には true、変更が無い場合には false を返す。物性値が変化すると式 22.29 の$[U]$の逆行列を再計算せねばならない。このような判定を設けることで無駄な逆行列の計算を回避する。

プログラム **22.4**　温湿度変化に対する物性値更新の **virtual** メソッド定義

Popolo.HVAC.ThermalLoad.WallLayer class

```
/// <summary>壁層端部の温度に応じて物性値を更新する</summary>
/// <param name="temperature1">温度 1</param>
/// <param name="temperature2">温度 2</param>
/// <returns>物性値変更の有無</returns>
public virtual bool UpdateState(double temperature1, double temperature2) { return false; }

/// <summary>壁層端部の温湿度に応じて物性値を更新する</summary>
/// <param name="temperature1">温度 1</param>
/// <param name="temperature2">温度 2</param>
/// <param name="humidity1">絶対湿度 1</param>
/// <param name="humidity2">絶対湿度 2</param>
/// <returns>物性値変更の有無</returns>
public virtual bool UpdateState
  (double temperature1, double temperature2, double humidity1, double humidity2) { return false; }
```

2）中空層クラス

プログラム 22.5 に中空層を表す AirGapLayer クラスを示す。WallLayer クラスとほぼ同じであるが、層の厚みではなく密閉されているか否かに応じて熱抵抗を決定する点が異なる。計算速度を重視して熱抵抗は固定値としているが、中空層の傾きを引数に加えて、傾きに応じた熱抵抗を計算しても良いし、WallLayer クラスの UpdateState メソッドを上書きして両端温度の関数としても良い。

プログラム **22.5　AirGapLayer** クラス

Popolo.HVAC.ThermalLoad.AirGapLayer class

```
/// <summary>空気層</summary>
public class AirGapLayer : WallLayer
{
  /// <summary>密閉されているか否か</summary>
  public bool IsSealed { get; private set; }

  /// <summary>インスタンスを初期化する</summary>
  /// <param name="name">名称</param>
  /// <param name="isSealed">密閉されているか否か</param>
  /// <param name="thickness">壁層の厚み[m]</param>
  public AirGapLayer(string name, bool isSealed, double thickness)
  {
    //年平均として常温 20C/60%程度の値を採用
    ThermalConductivity = MoistAir.GetThermalConductivity(20);
    VolSpecificHeat = MoistAir.GetSpecificHeat(0.01) * 1.2;

    IsVariableProperties = false;
    Name = name;
    Thickness = thickness;
    IsSealed = isSealed;
    if (isSealed) HeatConductance = 1 / 0.15;
    else HeatConductance = 1 / 0.07;
    HeatCapacity_B = HeatCapacity_F = 0.5 * VolSpecificHeat * Thickness * 1000;
  }
}
```

3）相変化材料による壁層

プログラム 22.6 に相変化材料による壁層を表す PCMWallLayer クラスの列挙型、インスタンス変数、プロパティの定義を示す。1~11 行は相を表す列挙型であり、固体状態、平衡状態、液体状態の 3 つの相としている。相変化材料は相によって物性が変化するため、14 行に示すように、それぞれの相に対して WallLayer を作成して物性値を保持する。式 22.36~22.38に示したように相変化が生じる際には温度の割戻しを行うため、相変化前後の相を特定する必要がある。このため、23, 26 行に示すように、現在の相に加えて過去の相の状態を保持するプロパティを用意する。

プログラム **22.6**　列挙型、インスタンス変数、プロパティの定義

Popolo.HVAC.ThermalLoad.PCMWallLayer class

```
/// <summary>相</summary>
[Flags]
public enum State
{
  /// <summary>固体</summary>
  Solid = 1,
  /// <summary>平衡</summary>
  Equilibrium = 2,
  /// <summary>液体</summary>
  Liquid = 4
}

/// <summary>各相の物性</summary>
private Dictionary<State, WallLayer> layers = new Dictionary<State, WallLayer>();

/// <summary>B 側の現在の相を取得する</summary>
public State CurrentState_B { get; private set; }

/// <summary>F 側の現在の相を取得する</summary>
public State CurrentState_F { get; private set; }

/// <summary>B 側の過去の相を取得する</summary>
public State LastState_B { get; private set; }

/// <summary>F 側の過去の相を取得する</summary>
public State LastState_F { get; private set; }

/// <summary>凝固点[C]を取得する</summary>
public double FreezingTemperature { get; private set; }

/// <summary>融解点[C]を取得する</summary>
public double MeltingTemperature { get; private set; }
```

プログラム 22.7 にコンストラクタおよびインスタンスメソッドを示す。相変化によって物性値が変化するため、コンストラクタで IsVariableProperties を true に設定する。26~34 行は凝固点および融点の温度から相の判定を行うメソッド、36~40 行は相の別によって熱容量を計算するメソッドである。42~72 行は相変化に伴う物性値の更新処理であり、プログラム 22.4 で示した virtual メソッドをオーバーライドする。48~55 行で現在の相と過去の相を特定し、相変化が生じたか否かを確認する。相変化があれば 57~69 行の処理（熱容量と熱抵抗の更新）を実行する。

プログラム 22.7　コンストラクタとインスタンスメソッド

Popolo.HVAC.ThermalLoad.PCMWallLayer class

```
/// <summary>インスタンスを初期化する</summary>
/// <param name="name">名称</param>
/// <param name="freezingTemperature">凝固点[C]</param>
/// <param name="meltingTemperature">融点[C]</param>
/// <param name="thickness">壁層の厚み[m]</param>
/// <param name="solidLayer">固体状態の物性</param>
/// <param name="equilibriumLayer">平衡状態の物性</param>
/// <param name="liquidLayer">液体状態の物性</param>
public PCMWallLayer
  (string name, double freezingTemperature, double meltingTemperature, double thickness,
  WallLayer solidLayer, WallLayer equilibriumLayer, WallLayer liquidLayer)

{
  IsVariableProperties = true;
  FreezingTemperature = freezingTemperature;
  MeltingTemperature = meltingTemperature;
  Thickness = thickness;
  layers.Add(State.Solid, solidLayer);
  layers.Add(State.Equilibrium, equilibriumLayer);
  layers.Add(State.Liquid , liquidLayer);

  CurrentState_F = CurrentState_B = State.Equilibrium;
  UpdateState(freezingTemperature, freezingTemperature);
}

/// <summary>温度から状態を判定する</summary>
/// <param name="temp">温度</param>
/// <returns>状態（固体・平衡・液相の別）</returns>
private State getState(double temp)
{
  if (temp <= FreezingTemperature) return State.Solid;
  if (temp < MeltingTemperature) return State.Equilibrium;
  else return State.Liquid;
}

/// <summary>熱容量[J/(m2K)]を計算する</summary>
/// <param name="state">相</param>
/// <returns>熱容量[J/(m2K)]</returns>
public double GetHeatCapacity(State state)
{ return 0.5 * layers[state].VolSpecificHeat * Thickness * 1000d; }

/// <summary>壁層端部の温度に応じて状態値を更新する</summary>
/// <param name="temperatureF">F 側温度</param>
/// <param name="temperatureB">B 側温度</param>
/// <returns>状態値変更の有無</returns>
public override bool UpdateState(double temperatureF, double temperatureB)
{
  LastState_F = CurrentState_F;
  LastState_B = CurrentState_B;
  CurrentState_F = getState(temperatureF);
  CurrentState_B = getState(temperatureB);

  bool phaseChanged = false;
  if ((CurrentState_F != LastState_F) || (CurrentState_B != LastState_B))
    phaseChanged = true;

  if (phaseChanged)
  {
    double res =
      Thickness / layers[CurrentState_F].ThermalConductivity +
      Thickness / layers[CurrentState_B].ThermalConductivity;
    HeatConductance = 2d / res;
    double th1000 = Thickness * 1000d;
    HeatCapacity_B = 0.5 * layers[CurrentState_B].VolSpecificHeat * th1000;
```

```
65     HeatCapacity_F = 0.5 * layers[CurrentState_F].VolSpecificHeat * th1000;
66
67     ThermalConductivity = HeatConductance * Thickness;
68     VolSpecificHeat = (HeatCapacity_F + HeatCapacity_B) / th1000;
69   }
70
71   return phaseChanged;
72 }
```

22.3.2　埋設配管クラスの作成

プログラム 22.8 に埋設配管を表す BuriedPipe クラスを示す。配管に関連する情報を保持し、水量から熱通過率を計算する機能を持つ。44~74 行はコンストラクタである。フィン効率は水量に影響を受けないため、この時点で式 22.55~22.60を適用して計算する。76~82 行と 84~90 行はそれぞれ水量と水温の設定処理であり、式 22.42 の実装である updateEffectiveness メソッド（92~125 行）を呼び出して熱通過率を更新する。

プログラム 22.8　BuriedPipe クラス

Popolo.HVAC.ThermalLoad.BuriedPipe class

```
1 /// <summary>埋設配管</summary>
2 public class BuriedPipe
3 {
4   /// <summary>水の密度[kg/m3]</summary>
5   private const double WATER_DENSITY = 1000;
6
7   /// <summary>水の比熱[J/kg]</summary>
8   private const double WATER_SPECIFICHEAT = 4186;
9
10  /// <summary>水温[C]を取得する</summary>
11  /// <remarks>配管内対流熱伝達率の計算に使用</remarks>
12  public double InletWaterTemperature { get; private set; } = 25;
13
14  /// <summary>水量[kg/s]を取得する</summary>
15  public double WaterFlowRate { get; private set; }
16
17  /// <summary>分岐の数を取得する</summary>
18  public int BranchNumber { get; private set; }
19
20  /// <summary>配管敷設ピッチ[m]を取得する</summary>
21  public double Pitch { get; private set; }
22
23  /// <summary>配管総延長[m]を取得する</summary>
24  public double Length { get; private set; }
25
26  /// <summary>内径[m]を取得する</summary>
27  public double InnerDiameter { get; private set; }
28
29  /// <summary>外径[m]を取得する</summary>
30  public double OuterDiameter { get; private set; }
31
32  /// <summary>配管材の熱伝導率[W/(mK)]を取得する</summary>
33 public double ThermalConductivityOfTube { get; private set; }
34
35  /// <summary>上側フィン効率[-]を取得する</summary>
36  public double UpperFinEfficiency { get; private set; }
37
38  /// <summary>下側フィン効率[-]を取得する</summary>
39  public double LowerFinEfficiency { get; private set; }
40
41  /// <summary>熱通過有効度[-]を取得する</summary>
42  public double Effectiveness { get; private set; }
43
44  /// <summary>インスタンスを初期化する</summary>
45  /// <param name="pitch">配管敷設ピッチ[m]</param>
46  /// <param name="length">総配管長[m]</param>
47  /// <param name="branchNumber">分岐の数[本]</param>
48  /// <param name="iDiameter">内径[m]</param>
49  /// <param name="oDiameter">外径[m]</param>
50  /// <param name="tubeConductivity">配管熱伝導率[W/(mK)]</param>
51  /// <param name="upperFinConductivity">上側フィン熱伝導率[W/mK]</param>
52  /// <param name="lowerFinConductivity">下側フィン熱伝導率[W/mK]</param>
53  /// <param name="upperFinResistance">上側フィン熱抵抗[(m2K)/W]</param>
54  /// <param name="lowerFinResistance">下側フィン熱抵抗[(m2K)/W]</param>
55  /// <param name="upperFinThickness">上側フィン厚み[m]</param>
56  /// <param name="lowerFinThickness">下側フィン厚み[m]</param>
```

```
 57   public BuriedPipe(double pitch, double length, int branchNumber, double iDiameter, double oDiameter,
 58    double tubeConductivity, double upperFinConductivity, double lowerFinConductivity,
 59    double upperFinResistance, double lowerFinResistance, double upperFinThickness, double lowerFinThickness)
 60   {
 61     Pitch = pitch;
 62     Length = length;
 63     BranchNumber = branchNumber;
 64     InnerDiameter = iDiameter;
 65     OuterDiameter = oDiameter;
 66     ThermalConductivityOfTube = tubeConductivity;
 67
 68     double bf1 = (pitch - oDiameter) / 2;
 69     double bf2 = oDiameter + (pitch - oDiameter);
 70     double ubL = bf1 * Math.Sqrt(1 / (upperFinResistance * upperFinConductivity * upperFinThickness));
 71     double ubR = bf1 * Math.Sqrt(1 / (lowerFinResistance * lowerFinConductivity * lowerFinThickness));
 72     UpperFinEfficiency = 1 / pitch * (bf2 * Math.Tanh(ubL) / ubL);
 73     LowerFinEfficiency = 1 / pitch * (bf2 * Math.Tanh(ubR) / ubR);
 74   }
 75
 76   /// <summary>体積流量[kg/s]を設定する</summary>
 77   /// <param name="flowRate">体積流量[m3/s]</param>
 78   public void SetFlowRate(double flowRate)
 79   {
 80     WaterFlowRate = flowRate;
 81     updateEffectiveness();
 82   }
 83
 84   /// <summary>水温[C]を設定する</summary>
 85   /// <param name="temperature">水温[C]</param>
 86   public void SetWaterTemperature(double temperature)
 87   {
 88     InletWaterTemperature = temperature;
 89     updateEffectiveness();
 90   }
 91
 92   /// <summary>熱通過率[-]を更新する</summary>
 93   private void updateEffectiveness()
 94   {
 95     if (WaterFlowRate <= 0)
 96     {
 97       Effectiveness = 0;
 98       return;
 99     }
100
101     //対流熱伝達率[W/m2K]の計算//////////////////////////
102     //動粘性係数[m2/s]・熱拡散率[m2/s]・熱伝導率[W/(m･K)]を計算
103     double v = Water.GetLiquidDynamicViscosity(InletWaterTemperature);
104     double a = Water.GetLiquidThermalDiffusivity(InletWaterTemperature);
105     double lambda = Water.GetLiquidThermalConductivity(InletWaterTemperature);
106
107     //配管内流速[m/s]を計算
108     double vFlow = WaterFlowRate / (WATER_DENSITY * BranchNumber);
109     double u = vFlow / (Math.Pow(InnerDiameter / 2, 2) * Math.PI);
110
111     //ヌセルト数を計算
112     double reNumber = u * InnerDiameter / v;
113     double prNumber = v / a;
114     double nuNumber = 0.023 * Math.Pow(reNumber, 0.8) * Math.Pow(prNumber, 0.4);
115
116     //ヌセルト数から対流熱伝達率を計算
117     double hi = nuNumber * lambda / InnerDiameter;
118
119     //伝熱係数KA, 移動単位数NTU, 熱通過率 ε の計算/////////
120     double ka = 1 / (InnerDiameter * hi)
121       + 1 / (2 * ThermalConductivityOfTube) * Math.Log(OuterDiameter / InnerDiameter);
122     ka = (Math.PI * Length) / ka;
123     double ntu = ka / (WaterFlowRate * WATER_SPECIFICHEAT);
124     Effectiveness = 1 - Math.Exp(-ntu);
125   }
126 }
```

22.3.3　壁クラスの作成

プログラム 22.9 に壁を表す Wall クラスのインスタンス変数を示す。壁体の温湿度の更新には行列[U]の逆行列を用いるが、壁材料の物性値や埋設配管の熱通過率が変化しないかぎりは、逆行列は変化しない。そこで、計算負荷を減らすために逆行列の計算の必要性を判定するフラグ（第 2 行）を用意する。また、埋設配管の温度や水量は壁材料の物性に比較して変化しやすい。そこで壁材料の物性

に関わる係数行列までは予め求めて保存（26 行 uMatrix）しておき、必要に応じて埋設配管に関わる情報を足し込むことで行列[*U*]（26 行 uMatrixWithTubeEffect）を作成するという方法をとる。この係数行列更新の判定のためのフラグ（第 5 行）を用意する。8 行は壁層リストであり、物性変化があり得る層の番号を 11 行の配列に保存する。14 行は埋設配管であり、複数の層に配管を敷設できるように配列で管理する。17 行は温湿度ベクトル[*TW*]であり、19 行以降は計算に必要な各種の係数を保持する行列やベクトルである。

プログラム 22.9　Wall クラスのインスタンス変数

Popolo.HVAC.ThermalLoad.Wall class

```
 1 /// <summary>逆行列更新判定フラグ</summary>
 2 private bool needToUpdateUINVMatrix = true;
 3
 4 /// <summary>係数行列更新判定フラグ</summary>
 5 internal bool needToUpdateUMatrix = false;
 6
 7 /// <summary>壁層リスト</summary>
 8 private WallLayer[] layers;
 9
10 /// <summary>物性変化があり得る層のリスト</summary>
11 private int[] variableLayers;
12
13 /// <summary>埋設配管リスト</summary>
14 private Dictionary<int, BuriedPipe> bPipes = new Dictionary<int, BuriedPipe>();
15
16 /// <summary>温湿度分布を保持するベクトル</summary>
17 private IVector tempAndHumid;
18
19 /// <summary>顕熱抵抗[m2K/W]・顕熱容量[J/(m2K)]リスト</summary>
20 private double[] resS, capS;
21
22 /// <summary>透湿抵抗[(kg/kg)･m2/(kg/s)]・水分容量[kg/m2]リスト</summary>
23 private double[] resL, capL, cNu, cKappa;
24
25 /// <summary>計算用行列</summary>
26 private IMatrix uMatrix, umWithTubeEffect, uxMatrix;
27
28 /// <summary>計算用配列（顕熱平衡・床冷暖房）</summary>
29 private double[] uSF, uSB, uP, uPF, uPM, uPB;
30
31 /// <summary>計算用配列（熱水分同時移動解析用）</summary>
32 private double[] uSF2, uSB2, uLF2, uLB2, uSF3, uSB3, uLF3, uLB3;
```

プログラム 22.10 にプロパティの定義を示す。7~13 行は温湿度分布を表すベクトルであり、部分行列クラスである VectorView を用いることで、温湿度ベクトル[*TW*]から温度部分と湿度部分のベクトルを切り出す。21~31 行は計算時間間隔であり、変更時には逆行列の再計算が必要となるため、needToUpdateUMatrix フラグをたてる。ただし、無駄な再計算が起こらないように設定値が従前の値と変わらない場合には無視する。このように負荷が大きい処理に関しては無駄な計算が発生しないように気を配る必要がある。33~61 行は放射および対流熱伝達率の設定処理であり、この場合にも逆行列再計算フラグをたてる必要がある。64 行は湿気伝達率であり、式 22.67 を用いてルイスの関係により計算する。66~76 行はその他の F 側条件設定処理である。79 行は F 側の壁表面であり、詳細は多数室の連成計算とともに第 25 章で解説する。B 側に関するソースコードは省略したが F 側と同様のプロパティを用意する。81 行以降は式 22.50~22.52、式 22.82~22.86の各種係数である。これらの係数はプログラム開発者が多数室連成計算に用いるためのものであり、モデル利用者にとっては無用の係数であるため、アクセス修飾子を internal として隠蔽する。90 行も多数室連成計算（プログラム 25.18 で用いる）に用いる変数であり、壁の逆行列が変更されたか否かの情報を保持する。

プログラム **22.10**　プロパティの定義

Popolo.HVAC.ThermalLoad.Wall class

```
/// <summary>水分移動を解くか否か</summary>
public bool ComputeMoistureTransfer { get; private set; }

/// <summary>質点の数を取得する</summary>
public int NodeNumber { get { return layers.Length + 1; } }

/// <summary>温度分布を取得する</summary>
public IVector Temperatures
{ get { return new VectorView(tempAndHumid, 0, NodeNumber); } }

/// <summary>湿度度分布を取得する</summary>
public IVector Humidities
{ get { return new VectorView(tempAndHumid, NodeNumber, NodeNumber); } }

/// <summary>壁面積[m2]を設定・取得する</summary>
public double Area { get; set; } = 1.0d;

/// <summary>計算時間間隔[sec]</summary>
private double timeStep = 3600;

/// <summary>計算時間間隔[sec]を設定・取得する</summary>
public double TimeStep
{
  get { return timeStep; }
  set
  {
    if (value <= 0 || timeStep == value) return;
    timeStep = value;
    needToUpdateUMatrix = true;
  }
}

/// <summary>F 側と B 側の対流・放射熱伝達率[W/(m2K)]</summary>
private double cCoefF, rCoefF, cCoefB, rCoefB;

/// <summary>F 側の総合熱伝達率[W/(m2K)]を取得する</summary>
public double FilmCoefficientF { get { return cCoefF + rCoefF; } }

/// <summary>F 側の対流熱伝達率[W/(m2K)]を設定・取得する</summary>
public double ConvectiveCoefficientF
{
  get { return cCoefF; }
  set
  {
    if (cCoefF == value) return;
    needToUpdateUMatrix = true;
    cCoefF = value;
  }
}

/// <summary>F 側の放射熱伝達率[W/(m2K)]を設定・取得する</summary>
public double RadiativeCoefficientF
{
  get { return rCoefF; }
  set
  {
    if (rCoefF == value) return;
    needToUpdateUMatrix = true;
    rCoefF = value;
  }
}

/// <summary>F 側の湿気伝達率[(kg/s)/((kg/kg)m2)]を取得する</summary>
public double MoistureCoefficientF { get { return 1.0d / resL[0]; } }

/// <summary>F 側の短波長吸収率[-]を設定・取得する</summary>
public double ShortWaveAbsorptanceF { get; set; } = 0.7;

/// <summary>F 側の長波長放射率[-]を設定・取得する</summary>
public double LongWaveEmissivityF { get; set; } = 0.9;

/// <summary>F 側の相当温度[C]を設定・取得する</summary>
public double SolAirTemperatureF { get; set; }

/// <summary>F 側の絶対湿度[kg/kg]を設定・取得する</summary>
public double HumidityRatioF { get; set; }

/// <summary>F 側の壁表面を取得する</summary>
```

```
79 internal wallWindowSurface SurfaceF { get; private set; }
80
81 /// <summary>境界条件による係数（F 側×温度）</summary>
82 internal double IF2_F { get; private set; }
83
84 /// <summary>F 側相当温度による係数（F 側顕熱×温度）</summary>
85 internal double FFS2_F { get; private set; }
86
87 ・・・以下略・・・
88
89 /// <summary>逆行列が更新されたか否かを設定・取得する</summary>
90 internal bool invMatrixUpdated { get; set; } = true;
```

プログラム 22.11 にコンストラクタを示す。引数は壁の面積、壁層のリスト、熱水分同時移動解析とするか否か、の 3 つである。4 行のコンストラクタは 3 つ目の引数を省略したコンストラクタであり、この場合には水分移動を考慮しない顕熱流のモデルとする。12~15 行で引数をインスタンス変数に保存し、17,18 行では壁表面を作成する（第 25 章で解説）。19~52 行で計算に必要な記憶領域を確保する。熱水分同時移動の場合には記憶領域が 2 倍（温度と絶対湿度）になることに注意する。係数行列は未作成のため、63 行で係数行列更新フラグをたてておく。

プログラム 22.11　コンストラクタ

Popolo.HVAC.ThermalLoad.Wall class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="area">面積[m2]</param>
 3 /// <param name="layers">壁構成</param>
 4 public Wall(double area, WallLayer[] layers) : this(area, layers, false) { }
 5
 6 /// <summary>新しいインスタンスを初期化する</summary>
 7 /// <param name="area">面積[m2]</param>
 8 /// <param name="layers">壁構成</param>
 9 /// <param name="computeMoistureTransfer">水分移動を計算するか否か</param>
10 public Wall(double area, WallLayer[] layers, bool computeMoistureTransfer)
11 {
12   Area = area;
13   this.layers = new WallLayer[layers.Length];
14   for (int i = 0; i < layers.Length; i++) this.layers[i] = (WallLayer)layers[i].Clone();
15   ComputeMoistureTransfer = computeMoistureTransfer;
16   SurfaceF = new wallWindowSurface(this, true);
17   SurfaceB = new wallWindowSurface(this, false);
18
19   //計算領域を確保
20   int mNum = this.layers.Length + 1; //質点数
21   capS = new double[mNum];
22   resS = new double[mNum + 1];
23   int ssNum = mNum; //未知変数の数
24   if (computeMoistureTransfer)
25   {
26     ssNum *= 2; //絶対湿度も未知のため倍
27     uSF2 = new double[mNum];
28     uSB2 = new double[mNum];
29     uSF3 = new double[mNum];
30     uSB3 = new double[mNum];
31     uLF2 = new double[mNum];
32     uLB2 = new double[mNum];
33     uLF3 = new double[mNum];
34     uLB3 = new double[mNum];
35     cNu = new double[mNum];
36     cKappa = new double[mNum];
37     capL = new double[mNum];
38     resL = new double[mNum + 1];
39   }
40   else
41   {
42     uSF = new double[ssNum];
43     uSB = new double[ssNum];
44   }
45   tempAndHumid = new Vector(ssNum);
46   uMatrix = new Matrix(ssNum, ssNum);
47   umWithTubeEffect = new Matrix(ssNum, ssNum);
48   uxMatrix = new Matrix(ssNum, ssNum);
49   uP = new double[mNum];
50   uPF = new double[mNum];
51   uPM = new double[mNum];
```

```
52   uPB = new double[mNum];
53
54   //物性変化があり得る層の番号を保存
55   List<int> tal = new List<int>();
56   for (int i = 0; i < layers.Length; i++) if (layers[i].IsVariableProperties) tal.Add(i);
57   variableLayers = tal.ToArray();
58
59   ConvectiveCoefficientF = ConvectiveCoefficientB = 5.3;
60   RadiativeCoefficientF = RadiativeCoefficientB = 4.5;
61   needToUpdateUMatrix = true;
62 }
```

プログラム 22.12 に係数行列[U]の更新処理を示す。4 行で更新の必要性のフラグを確認し、不要であれば終了する。更新する場合には逆行列の再計算も必要になるため、5 行で逆行列更新フラグをたてる。8~39 行は熱容量と熱抵抗の初期化処理であり、式 22.19~22.23の実装である。43 行以降が係数行列の更新処理である。熱水分同時移動の場合には式 22.72 と 22.76 に従い、45~89 行で処理する。顕熱流のみの場合には式 22.24 と 22.45 に従い、92~109 行で処理する。ただし、この時点では放射冷暖房の埋設配管の影響（u_{PF}やu_{PF2}など）は反映しない。

プログラム 22.12　係数行列[U]の更新処理

Popolo.HVAC.ThermalLoad.Wall class

```
 1 /// <summary>係数行列を更新する</summary>
 2 private void updateUMatrix()
 3 {
 4   if (!needToUpdateUMatrix) return;
 5   needToUpdateUINVMatrix = true;
 6
 7   int mNum = layers.Length + 1; //質点数
 8   //熱容量,熱抵抗,水分容量,透湿抵抗の配列を作成
 9   for (int i = 0; i < mNum; i++)
10   {
11     capS[i] = 0;
12     if (i != 0) capS[i] += layers[i - 1].HeatCapacity_B;
13     if (i != mNum - 1) capS[i] += layers[i].HeatCapacity_F;
14     if (i != 0) resS[i] = 1 / layers[i - 1].HeatConductance;
15     if (ComputeMoistureTransfer)
16     {
17       capL[i] = cKappa[i] = cNu[i] = 0;
18       if (i != 0)
19       {
20         capL[i] += layers[i - 1].WaterCapacity;
21         cKappa[i] += layers[i - 1].KappaC;
22         cNu[i] += layers[i - 1].NuC;
23       }
24       if (i != mNum - 1)
25       {
26         capL[i] += layers[i].WaterCapacity;
27         cKappa[i] += layers[i].KappaC;
28         cNu[i] += layers[i].NuC;
29       }
30       if (i != 0) resL[i] = 1 / layers[i - 1].MoistureConductivity;
31     }
32   }
33   resS[0] = 1 / FilmCoefficientF;
34   resS[resS.Length - 1] = 1 / FilmCoefficientB;
35   if (ComputeMoistureTransfer)
36   {
37     resL[0] = AIR_SPECIFICHEAT / cCoefF;
38     resL[resL.Length - 1] = AIR_SPECIFICHEAT / cCoefB;
39   }
40
41   //係数行列[U]を作成
42   uMatrix.Initialize(0);
43   if (ComputeMoistureTransfer)
44   {
45     //熱水分同時移動
46     for (int i = 0; i < mNum; i++)
47     {
48       if (capS[i] == 0 && capL[i] == 0)
49         throw new Exception("Vacuum wall layer is not supported");
50       double cS = capS[i];
51       double cL = capL[i];
52       double dtS = timeStep;
```

```
53      double dtL = timeStep;
54      if (cS == 0 && cNu[i] == 0) cS = dtS = 1;
55      if (cL == 0 && cKappa[i] == 0) cL = dtL = 1;
56      double cSL = cS * (cL + cKappa[i]) + VAPORIZATION_LATENT_HEAT * cNu[i] * cL;
57      double cSLS = dtS / cSL;
58      double cSLL = dtL / cSL;
59      uSF2[i] = cSLS * (cL + cKappa[i]) / resS[i];
60      uSB2[i] = cSLS * (cL + cKappa[i]) / resS[i + 1];
61      uLF2[i] = cSLS * VAPORIZATION_LATENT_HEAT * cKappa[i] / resL[i];
62      uLB2[i] = cSLS * VAPORIZATION_LATENT_HEAT * cKappa[i] / resL[i + 1];
63      uSF3[i] = cSLL * cNu[i] / resS[i];
64      uSB3[i] = cSLL * cNu[i] / resS[i + 1];
65      uLF3[i] = cSLL * (cS + VAPORIZATION_LATENT_HEAT * cNu[i]) / resL[i];
66      uLB3[i] = cSLL * (cS + VAPORIZATION_LATENT_HEAT * cNu[i]) / resL[i + 1];
67
68      if (i != 0)
69      {
70        uMatrix[i, i - 1] = -uSF2[i];
71        uMatrix[i, i - 1 + mNum] = -uLF2[i];
72        uMatrix[i + mNum, i - 1] = -uSF3[i];
73        uMatrix[i + mNum, i - 1 + mNum] = -uLF3[i];
74      }
75      if (i != mNum - 1)
76      {
77        uMatrix[i, i + 1] = -uSB2[i];
78        uMatrix[i, i + 1 + mNum] = -uLB2[i];
79        uMatrix[i + mNum, i + 1] = -uSB3[i];
80        uMatrix[i + mNum, i + 1 + mNum] = -uLB3[i];
81      }
82      if (capS[i] == 0 && cNu[i] == 0) uMatrix[i, i] = uSF2[i] + uSB2[i];
83      else uMatrix[i, i] = 1d + uSF2[i] + uSB2[i];
84      if (capL[i] == 0 && cKappa[i] == 0) uMatrix[i + mNum, i + mNum] = uLF3[i] + uLB3[i];
85      else uMatrix[i + mNum, i + mNum] = 1d + uLF3[i] + uLB3[i];
86      uMatrix[i, i + mNum] = uLF2[i] + uLB2[i];
87      uMatrix[i + mNum, i] = uSF3[i] + uSB3[i];
88    }
89  }
90  else
91  {
92    //顕熱流のみ
93    for (int i = 0; i < mNum; i++)
94    {
95      if (capS[i] == 0)
96      {
97        uSF[i] = 1 / resS[i];
98        uSB[i] = 1 / resS[i + 1];
99      }
100     else
101     {
102       uSF[i] = timeStep / (capS[i] * resS[i]);
103       uSB[i] = timeStep / (capS[i] * resS[i + 1]);
104     }
105     if (i != 0) uMatrix[i, i - 1] = -uSF[i];
106     if (i != mNum - 1) uMatrix[i, i + 1] = -uSB[i];
107     if (capS[i] == 0) uMatrix[i, i] = uSF[i] + uSB[i];
108     else uMatrix[i, i] = 1d + uSF[i] + uSB[i];
109   }
110 }
111 needToUpdateUMatrix = false;
112 }
```

プログラム 22.13 に逆行列の更新処理を示す。逆行列は係数行列が変化した場合または埋設配管の熱通過率が変化した場合に更新する必要があり、その判定を 8 行で行う。12~35 行で埋設配管の影響（式 22.45 と式 22.76）を加え、37 行で逆行列を計算する。39~70 行は表面温度計算の係数（式 22.51 と式 22.52 または式 22.83~22.86）の更新処理である。ただし、係数 *IF* と *IF*2 は壁体内部の温湿度の影響を受けるため、この段階ではまだ計算しない。

プログラム 22.13　逆行列**[*UX*]**の更新処理

Popolo.HVAC.ThermalLoad.Wall class

```
1 /// <summary>逆行列を更新する</summary>
2 private void updateInverseMatrix()
3 {
4   //係数行列（配管の影響を除く）を更新
5   updateUMatrix();
6
```

```
 7   //逆行列を更新
 8   if (needToUpdateUINVMatrix)
 9   {
10     needToUpdateUINVMatrix = false;
11
12     //埋設配管の影響を行列に反映
13     umWithTubeEffect.Initialize(0);
14     for (int i = 0; i < uMatrix.Rows; i++)
15       for (int j = 0; j < uMatrix.Columns; j++)
16         umWithTubeEffect[i, j] = uMatrix[i, j];
17     foreach (int key in bPipes.Keys)
18     {
19       BuriedPipe bp = bPipes[key];
20       double cS = capS[key];
21       if (ComputeMoistureTransfer)
22       {
23         double cSL = capS[key] * (capL[key] + cKappa[key]) + VAPORIZATION_LATENT_HEAT * cNu[key] * capL[key];
24         cS = cSL / (capL[key] + cKappa[key]);
25       }
26       uP[key] = WATER_SPECIFICHEAT * bp.WaterFlowRate * bp.Effectiveness * TimeStep / (cS * Area);
27       double bf = uP[key] / (resS[key + 1] * bp.UpperFinEfficiency + resS[key] * bp.LowerFinEfficiency);
28       uPF[key] = bf * resS[key + 1] * (1 - bp.UpperFinEfficiency);
29       uPM[key] = bf * (resS[key] + resS[key + 1]);
30       uPB[key] = bf * resS[key] * (1 - bp.LowerFinEfficiency);
31
32       if (key != 0) umWithTubeEffect[key, key - 1] -= uPF[key];
33       if (key != layers.Length) umWithTubeEffect[key, key + 1] -= uPB[key];
34       umWithTubeEffect[key, key] += uPM[key];
35     }
36     //逆行列を計算
37     LinearAlgebra.GetInverse(ref umWithTubeEffect, ref uxMatrix);
38
39     //係数FFとBFを更新
40     if (ComputeMoistureTransfer)
41     {
42       IMatrix ux = uxMatrix;
43       int n0 = layers.Length;
44       int n1 = n0 + 1;
45       int n2 = n0 * 2 + 1;
46       FFS2_F = ux[0, 0] * (uSF2[0] + uPF[0]) + ux[0, n1] * uSF3[0];
47       FFS2_B = ux[n0, 0] * (uSF2[0] + uPF[0]) + ux[n0, n1] * uSF3[0];
48       FFS3_F = ux[n1, 0] * (uSF2[0] + uPF[0]) + ux[n1, n1] * uSF3[0];
49       FFS3_B = ux[n2, 0] * (uSF2[0] + uPF[0]) + ux[n2, n1] * uSF3[0];
50       BFS2_F = ux[0, n0] * (uSB2[n0] + uPB[n0]) + ux[0, n2] * uSB3[n0];
51       BFS2_B = ux[n0, n0] * (uSB2[n0] + uPB[n0]) + ux[n0, n2] * uSB3[n0];
52       BFS3_F = ux[n1, n0] * (uSB2[n0] + uPB[n0]) + ux[n1, n2] * uSB3[n0];
53       BFS3_B = ux[n2, n0] * (uSB2[n0] + uPB[n0]) + ux[n2, n2] * uSB3[n0];
54       FFL2_F = ux[0, 0] * uLF2[0] + ux[0, n1] * uLF3[0];
55       FFL2_B = ux[n0, 0] * uLF2[0] + ux[n0, n1] * uLF3[0];
56       FFL3_F = ux[n1, 0] * uLF2[0] + ux[n1, n1] * uLF3[0];
57       FFL3_B = ux[n2, 0] * uLF2[0] + ux[n2, n1] * uLF3[0];
58       BFL2_F = ux[0, n0] * uLB2[n0] + ux[0, n2] * uLB3[n0];
59       BFL2_B = ux[n0, n0] * uLB2[n0] + ux[n0, n2] * uLB3[n0];
60       BFL3_F = ux[n1, n0] * uLB2[n0] + ux[n1, n2] * uLB3[n0];
61       BFL3_B = ux[n2, n0] * uLB2[n0] + ux[n2, n2] * uLB3[n0];
62     }
63     else
64     {
65       int num = uxMatrix.Rows - 1;
66       FFS2_F = uxMatrix[0, 0] * (uSF[0] + uPF[0]);
67       FFS2_B = uxMatrix[num, 0] * (uSF[0] + uPF[0]);
68       BFS2_F = uxMatrix[0, num] * (uSB[num] + uPB[num]);
69       BFS2_B = uxMatrix[num, num] * (uSB[num] + uPB[num]);
70     }
71
72     //逆行列変更フラグON
73     invMatrixUpdated = true;
74   }
75 }
```

プログラム 22.14 に温湿度の更新処理を示す。7~37 行で式 22.44 のベクトル[*TH**]または式 22.75 のベクトル[*TWH**]を作成する。39 行で逆行列を用いて温湿度を更新する。温湿度の影響を受けて物性値が変化する壁材料がある場合には、41~88 行で物性値を更新する。46~49 行で物性値更新を行うが、不要な逆行列の計算を防止するため、物性値が更新された場合のみ 50 行で逆行列更新フラグをたてる。特に PCM で相変化があった場合には 52~87 行で式 22.36~22.38 を適用して温度を割り戻

す。90~119 行は壁体内部の温湿度に影響を受ける係数 *IF* または *IF*2 の更新処理である。

プログラム 22.14　温湿度の更新処理

Popolo.HVAC.ThermalLoad.Wall class

```
/// <summary>壁体内部の温湿度を更新する</summary>
public void Update()
{
  //逆行列を更新
  updateInverseMatrix();

  //相当温度と絶対湿度で係数ベクトルを作成
  int mNum = layers.Length + 1;
  int last = mNum - 1;
  Vector tempAndHumid2 = new Vector(tempAndHumid.Length);
  tempAndHumid2.Initialize(0);
  if (ComputeMoistureTransfer)
  {
    tempAndHumid2[0] = uSF2[0] * SolAirTemperatureF + uLF2[0] * HumidityRatioF;
    tempAndHumid2[last] = uSB2[last] * SolAirTemperatureB + uLB2[last] * HumidityRatioB;
    tempAndHumid2[mNum] = uSF3[0] * SolAirTemperatureF + uLF3[0] * HumidityRatioF;
    tempAndHumid2[last + mNum] = uSB3[last] * SolAirTemperatureB + uLB3[last] * HumidityRatioB;
    for (int i = 0; i < mNum; i++)
    {
      if (capS[i] != 0 || cNu[i] != 0) tempAndHumid2[i] += tempAndHumid[i];
      if (capL[i] != 0 || cKappa[i] != 0) tempAndHumid2[i + mNum] += tempAndHumid[i + mNum];
    }
  }
  else
  {
    tempAndHumid2[0] = uSF[0] * SolAirTemperatureF;
    tempAndHumid2[last] = uSB[last] * SolAirTemperatureB;
    for (int i = 0; i < tempAndHumid2.Length; i++)
      if (capS[i] != 0) tempAndHumid2[i] += tempAndHumid[i];
  }
  //埋設配管の影響を加える
  foreach (int key in bPipes.Keys)
  {
    if (key == 0) tempAndHumid2[0] += uPF[0] * SolAirTemperatureF;
    if (key == last) tempAndHumid2[last] += uPB[last] * SolAirTemperatureB;
    tempAndHumid2[key] += uP[key] * bPipes[key].InletWaterTemperature;
  }
  //逆行列で解を求める
  LinearAlgebra.Multiplicate(uxMatrix, tempAndHumid2, ref tempAndHumid, 1, 0);

  //物性変化があり得る層の状態を計算して行列更新の要否を確認
  for (int i = 0; i < variableLayers.Length; i++)
  {
    int lnum = variableLayers[i];
    bool flg;
    if (ComputeMoistureTransfer)
      flg = layers[lnum].UpdateState
        (tempAndHumid[lnum], tempAndHumid[lnum + 1], tempAndHumid[lnum + mNum], tempAndHumid[lnum + mNum] +
1);
    else flg = layers[lnum].UpdateState(tempAndHumid[lnum], tempAndHumid[lnum + 1]);
    if (flg) needToUpdateUMatrix = true;

    //PCM の場合には温度を調整
    const PCMWallLayer.State sOrE = PCMWallLayer.State.Solid | PCMWallLayer.State.Equilibrium;
    PCMWallLayer pwl = layers[lnum] as PCMWallLayer;
    if (flg && (pwl != null))
    {
      if (pwl.CurrentState_F != pwl.LastState_F)
      {
        double temp;
        if ((pwl.CurrentState_F | pwl.LastState_F) == sOrE) temp = pwl.FreezingTemperature;
        else temp = pwl.MeltingTemperature;
        double cap1 = pwl.GetHeatCapacity(pwl.LastState_F);
        double cap2 = pwl.GetHeatCapacity(pwl.CurrentState_F);
        if (lnum == 0) tempAndHumid[0] = cap1 / cap2 * (tempAndHumid[0] - temp) + temp;
        else
        {
          double cap3 = layers[lnum - 1].HeatCapacity_F;
          tempAndHumid[lnum] = (tempAndHumid[lnum] * (cap1 + cap3) + temp * (cap2 - cap1)) / (cap2 + cap3);
        }
      }
      if (pwl.CurrentState_B != pwl.LastState_B)
      {
        double temp;
        if ((pwl.CurrentState_B | pwl.LastState_B) == sOrE) temp = pwl.FreezingTemperature;
```

```
76          else temp = pwl.MeltingTemperature;
77          double cap1 = pwl.GetHeatCapacity(pwl.LastState_B);
78          double cap2 = pwl.GetHeatCapacity(pwl.CurrentState_B);
79          if (lnum == layers.Length - 1)
80            tempAndHumid[layers.Length - 1] = cap1 / cap2 * (tempAndHumid[layers.Length - 1] - temp) + temp;
81          else
82          {
83            double cap3 = layers[lnum].HeatCapacity_B;
84            tempAndHumid[lnum + 1] =
85              (tempAndHumid[lnum + 1] * (cap1 + cap3) + temp * (cap2 - cap1)) / (cap2 + cap3);
86          }
87        }
88      }
89    }
90
91    //係数 IF を更新
92    if (ComputeMoistureTransfer)
93    {
94      int nm = layers.Length;
95      int nm1 = nm + 1;
96      int nm2 = 2 * nm + 1;
97      IF2_F = IF2_B = IF3_F = IF3_B = 0;
98      for (int i = 0; i <= nm; i++)
99      {
100       double bf = tempAndHumid[i];
101       int ipn = i + nm1;
102       if (bPipes.ContainsKey(i)) bf += uP[i] * bPipes[i].InletWaterTemperature;
103       IF2_F += uxMatrix[0, i] * bf + uxMatrix[0, ipn] * tempAndHumid[ipn];
104       IF2_B += uxMatrix[nm, i] * bf + uxMatrix[nm, ipn] * tempAndHumid[ipn];
105       IF3_F += uxMatrix[nm1, i] * bf + uxMatrix[nm1, ipn] * tempAndHumid[ipn];
106       IF3_B += uxMatrix[nm2, i] * bf + uxMatrix[nm2, ipn] * tempAndHumid[ipn];
107     }
108   }
109   else
110   {
111     int num = uxMatrix.Rows - 1;
112     IF2_F = IF2_B = 0;
113     for (int i = 0; i <= num; i++)
114     {
115       double bf = tempAndHumid[i];
116       if (bPipes.ContainsKey(i)) bf += uP[i] * bPipes[i].InletWaterTemperature;
117       IF2_F += uxMatrix[0, i] * bf;
118       IF2_B += uxMatrix[num, i] * bf;
119     }
120   }
    }
```

プログラム 22.15 に温湿度の初期化処理を示す。1~12 行が顕熱移動のみの場合、14~29 行が熱水分同時移動の場合の処理である。周囲の温湿度条件と壁体温湿度配列を引数の値で初期化する。係数行列[*U*]の更新フラグをたてた後、温湿度更新処理を呼び出すことで、内部温湿度の係数 *IF* または *IF*2 も更新する。

プログラム 22.15　温湿度初期化処理

Popolo.HVAC.ThermalLoad.Wall class

```
1 /// <summary>温度を初期化する</summary>
2 /// <param name="temperature">温度[C]</param>
3 public void Initialize(double temperature)
4 {
5   VectorView temp = new VectorView(tempAndHumid, 0, layers.Length + 1);
6   temp.Initialize(temperature);
7   for (int i = 0; i < variableLayers.Length; i++)
8     layers[variableLayers[i]].UpdateState(temperature, temperature);
9   SolAirTemperatureF = SolAirTemperatureB = temperature;
10  needToUpdateUMatrix = true;
11  Update();
12 }
13
14 /// <summary>温湿度を初期化する</summary>
15 /// <param name="temperature">温度[C]</param>
16 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
17 public void Initialize(double temperature, double humidityRatio)
18 {
19   if (!ComputeMoistureTransfer) return;
20   int mnum = layers.Length + 1;
21   new VectorView(tempAndHumid, 0, mnum).Initialize(temperature);
22   new VectorView(tempAndHumid, mnum, mnum).Initialize(humidityRatio);
```

```
23   for (int i = 0; i < variableLayers.Length; i++)
24     layers[variableLayers[i]].UpdateState(temperature, temperature, humidityRatio, humidityRatio);
25   SolAirTemperatureF = SolAirTemperatureB = temperature;
26   HumidityRatioF = HumidityRatioB = humidityRatio;
27   needToUpdateUMatrix = true;
28   Update();
29 }
```

プログラム 22.16 に埋設配管関連の処理を示す。1~38 行が配管設定処理である。引数で与えられた配管構造をもとに BuriedPipe のインスタンスを作成する。40~53 行は送水条件設定処理であり、温度または水量に変化があった場合には逆行列更新フラグをたてる。55~70 行と 72~81 行はそれぞれ埋設管からの熱流と出口温度の計算処理であり、式 22.53 と式 22.54 の実装である。

プログラム 22.16　埋設配管関連の処理

Popolo.HVAC.ThermalLoad.Wall class

```
 1 /// <summary>配管を追加する</summary>
 2 /// <param name="node">追加する質点番号</param>
 3 /// <param name="pitch">敷設ピッチ[m]</param>
 4 /// <param name="length">配管総延長[m]</param>
 5 /// <param name="branchNumber">分岐の数[本]</param>
 6 /// <param name="iDiameter">内径[m]</param>
 7 /// <param name="oDiameter">外径[m]</param>
 8 /// <param name="tubeConductivity">配管材の熱伝導率[W/(mK)]</param>
 9 public void AddPipe(int node, double pitch, double length, int branchNumber,
10   double iDiameter, double oDiameter, double tubeConductivity)
11 {
12   updateUMatrix();  //resS を設定
13   needToUpdateUINVMatrix = true;
14   double lambdaUF, lambdaLF, thkUF, thkLF;
15   if (node == 0)
16   {
17     lambdaUF = layers[0].ThermalConductivity;
18     thkUF = Math.Min(oDiameter, 0.5 * layers[0].Thickness);
19   }
20   else
21   {
22     lambdaUF = layers[node - 1].ThermalConductivity;
23     thkUF = Math.Min(oDiameter, layers[node - 1].Thickness);
24   }
25   if (node == layers.Length)
26   {
27     lambdaLF = layers[node - 1].ThermalConductivity;
28     thkLF = Math.Min(oDiameter, 0.5 * layers[node - 1].Thickness);
29   }
30   else
31   {
32     lambdaLF = layers[node].ThermalConductivity;
33     thkLF = Math.Min(oDiameter, layers[node].Thickness);
34   }
35   BuriedPipe bp = new BuriedPipe(pitch, length, branchNumber, iDiameter, oDiameter,
36     tubeConductivity, lambdaUF, lambdaLF, resS[node], resS[node + 1], thkUF, thkLF);
37   bPipes[node] = bp;
38 }
39
40 /// <summary>送水条件を設定する</summary>
41 /// <param name="mNumber">質点番号</param>
42 /// <param name="flowRate">通水量[kg/s]</param>
43 /// <param name="temperature">水温[C]</param>
44 public void SetInletWater(int mNumber, double flowRate, double temperature)
45 {
46   BuriedPipe bp = bPipes[mNumber];
47   if (bp.WaterFlowRate != flowRate || bp.InletWaterTemperature != temperature)
48   {
49     needToUpdateUINVMatrix = true;
50     bp.SetFlowRate(flowRate);
51     bp.SetWaterTemperature(temperature);
52   }
53 }
54
55 /// <summary>埋設管からの熱流[W]を取得する</summary>
56 /// <param name="mNumber">質点番号</param>
57 /// <returns>埋設管からの熱流[W]</returns>
58 public double GetHeatTransferFromPipe(int mNumber)
59 {
60   BuriedPipe bp = bPipes[mNumber];
```

```
  if (bp.WaterFlowRate == 0) return 0;
  double tm1, tp1;
  if (mNumber == 0) tm1 = SolAirTemperatureF;
  else tm1 = tempAndHumid[mNumber - 1];
  if (mNumber == layers.Length + 1) tp1 = SolAirTemperatureB;
  else tp1 = tempAndHumid[mNumber + 1];
  double bf = uP[mNumber] * bp.InletWaterTemperature + uPF[mNumber] * tm1
    - uPM[mNumber] * tempAndHumid[mNumber] + uPB[mNumber] * tp1;
  return bf * capS[mNumber] * Area / timeStep;
}

/// <summary>埋設配管の出口水温[C]を取得する</summary>
/// <param name="mNumber">質点番号</param>
/// <returns>埋設配管の出口水温[C]</returns>
public double GetOutletWaterTemperature(int mNumber)
{
  BuriedPipe bp = bPipes[mNumber];
  if (bp.WaterFlowRate == 0) return 0;
  double hs = GetHeatTransferFromPipe(mNumber);
  return bp.InletWaterTemperature - hs / (WATER_SPECIFICHEAT * bp.WaterFlowRate);
}
```

【例題 22.2】

表 22.4 の構成を持つ床の非定常熱伝導を計算せよ。内部の初期温度は 16 ℃、床上の相当温度は 30 ℃、床下の相当温度は 16 ℃ とし、600 秒刻みで 4 時間の計算を行うこととする。また、総合熱伝達率は上下ともに 9.3 W/(m^2·K)とする。

表 22.4　床の構成

No.	材料名称	熱伝導率 [W/(m·K)]	容積比熱 [kJ/(m^3·K)]	厚み [m]
0	フローリング	0.120	520	0.012
1	合板	0.160	720	0.009
2	ポリスチレンフォーム	0.035	80	0.020
3	合板	0.160	720	0.009
4	非密閉空気層	-	-	0.050
5	合板	0.160	720	0.009

【解】

プログラム 22.17 に処理を示す。4~10 行で壁層の配列を作り、11 行で Wall のインスタンスを作成する。面積は 5m^2 としたが、相当温度は一定のため、何 m^2 であっても温度分布は変わらない。13~18 行でタイムステップと初期化処理を行い 20~28 行で 600 秒ずつ計算を進めて書き出しを行う。出力をグラフ化すると図 22.8 が得られる。熱の流入により断熱層よりも上側の温度が上昇するが、120 min 程度経過するとほぼ定常となることがわかる。

プログラム 22.17　壁体熱流テスト 1

```
/// <summary>壁体熱流テスト</summary>
private static void wallTest1()
{
  WallLayer[] layers = new WallLayer[6];
  layers[0] = new WallLayer("フローリング", 0.120, 520, 0.012);
  layers[1] = new WallLayer("合板", 0.160, 720, 0.009);
  layers[2] = new WallLayer("ポリスチレンフォーム", 0.035, 80, 0.020);
  layers[3] = new WallLayer("合板", 0.160, 720, 0.009);
  layers[4] = new AirGapLayer("非密閉空気層", false, 0.05);
  layers[5] = new WallLayer("合板", 0.160, 720, 0.009);
  Wall wall = new Wall(5, layers);

  wall.TimeStep = 600;
  wall.Initialize(16);
  wall.SolAirTemperatureF = 30;
  wall.SolAirTemperatureB = 16;
  wall.ConvectiveCoefficientF = wall.ConvectiveCoefficientB = 4.8;
  wall.RadiativeCoefficientF = wall.RadiativeCoefficientB = 4.5;

  for (int i = 0; i <= 24; i++)
  {
    Console.Write((i * wall.TimeStep).ToString("F0"));
    for (int j = 0; j < wall.Temperatures.Length; j++)
      Console.Write("," + wall.Temperatures[j].ToString("F2"));
    Console.WriteLine();
    wall.Update();
  }
}
```

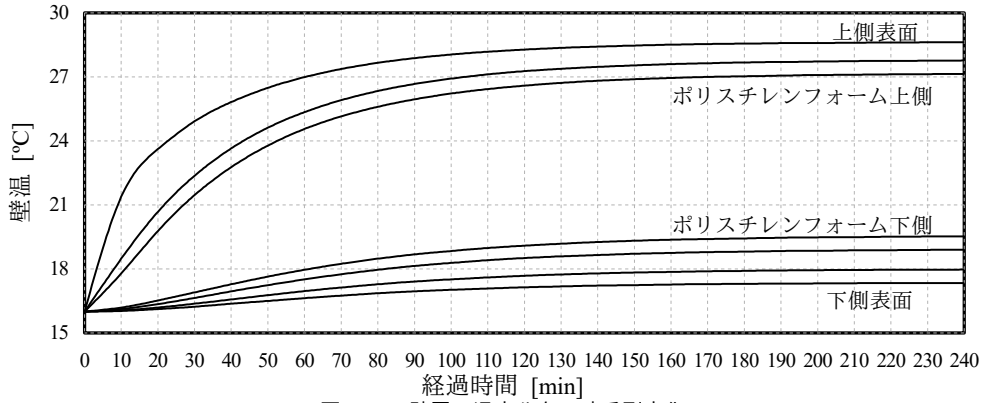


図 22.8　壁層の温度分布の時系列変化

【例題 22.3】

例題 22.2 の床の第 1 層（合板）を下記仕様の PCM に取り替えた場合、時系列温度変化はどのようになるか検討せよ。

表 22.5　PCM の物性

状態	温度区分	熱伝導率 [W/(m·K)]	容積比熱 [kJ/(m³·K)]
固相	~19 ºC	0.190	5,000
二相	19~23 ºC	0.205	21,000
液相	23 ºC ~	0.220	3,000

【解】

プログラム 22.18 に処理を示す。6~9 行で PCM 材料を設定した他は、ほぼ例題 22.2 と同じである。熱容量が大きいため、24 時間の計算を行う。出力をグラフ化すると図 22.9 が得られる。液相から二相へ、二相から固相へ相変化する 19 ºC と 23 ºC でそれぞれ変曲点が発生することが確認できる。

プログラム 22.18　壁体熱流テスト 2

```
 1 /// <summary>壁体熱流テスト:潜熱蓄熱材</summary>
 2 private static void wallTest2()
 3 {
 4   WallLayer[] layers = new WallLayer[6];
 5   layers[0] = new WallLayer("フローリング", 0.120, 520, 0.012);
 6   layers[1] = new PCMWallLayer("PCM", 19, 23, 0.02,
 7     new WallLayer("固体", 0.190, 5000, 0.02),
 8     new WallLayer("平衡", 0.205, 21000, 0.02),
 9     new WallLayer("液体", 0.220, 3000, 0.02));
10   layers[2] = new WallLayer("ポリスチレンフォーム", 0.035, 80, 0.020);
11   layers[3] = new WallLayer("合板", 0.160, 720, 0.009);
12   layers[4] = new AirGapLayer("非密閉空気層", false, 0.05);
13   layers[5] = new WallLayer("合板", 0.160, 720, 0.009);
14   Wall wall = new Wall(5, layers);
15
16   wall.TimeStep = 600;
17   wall.Initialize(16);
18   wall.ConvectiveCoefficientF = wall.ConvectiveCoefficientB = 4.8;
19   wall.RadiativeCoefficientF = wall.RadiativeCoefficientB = 4.5;
20
21   wall.SolAirTemperatureF = 30;
22   wall.SolAirTemperatureB = 16;
23
24   for (int i = 0; i <= 144; i++)
25   {
26     Console.Write((i * wall.TimeStep).ToString("F0"));
28     for (int j = 0; j < wall.Temperatures.Length; j++)
29       Console.Write("," + wall.Temperatures[j].ToString("F2"));
30     Console.WriteLine();
31     wall.Update();
32   }
33 }
```

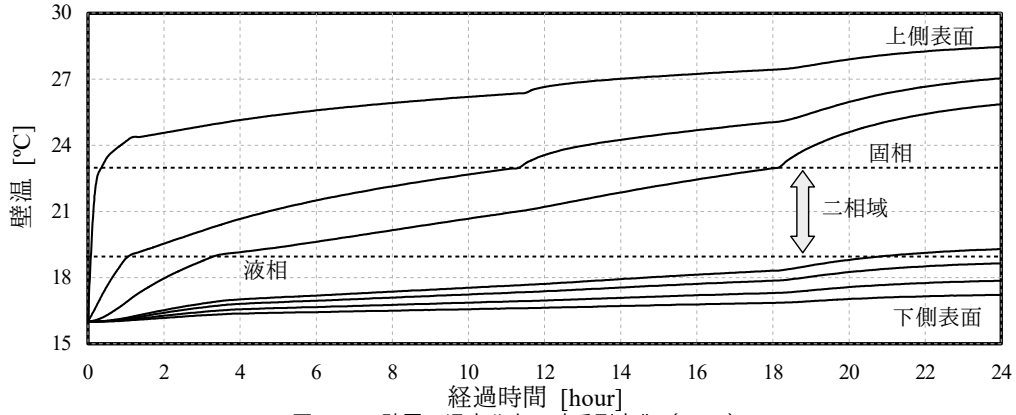


図 22.9　壁層の温度分布の時系列変化（PCM）

【例題 22.4】

例題 22.2 の床のフローリングと合板の間に下記仕様の配管を埋設した場合の時系列変化を検討せよ。

配管敷設ピッチ：0.05 m、配管総延長：200 m、分岐の数：40

配管内径：0.0040 m、配管外径：0.0046 m、床面積：5 m^2

水量：0.014 kg/s、入口水温：20 ℃

配管熱伝導率：0.47 W/(m·K)　（架橋ポリエチレン）

【解】

プログラム 22.19 に処理を示す。21、22 行が配管埋設の処理である。出力をグラフ化すると図 22.10 が得られる。初期温度（16 ℃）が入口水温（20 ℃）よりも低いため、最初は配管から熱が供給されるが、20 分程度で床が温まり、冷却側に変化する。2 時間程度で定常状態となり、配管からは 100W 程度の冷熱が供給される。例題 22.2 と比較すると、床冷房の効果により、床上表面温度が約 2 ℃ 低くなることがわかる。

プログラム 22.19　壁体熱流テスト 3

```
 1 /// <summary>壁体熱流テスト:床冷暖房</summary>
 2 private static void wallTest3()
 3 {
 4   WallLayer[] layers = new WallLayer[6];
 5   layers[0] = new WallLayer("フローリング", 0.120, 520, 0.012);
 6   layers[1] = new WallLayer("合板", 0.160, 720, 0.009);
 7   layers[2] = new WallLayer("ポリスチレンフォーム", 0.035, 80, 0.020);
 8   layers[3] = new WallLayer("合板", 0.160, 720, 0.009);
 9   layers[4] = new AirGapLayer("非密閉空気層", false, 0.05);
10   layers[5] = new WallLayer("合板", 0.160, 720, 0.009);
11   Wall wall = new Wall(5, layers);
12
13   wall.TimeStep = 600;
14   wall.Initialize(16);
15   wall.ConvectiveCoefficientF = wall.ConvectiveCoefficientB = 4.8;
16   wall.RadiativeCoefficientF = wall.RadiativeCoefficientB = 4.5;
17
18   wall.SolAirTemperatureF = 30;
19   wall.SolAirTemperatureB = 16;
20
21   wall.AddPipe(1, 0.05, 200, 40, 0.004, 0.0046, 0.47);
22   wall.SetInletWater(1, 0.014, 20);
23
24   for (int i = 0; i <= 24; i++)
25   {
26     Console.Write((i * wall.TimeStep).ToString("F0"));
28     for (int j = 0; j < wall.Temperatures.Length; j++)
29       Console.Write("," + wall.Temperatures[j].ToString("F2"));
30     Console.WriteLine("," + wall.GetHeatTransferFromPipe(1).ToString("F2")
31       + "," + wall.GetOutletWaterTemperature(1).ToString("F2"));
32     wall.Update();
33   }
34 }
```

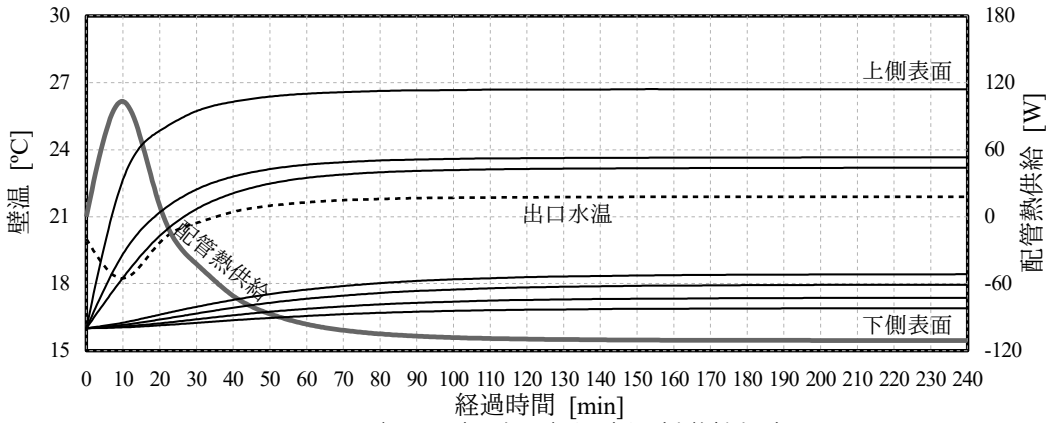


図 22.10　壁層の温度分布の時系列変化（床放射冷房）

【例題 22.5】

下記の仕様の木繊維板が 20 ºC、0.008 kg/kg で平衡状態にあったとする。温度が 21 ºC にステップ変化した場合に、壁体温湿度分布はどのように変動するか計算せよ。壁層は 3 分割する。また、熱水分同時移動解析とした場合と顕熱流のみを考慮した場合とで結果を比較せよ。

熱伝導率：0.1116 W/(m^2·K)、容積比熱：585 kJ/(m^3·K)、厚み：0.018 m

湿気伝導率：4.694 × 10^{-6} (kg/s)/((kg/kg)·m)、空隙率：0.788

吸湿係数 κ：3080 kg/m^3、放湿係数 ν：1.715 kg/(m^3·K)

【解】

プログラム 22.20 に処理を示す。5~8 行で壁層リストを作り、これを用いて 10、11 行で Wall インスタンスを作成する。WallA は熱水分同時移動モデルである。16~19 行で境界条件を定め、B 側の熱伝達率を 0 にすることで断熱境界とする。F 側温度を 21ºC とし、4 時間（240×60 秒）の計算を行う。出力をグラフ化すると図 22.11 が得られる。顕熱流のみの計算結果に比較すると、水分移動を考慮した方が温度変動が緩やかであることがわかる。温度上昇により平衡含水率が低下して壁の絶対湿度が上昇する。この結果、水分放出が生じて気化熱により温度上昇が緩慢になる。

プログラム 22.20　壁体熱流テスト 4

```
 1 /// <summary>壁体熱流テスト:熱水分同時移動</summary>
 2 /// <remarks>松本衛:博士論文 pp.8-10 より</remarks>
 3 private static void wallTest4()
 4 {
 5   WallLayer[] layers = new WallLayer[3];
 6   for(int i=0;i<layers.Length;i++)
 7     layers[i] = new WallLayer
 8       ("木繊維板", 0.1116, 585, 0.000004694, 0.788, 3080, 1.715, 0.006);
 9
10   Wall wallA = new Wall(1, layers, true);
11   Wall wallB = new Wall(1, layers, false);
12
13   wallA.TimeStep = wallB.TimeStep = 60;
14   wallA.Initialize(20, 0.008);
15   wallB.Initialize(20);
16   wallA.ConvectiveCoefficientF = wallB.ConvectiveCoefficientF = 4.8;
17   wallA.ConvectiveCoefficientB = wallB.ConvectiveCoefficientB = 0;
18   wallA.RadiativeCoefficientF = wallB.RadiativeCoefficientF =
19     wallA.RadiativeCoefficientB = wallB.RadiativeCoefficientB = 0.0;
20
21   wallA.SolAirTemperatureF = wallB.SolAirTemperatureF = 21;
22   wallA.SolAirTemperatureB = wallB.SolAirTemperatureB = 20;
23   wallA.HumidityRatioF = wallB.HumidityRatioF = 0.008;
24   wallA.HumidityRatioB = wallB.HumidityRatioB = 0.008;
25   for (int i = 0; i < 240; i++)
26   {
28     Console.WriteLine((i * 60) + ", " +
29       wallA.Temperatures[0].ToString("F3") + ", " +
30       wallB.Temperatures[0].ToString("F3") + ", " +
31       wallA.Humidities[0].ToString("F6"));
32     wallA.Update();
33     wallB.Update();
34   }
35 }
```

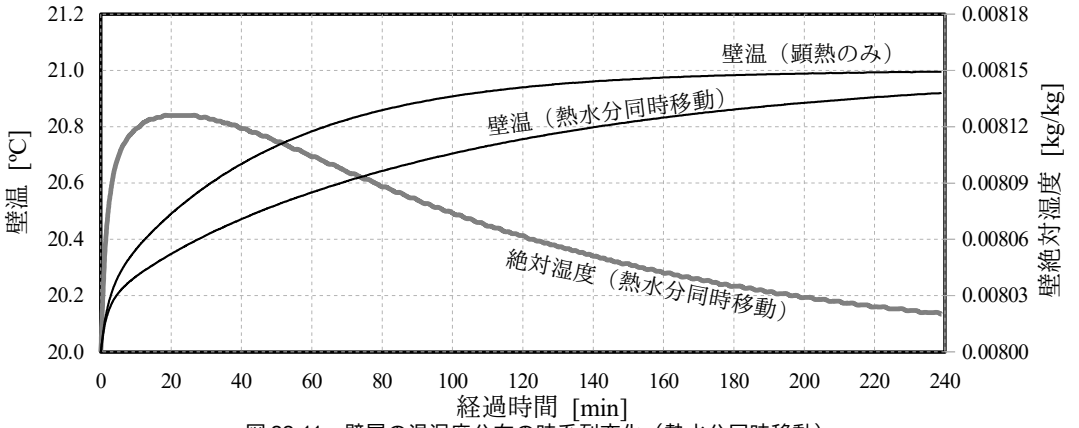


図 22.11　壁層の温湿度分布の時系列変化（熱水分同時移動）

【第 22 章 記号表】

A	：面積 **[m²]**	T^*	：現在の壁温 **[K]**
a_s	：日射吸収率 **[-]**	T_{cor}	：修正後温度 **[K]**
CAP	：熱容量 **[J/(m²·K)]**	T_m	：平均温度 **[K]**
c_{pw}	：水の定圧比熱 **[kJ/(kg·K)]**	T_{PC}	：相変化温度 **[K]**
$c\rho$	：容積比熱 **[J/(m³·K)]**	T_{Sol}	：相当温度 **[K]**
d	：壁層の厚み **[m]**	T_S	：平均放射温度 **[K]**
d_{Pi}	：内径 **[m]**	T_{ZN}	：ゾーンの乾球温度 **[K]**
d_{Po}	：外径 **[m]**	v	：屋外の平均風速 **[m/s]**
F_S	：天空への形態係数 **[-]**	v_s	：屋外壁面近傍の平均風速 **[m/s]**
H_S	：床冷暖房による供給熱 **[W/m²]**	α_w	：配管内表面対流熱伝達率 **[W/(m²·K)]**
I_W	：傾斜面に入射する日射 **[W/m²]**	$\alpha_{i(c+r)}$	：室内側総合熱伝達率 **[W/(m²·K)]**
K	：熱貫流率 **[W/(m²·K)]**	$\alpha_{i(c)}$	：室内側対流熱伝達率 **[W/(m²·K)]**
k_c	：対流熱伝達率比 **[-]**	$\alpha_{i(r)}$	：室内側放射熱伝達率 **[W/(m²·K)]**
k_r	：放射熱伝達率比 **[-]**	$\alpha_{o(c)}$	：屋外側対流熱伝達率 **[W/(m²·K)]**
L	：総配管長 **[m]**	ε	：長波長放射率 **[-]**
M	：壁層の数 **[層]**	ε_P	：配管の熱通過有効度 **[-]**
m_w	：水の質量流量 **[kg/s]**	Δt	：タイムステップ **[sec]**
Q_c	：対流による熱流 **[W]**	λ	：熱伝導率 **[W/(m·K)]**
Q_r	：放射による熱流 **[W]**	λ_P	：配管材料の熱伝導率 **[W/(m·K)]**
R	：熱抵抗 **[(m²·K)/W]**	η_P	：フィン効率 **[-]**
R_N	：夜間放射 **[W/m²]**	σ	：ステファン・ボルツマン係数 **=5.67×10-8 W/(m²·K⁴)**
R_{SLW}	：壁面以外からの放射 **[W/m²]**	ν	：単位絶対温度差に対する放湿係数 **[kg/m³/(kg/kg)]**
T	：現在の壁温 **[K]**	κ	：単位絶対湿度差に対する吸湿係数 **[kg/m³/K]**

subscripts

m	：壁層番号

【第 22 章 参考文献】

22.1） 石野久彌, 品川浩一, 他: 試して学ぶ熱負荷 HASPEE ~新最大熱負荷計算法~, 序文, 2012

22.2） 松尾陽, 横山浩一, 石野久彌, 川元昭吾: 空調設備の動的熱負荷計算入門, 日本建築設備士協会, 1980

22.3） Stephenson and Mitalas: Cooling Load Calculations by Thermal Response Factor Method, ASHRAE Transactions, Vol.73, 1967

22.4） 宇田川光弘: パソコンによる空気調和計算法, オーム社, 1986

22.5） 日本機械学会 伝熱工学資料 改訂第 5 版, I.基礎編, 第 2 章 対流熱伝達率, 丸善出版, 2013

22.6） 伊藤直明, 佐藤鑑, 岡樹生, 木村建一: 自然風下における建築物外壁面の対流熱伝達について, 環境工学における市街地風の変動とその影響に関する総合的研究 その 5, 日本建築学会論文報告集, 学術講演要旨集, 42, pp.522, 1967

22.7） 富樫英介, 中川純, 前真之, 高瀬幸造: シミュレーションに基づく潜熱蓄熱型放射暖房システムの性能評価 (第 1 報) 潜熱蓄熱材料を含む壁体および多数室連成モデルの開発, 空気調和・衛生工学会 学術講演会論文集, pp.1199-1202, 2011

22.8） 石田建一: 潜熱蓄熱パネルを用いたふく射暖冷房シミュレーション, 空気調和・衛生工学会 学術講演会論文集, pp.1041-1044, 1994

22.9）　鈴木憲三, 西安信, 荒谷登: 相当ふく射気温を用いた多数室建物の伝熱計算法 第 1 報 解析精度と放射伝熱算法の改善, 空気調和・衛生工学会論文集, No.41, pp.95-103, 1989
22.10）空気調和衛生工学便覧 第 14 版 第 17 章 冷暖房負荷 表 17.18 材料の熱定数表
22.11）鉾井修一: 湿った建築壁体の熱的特性に関する基礎的研究, 京都大学博士学位請求論文, 1986
22.12）松本衛: 建築壁体における熱・水分の同時移動および水分蓄積に関する研究, 京都大学博士学位請求論文, 1978.11
22.13）前田敏男, 松本衛: 室内壁表面の吸放湿を考えた場合の室内湿度の計算, 日本建築学会論文報告集, pp.45-48, 1959
22.14）尾崎明仁, 渡辺俊行, 高瀬秀芳, 辻丸達憲: 建築の熱・水分・空気連成シミュレーション その 1 壁体の吸放湿を考慮した温湿度計算, 日本建築学会大会学術講演梗概集, pp.439-440, 2004
22.15）宇田川光弘, 近藤靖史, 秋元孝之, 長井達夫: シリーズ<建築光学> 建築環境工学 熱環境と空気環境 初版, p.125, 朝倉書店, 2009
22.16）松波晴人, 松本衛: 蒸気拡散支配・熱・水分移動方程式の適応限界 : 非線形近似の適応範囲, 日本建築学会学術講演梗概集, pp.927-928, 1992
22.17）尾崎明仁, 浦野良美, 渡辺俊行, 龍有二, 村田義郎, 平佐幸男: 期間熱負荷に及ぼす降水の影響について, 日本建築学会計画系論文報告集, Vol.400, pp.13-23, 1989
22.18）尾崎明仁, 須貝高, 渡辺俊行, 龍有二, 赤司泰義, 山崎繁, 湯浅孝, 佐藤章造: 水分ポテンシャルによる湿気移動解析 -湿流の駆動力-, 日本建築学会計画系論文報告集, Vol.488, pp.17-24, 1996
22.19）高田暁, 宇野勇治, 太田昌宏: 土壁の吸放湿特性に関する研究 平衡含水率曲線の測定と居室の温湿度解析, 日本建築学会 学術講演梗概集, pp.237-238, 2013
22.20）大橋達也, 原田和典, 宮林正幸: スギ集成材の平衡含水率・寸法変化率・熱伝導率と燃え止まり性能の関係, 日本建築学会近畿支部研究報告集, 構造系 (53), pp.265-268, 2013
22.21）李在永, 原田和典: 高強度コンクリートの平衡含水率と熱伝導率の測定, 日本建築学会近畿支部研究報告集, 環境系 (51), pp.109-112, 2011
22.22）ASHRAE Fundamentals Handbook, Chapter 3 Heat Transfer,Table.3 Emittances and Absorptances for Some Surfaces, pp.3.8, 1997

第23章　熱負荷計算 2: 日射遮蔽
(Heat Load Calculation 2: Window Coverings)

23.1　概要

建築に窓を設けることには、日照による明るさ感の向上、冬季日射熱取得による暖房負荷の軽減、優れた眺望の獲得などの長所がある一方で、夏季日射熱取得による冷房負荷の増大という短所がある。このため、日射を適切に遮蔽して熱取得を制御する技術が生み出されてきた。例えば、建物の南側に落葉樹を植樹すれば、夏季には枝葉が日射熱を遮る一方で、冬季には落葉により日射が建物まで到達する。また、窓面上部に設けられた庇は太陽高度の高い夏季の日射を遮る一方で、高度の低い冬季の日射を招き入れる。建物内に設けられることの多いカーテンやブラインドも日射遮蔽技術の代表例である。

日射遮蔽技術の効果を知るためには、太陽と遮蔽物の位置関係を表現し、時々刻々変化する太陽位置に対して、遮蔽物が窓面にどのように影を落とすのか、遮蔽物がどのように日射を反射するのか、などを計算する必要がある。本章では代表的な屋外側日射遮蔽技術である庇・袖壁・ルーバーと、屋内側日射遮蔽技術であるブラインドの計算方法について解説する。

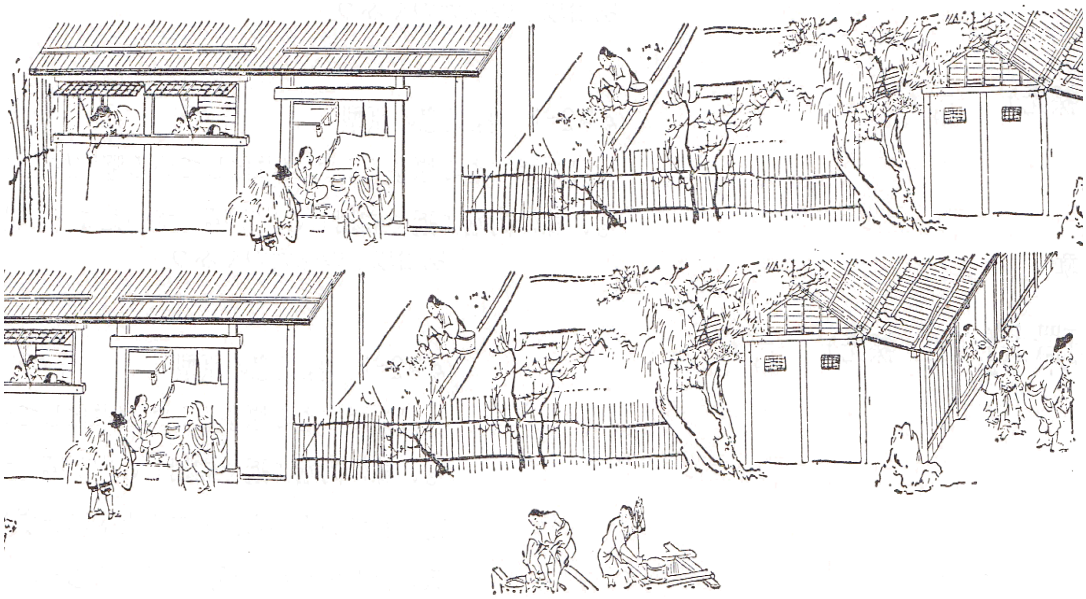


図 23.1　信貴山縁起[23.1)]

（平安時代の絵巻物であるが、日射遮蔽を目的とした蔀戸と連子窓の存在が確認できる）

23.2　理論

23.2.1　庇・袖壁・格子ルーバー

詳細は第 24 章で解説するが、ガラスは短波長を透過し、長波長を反射するという特性を持つため、一旦、室内の物体に吸収されてしまった日射熱は屋外に放出されない。例えば屋内側に設けたブラインドは短波長の一部を屋外に反射できるが、一部の日射熱はブラインドが吸収してしまい、その吸収熱は放射（長波長）により直接にガラス面を透過して屋外に放出させることはできない。従って、日射はガラスを透過させる前に遮蔽することが効果的であり、屋外側で日射を遮蔽する庇・袖壁・格子ルーバーなどはその代表的技術である。可動・仮設的な設備ではなく意匠として成立させながら建築的に取り込んだものは特にブリーズ・ソレイユ（Brise-Soleil）と呼ばれる[†1)]。

日除けの計算については既に多くの文献で理論と計算法が示されている[23.8) 23.9) 23.10)]。本書では宇田川の方法に従い、解析的に計算ができる庇・袖壁・格子ルーバーについて解説する。第 25 章の熱負荷計算で必要となる最低限の基礎式のみを挙げるため、図表を交えた詳細な解説に関しては原著を確認されたい。また、落葉樹のように複雑な形状の場合に関しては、樋口らがモンテカルロ法を用いて近似する方法を提案している[23.7)]。

1）庇

庇が落とす影[†2)]の形状は庇と窓の位置関係や太陽位置によって様々であるが、図 23.2 に示すようにモデル化すれば、式 23.1 で日陰部分の面積を計算することができる。図の p 点は庇の端部から落ちた影の先端であり、この位置に応じて d_{shWA}, d_{shWB}, d_{shHA}, d_{shHB} の計算式を式 23.2~23.10に示すように切り替える。なお、d_{pW} と d_{pH} の計算法については第 6 章 6.2.2 節で解説した。

$$A_{sh}=d_{shWA}d_{shHA}+0.5\left(d_{shWA}+d_{shWB}\right)\left(d_{shHB}-d_{shHA}\right) \tag{23.1}$$

$$d'_{shHA}=\begin{cases} d_{mW}\dfrac{d_{pH}}{d_{pW}}-d_{mH} & \left(d_{mW}<d_{pW}\right) \\ d_{pH}-d_{mH} & \left(d_{pW}\le d_{mW}\right) \end{cases} \tag{23.2}$$

$$d_{shHA}=min\left(d_{wH},max\left(0,d'_{shHA}\right)\right) \tag{23.3}$$

$$d'_{shHB}=\begin{cases} \left(d_{mW}+d_{wW}\right)\dfrac{d_{pH}}{d_{pW}}-d_{mH} & \left(d_{mW}+d_{wW}<d_{pW}\right) \\ d_{pH}-d_{mH} & \left(d_{pW}\le d_{mW}+d_{wW}\right) \end{cases} \tag{23.4}$$

$$d_{shHB}=min\left(d_{wH},max\left(0,d'_{shHB}\right)\right) \tag{23.5}$$

$$d'_{shWA}=\begin{cases} 0 & \left(d_{pH}\le d_{mH}\right) \\ \left(d_{mW}+d_{wW}\right)-d_{mH}\dfrac{d_{pW}}{d_{pH}} & \left(d_{mH}<d_{pH}\right) \end{cases} \tag{23.6}$$

$$d_{shWA}=min\left(d_{wW},max\left(0,d'_{shWA}\right)\right) \tag{23.7}$$

$$d'_{shWB}=\begin{cases} \left(d_{mW}+d_{wW}\right)-d_{pW} & \left(d_{pH}\le d_{mH}+d_{wH}\right) \\ \left(d_{mW}+d_{wW}\right)-\left(d_{mH}+d_{wH}\right)\dfrac{d_{pW}}{d_{pH}} & \left(d_{mH}+d_{wH}<d_{pH}\right) \end{cases} \tag{23.8}$$

†1　名付け親はル・コルビジェであり[23.6)]、「Brise」と「Soleil」はそれぞれフランス語で「砕く」と「日射」を意味する。自然との相対し方がいかにも海外の建築家らしい。

†2　「日影」は日照がある部分、「日陰」または「影」は日照が無い部分である。

$$d_{shWB}=min\left(d_{wH},max\left(0,d'_{shWB}\right)\right) \tag{23.9}$$

$$d_{mW}=\begin{cases} d_{mWA} & (0<d_{pW}) \\ d_{mWB} & (d_{pW}\leq 0) \end{cases} \tag{23.10}$$

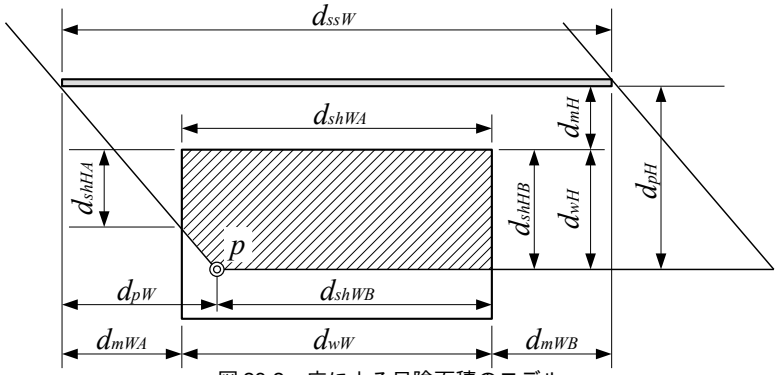


図 23.2　庇による日陰面積のモデル

2）無限長の庇

庇長さが無限大とみなせる場合には d_{pW}=0、d_{shWA}=d_{shWB}、d_{shHA}=d_{shHB} となるため、これらの条件を式 23.1~23.3に代入して、日陰の面積は式 23.11 で計算できる。

$$A_{sh}=min\left(d_{wH},max\left(0,d_{pH}-d_{mH}\right)\right)\times d_{wW} \tag{23.11}$$

3）袖壁

袖壁は庇を 90º 回転させたものとみなすことで計算できる。具体的には式 23.2~23.11における d_{pH} と d_{pW}、d_{wH} と d_{wW}、d_{shHA} と d_{shWA}、d_{shHA} と d_{shWA}、d_{mH} と d_{mW} を入れ換えた後に式 23.1 を適用する。

4）格子ルーバー（バルコニー）

窓の四周に日除けがまわった場合の計算は、図 23.3 に示すようにモデル化する。図 23.3 左は日除けが窓よりも大きい場合（格子ルーバー）、右は日除けが窓よりも小さい場合（開口部のあるバルコニーなど）である。

窓面端部から日影の上下左右端部までの距離をそれぞれ、d_{HA}、d_{HB}、d_{WA}、d_{WB} とすれば、日陰面積は式 23.12 で計算できる。ただし図 23.3 左の場合には d_{HB}=d_{WB}=0 である。d_{HA}、d_{HB}、d_{WA}、d_{WB} は式 23.12~23.16で計算する。

$$A_{sh}=d_{wW}\left(d_{HA}+d_{HB}\right)+\left(d_{WA}+d_{WB}\right)\left(d_{wH}-d_{HA}-d_{HB}\right) \tag{23.12}$$

$$d_{WA}=min\left(d_{wW},max\left(0,d_{pW}-d_{mWA}\right)\right) \tag{23.13}$$

$$d_{WB}=min\left(d_{wW},max\left(0,-d_{pW}-d_{mWB}\right)\right) \tag{23.14}$$

$$d_{HA}=min\left(d_{wH},max\left(0,d_{pH}-d_{mHA}\right)\right) \tag{23.15}$$

$$d_{HB}=min\left(d_{wH},max\left(0,-d_{pH}-d_{mHB}\right)\right) \tag{23.16}$$

5）日陰面積率

窓の計算を行う場合には窓面に対する日陰の面積の比率（日陰面積率）を用いることが多い。予め式 23.17 の係数 R_s を各部に乗じて縮尺を変更しておけば、窓面積が 1 m^2 に調整されるため、式 23.1、式 23.11、式 23.12 を適用して日陰面積率を直接に計算できる。

$$R_s=\left(d_{wW}\times d_{wH}\right)^{-0.5} \tag{23.17}$$

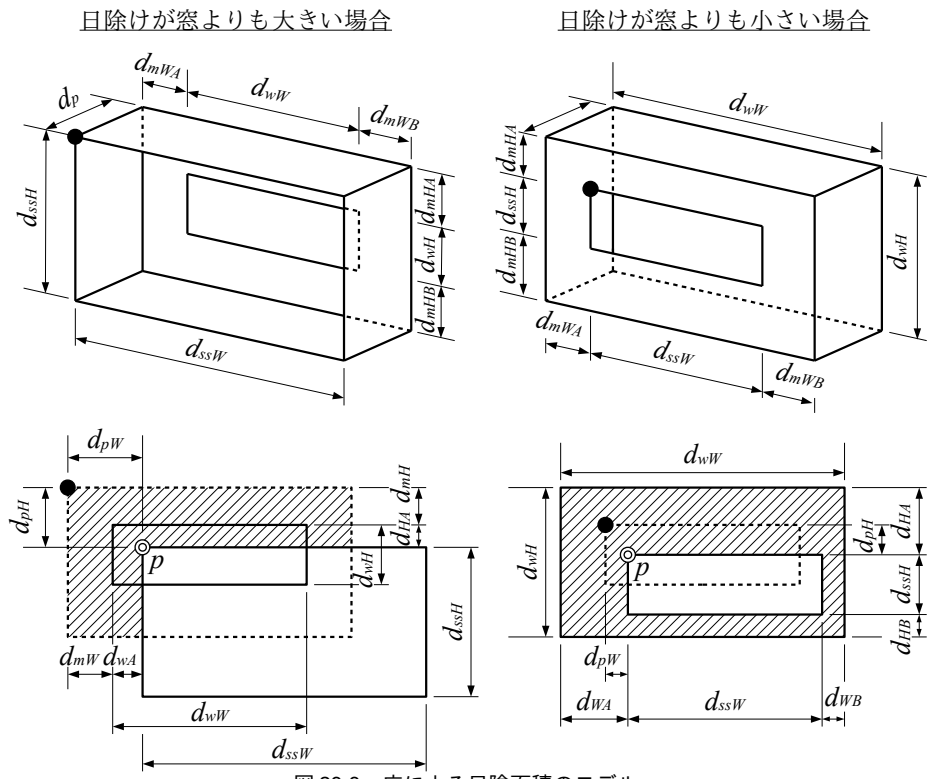


図 23.3　庇による日陰面積のモデル

23.2.2　ブラインド

オフィスで用いられる代表的な日射遮蔽技術としてヴェネシャン・ブラインド（Venetian blind[†1)]）があり、近年では住宅においても使用される例が増えている。スラットと呼ばれる水平の羽根が重なった構造となっており、この角度（スラット角）を変えることで入射する日射に対する特性を調整することができる。従って、スラット角に応じて透過率・反射率・吸収率が変化するため、単純なカーテンと比べるとやや計算が複雑である。本書では ISO 15099 に定められた計算法について解説する[23.11)]。本計算法はスラットの断面形状が平であり、その表面が完全拡散反射[†2)]であることを仮定している。なお、断面形状が複雑な場合や、鏡面反射成分[†3)]を考慮する必要がある場合に関しては、木下らが計算法を提案している[23.4) 23.5)]。

1）モデル構造と基礎式

図 23.4 にヴェネシャン・ブラインドの熱移動モデルを示す。2 枚のスラットで囲まれた 1 スパンをモデル化する。上側を U、下側を D と表現する。スラットは N 個の領域に分割し[†4)]、屋外側から順番に通し番号をつける。τ と ρ はそれぞれ透過率と反射率であり、上側と下側で別々の値を設定可能とする。実際にブラインドメーカーの多くは表裏で色を変えたツートンカラーの製品を販売しており、このような場合には透過率と反射率で調整する。スラット材の色および表面仕上げと反射率の関係を

†1　名前が示す通り、水の都ヴェネチアが発祥だと言われている。水面に反射する太陽光が眩しいため、夏の日中にはスラットを閉め切り、夕方には開放して涼しい風を入れるという使い方をしていたらしい。

†2　このような理想的な反射をランバート反射と呼ぶ。

†3　文献 23.3 には JIS R 3106 によって測定したスラット材表面の光学特性が記載されている。ホワイトやグレイに関しては艶の有無に関わらず鏡面反射成分は 10%を超えない。一方、ブラックの場合には艶ありで 80%、艶なしで 20%が鏡面反射成分となり、ランバート反射の仮定はやや現実にそぐわない。ただし、ブラックの場合にはそもそも全反射率が 5%程度しか無く、日射の殆どが吸収されるため、窓全体での計算に関しての誤差は大きくないだろう。

†4　分割数は任意であるが、ISO 15099 によれば 5 よりも大きな値としても大した精度向上は望めない。

表 23.1 に示す。

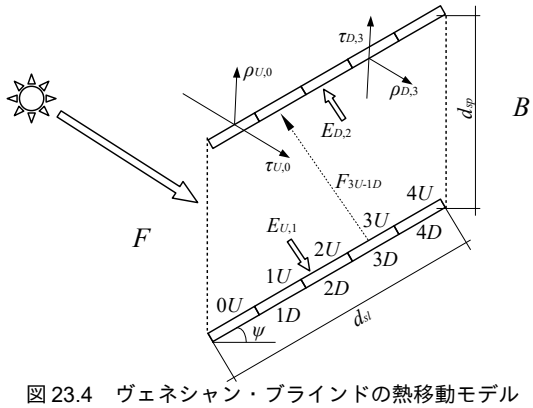


図 23.4　ヴェネシャン・ブラインドの熱移動モデル

表 23.1　スラット材の色と表面仕上げと反射率[23.3)]

表面仕上げ	ホワイト	アイボリー	グレイ	ブラック
艶あり	0.73	0.66	0.55	0.05
艶なし	0.75	0.69	0.56	0.05

$F_{q\to s}$ [-]は領域 q から領域 s をみる形態係数である。ψ [radian]はスラット角、d_{sp} [m]と d_{sl} [m]はそれぞれスパンとスラットの長さである。$E_{U,i}$ [W]および $E_{D,i}$ [W]は反射や透過を繰り返して最終的に i 番目の分割領域に上側または下側から入射する日射量であり、式 23.18 と 23.19 で計算できる。ただし E_{bnd} [W]は屋外あるいは屋内から直接に入射する日射であり、計算法は後述する。

$$E_{U,i}=\tau_U\sum_{j=0}^{N}E_{U,j}F_{Dj\to Ui}+\rho_D\sum_{j=0}^{N}E_{D,j}F_{Dj\to Ui}+E_{bnd,U,i} \tag{23.18}$$

$$E_{D,i}=\tau_D\sum_{j=0}^{N}E_{D,j}F_{Uj\to Di}+\rho_U\sum_{j=0}^{N}E_{U,j}F_{Uj\to Di}+E_{bnd,D,i} \tag{23.19}$$

式 23.18 と 23.19 を行列表現すると式 23.20 となり、これを[E]について解くと式 23.21 が得られる。[E]と[B]は長さ $2N$ のベクトルであり、s 番目の要素 E_s と B_s はそれぞれ式 23.22 と 23.23 で表される。[A]は $2N\times 2N$ の係数行列、[IA]は単位行列から[A]を減じた行列であり、s 行 q 列の成分 IA_{sq} は式 23.24 で表される。δ_{sq} はクロネッカーのデルタ（$s=q$ の場合に 1、その他の成分は 0）である。

$$[\boldsymbol{E}]=[\boldsymbol{A}][\boldsymbol{E}]+[\boldsymbol{B}] \tag{23.20}$$

$$[\boldsymbol{E}]=([\boldsymbol{I}]-[\boldsymbol{A}])^{-1}[\boldsymbol{B}]=[\boldsymbol{IA}]^{-1}[\boldsymbol{B}] \tag{23.21}$$

$$E_s=\begin{cases}E_{U,s} & (s<N)\\ E_{D,(s-N)} & (N\le s)\end{cases} \tag{23.22}$$

$$B_s=\begin{cases}E_{bnd,U,s} & (s<N)\\ E_{bnd,D,(s-N)} & (N\le s)\end{cases} \tag{23.23}$$

$$IA_{sq}=\begin{cases}\delta_{sq}-\tau_U F_{Dq\to Us} & ((s<N)\wedge(q<N))\\ -\rho_D F_{D(q-N)\to Us} & ((s<N)\wedge(N\le q))\\ \delta_{sq}-\tau_D F_{U(q-N)\to D(s-N)} & ((N\le s)\wedge(N\le q))\\ -\rho_U F_{Uq\to D(s-N)} & ((N\le s)\wedge(q<N))\end{cases} \tag{23.24}$$

式 23.21 を計算してそれぞれのスラットに入射する日射が計算できれば式 23.25 および式 23.26 を用いて F 側開口部と B 側開口部へ入射する日射を求めることができる。

$$E_{F,dif}=\tau_U\sum_{j=0}^{N}E_{U,j}F_{Dj\to F}+\rho_D\sum_{j=0}^{N}E_{D,j}F_{Dj\to F}+\tau_D\sum_{j=0}^{N}E_{D,j}F_{Uj\to F}+\rho_U\sum_{j=0}^{N}E_{U,j}F_{Uj\to F} \tag{23.25}$$

$$E_{B,dif}=\tau_U\sum_{j=0}^{N}E_{U,j}F_{Dj\to B}+\rho_D\sum_{j=0}^{N}E_{D,j}F_{Dj\to B}+\tau_D\sum_{j=0}^{N}E_{D,j}F_{Uj\to B}+\rho_U\sum_{j=0}^{N}E_{U,j}F_{Uj\to B} \tag{23.26}$$

2）モデル内の要素間の形態係数

式 23.18 と 23.19 を計算するためにはスラットの分割領域間の形態係数が必要となる。各々の分割領域の面積は等しいため、形態係数の相反則により、*i* 番目の領域から *j* 番目の領域を見る形態係数と、*j* 番目の領域から *i* 番目の領域を見る形態係数は等しい（$F_{Ui\text{-}Dj}=F_{Dj\text{-}Ui}$）。また、それぞれのスラットは平行しているため、隣接する分割領域相互の形態係数も等しい（$F_{Ui\text{-}Dj}=F_{U(i\text{-}1)\text{-}D(j\text{-}1)}=F_{U(i+1)\text{-}D(j+1)}$）。従って、図 23.5 に示すように 2*N*-1 個（F 側と B 側にそれぞれ *N*-1 ずつ動いたスラット）の形態係数を計算すれば、モデルに含まれる総てのスラット間の形態係数を得ることができる。図 23.5 における *n* 番目のスラットへの形態係数を F_{Pn} とすれば、*i* 番目から *j* 番目のスラットへの形態係数 $F_{Ui\text{-}Dj}$ は $F_{P(j\text{-}i+N\text{-}1)}$ となる。スラット端部間の距離を d_n とすれば、これを用いてスラット間の形態係数 F_{Pn} を式 23.27 で計算できる。ただし d_n は式 23.28 で計算する。また、d_{Rsp} [-]は分割領域長さを基準としたスラットのスパン d_{sp} の長さであり、式 23.29 で計算する。

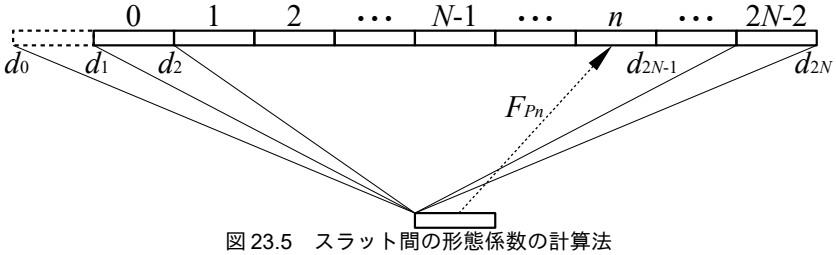


図 23.5　スラット間の形態係数の計算法

$$F_{Pn}=0.5\left(d_n+d_{n+2}\right)-d_{n+1} \tag{23.27}$$

$$d_n=\sqrt{\ 2(n-N)\,d_{Rsp}\sin\psi+(n-N)^2+d_{Rsp}^2} \tag{23.28}$$

$$d_{Rsp}=N\,d_{sp}/d_{sl} \tag{23.29}$$

i 番目のスラット分割領域の表側から F 側の開口部への形態係数 $F_{Ui\text{-}F}$ は式 23.30 で計算する。形状が対称であるため、これは(*N*-*i*)番目のスラット分割領域の裏側から B 側の開口部への形態係数 $F_{D(N\text{-}i)\text{-}B}$ に等しい。また、形態係数の相反則により式 23.30 の第 3 項と第 4 項が成立する。

$$F_{Ui-F}=F_{D(N-i)-B}=F_{F-Ui}/d_{Rsp}=F_{B-D(N-i)}/d_{Rsp}=0.5\left(d_{N-i}-d_{N-1-i}+1\right) \tag{23.30}$$

i 番目のスラット分割領域の表側から B 側の開口部への形態係数 $F_{Ui\text{-}B}$ は、形態係数の総和則により式 23.31 で計算できる。

$$F_{Ui-B}=1-\left(F_{Ui-F}+\sum_{j=0}^{N}F_{Ui-Dj}\right) \tag{23.31}$$

3）拡散日射に対する透過率・反射率・吸収率

F 側開口部から入射する拡散日射は、F 側開口部から各分割領域に対する形態係数によって振り分ける。従って、1 単位の日射が与えられたとき、式 23.18 と 23.19 における E_{bnd} は式 23.32 および式 23.33 である。

$$E_{bnd,U,i}=F_{F-Ui} \tag{23.32}$$

$$E_{bnd,D,i}=F_{F-Di} \tag{23.33}$$

式 23.21 の[*B*]に式 23.32 および式 23.33 を代入して行列演算を行うことで[*E*]を求め、さらに式 23.25

および式 23.26 に代入すればそれぞれの開口部に入射する日射が得られる。ただし、B 側開口部に関してはF 側開口部から直接に B 側に到達する拡散日射（F 側開口部から B 側開口部への形態係数 $F_{F\to B}$ に等しい）を足し込む必要がある。これらは 1 単位の日射に対する値であるから、F 側から入射する拡散日射に対する透過率 $\tau_{dif,F}$ と反射率 $\rho_{dif,F}$ に等しい（式 23.34、式 23.35）。B 側開口部から入射する拡散日射についても同様の方法で計算することができる。なお、透過率と反射率の合計値を 1 から差し引けば吸収率 $\alpha_{dif,F}$ が求まる。

$$\tau_{dif,F} = E_{B,dif} + F_{F\to B} \tag{23.34}$$

$$\rho_{dif,F} = E_{F,dif} \tag{23.35}$$

4）直達日射に対する透過率・反射率・吸収率

直達日射に対する透過率などを計算するためには、まず、スラットのどの範囲に日射が入射するかを計算する必要がある。図 23.6 に示すように、スラット角 ψ と見かけの太陽高度 ϕ の関係によって、スラットの表側または裏側に日射が入射する。日射の入射範囲 d_{Rrad} は式 23.36 で計算できる。d_{Rrad} は分割領域長さで基準化された値であるため、$N < d_{Rrad}$ の場合にはすべてのスラットに日射が届く。なお、スラットの厚みは無視するため、$-\psi = \phi$ の場合には日射は 100 %透過する。

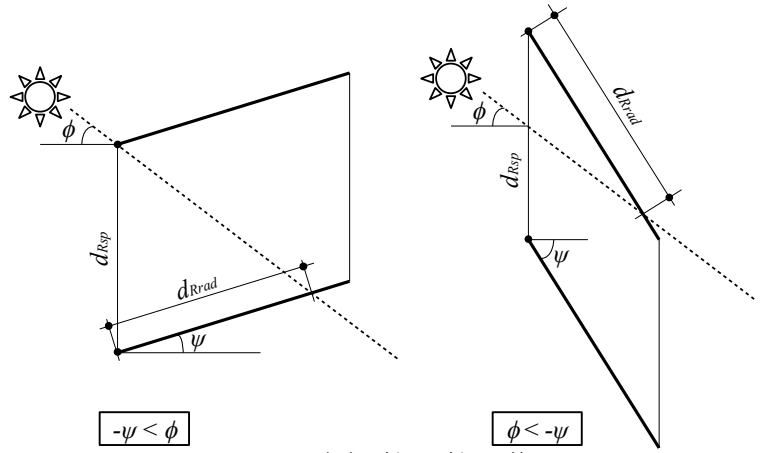


図 23.6　直達日射の入射する範囲

$$d_{Rrad} = \frac{d_{Rsp}}{\cos\psi\,|\tan\psi + \tan\varphi|} \tag{23.36}$$

それぞれの分割領域に入射する日射は式 23.37 で計算する。拡散日射の場合と同様に、これらを式 23.21 に設定して[E]を求め、式 23.25 および式 23.26 により開口部へ入射する日射を計算する。この結果を用いて、直達日射に対する透過率と反射率は式 23.38 と 23.39 で計算できる。ただし、式 23.38 の第二項はスラットに一度もあたらずに室内側へ到達する成分である[†1)]。

$$E_{bnd,i} = \frac{Min(1, Max(0, d_{Rrad} - i))}{d_{Rrad}} \tag{23.37}$$

$$\tau_{dir,F} = E_{B,dif} + \frac{Max(0, d_{Rrad} - N)}{d_{Rrad}} \tag{23.38}$$

$$\rho_{dir,F} = E_{F,dif} \tag{23.39}$$

†1　ISO 15099 ではスラットにあたって拡散光となる成分と直接に室内に入射する成分を分離し、直達日射に対して二種類の透過率を定義している。第 24 章に述べるように本書の窓モデルでは両者を区別しないため、ブラインドのモデルにおいても両者を合わせて「透過率」とした。

23.3　計算法

23.3.1　庇・袖壁・格子ルーバーの計算

外部日除けによる日陰面積率を計算する SunShade クラスを開発する。

プログラム 23.1 に日除けの形状を示す列挙型の定義を示す。

プログラム 23.1　日除け形状の列挙型定義

Popolo.ThermalLoad.SunShade class

```
/// <summary>日除けの形状</summary>
public enum Shapes
{
  /// <summary>無し</summary>
  None = 0,
  /// <summary>水平庇</summary>
  Horizontal = 1,
  /// <summary>水平庇（無限大長）</summary>
  LongHorizontal = 2,
  /// <summary>袖壁（左）</summary>
  VerticalLeft = 3,
  /// <summary>袖壁（右）</summary>
  VerticalRight = 4,
  /// <summary>袖壁（両方）</summary>
  VerticalBoth = 5,
  /// <summary>袖壁（無限大長:左）</summary>
  LongVerticalLeft = 6,
  /// <summary>袖壁（無限大長:右）</summary>
  LongVerticalRight = 7,
  /// <summary>袖壁（無限大長:両方）</summary>
  LongVerticalBoth = 8,
  /// <summary>格子ルーバー</summary>
  Grid = 9
}
```

プログラム 23.2 にプロパティ定義を示す。ただし、日除けの形状によっては使用しないプロパティもある。

プログラム 23.2　プロパティの定義

Popolo.ThermalLoad.SunShade class

```
/// <summary>傾斜面を取得する</summary>
public ImmutableIncline Incline { get; private set; }

/// <summary>形状を取得する</summary>
public Shapes Shape { private set; get; }

/// <summary>窓高さ[m]を取得する</summary>
public double WinHeight { private set; get; }

/// <summary>窓幅[m]を取得する</summary>
public double WinWidth { private set; get; }

/// <summary>張り出し幅[m]を取得する</summary>
public double Overhang { private set; get; }

/// <summary>上部マージン[m]を取得する</summary>
public double TopMargin { private set; get; }

/// <summary>下部マージン[m]を取得する</summary>
public double BottomMargin { private set; get; }

/// <summary>左側マージン[m]を取得する</summary>
public double LeftMargin { private set; get; }

/// <summary>右側マージン[m]を取得する</summary>
public double RightMargin { private set; get; }
```

列挙型の定義からわかるように、日除けの形状は様々であり、コンストラクタの引数のみで初期化の方法を切り替えると間違いが生じやすい。そこでコンストラクタは隠蔽し、代わりに初期化を行う static メソッドを用意する。プログラム 23.3 に日除けの初期化処理を示す。4~11 行はサイズ 0 の日除けの作成処理である。太陽位置に関わらず、常に日陰面積率として 0.0 を出力する。このような日除けを作っておけば null 参照エラーが生じにくい。以降の行のメソッドではいずれも日除けの形状に応

じてプロパティを初期化し、SunShade クラスのインスタンスを出力する。

プログラム **23.3**　日除けの初期化処理

Popolo.ThermalLoad.SunShade class

```
/// <summary>コンストラクタを隠蔽</summary>
private SunShade() { }

/// <summary>サイズ 0 の日除けを作成する</summary>
/// <returns>サイズ 0 の日除け</returns>
public static SunShade MakeEmptySunShade()
{
  SunShade ss = new SunShade();
  ss.Shape = Shapes.None;
  return ss;
}

/// <summary>水平庇（無限長）を作成する</summary>
/// <param name="wWidth">窓幅[m]</param>
/// <param name="wHeight">窓高[m]</param>
/// <param name="depth">張り出し幅[m]</param>
/// <param name="tMargin">上部マージン[m]</param>
/// <param name="incline">傾斜面</param>
/// <returns>水平庇（無限長）</returns>
public static SunShade MakeHorizontalSunShade
  (double wWidth, double wHeight, double depth, double tMargin, ImmutableIncline incline)
{
  SunShade ss = new SunShade();
  ss.Shape = Shapes.LongHorizontal;
  ss.WinWidth = wWidth;
  ss.WinHeight = wHeight;
  ss.Overhang = depth;
  ss.TopMargin = tMargin;
  ss.Incline = incline;
  return ss;
}

/// <summary>水平庇を作成する</summary>
/// <param name="wWidth">窓幅[m]</param>
/// <param name="wHeight">窓高[m]</param>
/// <param name="depth">張り出し幅[m]</param>
/// <param name="lMargin">左側マージン[m]</param>
/// <param name="rMargin">右側マージン[m]</param>
/// <param name="tMargin">上部マージン[m]</param>
/// <param name="incline">傾斜面</param>
/// <returns>水平庇</returns>
public static SunShade MakeHorizontalSunShade(double wWidth, double wHeight, double depth,
  double lMargin, double rMargin, double tMargin, ImmutableIncline incline)
{
  SunShade ss = MakeHorizontalSunShade(wWidth, wHeight, depth, tMargin, incline);
  ss.Shape = Shapes.Horizontal;
  ss.LeftMargin = lMargin;
  ss.RightMargin = rMargin;
  return ss;
}

/// <summary>袖壁（無限長）を作成する</summary>
/// <param name="wWidth">窓幅[m]</param>
/// <param name="wHeight">窓高[m]</param>
/// <param name="depth">張り出し幅[m]</param>
/// <param name="sMargin">横側マージン[m]</param>
/// <param name="isLeftSide">左側か否か（右の場合は false）</param>
/// <param name="incline">傾斜面</param>
/// <returns>袖壁（無限長）</returns>
public static SunShade MakeVerticalSunShade(double wWidth, double wHeight, double depth,
  double sMargin, bool isLeftSide, ImmutableIncline incline)
{
  SunShade ss = new SunShade();
  ss.WinWidth = wWidth;
  ss.WinHeight = wHeight;
  ss.Overhang = depth;
  if (isLeftSide)
  {
    ss.Shape = Shapes.LongVerticalLeft;
    ss.LeftMargin = sMargin;
  }
  else
  {
    ss.Shape = Shapes.LongVerticalRight;
    ss.RightMargin = sMargin;
```

```
 76   }
 77   ss.Incline = incline;
 78   return ss;
 79 }
 80
 81 /// <summary>袖壁を作成する</summary>
 82 /// <param name="wWidth">窓幅[m]</param>
 83 /// <param name="wHeight">窓高[m]</param>
 84 /// <param name="depth">張り出し幅[m]</param>
 85 /// <param name="sMargin">横側マージン[m]</param>
 86 /// <param name="isLeftSide">左側か否か（右の場合は false）</param>
 87 /// <param name="tMargin">上部マージン[m]</param>
 88 /// <param name="bMargin">下部マージン[m]</param>
 89 /// <param name="incline">傾斜面</param>
 90 /// <returns>袖壁</returns>
 91 public static SunShade MakeVerticalSunShade(double wWidth, double wHeight, double depth,
 92   double sMargin, bool isLeftSide, double tMargin, double bMargin, ImmutableIncline incline)
 93 {
 94   SunShade ss = MakeVerticalSunShade(wWidth, wHeight, depth, sMargin, isLeftSide, incline);
 95   if (isLeftSide) ss.Shape = Shapes.VerticalLeft;
 96   else ss.Shape = Shapes.VerticalRight;
 97   ss.TopMargin = tMargin;
 98   ss.BottomMargin = bMargin;
 99   return ss;
100 }
101
102 /// <summary>袖壁（無限長）を作成する</summary>
103 /// <param name="wWidth">窓幅[m]</param>
104 /// <param name="wHeight">窓高[m]</param>
105 /// <param name="depth">張り出し幅[m]</param>
106 /// <param name="lMargin">左側マージン[m]</param>
107 /// <param name="rMargin">右側マージン[m]</param>
108 /// <param name="incline">傾斜面</param>
109 /// <returns>袖壁（無限長）</returns>
110 public static SunShade MakeVerticalSunShade(double wWidth, double wHeight, double depth,
111   double lMargin, double rMargin, ImmutableIncline incline)
112 {
113   SunShade ss = MakeVerticalSunShade(wWidth, wHeight, depth, lMargin, true, incline);
114   ss.RightMargin = rMargin;
115   ss.Shape = Shapes.LongVerticalBoth;
116   return ss;
117 }
118
119 /// <summary>袖壁を作成する</summary>
120 /// <param name="wWidth">窓幅[m]</param>
121 /// <param name="wHeight">窓高[m]</param>
122 /// <param name="depth">張り出し幅[m]</param>
123 /// <param name="lMargin">左側マージン[m]</param>
124 /// <param name="rMargin">右側マージン[m]</param>
125 /// <param name="tMargin">上部マージン[m]</param>
126 /// <param name="bMargin">下部マージン[m]</param>
127 /// <param name="incline">傾斜面</param>
128 /// <returns>袖壁</returns>
129 public static SunShade MakeVerticalSunShade(double wWidth, double wHeight, double depth,
130   double lMargin, double rMargin, double tMargin, double bMargin, ImmutableIncline incline)
131 {
132   SunShade ss = MakeVerticalSunShade(wWidth, wHeight, depth, lMargin, rMargin, incline);
133   ss.Shape = Shapes.VerticalBoth;
134   ss.TopMargin = tMargin;
135   ss.BottomMargin = bMargin;
136   return ss;
137 }
138
139 /// <summary>格子ルーバーを作成する</summary>
140 /// <param name="wWidth">窓幅[m]</param>
141 /// <param name="wHeight">窓高[m]</param>
142 /// <param name="depth">張り出し幅[m]</param>
143 /// <param name="lMargin">左側マージン[m]</param>
144 /// <param name="rMargin">右側マージン[m]</param>
145 /// <param name="tMargin">上部マージン[m]</param>
146 /// <param name="bMargin">下部マージン[m]</param>
147 /// <param name="incline">傾斜面</param>
148 /// <returns>格子ルーバー</returns>
149 public static SunShade MakeGridSunShade(double wWidth, double wHeight, double depth,
150   double lMargin, double rMargin, double tMargin, double bMargin, ImmutableIncline incline)
151 {
152   SunShade ss = new SunShade();
153   ss.Shape = Shapes.Grid;
154   ss.WinWidth = wWidth;
155   ss.WinHeight = wHeight;
```

```
156   ss.Overhang = depth;
157   ss.LeftMargin = lMargin;
158   ss.RightMargin = rMargin;
159   ss.TopMargin = tMargin;
160   ss.BottomMargin = bMargin;
161   ss.Incline = incline;
162   return ss;
163 }
```

プログラム 23.4 に日陰面積の計算処理を示す。1~35 行は水平庇の計算処理であり、式 23.1~23.10 の実装である。37~47 行は無限長の水平庇であり式 23.11 の実装である。49~67 行は格子ルーバーの場合であり、式 23.12~23.16の実装である。これらはいずれも static メソッドであり、計算に必要な情報がすべて引数として与えられる。SunShade クラスのインスタンスは、プロパティの情報とこれらのメソッドを組み合わせて自身の日陰面積率を計算する。

プログラム 23.4　日陰面積率の計算処理

Popolo.ThermalLoad.SunShade class

```
  1 /// <summary>水平庇の日陰面積[m2]を計算する</summary>
  2 /// <param name="dpW">庇端部から影端部までの水平距離[m]</param>
  3 /// <param name="dpH">庇端部から影端部までの垂直距離[m]</param>
  4 /// <param name="dwW">窓幅[m]</param>
  5 /// <param name="dwH">窓高さ[m]</param>
  6 /// <param name="dmWA">庇と窓の左端距離[m]</param>
  7 /// <param name="dmH">庇と窓の上端距離[m]</param>
  8 /// <param name="dmWB">庇と窓の右端距離[m]</param>
  9 /// <returns>水平庇の日陰面積[m2]</returns>
 10 private static double computeShadowArea_H
 11   (double dpW, double dpH, double dwW, double dwH, double dmWA, double dmH, double dmWB)
 12 {
 13   if (dpH <= dmH) return 0;
 14
 15   double dmW;
 16   if (0 < dpW) dmW = dmWA;
 17   else dmW = dmWB;
 18   dpW = Math.Abs(dpW);
 19
 20   double dshHA, dshHB, dshWA, dshWB;
 21   double dshHAd, dshHBd, dshWAd, dshWBd;
 22   if (dmW < dpW) dshHAd = dmW * (dpH / dpW) - dmH;
 23   else dshHAd = dpH - dmH;
 24   dshHA = Math.Min(dwH, Math.Max(0, dshHAd));
 25   if (dmW + dwW < dpW) dshHBd = (dmW + dwW) * (dpH / dpW) - dmH;
 26   else dshHBd = dpH - dmH;
 27   dshHB = Math.Min(dwH, Math.Max(0, dshHBd));
 28   dshWAd = (dmW + dwW) - dmH * (dpW / dpH);
 29   dshWA = Math.Min(dwW, Math.Max(0, dshWAd));
 30   if (dpH < dmH + dwH) dshWBd = (dmW + dwW) - dpW;
 31   else dshWBd = (dmW + dwW) - (dmH + dwH) * dpW / dpH;
 32   dshWB = Math.Min(dwW, Math.Max(0, dshWBd));
 33
 34   return dshWA * dshHA + 0.5 * (dshWA + dshWB) * (dshHB - dshHA);
 35 }
 36
 37 /// <summary>無限長の水平庇の日陰面積[m2]を計算する</summary>
 38 /// <param name="dpH">庇端部から影端部までの垂直距離[m]</param>
 39 /// <param name="dwW">窓幅[m]</param>
 40 /// <param name="dwH">窓高さ[m]</param>
 41 /// <param name="dmH">庇と窓の上端距離[m]</param>
 42 /// <returns>無限長の水平庇の日陰面積[m2]</returns>
 43 private static double computeShadowArea_LH(double dpH, double dwW, double dwH, double dmH)
 44 {
 45   if (dpH <= dmH) return 0;
 46   return dwW * Math.Min(dpH - dmH, dwH);
 47 }
 48
 49 /// <summary>格子ルーバーの日陰面積[m2]を計算する</summary>
 50 /// <param name="dpW">庇端部から影端部までの水平距離[m]</param>
 51 /// <param name="dpH">庇端部から影端部までの垂直距離[m]</param>
 52 /// <param name="dwW">窓幅[m]</param>
 53 /// <param name="dwH">窓高さ[m]</param>
 54 /// <param name="dmWA">庇と窓の左端距離[m]</param>
 55 /// <param name="dmHA">庇と窓の上端距離[m]</param>
 56 /// <param name="dmWB">庇と窓の右端距離[m]</param>
 57 /// <param name="dmHB">庇と窓の下端距離[m]</param>
 58 /// <returns>格子ルーバーの日陰面積[m2]</returns>
```

```
59 private static double computeShadowArea_Grid
60   (double dpW, double dpH, double dwW, double dwH, double dmWA, double dmHA, double dmWB, double dmHB)
61 {
62   double dWA = Math.Min(Math.Max(0, dpW - dmWA), dwW);
63   double dWB = Math.Min(Math.Max(0, -(dpW + dmWB)), dwW);
64   double dHA = Math.Min(Math.Max(0, dpH - dmHA), dwH);
65   double dHB = Math.Min(Math.Max(0, -(dpH + dmHB)), dwH);
66   return dwW * (dHA + dHB) + (dWA + dWB) * (dwH - dHA - dHB);
67 }
```

プログラム 23.5 に日陰面積率の計算処理を示す。6~11 行は、太陽位置や日除け形状により即座に日陰面積率が決定できる場合の処理である。16~49 行で、日除けの形状別に面積率を計算する。

プログラム 23.5　日陰面積率の計算処理

Popolo.ThermalLoad.SunShade class

```
 1 /// <summary>日影面積率[-]を計算する</summary>
 2 /// <param name="sun">太陽</param>
 3 /// <returns>日影面積率[-]</returns>
 4 public double GetShadowRate(ImmutableSun sun)
 5 {
 6   //日の出前と日没後はすべて影
 7   if (sun.Altitude <= 0) return 1;
 8   //日除けが無ければ影は無し
 9   if (Shape == Shapes.None) return 0;
10   //太陽が裏面にある場合にはすべて影
11   if (Incline.GetDirectSolarRadiationRate(sun) <= 0) return 1;
12
13   double dpW = Overhang * Math.Tan(Incline.HorizontalAngle - sun.Orientation);
14   double dpH = Overhang * Incline.GetTangentProfileAngle(sun);
15   double sr = 0;
16   switch (Shape)
17   {
18     case Shapes.Horizontal:
19       sr = computeShadowArea_H(dpW, dpH, WinWidth, WinHeight, LeftMargin, TopMargin, RightMargin);
20       break;
21     case Shapes.VerticalLeft:
22       sr = computeShadowArea_H(dpH, dpW, WinHeight, WinWidth, TopMargin, LeftMargin, BottomMargin);
23       break;
24     case Shapes.VerticalRight:
25       sr = computeShadowArea_H(dpH, -dpW, WinHeight, WinWidth, TopMargin, RightMargin, BottomMargin);
26       break;
27     case Shapes.VerticalBoth:
28       if (dpW < 0)
29         sr = computeShadowArea_H(dpH, -dpW, WinHeight, WinWidth, TopMargin, RightMargin, BottomMargin);
30       else sr = computeShadowArea_H(dpH, dpW, WinHeight, WinWidth, TopMargin, LeftMargin, BottomMargin);
31       break;
32     case Shapes.LongHorizontal:
33       sr = computeShadowArea_LH(dpH, WinWidth, WinHeight, TopMargin);
34       break;
35     case Shapes.LongVerticalLeft:
36       sr = computeShadowArea_LH(dpW, WinHeight, WinWidth, LeftMargin);
37       break;
38     case Shapes.LongVerticalRight:
39       sr = computeShadowArea_LH(-dpW, WinHeight, WinWidth, RightMargin);
40       break;
41     case Shapes.LongVerticalBoth:
42       if (dpW < 0) sr = computeShadowArea_LH(-dpW, WinHeight, WinWidth, RightMargin);
43       else sr = computeShadowArea_LH(dpW, WinHeight, WinWidth, LeftMargin);
44       break;
45     case Shapes.Grid:
46       sr = computeShadowArea_Grid
47         (dpW, dpH, WinWidth, WinHeight, LeftMargin, TopMargin, RightMargin, BottomMargin);
48       break;
49   }
50   return sr / (WinHeight * WinWidth);
51 }
```

【例題 23.1】

南面と西面の窓について、庇、袖壁、格子ルーバーを設けた場合の効果を評価するため、7 月 20 日の 1 日の日陰面積率の推移を計算せよ。ただし、窓の大きさは 3,800×2,700mm とする。また、日除けと窓とのマージンは上下左右ともに 100mm とし、日除けの奥行きは 600mm とする。

【解】

プログラム 23.6 に計算処理を示す。出力をグラフ化した結果を図 23.7 に示す。南面の場合、太陽高度が比較的低い 8:00 頃や 15:00 は水平庇と袖壁の日陰面積率は同程度であり、両者を足し合わせた格子ルー

バーの日陰面積率は 100 %となる。太陽高度が高くなるにつれて袖壁の効果は小さく、庇の効果は大きくなる。西面では午後に太陽光が入射する。昼頃は太陽高度が高いために袖壁の上部から日射が入射し、夕方は袖壁と平行する方向から日射が入射するため、時間帯を問わず袖壁の効果は小さい。従って水平庇と格子の効果はほぼ同等となる。太陽高度がさがるにつれて庇の効果も低下し、日没頃には日陰面積率はほぼ 0 となる。

プログラム 23.6　日除けの形状と日陰面積率の関係の計算

```
 1 private static void sunShadeTest()
 2 {
 3   Incline incS = new Incline(Incline.Orientation.S, 0.5 * Math.PI);
 4   Incline incW = new Incline(Incline.Orientation.W, 0.5 * Math.PI);
 5   SunShade[] ss = new SunShade[6];
 6   ss[0] = SunShade.MakeHorizontalSunShade(3.8, 2.7, 0.6, 0.1, 0.1, 0.1, incS);
 7   ss[1] = SunShade.MakeVerticalSunShade(3.8, 2.7, 0.6, 0.1, 0.1, 0.1, 0.1, incS);
 8   ss[2] = SunShade.MakeGridSunShade(3.8, 2.7, 0.6, 0.1, 0.1, 0.1, 0.1, incS);
 9   ss[3] = SunShade.MakeHorizontalSunShade(3.8, 2.7, 0.6, 0.1, 0.1, 0.1, incW);
10   ss[4] = SunShade.MakeVerticalSunShade(3.8, 2.7, 0.6, 0.1, 0.1, 0.1, 0.1, incW);
11   ss[5] = SunShade.MakeGridSunShade(3.8, 2.7, 0.6, 0.1, 0.1, 0.1, 0.1, incW);
12
13   DateTime dTime = new DateTime(2015, 7, 20, 7, 0, 0);
14   Sun sun = new Sun(Sun.City.Tokyo);
15   while (true)
16   {
17     if (dTime.Hour == 19) break;
18     Console.Write(dTime.ToShortTimeString());
19     sun.Update(dTime);
20
21     for (int i = 0; i < ss.Length; i++) Console.Write(", " + ss[i].GetShadowRate(sun).ToString("F3"));
22     Console.WriteLine();
23     dTime = dTime.AddMinutes(5);
24   }
25 }
```

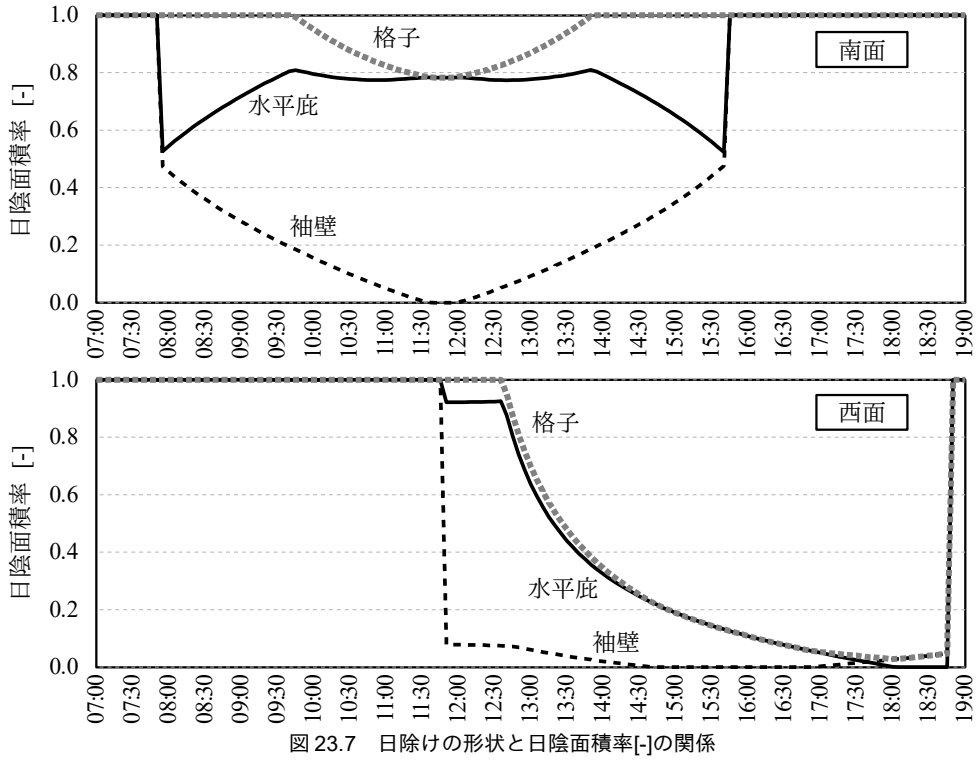


図 23.7　日除けの形状と日陰面積率[-]の関係

23.3.2　ブラインドの計算

1）日射遮蔽物インターフェースの作成

窓ガラスの内外表面または中空層に取り付ける日射遮蔽物のクラスを作成する。プログラム 23.7

に日射遮蔽物を表わす IShadingDevice インターフェースを示す。ブラインド、ロールスクリーン、障子、遮光フィルムなどの具体的な日射遮蔽物はこのインターフェースを実装することで作成する。6 行は有効判定であり、例えばブラインドやロールスクリーンが下がっているか否かを切り替えるために使用する。9 行は光学特性の変化判定である。窓全体の計算において、日射遮蔽物の光学特性の再取得・再計算が必要になるタイミングを判定するために用いる。 12 行は見かけの太陽高度設定処理である。14~21 行が主たるメソッドであり、日射遮蔽物の透過率と反射率を出力する。通常は、入射する日射が拡散日射であるか否か、入射する方向が F 側からか否かによって特性が変わるため、これらを最初の 2 つの引数で判定する。HasPropertyChanged プロパティが true の場合には本メソッドを呼び出して光学特性を更新する。

プログラム 23.7　日射遮蔽物インターフェースの定義

Popolo.ThermalLoad.IShadingDevice interface

```
 1 /// <summary>窓ガラス中空層の日射遮蔽物</summary>
 2 public interface IShadingDevice
 3 {
 4
 5   /// <summary>有効か否か</summary>
 6   bool Pulldowned { get; set; }
 7
 8   /// <summary>光学特性に変化があるか否か</summary>
 9   bool HasPropertyChanged { get; }
10
11   /// <summary>見かけの太陽高度[radian]を設定・取得する</summary>
12   double ProfileAngle { get; set; }
13
14   /// <summary>光学特性を計算する</summary>
15   /// <param name="isDiffuseIrradianceProperties">拡散日射か否か</param>
16   /// <param name="irradianceFromSideF">F 側開口からの日射か否か</param>
17   /// <param name="transmittance">透過率[-]</param>
18   /// <param name="reflectance">反射率[-]</param>
19   void ComputeOpticalProperties
20     (bool isDiffuseIrradianceProperties, bool irradianceFromSideF,
21     out double transmittance, out double reflectance);
22
23 }
```

プログラム 23.8 は IShadingDevice の実装例であり、常に透過率が 100 %となるサイズ 0 の日射遮蔽物 EmptyShadingDevice である。標準の窓ガラスにはこの EmptyShadingDevice を設定しておけば、日射遮蔽物の有無を判定しなくても良くなるため、ソースコードが簡単になる。

プログラム 23.8　サイズ 0 の日射遮蔽物

Popolo.ThermalLoad.EmptyShadingDevice class

```
 1 /// <summary>常に透過率 100%の日射遮蔽物</summary>
 2 public class EmptyShadingDevice : IShadingDevice
 3 {
 4   /// <summary>有効か否か</summary>
 5   public bool Pulldowned { get; set; }
 6
 7   /// <summary>光学特性に変化があるか否か</summary>
 8   public bool HasPropertyChanged { get; private set; }
 9
10   /// <summary>見かけの太陽高度[radian]を設定・取得する</summary>
11   public double ProfileAngle { get; set; }
12
13   /// <summary>新しいインスタンスを初期化する</summary>
14   public EmptyShadingDevice() { HasPropertyChanged = true; }
15
16   /// <summary>光学特性を計算する</summary>
17   /// <param name="isDiffuseIrradianceProperties">拡散日射か否か</param>
18   /// <param name="irradianceFromSideF">F 側開口からの日射か否か</param>
19   /// <param name="transmittance">透過率[-]</param>
20   /// <param name="reflectance">反射率[-]</param>
21   public void ComputeOpticalProperties
22     (bool isDiffuseIrradianceProperties, bool irradianceFromSideF,
23     out double transmittance, out double reflectance)
24   {
```

```
    transmittance = 1.0;
    reflectance = 0.0;
    HasPropertyChanged = false;
  }
}
```

2）ベネシャンブラインドクラスの作成

ベネシャンブラインドを表わす VenetianBlind クラスを作成する。プログラム 23.7 で示した IShadingDevice を実装する。プログラム 23.9 に定数宣言ならびにインスタンス変数、プロパティの定義を示す。ISO15099 に従い、スラット分割数は 5 とする。不要な逆行列演算が生じないように、条件変更に関する判定フラグを 5 行に定義する。また、直達日射は見かけの太陽高度の影響を受けるため、直達日射に関する光学特性更新処理の必要性判定フラグを 8 行に定義する。光学特性としては直達日射に対するものか拡散日射に対するものか、また、F 側からの日射か B 側からの日射かによって変わるため、2×2=4 通りが存在する。ただし、本書のモデルでは、一旦室内に入射した日射はすべて拡散反射すると仮定するため、室内側（B 側）からの直達日射は存在しない。従って 4-1=3 通りの光学特性の計算が必要となり、これらを 10~17 行の変数で管理する。43 行以降はプロパティであるが、見かけの太陽高度やスラット角など、形態係数の再計算が必要となる変更が加えられる場合には、状態値更新フラグをたてる。

プログラム 23.9　定数宣言ならびにインスタンス変数、プロパティの定義

Popolo.ThermalLoad.VenetianBlind class

```
/// <summary>スラット分割数</summary>
private const int SP_N = 5;

/// <summary>逆行列の更新が必要か否か</summary>
private bool needUpdateInverseMatrix = true;

/// <summary>直達日射に対する特性更新が必要か否か</summary>
private bool needUpdateDirectIrradianceProperties = true;

/// <summary>F 側開口部からの拡散日射に対する光学特性</summary>
private double difTauF, difRhoF;

/// <summary>B 側開口部からの拡散日射に対する光学特性</summary>
private double difTauB, difRhoB;

/// <summary>F 側開口部からの直達日射に対する光学特性</summary>
private double dirTauF, dirRhoF;

/// <summary>スラットからの形態係数行列</summary>
private double[,] vFacSlt = new double[SP_N + 2, SP_N + 2];

/// <summary>F 側開口部からの形態係数ベクトル</summary>
private double[] vFacOF = new double[SP_N * 2 + 1];

/// <summary>B 側開口部からの形態係数ベクトル</summary>
private double[] vFacOB = new double[SP_N * 2 + 1];

/// <summary>行列</summary>
private IMatrix iaMatrix = new Matrix(SP_N * 2, SP_N * 2);

/// <summary>逆行列</summary>
private IMatrix iaMatrixINV = new Matrix(SP_N * 2, SP_N * 2);

/// <summary>基準化したスパン長さ[-]</summary>
private double dRspan;

/// <summary>見かけの太陽高度[radian]</summary>
private double pAngle;

/// <summary>スラット角[radian]</summary>
private double sAngle;

/// <summary>有効か否かを設定・取得する</summary>
public bool Pulldowned
{
```

```
46   get { return pullDowned; }
47   set
48   {
49     HasPropertyChanged = (pullDowned != value);
50     pullDowned = value;
51   }
52 }
53
54 /// <summary>光学特性に変化があるか否か</summary>
55 public bool HasPropertyChanged { get; private set; }
56
57 /// <summary>見かけの太陽高度[radian]を設定・取得する</summary>
58 public double ProfileAngle
59 {
60   get { return pAngle; }
61   set
62   {
63     if (pAngle == value) return;
64     pAngle = value;
65     needUpdateDirectIrradianceProperties = true;
66     HasPropertyChanged = true;
67   }
68 }
69
70 /// <summary>スラット角[radian]を設定・取得する (0 で水平) </summary>
71 public double SlatAngle
72 {
73   get { return sAngle; }
74   set
75   {
76     if (sAngle == value) return;
77     sAngle = value;
78     needUpdateInverseMatrix = true;
79     HasPropertyChanged = true;
80   }
81 }
82
83 /// <summary>上側透過率[-]を取得する</summary>
84 public double UpsideTransmittance { get; private set; }
85
86 /// <summary>下側透過率[-]を取得する</summary>
87 public double DownsideTransmittance { get; private set; }
88
89 /// <summary>上側反射率[-]を取得する</summary>
90 public double UpsideReflectance { get; private set; }
91
92 /// <summary>下側反射率[-]を取得する</summary>
93 public double DownsideReflectance { get; private set; }
```

プログラム 23.10 にコンストラクタを示す。スラットの幾何学形状と光学特性（透過率と反射率）が入力である。スラット幅と間隔は絶対長さではなく式 23.29 で示した規準化したスパン長さが大切であるため、これを 112 行で計算して保存する。

プログラム 23.10　コンストラクタ

Popolo.ThermalLoad.VenetianBlind class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="slatWidth">スラット幅</param>
 3 /// <param name="slatSpan">スラット間隔</param>
 4 /// <param name="upsideTransmittance">スラット上側表面透過率[-]</param>
 5 /// <param name="downsideTransmittance">スラット下側表面透過率[-]</param>
 6 /// <param name="upsideReflectance">スラット上側表面反射率[-]</param>
 7 /// <param name="downsideReflectance">スラット下側表面反射率[-]</param>
 8 /// <remarks>slatWidth と slatSpan の単位は同じであれば任意</remarks>
 9 public VenetianBlind(double slatWidth, double slatSpan, double upsideTransmittance,
10   double downsideTransmittance, double upsideReflectance, double downsideReflectance)
11 {
12   dRspan = SP_N * slatSpan / slatWidth;
13   UpsideTransmittance = upsideTransmittance;
14   DownsideTransmittance = downsideTransmittance;
15   UpsideReflectance = upsideReflectance;
16   DownsideReflectance = downsideReflectance;
17 }
```

スラット角変更時には形態係数を更新する必要がある。この処理をプログラム 23.11 に示す。式 23.28 と式 23.29 を用いて 4~11 行でスラット間の距離 d_n を更新する。これと式 23.27 を用いて 16, 17

行で分割領域間の形態係数を算出する。また、式 23.30 と 23.31 の関係をもとに、17~27 行でそれぞれの分割領域から開口部への形態係数を算出する。さらに、形態係数の相反法則（式 23.30）にもとづき、開口部から分割領域への形態係数を 30~37 行で計算する。

プログラム 23.11　形態係数の更新処理

Popolo.ThermalLoad.VenetianBlind class

```
 1 /// <summary>形態係数を更新する</summary>
 2 private void updateViewFactor()
 3 {
 4   //スラット間距離を更新
 5   double[] dn = new double[SP_N * 2 + 1];
 6   double sinPsi = Math.Sin(SlatAngle);
 7   for (int i = 0; i < dn.Length; i++)
 8   {
 9     int bf = (i - SP_N);
10     dn[i] = Math.Sqrt(dRspan * (2 * bf * sinPsi + dRspan) + bf * bf);
11   }
12
13   //スラットからみた形態係数を更新
14   double sum;
15   double[] ff = new double[2 * SP_N - 1];
16   for (int i = 0; i < ff.Length; i++) ff[i] = 0.5 * (dn[i] + dn[i + 2]) - dn[i + 1];
17   for (int i = 0; i < SP_N; i++)
18   {
19     sum = 0;
20     for (int j = 0; j < SP_N; j++)
21     {
22       vFacSlt[i, j] = ff[j - i + SP_N - 1];
23       sum += vFacSlt[i, j];
24     }
25     vFacSlt[i, SP_N] = vFacSlt[SP_N + 1, SP_N - 1 - i] = 0.5 * (dn[SP_N - i] - dn[SP_N - 1 - i] + 1);
26     vFacSlt[i, SP_N + 1] = vFacSlt[SP_N, SP_N - 1 - i] = 1 - (sum + vFacSlt[i, SP_N]);
27   }
28
29   //開口からみた形態係数を更新
30   sum = 0;
31   for (int i = 0; i < SP_N; i++)
32   {
33     vFacOF[i] = vFacOB[vFacOB.Length - 2 - i] = vFacSlt[i, SP_N] / dRspan;
34     vFacOF[i + SP_N] = vFacOB[SP_N - 1 - i] = vFacSlt[SP_N, i] / dRspan;
35     sum += vFacOF[i] + vFacOF[i + SP_N];
36   }
37   vFacOF[SP_N * 2] = vFacOB[SP_N * 2] = 1 - sum;
38 }
```

プログラム 23.12 に光学特性計算のための private メソッドを示す。1~24 行はそれぞれのスラット分割領域から開口部へ向かう日射の計算処理であり、式 23.25 と式 23.26 の実装である。ただしベクトル[*E*]は 26~70 行のメソッドで計算され、引数として与えられる。26~43 行は拡散日射に対する光学特性計算処理である。33 行で式 23.32 と式 23.33 によりスラットに直接入射する日射を求め、34 行で式 23.21 を適用してベクトル[*E*]を計算する。ただし、計算に必要な逆行列$[IA]^{-1}$は 72~93 行のメソッドで予め更新しておく。35 行でスラットから開口に向かう日射を計算し、最後に 36 行で開口部から開口部に直接向かう日射を加算する。38~42 行は B 側開口部の処理であり、F 側と同様である。45~70 行は直達日射に対する光学特性計算処理である。スラット分割領域に入射する日射を式 23.36~23.39を用いて 52~63 行で計算する。その後の処理は拡散日射と同様である。

プログラム 23.12　光学特性計算のための private メソッド

Popolo.ThermalLoad.VenetianBlind class

```
 1 /// <summary>スラットから開口部への入射を計算する</summary>
 2 /// <param name="eVec">各スラットに入射する日射</param>
 3 /// <param name="sideF">F 側開口</param>
 4 /// <param name="sideB">B 側開口</param>
 5 private void computeRateToOpenings(IVector eVec, out double sideF, out double sideB)
 6 {
 7   double td1, td2, td3, td4, rd1, rd2, rd3, rd4;
 8   td1 = td2 = td3 = td4 = rd1 = rd2 = rd3 = rd4 = 0;
```

```
  for (int i = 0; i < SP_N; i++)
  {
    td1 += eVec[i] * vFacSlt[SP_N + 1, i];
    td2 += eVec[i + SP_N] * vFacSlt[SP_N + 1, i];
    td3 += eVec[i + SP_N] * vFacSlt[i, SP_N + 1];
    td4 += eVec[i] * vFacSlt[i, SP_N + 1];
    rd1 += eVec[i] * vFacSlt[SP_N, i];
    rd2 += eVec[i + SP_N] * vFacSlt[SP_N, i];
    rd3 += eVec[i + SP_N] * vFacSlt[i, SP_N];
    rd4 += eVec[i] * vFacSlt[i, SP_N];
  }
  sideB = td1 * UpsideTransmittance + td2 * DownsideReflectance
    + td3 * DownsideTransmittance + td4 * UpsideReflectance;
  sideF = rd1 * UpsideTransmittance + rd2 * DownsideReflectance
    + rd3 * DownsideTransmittance + rd4 * UpsideReflectance;
}

/// <summary>拡散日射に対する光学特性を計算する</summary>
private void updateDiffuseIrradianceProperties()
{
  IVector bVec = new Vector(SP_N * 2);
  IVector eVec = new Vector(SP_N * 2);

  //F 側開口からの拡散日射に対する透過率・反射率
  for (int i = 0; i < bVec.Length; i++) bVec[i] = vFacOF[i];
  LinearAlgebra.Multiplicate(iaMatrixINV, bVec, ref eVec, 1, 0);
  computeRateToOpenings(eVec, out difRhoF, out difTauF);
  difTauF += vFacOF[SP_N * 2];

  //B 側開口からの拡散日射に対する透過率・反射率
  for (int i = 0; i < bVec.Length; i++) bVec[i] = vFacOB[i];
  LinearAlgebra.Multiplicate(iaMatrixINV, bVec, ref eVec, 1, 0);
  computeRateToOpenings(eVec, out difTauB, out difRhoB);
  difTauB += vFacOB[SP_N * 2];
}

/// <summary>直達日射に対する光学特性を計算する</summary>
private void updateDirectIrradianceProperties()
{
  //スラットに直接入射する日射を計算
  IVector bVec = new Vector(SP_N * 2);
  IVector eVec = new Vector(SP_N * 2);

  if (-SlatAngle == ProfileAngle)
  {
    dirTauF = 1.0;
    dirRhoF = 0;
    return;
  }

  double dRrad = dRspan / (Math.Cos(SlatAngle) * Math.Abs(Math.Tan(SlatAngle) + Math.Tan(ProfileAngle)));
  int bf = 0;
  if (ProfileAngle < -SlatAngle) bf = SP_N;
  for (int i = 0; i < SP_N; i++) bVec[i + bf] = Math.Min(1, Math.Max(0, dRrad - i)) / dRrad;
  LinearAlgebra.Multiplicate(iaMatrixINV, bVec, ref eVec, 1, 0);

  //開口部へ向かう日射を計算
  computeRateToOpenings(eVec, out dirRhoF, out dirTauF);
  dirTauF += Math.Max(0, dRrad - SP_N) / dRrad;

  needUpdateDirectIrradianceProperties = false;
}

/// <summary>逆行列を更新する</summary>
private void updateInverseMatrix()
{
  //形態係数を更新
  updateViewFactor();

  for (int i = 0; i < SP_N; i++)
  {
    for (int j = 0; j < SP_N; j++)
    {
      double dlt = 0;
      if (i == j) dlt = 1;
      iaMatrix[i, j] = dlt - (UpsideTransmittance * vFacSlt[i, j]);
      iaMatrix[i, j + SP_N] = -(DownsideReflectance * vFacSlt[i, j]);
      iaMatrix[i + SP_N, j + SP_N] = dlt - (DownsideTransmittance * vFacSlt[j, i]);
      iaMatrix[i + SP_N, j] = -(UpsideReflectance * vFacSlt[j, i]);
    }
```

```
 89   }
 90   LinearAlgebra.GetInverse(ref iaMatrix, ref iaMatrixINV);
 91
 92   needUpdateInverseMatrix = false;
 93 }
```

プログラム 23.13 に光学特性の更新処理を示す。10~15 行はブラインド不使用時の計算であり、この場合には透過率が 100 %となる。17~24 行で必要に応じて光学特性を更新し、直達・拡散の別、F 側・B 側の別に応じて 26~60 行で出力を設定する。43 行に示すように、直達日射の場合には拡散日射とは異なり、スラット角あるいは見かけの太陽高度の変化に併せて光学特性更新処理が必要になる場合があることに注意する。

プログラム 23.13　光学特性の更新処理

Popolo.ThermalLoad.VenetianBlind class

```
  1 /// <summary>光学特性を計算する</summary>
  2 /// <param name="isDiffuseIrradianceProperties">拡散日射か否か</param>
  3 /// <param name="irradianceFromSideF">F 側開口からの日射か否か</param>
  4 /// <param name="transmittance">透過率[-]</param>
  5 /// <param name="reflectance">反射率[-]</param>
  6 public void ComputeOpticalProperties
  7   (bool isDiffuseIrradianceProperties, bool irradianceFromSideF,
  8   out double transmittance, out double reflectance)
  9 {
 10   if (!Pulldowned)
 11   {
 12     transmittance = 1.0;
 13     reflectance = 0.0;
 14     return;
 15   }
 16
 17   //必要に応じて逆行列を更新
 18   if (needUpdateInverseMatrix)
 19   {
 20     updateInverseMatrix();
 21     updateDirectIrradianceProperties();
 22     updateDiffuseIrradianceProperties();
 23     HasPropertyChanged = false;
 24   }
 25
 26   //拡散日射に対する光学特性
 27   if (isDiffuseIrradianceProperties)
 28   {
 29     if (irradianceFromSideF)
 30     {
 31       transmittance = difTauF;
 32       reflectance = difRhoF;
 33     }
 34     else
 35     {
 36       transmittance = difTauB;
 37       reflectance = difRhoB;
 38     }
 39   }
 40   //直達日射に対する光学特性
 41   else
 42   {
 43     if (needUpdateDirectIrradianceProperties)
 44     {
 45       updateDirectIrradianceProperties();
 46       HasPropertyChanged = false;
 47     }
 48
 49     if (irradianceFromSideF)
 50     {
 51       transmittance = dirTauF;
 52       reflectance = dirRhoF;
 53     }
 54     else
 55     {
 56       //F 側と同じ値とする
 57       transmittance = dirTauF;
 58       reflectance = dirRhoF;
 59     }
 60   }
 61 }
```

【例題 23.2】

スラット幅 22.5mm、スラット間隔 25mm のブラインドについて、スラット角と透過率・反射率との関係を計算せよ。ただし、スラットの透過率は上下ともに 0 とし、下側反射率は 0.5、上側反射率は 0.1, 0.3, 0.5, 0.7, 0.9 の 5 種類とする。また、見かけの太陽高度は 30º で固定する。

【解】

計算処理をプログラム 23.14 に示す。出力をグラフ化すると図 23.8 が得られる。直達日射に関しては、見かけの太陽高度とスラット角が一致する 30º で透過率が 1.0 となり、ここから離れるにつれて透過率が下がる。スラット角が大きくなると入射がまずスラットの上側表面にあたるにため、上側表面の反射率の影響を大きく受けるようになる。スラット角が 90º またはで-90º の条件では全閉となるため、スラット表面の反射率とブラインドとしての反射率が一致する。拡散日射の光学特性は形態係数に依存し、透過率は全条件ともに 0 のため、0º を中心に対称形となる。また、拡散日射は指向性を持たないため、反射率に関しても直達日射に比較するとスラット角に対する感度がゆるやかになる。

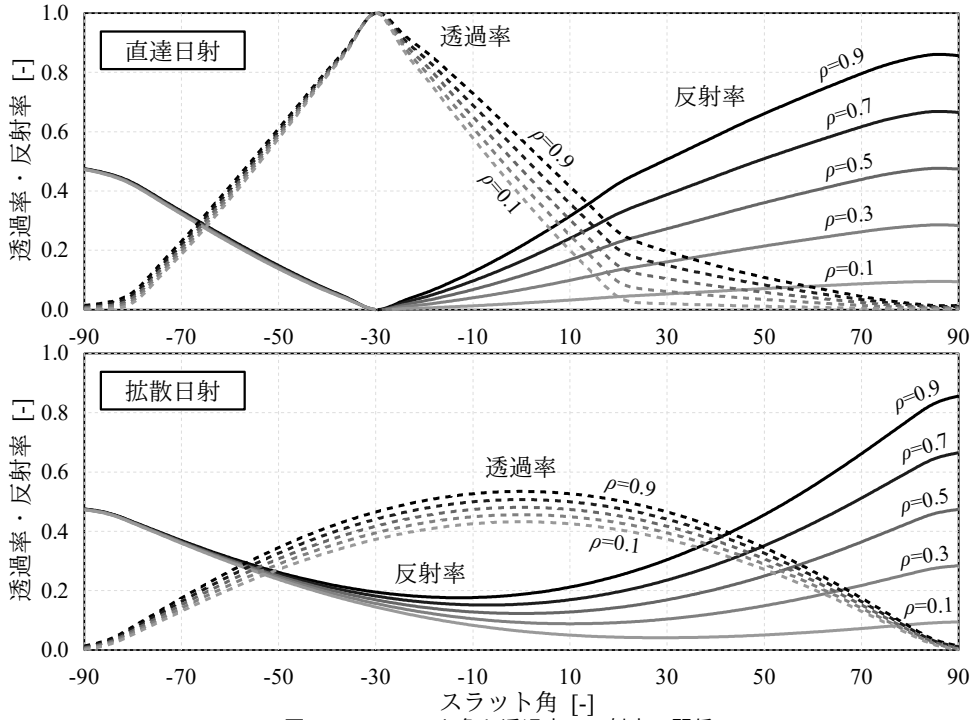


図 23.8　スラット角と透過率・反射率の関係

プログラム **23.14**　ブラインドの光学特性の計算

```
 1 private static void venetianBlindTest()
 2 {
 3   VenetianBlind[] blinds = new VenetianBlind[5];
 4   double[] sltRho = new double[] { 0.1, 0.3, 0.5, 0.7, 0.9 };
 5   for (int i = 0; i < blinds.Length; i++)
 6   {
 7     blinds[i] = new VenetianBlind(25, 22.5, 0, 0, sltRho[i], 0.5);
 8     blinds[i].ProfileAngle = 30 / 180d * Math.PI;
 9   }
10
11   using (StreamWriter sWriter = new StreamWriter
12     ("VenetianBlindTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
13   {
14     sWriter.Write("SlatAngle");
15     for (int i = 0; i < 5; i++) sWriter.Write(", " + sltRho[i] + ":DirTau");
16     for (int i = 0; i < 5; i++) sWriter.Write(", " + sltRho[i] + ":DirRho");
17     for (int i = 0; i < 5; i++) sWriter.Write(", " + sltRho[i] + ":DifTau");
18     for (int i = 0; i < 5; i++) sWriter.Write(", " + sltRho[i] + ":DifRho");
19     sWriter.WriteLine();
20
21     double[] drTau = new double[blinds.Length];
```

```
22      double[] drRho = new double[blinds.Length];
23      double[] dfTau = new double[blinds.Length];
24      double[] dfRho = new double[blinds.Length];
25      for (int i = -90; i <= 90; i += 5)
26      {
27        sWriter.Write(i);
28        for (int j = 0; j < blinds.Length; j++)
29        {
30          blinds[j].SlatAngle = i / 180d * Math.PI;
31          blinds[j].ComputeOpticalProperties(false, true, out drTau[j], out drRho[j]);
32          blinds[j].ComputeOpticalProperties(true, true, out dfTau[j], out dfRho[j]);
33        }
34
35        for (int j = 0; j < 5; j++) sWriter.Write(", " + drTau[j]);
36        for (int j = 0; j < 5; j++) sWriter.Write(", " + drRho[j]);
37        for (int j = 0; j < 5; j++) sWriter.Write(", " + dfTau[j]);
38        for (int j = 0; j < 5; j++) sWriter.Write(", " + dfRho[j]);
39        sWriter.WriteLine();
40      }
41    }
42 }
```

【第 23 章 記号表】

A_{sh}	：日陰面積 **[m²]**	N	：スラットの分割数 **[個]**
d	：長さ **[m]**	R_s	：窓面積を無次元化するための縮尺比 **[-]**
d_R	：基準化した長さ **[-]**	α	：吸収率 **[-]**
d_{Rrad}	：直達日射の到達距離 **[-]**	δ	：クロネッカーのデルタ
d_{sl}	：スラット幅 **[m]**	ψ	：スラット角 **[radian]**
d_{sp}	：スラットスパン **[m]**	ϕ	：見かけの太陽高度 **[radian]**
E	：入射 **[W]**	ρ	：反射率 **[-]**
F	：形態係数 **[-]**	τ	：透過率 **[-]**
添字：			
B	：裏側	m	：マージン
bnd	：境界条件	p	：影の落ちる点
D	：下側	sh	：影
dif	：拡散日射	U	：上側
dir	：直達日射	W	：水平距離
F	：表側	w	：窓
H	：垂直距離		

【第 23 章 参考文献】

23.1） 近代日本建築学発達史, 日本建築学会, 第 8 編 環境工学, p.1404, 1972

23.2） JIS 3107 1998: 板ガラス類の熱抵抗及び建築における熱貫流率の算定方法

23.3） 窓の遮熱性能計算・試験方法の JIS・ISO 化 成果報告書: 経済産業省委託 平成 24 年度国際標準開発事業, 一般社団法人 日本建材・住宅設備産業協会, 平成 25 年 3 月

23.4） 木下泰斗, 赤坂裕, 二宮秀與: ベネシャンブラインドの光学特性の計算法, 日本建築学会環境系論文集, No.617, pp.1-8, 2007

23.5） 木下泰斗, 赤坂裕, 二宮秀與: 板ガラスとベネシャンブラインドとを組み合わせた光学特性の計算法, 日本建築学会環境系論文集, Vol.74, No.639, pp.569-577, 2009

23.6） Le Corbusier: Promlèmes de l'ensoleillement Le Brise-soleil, Œuvre complète, pp.1938–1946, Les éditions d'architecture, Artemis, Zurich, 1946

23.7） 樋口佳樹, 宇田川光弘, 佐藤誠, 木村建一, 周囲環境を考慮した住宅の熱負荷シミュレーションに関する研究 その 1 建築屋外における日射と長波長放射の計算モデル, 日本建築学会計画系論文集, No.544, pp.9-15, 2001

23.8） 宇田川光弘: パソコンによる空気調和計算法, 第 6 章

23.9） 松尾陽, 横山浩一, 石野久彌, 川元省吾: 空調設備の動的熱負荷計算入門, 第二章 日射受熱量, pp.24-28, 日本建築設備士協会, 1980

23.10） 石野久彌, 相賀洋 他: 試して学ぶ熱負荷 HASPEE ~新最大熱負荷計算法~, p.38 ルーバひさしによるガラス面の日照面積率の求め方, 2012, 丸善出版

23.11） ISO 15099: 2003, Thermal performance of windows, doors and shading devices - Detailed calculations

第24章　熱負荷計算 3: 窓面熱取得
(Heat Load Calculation 3: Heat gain from window)

24.1　概要

第 22 章で述べたように、コンクリートやタイルなどを材料とする建築外壁は、日射吸収率の値に応じて日射の一部を吸収し、残余を反射する。窓面はこのような一般外壁と比較して異なった特性を持つために注意が必要である。即ち、窓の材料であるガラスは吸収と反射に加え、一部を透過させる特性を持つため、直接的に室内に入り込む日射成分が発生する。また、吸収・反射・透過の比率は一定値ではなく、日射がガラスに入射する角度に大きく影響を受け、時々刻々変化する太陽の位置によって変動する。さらに、窓面には第 24 章で解説したように日射を遮蔽するための日除けが設けられることもあり、このような場合には日陰となる面積を計算して直達日射の大きさを推定することも必要となる。本章では熱負荷計算で必要となる透過日射熱取得、吸収日射熱取得、貫流熱取得を計算する方法を開設する。

図 24.1　最初期のガラス建築である水晶宮（ロンドン万博）[24.1)]

24.2　理論

図 24.2 に示すように窓の熱的性能は断熱性能と遮熱性能として捉えることができる。内外温度差により生じる熱流に関する性能が断熱性能、日射による熱取得に関する性能が遮熱性能である。

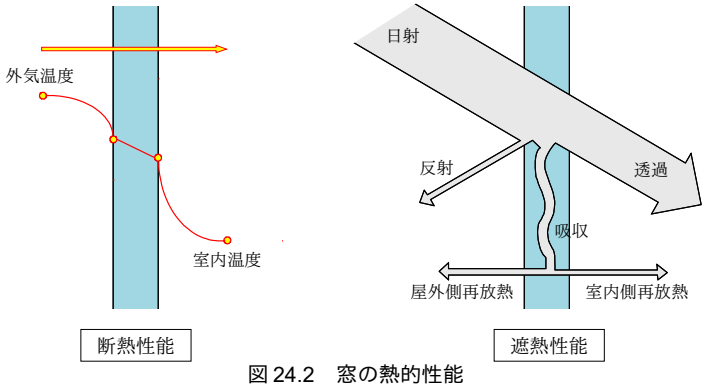


図 24.2　窓の熱的性能

24.2.1　断熱性能

内外温度差による熱流 Q_{WO} [W]は、式 24.1 に示すように一般外壁と同様の方法で計算する[†1)]。A_W [m²]は窓面積[†2)]、$T_{SolW,o}$ [K]は屋外側相当温度、$T_{SolW,i}$ [K]は屋内側相当温度、R_W [(m²·K)/W]は熱抵抗、K_W [W/(m²·K)]は熱貫流率である。屋外側相当温度 $T_{SolW,o}$ は外気温度 T_o [K]に放射の影響を加味した温度であり、式 24.2 で計算する。第 2 項は夜間放射 R_N [W/m²]の影響であり一般外壁と同じであるが、窓の場合には日射を吸収するだけではなく透過させるため、第 3 項がやや異なる。第 3 項の $Q_{W,\alpha}$ [W]は窓に吸収された日射の内、室内側へ放熱される熱であり、吸収日射熱取得と呼ばれる。この計算法に関しては後述する。ε_{LW} [-]はガラス表面の長波長放射率、F_S [-] はガラス面から天空への形態係数、$\alpha_{(c+r),o}$ [W/(m²·K)]は外表面総合熱伝達率である。屋内側相当温度 $T_{SolW,i}$ は壁の屋内側相当温度と同じであり、式 22.1 で計算する。

$$Q_{WO}=A_W\frac{T_{SolW,o}-T_{SolW,i}}{R_W}=K_W A_W\left(T_{SolW,o}-T_{SolW,i}\right) \tag{24.1}$$

$$T_{SolW,o}=T_o-\frac{\varepsilon_{LW}F_S R_N}{\alpha_{(c+r),o}}+\frac{Q_{W,\alpha}}{K_W A_W} \tag{24.2}$$

壁体の場合と同様に室温との連成計算に備えて室内側の窓表面温度 T_B [K]を室内外の相当温度の関数として表現すると式 24.3 となる。ただし係数 FF_B と BF_B は式 24.4 である。

$$T_B=FF_B\cdot T_{SolW}+BF_B\cdot T_{SolW,i} \tag{24.3}$$

$$FF_B=1-\frac{K_W}{\alpha_{(c+r),B}}\quad,\quad BF_B=\frac{K_W}{\alpha_{(c+r),B}} \tag{24.4}$$

窓の熱抵抗 R_W は窓ガラスメーカーのカタログに直接に記載されている場合が多いが、特別な仕様のガラスについて計算をしたい場合には一般の多層壁と同様に、内外表面の熱抵抗にガラスの抵抗を加算して式 24.5 で計算する。現在は、断熱性能と遮熱性能の向上を目的に、中空層を挟んでガラス

†1　本章で記載した計算法の多くは、参考文献 24.3 および 24.4 の内容を簡略化したものである。原典には、より詳細な計算法の記載があるため、必要に応じて参照されたい。

†2　通常の窓はガラスとサッシから構成されており、安価なアルミサッシはガラス面よりも熱的な弱点になることもある。両者を明確に分離して窓面をモデル化する方法もあるが、1 つの窓について 2 種類の面積が発生してわかりづらくなるため、本書ではガラス面のみのモデルとしている。サッシの断熱性能が低く、計算に反映する必要がある場合には第 22 章の一般外壁としてサッシ部分をモデルに追加すれば同じ効果が得られる。

を二重または三重に連ねた窓が商品化されており、この場合には中空層の熱抵抗も計算に含める必要がある。$\alpha_{(c+r)}$ [W/(m²·K)]は内外表面の総合熱伝達率であり、添字の F と B はそれぞれ表側と裏側を表す。$R_{G,n}$ [(m²·K)/W]は第 n 層のガラス自体の熱抵抗、$R_{A,n}$ [(m²·K)/W]は第 n 層と第 n+1 層のガラスに挟まれた中空層の熱抵抗である。ガラスの熱抵抗 R_G はガラス厚 d_G [m]をガラスの熱伝導率 λ_G （1.0 W/(m·K)とする）で除して式 24.6 で計算する。

$$R_W=\frac{1}{\alpha_{(c+r),B}}+\sum_{n=0}^{N}R_{G,n}+\sum_{n=0}^{N-1}R_{A,n}+\frac{1}{\alpha_{(c+r),F}} \tag{24.5}$$

$$R_G=d_G/\lambda_G \tag{24.6}$$

中空層の中には通常は乾き空気を封入するが、近年では熱伝導率の低いアルゴンガスを用いた複層ガラスも製品化されている。このような場合の中空層の熱抵抗は中空層を挟むガラスの内外表面温度、ガラスの角度、熱流の向き、ガラス表面の放射率などに影響を受ける。図 24.3 に中空層の内外ガラス表面温度差と気体熱コンダクタンス†1)の関係を計算した結果を示す†2)。左は乾き空気、右はアルゴンガスを封入した場合である。それぞれ水平・垂直・45 度の 3 種類の角度と、5mm と 10mm の 2 種類の中空層厚さについて計算した。幅 5mm の場合には対流効果は生じず、角度によらず一定値である。10mm で 45 度ないしは垂直の場合には、ガラス表面温度差によっては対流効果が生じる。

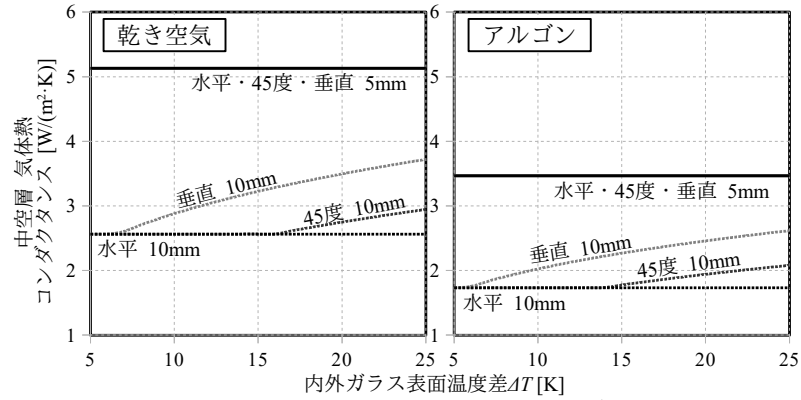


図 24.3　中空層の内外ガラス表面温度差と気体熱コンダクタンスの関係

気体熱コンダクタンスに放射熱コンダクタンスを加えた値が、中空層の熱コンダクタンスである。式 22.10 で、ガラスの放射率を 0.84、温度を 300 K とすれば、放射熱コンダクタンスは約 4.2 W/(m²·K)となる。また、金属膜を塗布して放射率を 0.2 程度に抑えた Low-E ガラスと呼ばれるガラスがあるが、この場合に式 22.10 を適用すると放射熱コンダクタンスは約 1.0 W/(m²·K)となる。従って、中空層の熱コンダクタンスは、空気+非 Low-E で 6.7 W/(m²·K)程度、アルゴンガス+非 Low-E で 6.0 W/(m²·K)程度、空気+Low-E で 3.5 W/(m²·K)程度、アルゴンガス+Low-E で 2.8 W/(m²·K)程度の値となろう。

建物全体の熱負荷計算の各タイムステップにおいて中空層の熱コンダクタンスを再計算することはかなり計算速度を下げ、煩雑でもあるため、本書のモデルでは一定値として計算を行う方針とする。

24.2.2　遮熱性能

1）透過率・吸収率・反射率

ガラス面に入射した日射は、一部がガラスを透過し、一部はガラスに吸収され、残余は反射され

†1　対流熱移動に加えて気体の伝導を考慮するため、このような名称としている。

†2　参考文献 24.3 による。

る。代表的なガラスの光学特性を表 24.1 に示す。

表 24.1　各種ガラスの垂直入射時の光学的性能[24.4)]

種類	透過率	ガラス面 反射率	薄膜面 反射率
透明板ガラス（3mm）	85.9 %	7.7 %	-
透明板ガラス（6mm）	80.6 %	7.3 %	-
網入りガラス（6.8mm）	77.7 %	7.6 %	-
透明合わせガラス（6mm、中間膜 0.76mm）	75.2 %	7.0 %	-
熱線吸収板ガラス（グリーン・5mm）	51.4 %	5.7 %	-
熱線吸収板ガラス（グリーン・6mm）	46.7 %	5.5 %	-
熱線反射ガラス（シルバー・6mm）	62.7 %	21.1 %	25.2 %
高遮蔽性能熱線反射ガラス（SGY32・6mm）	29.1 %	10.3 %	25.0 %
高遮蔽性能熱線反射ガラス（SS8・8mm）	6.3 %	36.2 %	47.7 %
Low-E ガラス（日射取得型・3mm）	70.3 %	10.6 %	11.7 %
Low-E ガラス（日射中庸型・3mm）	60.8 %	21.6 %	24.4 %
Low-E ガラス（日射遮蔽型・3mm）	39.6 %	35.5 %	42.7 %

ガラスメーカーのカタログにはガラス別にこのような光学特性が記載されているため、具体的なガラス種別が既に決定している場合にはその値を用いる方が良い。これらの値はガラス面に日射が垂直に入射したときの値であるが、実際には透過率・吸収率・反射率は入射角への依存性がある。この計算法に関しては既に第 18 章で解説した。各種ガラスの入射角特性近似係数を表 24.2 に示す。

表 24.2　各種ガラスの入射角特性近似係数[24.4)]

ガラス種別	-	m_1	m_2	m_3	m_4	m_5
透明フロートガラス	日射透過率	2.552	1.364	-11.388	13.617	-5.146
	日射反射率	5.189	-12.392	16.593	-11.851	3.461
熱線吸収ガラス	日射透過率	1.760	3.770	-14.901	16.422	-6.052
	日射反射率	5.189	-12.392	16.593	-11.851	3.461
熱線反射ガラス	日射透過率	3.297	-1.122	-8.408	12.206	-4.972
	日射反射率	5.842	-15.264	-21.642	-15.948	4.727
Low-E ガラス	日射透過率	2.273	1.631	-10.358	11.769	-4.316
	日射反射率（表）	5.084	-12.646	18.213	-13.967	4.316
	日射反射率（裏）	4.387	-9.175	11.152	-7.416	2.052

第 18 章では垂直入射時の透過率・吸収率・反射率をそれぞれ $\tau(0)$、$\alpha(0)$、$\rho(0)$と表現し、入射角 θ のときの値を $\tau(\theta)$、$\alpha(\theta)$、$\rho(\theta)$と表現したが、本章では、以降、入射角特性を考慮した後の値を単純に τ、α、ρ と表現することにする。

ガラスの透過率は波長によっても変化し、短波長が主たる成分である太陽光線は透過するが、我々の居住環境に存在する物体が発する長波長は殆ど透過しない。図 24.4 にガラスの波長別透過率と太陽光線の分光エネルギーを示す[24.5)]。従って、一旦、日射が室内の壁体などに吸収されてしまうと、壁体からの長波長放射は透過しないため、日射が室内に閉じ込められることになる。

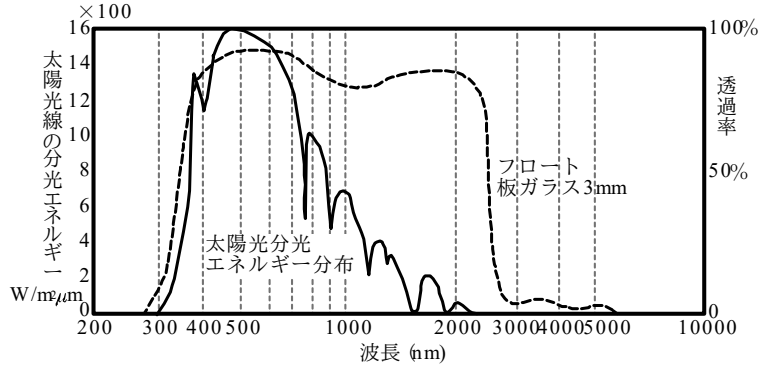


図 24.4　ガラスの波長別透過率と太陽光線の分光エネルギー[24.5)]

【例題 24.1】

黒体からの放射がピークとなる波長 λ_{max} [m]は黒体の絶対温度 T [K]に依存し、式 24.7 のウィーンの変位則に従う。太陽光と室内常温物体のピーク波長を計算し、ガラス波長別透過率との関係を考察せよ。ただし太陽の温度は約 6,000 K である。

$$\lambda_{max}=\frac{2.898\times10^{-3}}{T} \tag{24.7}$$

【解】

太陽と室内の物体を完全黒体とみなして計算を行う。室温を 26 ºC 前後とすれば、絶対温度表記で約 300 K である。300 K と 6000 K を式 24.7 に代入すると

$\lambda_{max,300} = 2.898 \times 10^{-3} / 300 = 9.659 \times 10^{-6}$ m = 9659 nm

$\lambda_{max,6000} = 2.898 \times 10^{-3} / 6000 = 4.83 \times 10^{-7}$ m = 483 nm

となる。図 24.4 によれば、太陽光の放射がピークとなる波長である 483 nm におけるガラスの透過率は 100 %に近く、室内物体のピークである 9659 nm における透過率はほぼ 0 %である。従って、ガラスは太陽の短波長放射を透過し、室内物体からの長波長放射を遮断する特性を持っていると言える。

2）総合透過率、総合反射率、総合吸収率

複層ガラスの多重反射を考慮した吸収・透過・反射は各層の吸収率 α・透過率 τ・反射率 ρ を用いて計算できる。複層ガラスの多重反射を図 24.5 に示すようにモデル化する。各層のガラスは正面と背面とで異なった特性を示しうるとし、それぞれの値を添字の F と B で表現する。また、正面側から順に層番号を 0, 1, 2…, n, n+1…, N-1, N と表現する。

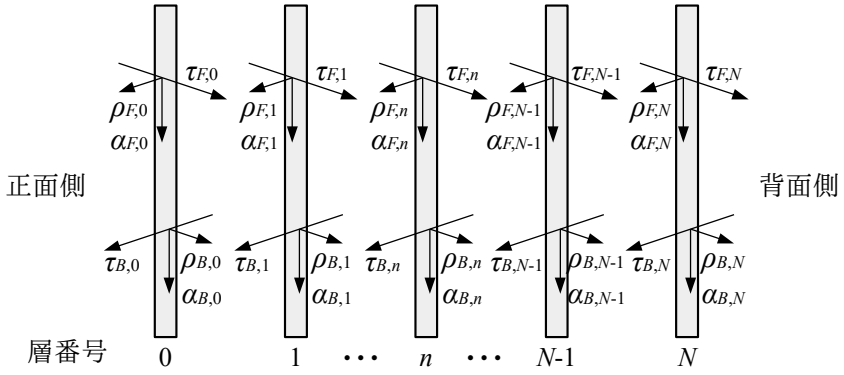


図 24.5　複層ガラスの多重反射モデル

第 i 層から第 j 層までの多重反射を考慮した透過率と反射率は式 24.8~24.12で計算できる。ただし、添字の F は正面側からの日射、添字の B は裏側からの日射に対する値である。また、$\tau_{F,(0\to0)}$、$\tau_{B,(0\to0)}$、$\tau_{F,(N\to N)}$、$\tau_{B,(N\to N)}$、$\rho_{F,(0\to0)}$、$\rho_{B,(0\to0)}$、$\rho_{F,(N\to N)}$、$\rho_{B,(N\to N)}$はそれぞれ $\tau_{F,0}$、$\tau_{B,0}$、$\tau_{F,N}$、$\tau_{B,N}$、$\rho_{F,0}$、$\rho_{B,0}$、$\rho_{F,N}$、$\rho_{B,N}$である。従って i=j=0 または i=j=N から始めて逐次計算を行うことで任意の 0→j または i→N の値を計算することができる。特に i=0、j=N の場合の τ と ρ を総合透過率 τ_T および総合反射率 ρ_T と呼ぶ。

$$\tau_{F,(i\to j)}=\tau_{F(i\to j-1)}\tau_{F,j}X_r \tag{24.8}$$

$$\tau_{B,(i\to j)}=\tau_{B(i\to j-1)}\tau_{B,j}X_r \tag{24.9}$$

$$\rho_{F,(i\to j)}=\rho_{F,(i\to j-1)}+\tau_{F,(i\to j-1)}\rho_{F,j}\tau_{F,(i\to j-1)}X_r \tag{24.10}$$

$$\rho_{B,(i\to j)}=\rho_{B,j}+\tau_{F,j}\rho_{F,(i\to j-1)}\tau_{F,j}X_r \tag{24.11}$$

$$X_r=\left(1-\rho_{B,(i\to j-1)}\rho_{F,j}\right)^{-1} \tag{24.12}$$

式 24.8~24.12の結果を式 24.13 と 24.14 に代入すれば、多重反射後に各層が吸収する日射の比率を計算できる。α_{TF} は正面からの日射が各層に吸収される比率、α_{TB} は裏側からの日射が各層に吸収される

比率である。

$$\alpha_{TF,j}=\frac{\tau_{F,(0\to j-1)}\alpha_{F,j}}{1-\rho_{B,(0\to j-1)}\rho_{F,(j\to N)}}+\frac{\tau_{F,(0\to j)}\rho_{F,(j+1\to N)}\alpha_{B,j}}{1-\rho_{B,(0\to j)}\rho_{F,(j+1\to N)}} \tag{24.13}$$

$$\alpha_{TB,j}=\frac{\tau_{B,(j\to N)}\rho_{B,(j-1\to N)}\alpha_{F,j}}{1-\rho_{B,(0\to j-1)}\rho_{F,(j\to N)}}+\frac{\tau_{B,(j+1\to N)}\alpha_{B,j}}{1-\rho_{B,(0\to j)}\rho_{F,(j+1\to N)}} \tag{24.14}$$

ある層に吸収された日射は、ガラスと空気層を伝わり、最終的に正面側および背面側に放熱される。この放熱の比率はj層から正面および背面までの熱抵抗の比率を用いて式 24.15 と式 24.16 で計算できる。$R_{\alpha F}$は正面側に放熱される比率、$R_{\alpha B}$は背面側に放熱される比率である。各層での総合吸収率$\alpha_{TB,j}$の内、室内側へ配分される部分を積算した値を吸収日射取得率 α_{TSum} と呼び、式 24.17 で計算する。直達日射と拡散日射とで透過率・反射率・吸収率は異なるため、総合透過率・総合反射率・総合吸収率・吸収日射取得率は、両方の日射に対して計算する必要がある。

$$R_{\alpha F,j}=1-R_{\alpha B,j}=\left(\frac{1}{\alpha_{(c+r),B}}+\sum_{n=j+1}^{N}R_{G,n}+\sum_{n=j}^{N-1}R_{A,n}+0.5R_{G,j}\right)/R_G \tag{24.15}$$

$$R_{\alpha B,j}=1-R_{\alpha F,j}=\left(\frac{1}{\alpha_{(c+r),F}}+\sum_{n=0}^{j-1}R_{G,n}+\sum_{n=0}^{j-1}R_{A,n}+0.5R_{G,j}\right)/R_G \tag{24.16}$$

$$\alpha_{TSumF}=\sum_{j=0}^{N}R_{\alpha B,j}\alpha_{TF,j}\quad,\quad\alpha_{TSumB}=\sum_{j=0}^{N}R_{\alpha B,j}\alpha_{TB,j} \tag{24.17}$$

現実の窓にはブラインドやスクリーンなどの日射遮蔽が設けられることが多い。拡散反射成分と拡散透過成分の大きいこれらの日射遮蔽物に式 24.8~24.14を適用することは正確ではないが、実用上はガラスと同様に日射の直進透過と正反射を前提として多重反射を計算する†1)。ブラインドに関しては第 23 章で示した方法によりスラット角と見かけの太陽高度に応じて光学特性を計算する。また、スクリーンに関しては表 24.3 の値を用いる。日射遮蔽が設けられた場合にも中空層の熱抵抗 R_A は変化しないものとみなし†2)、日射遮蔽の F 側と B 側の熱抵抗はそれぞれ 0.5 R_A とする。

表 24.3　スクリーンの光学特性[24.4)]

種別	素材	色	透過率	反射率
一般	ポリエステル繊維	ホワイト	0.30	0.63
		グレイ	0.23	0.52
		アイボリー	0.26	0.59
		ベージュ	0.24	0.56
		セピア	0.18	0.37
遮光	ポリエステル繊維 屋外側アクリルコーティング	室内側ホワイト 室外側ホワイト	0.00	0.64
高反射	ガラス繊維 PVC コーティング 屋外側アルミ蒸着	室内側ホワイト 室外側アルミ色	0.06	0.77

3）窓面からの日射熱取得

日射による窓面からの熱取得は、窓を透過して直接に室内に入射する透過日射熱取得 $Q_{W\tau}$ [W]とガラスやブラインド等に吸収されてから室内に流入する吸収日射熱取得 $Q_{W\alpha}$ [W]に分けられる。透過日射熱取得は総合透過率 τ_T を用いて式 24.18 で計算する。I_D [W/m^2]と I_{SR} [W/m^2]は傾斜面へ入射する直達日射と拡散日射であり、計算法は第 6 章で解説した。$\tau_{TF,Dir}$ と $\tau_{TF,Dif}$ はそれぞれ屋外側からの直達日射

†1　木下らは板ガラスとブラインドを組み合わせた窓の光学特性について、拡散反射と透過を考慮した多重反射計算方法を提案し、本書のような簡易モデルとの誤差について報告している[24.6)]。外側ブラインドの場合などには比較的誤差が大きく、誤差率は 5 %を超える場合があるとしている。

†2　現実には、対流成分は殆ど変化ないが、放射成分は日射遮蔽物の影響を受けると予想できる。しかしこれを厳密に扱おうとすると日射遮蔽物を介して見えがかりになっている次層のガラス表面積比の形態係数などを知る必要があり、計算が極めて煩雑になる。建物全体の熱負荷計算という目的においてはこのような仮定を置いても問題ないと予想する。

と拡散日射に対する総合透過率である。

$$Q_{W\tau}=A_W\left(I_D\tau_{TF,Dir}+I_{SR}\tau_{TF,Dif}\right) \tag{24.18}$$

吸収日射熱取得 $Q_{W\alpha}$ は、式 24.19~24.21に示すように屋外からの日射に対する吸収日射熱取得 $Q_{W\alpha F}$ と屋内からの日射 I_{BSW} [W/m²]に対する吸収日射熱取得 $Q_{W\alpha B}$ の和である。屋内からの日射とは、屋外から入射した日射が室内で反射を繰り返しながら窓面に入射する日射の内、窓面で反射しない部分の積算値である。計算法は第 25 章で示す。室内で反射を繰り返すうちに殆どが拡散光になると推測できるため、本書のモデルでは、屋内からの日射に対する吸収日射熱取得 $Q_{W\alpha B}$ に関しては直達日射を考慮しない。I_{BSW} は反射成分を控除した値であるため、吸収日射熱取得率 $\alpha_{TSumB,dif}$ を(1-$\rho_{TB,dif}$)で除している。

$$Q_{W\alpha}=Q_{W\alpha F}+Q_{W\alpha B} \tag{24.19}$$

$$Q_{W\alpha F}=A_W\left(I_D\alpha_{TSumF,Dir}+I_{SR}\alpha_{TSumF,Dif}\right) \tag{24.20}$$

$$Q_{W\alpha B}=A_W I_{BSW}\frac{\alpha_{TSumB,dif}}{1-\rho_{TB,dif}} \tag{24.21}$$

透過日射熱取得 $Q_{W\tau}$ に関しては窓以外の室内の壁などにも吸収されるが、吸収日射熱取得 $Q_{W\alpha}$ は窓面に吸収されて窓表面から室内に放出される。そこで、吸収日射熱取得 $Q_{W\alpha}$ に関しては式 24.2 に示した窓単体の熱収支式の中で表現して計算を簡易化する。

24.3　計算法

窓を表わす Window クラスを作成する。

プログラム 24.1 に列挙型定義を示す。

プログラム 24.1　列挙型定義

Popolo.ThermalLoad.Window class

```
 1 /// <summary>ガラス種別</summary>
 2 public enum GlassTypes
 3 {
 4   /// <summary>透明フロート</summary>
 5   Transparent,
 6   /// <summary>熱線吸収</summary>
 7   HeatAbsorbing,
 8   /// <summary>熱線反射</summary>
 9   HeatReflecting,
10   /// <summary>Low-E</summary>
11   LowEmissivity
12 }
```

プログラム 24.2 にインスタンス変数およびプロパティを示す。2 行は入射角特性の係数であり、表側と裏側、透過率と反射率があるため、各ガラス層につき 4 種類ずつある。4~13 行は各層の熱抵抗と光学特性である。吸収日射熱取得はガラス表面の総合熱伝達率によって変化するため、熱伝達率変更時の再計算負荷を減らすため、各層の総合吸収率を 16 行のリストに保存する。また、19 行の変数に太陽位置を保存するすることで、太陽位置が変化しない場合には更新処理をスキップする。40~65 行は総合透過率・総合吸収率・吸収日射取得率である。本書のモデルでは室内側で反射した日射はすべて拡散日射とみなすため、屋内側からの直達日射に対する値は定義しない。76~110 行は F 側境界条件などの設定である。省略したが B 側にも同じプロパティを定義する。112~116 行のプロパティは多数室連成計算用であり、第 25 章で解説する。

プログラム **24.2**　インスタンス変数およびプロパティ

Popolo.ThermalLoad.Window class

```
/// <summary>入射角特性近似係数</summary>
private double[][] tau_CF, tau_CB, rho_CF, rho_CB;

/// <summary>中空層熱抵抗[(m2K)/W]</summary>
private double[] agapRes;

/// <summary>ガラス熱抵抗[(m2K)/W]</summary>
private double[] glassRes;

/// <summary>各層の光学特性
/// 0:透過,1:反射,2:吸収
/// F:正面,B:裏,Dir:直達,Dif:拡散</summary>
private double[,] opFDir, opBDir, opFDif, opBDif;

/// <summary>各層の吸収率リスト</summary>
private double[] absFDir, absFDif, absBDif;

/// <summary>各層のガラスの垂直入射時の透過率と反射率</summary>
private double[,] taurhoF, taurhoB;

/// <summary>従前の太陽高度と太陽方位</summary>
private double lstAlt, lstOri;

/// <summary>中空層日射遮蔽物リスト</summary>
private IShadingDevice[] sDevices;

/// <summary>窓面積[m2]</summary>
private double area = 1;

/// <summary>窓面積[m2]を設定・取得する</summary>
public double Area
{
  set { if (0 < value) { area = value; } }
  get { return area; }
}

/// <summary>外表面側傾斜面を取得する</summary>
public ImmutableIncline OutsideIncline { get; private set; }

/// <summary>屋外からの直達日射の総合透過率[-]を取得する</summary>
public double DirectSolarIncidentTransmittance { get; private set; }

/// <summary>屋外からの直達日射の総合反射率[-]を取得する</summary>
public double DirectSolarIncidentReflectance { get; private set; }

/// <summary>屋外からの直達日射の吸収日射取得率[-]を取得する</summary>
public double DirectSolarIncidentAbsorptance { get; private set; }

/// <summary>屋外からの拡散日射の総合透過率[-]を取得する</summary>
public double DiffuseSolarIncidentTransmittance { get; private set; }

/// <summary>屋外からの拡散日射の総合反射率[-]を取得する</summary>
public double DiffuseSolarIncidentReflectance { get; private set; }

/// <summary>屋外からの拡散日射の吸収日射取得率[-]を取得する</summary>
public double DiffuseSolarIncidentAbsorptance { get; private set; }

/// <summary>屋内からの拡散日射の総合透過率[-]を取得する</summary>
public double DiffuseSolarLostTransmittance { get; private set; }

/// <summary>屋内からの拡散日射の総合反射率[-]を取得する</summary>
public double DiffuseSolarLostReflectance { get; private set; }

/// <summary>屋内からの拡散日射の吸収日射取得率[-]を取得する</summary>
public double DiffuseSolarLostAbsorptance { get; private set; }

/// <summary>ガラスの層数を取得する</summary>
public int GlazingNumber { get; private set; }

/// <summary>日除けを設定・取得する</summary>
public SunShade SunShade { get; set; }

/// <summary>F 側と B 側の対流・放射熱伝達率[W/(m2K)]</summary>
private double cCoefF, rCoefF, cCoefB, rCoefB;

/// <summary>F 側対流熱伝達率[W/(m2K)]を設定・取得する</summary>
public double ConvectiveCoefficientF
{
```

```
 79   get { return cCoefF; }
 80   set
 81   {
 82     cCoefF = value;
 83     updateFilmCoefficent();
 84   }
 85 }
 86
 87 /// <summary>F 側放射熱伝達率[W/(m2K)]を設定・取得する</summary>
 88 public double RadiativeCoefficientF
 89 {
 90   get { return rCoefF; }
 91   set
 92   {
 93     rCoefF = value;
 94     updateFilmCoefficent();
 95   }
 96 }
 97
 98 /// <summary>F 側総合熱伝達率[W/(m2K)]を取得する</summary>
 99 public double FilmCoefficientF
100 { get { return 1d / (2 * agapRes[0]); } }
101
102 /// <summary>F 側の短波長放射率[-]を取得する</summary>
103 public double ShortWaveEmissivityF
104 { get { throw new Exception("Not Implemented"); } }
105
106 /// <summary>F 側の長波長放射率[-]を設定・取得する</summary>
107 public double LongWaveEmissivityF { get; set; } = 0.9;
108
109 /// <summary>F 側相当温度を設定・取得する</summary>
110 public double SolAirTemperatureF { get; set; }
111
112 /// <summary>内側表面を取得する</summary>
113 internal wallWindowSurface InsideSurface { get; private set; }
114
115 /// <summary>外側表面を取得する</summary>
116 internal wallWindowSurface OutsideSurface { get; private set; }
```

プログラム 24.3 にコンストラクタを示す。各層のガラスの透過率と反射率のリストならびに屋外側外表面の傾斜面情報を入力とする。1~68 行は表側と裏側で透過率と反射率を変える場合、70~77 行は同じ透過率と反射率にする場合のコンストラクタである。11~31 行でプロパティを初期化し、配列の記憶領域を確保する。それぞれの中空層にブラインドなどの日射遮蔽物を設置しえるモデルとするため、配列数は $2N$+1 確保する。特に空気層の熱抵抗に関しては両端部があるため $2N$+2 確保する必要があることに注意する。ただし標準では日射遮蔽物は無く、33~42 行に示すように空の日射遮蔽物（常に透過率 100 %）を設置する。53~60 行は熱抵抗と入射角特性の初期化処理であり、透明フロート+空気層を想定した値とする。

プログラム 24.3　コンストラクタ

Popolo.ThermalLoad.Window class

```
  1 /// <summary>N 層の多層ガラスの新しいインスタンスを初期化する</summary>
  2 /// <param name="area">面積[m2]</param>
  3 /// <param name="transmittanceF">正面側透過率リスト[-] (0=F, N-1=B) </param>
  4 /// <param name="reflectanceF">正面側反射率リスト[-] (0=F, N-1=B) </param>
  5 /// <param name="transmittanceB">裏側透過率リスト[-] (0=F, N-1=B) </param>
  6 /// <param name="reflectanceB">裏側反射率リスト[-] (0=F, N-1=B) </param>
  7 /// <param name="outsideIncline">外側傾斜面</param>
  8 public Window(double area, double[] transmittanceF, double[] reflectanceF,
  9   double[] transmittanceB, double[] reflectanceB, ImmutableIncline outsideIncline)
 10 {
 11   Area = area;
 12   GlazingNumber = transmittanceF.Length;
 13   this.SunShade = SunShade.MakeEmptySunShade();
 14   this.OutsideIncline = outsideIncline;
 15   lstAlt = lstOri = -999;
 16
 17   tau_CF = new double[GlazingNumber][];
 18   tau_CB = new double[GlazingNumber][];
 19   rho_CF = new double[GlazingNumber][];
 20   rho_CB = new double[GlazingNumber][];
```

```
21   taurhoF = new double[GlazingNumber, 2];
22   taurhoB = new double[GlazingNumber, 2];
23   opFDir = new double[GlazingNumber * 2 + 1, 3];
24   opBDir = new double[GlazingNumber * 2 + 1, 3];
25   opFDif = new double[GlazingNumber * 2 + 1, 3];
26   opBDif = new double[GlazingNumber * 2 + 1, 3];
27   absFDir = new double[GlazingNumber * 2 + 1];
28   absFDif = new double[GlazingNumber * 2 + 1];
29   absBDif = new double[GlazingNumber * 2 + 1];
30   agapRes = new double[GlazingNumber * 2 + 2];
31   glassRes = new double[GlazingNumber];
32
33   //空の日射遮蔽で初期化
34   sDevices = new IShadingDevice[GlazingNumber + 1];
35   for (int i = 0; i < sDevices.Length; i++)
36   {
37     sDevices[i] = new EmptyShadingDevice();
38     int i2 = i * 2;
39     opFDir[i2, 0] = opBDir[i2, 0] = opFDif[i2, 0] = opBDif[i2, 0] = 1.0;
40     opFDir[i2, 1] = opBDir[i2, 1] = opFDif[i2, 1] = opBDif[i2, 1] =
41       opFDir[i2, 2] = opBDir[i2, 2] = opFDif[i2, 2] = opBDif[i2, 2] = 0.0;
42   }
43
44   //垂直入射時の透過率と反射率を保存
45   for (int i = 0; i < GlazingNumber; i++)
46   {
47     taurhoF[i, 0] = transmittanceF[i];
48     taurhoB[i, 0] = transmittanceB[i];
49     taurhoF[i, 1] = reflectanceF[i];
50     taurhoB[i, 1] = reflectanceB[i];
51   }
52
53   //熱抵抗を初期化
54   RadiativeCoefficientF = RadiativeCoefficientB = 4.5;
55   ConvectiveCoefficientF = 18.5;
56   ConvectiveCoefficientB = 7.5;
57   updateFilmCoefficent();
58   agapRes[agapRes.Length - 1] = agapRes[agapRes.Length - 2] = 0.5 * 1 / 6.7;
59   for (int i = 2; i < agapRes.Length - 2; i++) agapRes[i] = 0.5 * 1 / 6.7;
60   for (int i = 0; i < glassRes.Length; i++) glassRes[i] = 0.006;
61
62   //内外表面作成
63   OutsideSurface = new wallWindowSurface(this, true);
64   InsideSurface = new wallWindowSurface(this, false);
65
66   //入射角特性を透明フロートガラスで初期化
67   for (int i = 0; i < GlazingNumber; i++) SetAngleDependence(i, GlassTypes.Transparent);
68 }
69
70 /// <summary>N層の多層ガラスの新しいインスタンスを初期化する</summary>
71 /// <param name="area">面積[m2]</param>
72 /// <param name="transmittance">透過率リスト[-] (0=F, N-1=B) </param>
73 /// <param name="reflectance">反射率リスト[-] (0=F, N-1=B) </param>
74 /// <param name="outsideIncline">外側傾斜面</param>
75 public Window(double area, double[] transmittance, double[] reflectance, ImmutableIncline outsideIncline) :
76   this(area, transmittance, reflectance, transmittance, reflectance, outsideIncline)
77 { }
```

プログラム 24.4 にモデル作成関連の処理を示す。6~31 行は、ガラス、中空層、表面熱伝達率の変更処理である。これらの値が変更された場合には、吸収日射の室内側按分比率が変化するため、33~43 行のメソッドで吸収日射熱取得を更新する必要がある。F 側からの直達日射、F 側からの拡散日射、B 側からの拡散日射をそれぞれを更新する。45~67 行が按分処理を行うメソッドであり、式 23.15~式 23.17 の実装である。69~77 行は窓全体の熱抵抗計算処理であり、式 23.5 の実装である。

プログラム 24.4　モデル作成関連の処理

Popolo.ThermalLoad.Window class

```
1 /// <summary>日射遮蔽物を設定する</summary>
2 /// <param name="number">設定する層番号 (屋外側:0,屋内側:N+1) </param>
3 /// <param name="sDevice">遮蔽物</param>
4 public void SetShadingDevice(int number, IShadingDevice sDevice) { sDevices[number] = sDevice; }
5
6 /// <summary>ガラスの熱抵抗[m2K/W]を設定する</summary>
7 /// <param name="number">層番号</param>
8 /// <param name="resistance">ガラスの熱抵抗[m2K/W]</param>
9 public void SetGlassResistance(int number, double resistance)
```

```
10 {
11   glassRes[number] = resistance;
12   updateAbsorptance();
13 }
14
15 /// <summary>中空層の熱抵抗[m2K/W]を設定する</summary>
16 /// <param name="number">層番号（層の右側）</param>
17 /// <param name="resistance">中空層の熱抵抗[m2K/W]</param>
18 public void SetAirGapResistance(int number, double resistance)
19 {
20   agapRes[2 * number + 2] = agapRes[2 * number + 3] = 0.5 * resistance;
21   updateAbsorptance();
22 }
23
24 /// <summary>総合熱伝達率[W/(m2K)]を更新する</summary>
25 private void updateFilmCoefficent()
26 {
27   agapRes[0] = agapRes[1] = 0.5 / (ConvectiveCoefficientF + RadiativeCoefficientF);
28   agapRes[agapRes.Length - 1] =
29     agapRes[agapRes.Length - 2] = 0.5 / (ConvectiveCoefficientB + RadiativeCoefficientB);
30   updateAbsorptance();
31 }
32
33 /// <summary>吸収日射取得率を更新する</summary>
34 private void updateAbsorptance()
35 {
36   double adF, adB;
37   integrateAbsorption(absFDir, agapRes, glassRes, out adF, out adB);
38   DirectSolarIncidentAbsorptance = adB;
39   integrateAbsorption(absFDif, agapRes, glassRes, out adF, out adB);
40   DiffuseSolarIncidentAbsorptance = adB;
41   integrateAbsorption(absBDif, agapRes, glassRes, out adF, out adB);
42   DiffuseSolarLostAbsorptance = adB;
43 }
44
45 /// <summary>総合吸収率を F 側と B 側に按分する</summary>
46 /// <param name="ttlA">総合吸収率リスト</param>
47 /// <param name="agapResist">中空層の熱抵抗リスト</param>
48 /// <param name="glassResistance">ガラスの熱抵抗リスト</param>
49 /// <param name="ttlAF">F 側按分量</param>
50 /// <param name="ttlAB">B 側按分量</param>
51 private static void integrateAbsorption
52   (double[] ttlA, double[] agapResist, double[] glassResistance, out double ttlAF, out double ttlAB)
53 {
54   double rSum1, rSum2 = 0;
55   rSum1 = rSum2 = ttlAF = ttlAB = 0;
56   for (int i = 0; i < agapResist.Length; i++) rSum1 += agapResist[i];
57   for (int i = 0; i < glassResistance.Length; i++) rSum1 += glassResistance[i];
58   for (int i = 0; i < ttlA.Length; i++)
59   {
60     rSum2 += agapResist[i];
61     if (i % 2 == 1) rSum2 += 0.5 * glassResistance[(i - 1) / 2];
62     else if (i != 0) rSum2 += 0.5 * glassResistance[i / 2 - 1];
63     double fRate = rSum2 / rSum1;
64     ttlAB += ttlA[i] * fRate;
65     ttlAF += ttlA[i] * (1 - fRate);
66   }
67 }
68
69 /// <summary>窓全体の熱抵抗[m2K/W]を取得する</summary>
70 /// <returns>窓全体の熱抵抗[m2K/W]</returns>
71 public double GetResistance()
72 {
73   double rg = 0;
74   for (int i = 0; i < glassRes.Length; i++) rg += glassRes[i];
75   for (int i = 0; i < agapRes.Length; i++) rg += agapRes[i];
76   return rg;
77 }
```

プログラム 24.5 に入射角特性の設定処理を示す。拡散日射に対する各層の透過率と反射率は見かけの太陽高度に影響を受けないため、この時点で計算して保存しておく（16~39 行）。42 行の総合特性更新処理については後述する。計算の度に入射角特性を求めることは手間なので、プログラム 24.1 で列挙型として定義した代表的なガラスに関しては、45~82 行に示すように標準的な入射角特性係数で初期化するメソッドを用意する。

プログラム **24.5**　入射角特性の設定処理

Popolo.ThermalLoad.Window class

```
/// <summary>入射角特性を設定する</summary>
/// <param name="layerNumber">層番号</param>
/// <param name="coefTF">F 側透過特性の近似係数</param>
/// <param name="coefTB">B 側透過特性の近似係数</param>
/// <param name="coefRF">F 側反射特性の近似係数</param>
/// <param name="coefRB">B 側反射特性の近似係数</param>
public void SetAngleDependence
  (int layerNumber, double[] coefTF, double[] coefTB, double[] coefRF, double[] coefRB)
{
  int ln = layerNumber;
  tau_CF[ln] = coefTF;
  tau_CB[ln] = coefTB;
  rho_CF[ln] = coefRF;
  rho_CB[ln] = coefRB;

  //拡散日射の規準化透過率と規準化反射率の計算
  double difCTF = 0;
  double difCTB = 0;
  double difCRF = 0;
  double difCRB = 0;
  for (int j = 0; j < tau_CF[ln].Length; j++)
  {
    difCTF += tau_CF[ln][j] / (j + 3);
    difCTB += tau_CB[ln][j] / (j + 3);
    difCRF += rho_CF[ln][j] / (j + 3);
    difCRB += rho_CB[ln][j] / (j + 3);
  }
  difCTF *= 2;
  difCTB *= 2;
  difCRF *= 2;
  difCRB *= 2;

  //拡散日射の透過率と反射率を計算
  opFDif[2 * ln + 1, 0] = difCTF * taurhoF[ln, 0];
  opFDif[2 * ln + 1, 1] = difCRF * taurhoF[ln, 1];
  opFDif[2 * ln + 1, 2] = 1 - (opFDif[2 * ln + 1, 0] + opFDif[2 * ln + 1, 1]);
  opBDif[2 * ln + 1, 0] = difCTB * taurhoB[ln, 0];
  opBDif[2 * ln + 1, 1] = difCRB * taurhoB[ln, 1];
  opBDif[2 * ln + 1, 2] = 1 - (opBDif[2 * ln + 1, 0] + opBDif[2 * ln + 1, 1]);

  //拡散日射に関する総合特性を更新
  updateDiffuseTotalProperties();
}

/// <summary>入射角特性を設定する</summary>
/// <param name="layerNumber">層番号</param>
/// <param name="type">ガラス種類</param>
public void SetAngleDependence(int layerNumber, GlassTypes type)
{
  int ln = layerNumber;
  switch (type)
  {
    case GlassTypes.HeatAbsorbing:
      SetAngleDependence(layerNumber,
        new double[] { 1.760, 3.770, -14.901, 16.422, -6.052 },
        new double[] { 1.760, 3.770, -14.901, 16.422, -6.052 },
        new double[] { 5.189, -12.392, 16.593, -11.851, 3.461 },
        new double[] { 5.189, -12.392, 16.593, -11.851, 3.461 });
      return;
    case GlassTypes.HeatReflecting:
      SetAngleDependence(layerNumber,
        new double[] { 3.297, -1.122, -8.408, 12.206, -4.972 },
        new double[] { 3.297, -1.122, -8.408, 12.206, -4.972 },
        new double[] { 5.842, -15.264, -21.642, -15.948, 4.727 },
        new double[] { 5.842, -15.264, -21.642, -15.948, 4.727 });
      return;
    case GlassTypes.LowEmissivity:
      SetAngleDependence(layerNumber,
        new double[] { 2.273, 1.631, -10.358, 11.769, -4.316 },
        new double[] { 2.273, 1.631, -10.358, 11.769, -4.316 },
        new double[] { 5.084, -12.646, 18.213, -13.967, 4.316 },
        new double[] { 4.387, -9.175, 11.152, -7.416, 2.052 });
      return;
    default:
      SetAngleDependence(layerNumber,
        new double[] { 2.552, 1.364, -11.388, 13.617, -5.146 },
        new double[] { 2.552, 1.364, -11.388, 13.617, -5.146 },
        new double[] { 5.189, -12.392, 16.593, -11.851, 3.461 },
```

```
79          new double[] { 5.189, -12.392, 16.593, -11.851, 3.461 });
80       return;
81   }
82 }
```

プログラム 24.6 に総合特性の計算処理を示す。F 側と B 側のそれぞれからの日射に対する各層の光学特性を引数にとり、総合透過率・総合反射率・総合吸収率を計算する。引数の opPropF と opPropB は *N* 行 3 列の 2 次元配列であり、行方向はガラスの層、列方向は第 1 列が透過率、第 2 列が反射率、第 3 列が吸収率を表わす。式 23.8~式 23.12 を用いて、18~29 行では *i*=*j*=*N* から 0 に向かって逐次計算を、31~52 行では *i*=*j*=0 から *N* に向かって逐次計算を行う。48~51 行では式 23.13 と式 23.14 を適用して総合吸収率を計算する。55~69 行は拡散日射に関して本メソッドを用いて総合特性を更新する処理である。

プログラム 24.6　総合特性の計算処理

Popolo.ThermalLoad.Window class

```
 1 /// <summary>総合特性を計算する</summary>
 2 /// <param name="opPropF">F 側入射に対する光学特性</param>
 3 /// <param name="opPropB">B 側入射に対する光学特性</param>
 4 /// <param name="ttlTF">出力:F 側入射に対する総合透過率</param>
 5 /// <param name="ttlRF">出力:F 側入射に対する総合反射率</param>
 6 /// <param name="ttlAF">出力:F 側入射に対する総合吸収率</param>
 7 /// <param name="ttlTB">出力:B 側入射に対する総合透過率</param>
 8 /// <param name="ttlRB">出力:B 側入射に対する総合反射率</param>
 9 /// <param name="ttlAB">出力:B 側入射に対する総合吸収率</param>
10 private static void computeTotalOProperties
11   (double[,] opPropF, double[,] opPropB, out double ttlTF, out double ttlRF, ref double[] ttlAF,
12   out double ttlTB, out double ttlRB, ref double[] ttlAB)
13 {
14   int ln = opPropF.GetLength(0);
15   double[] ttlTBi = new double[ln];
16   double[] ttlRFi = new double[ln];
17
18   ttlTF = opPropF[ln - 1, 0];
19   ttlRFi[ln - 1] = opPropF[ln - 1, 1];
20   ttlTBi[ln - 1] = opPropB[ln - 1, 0];
21   ttlRB = opPropB[ln - 1, 1];
22   for (int i = ln - 2; 0 <= i; i--)
23   {
24     double xr = 1 / (1 - ttlRB * opPropF[i, 1]);
25     ttlRFi[i] = ttlRFi[i + 1] + ttlTF * xr * opPropF[i, 1] * ttlTBi[i + 1];
26     ttlRB = opPropB[i, 1] + opPropB[i, 0] * xr * ttlRB * opPropF[i, 0];
27     ttlTF = ttlTF * xr * opPropF[i, 0];
28     ttlTBi[i] = opPropB[i, 0] * xr * ttlTBi[i + 1];
29   }
30
31   ttlTF = opPropF[0, 0];
32   ttlRF = opPropF[0, 1];
33   ttlTB = opPropB[0, 0];
34   ttlRB = opPropB[0, 1];
35   ttlAF[0] = opPropF[0, 2];
36   ttlAB[0] = 0;
37   for (int j = 1; j < ln; j++)
38   {
39     double tfm1 = ttlTF;
40     double tbm1 = ttlTB;
41     double rbm1 = ttlRB;
42     double xr = 1 / (1 - ttlRB * opPropF[j, 1]);
43     ttlRF = ttlRF + ttlTF * xr * opPropF[j, 1] * ttlTB;
44     ttlRB = opPropB[j, 1] + opPropB[j, 0] * xr * ttlRB * opPropF[j, 0];
45     ttlTF = ttlTF * xr * opPropF[j, 0];
46     ttlTB = opPropB[j, 0] * xr * ttlTB;
47     double bf = 1 / (1 - rbm1 * ttlRFi[j]);
48     ttlAF[j] = tfm1 * opPropF[j, 2] * bf;
49     ttlAF[j - 1] += tfm1 * ttlRFi[j] * opPropB[j - 1, 2] * bf;
50     ttlAB[j - 1] += ttlTBi[j] * opPropB[j - 1, 2] * bf;
51     ttlAB[j] = ttlTBi[j] * rbm1 * opPropF[j, 2] * bf;
52   }
53 }
54
55 /// <summary>拡散日射に関する総合特性を更新する</summary>
56 private void updateDiffuseTotalProperties()
57 {
```

```
58   double adF, adB, ttlTF, ttlRF, ttlTB, ttlRB;
59   computeTotalOProperties
60     (opFDif, opBDif, out ttlTF, out ttlRF, ref absFDif, out ttlTB, out ttlRB, ref absBDif);
61   DiffuseSolarIncidentTransmittance = ttlTF;
62   DiffuseSolarIncidentReflectance = ttlRF;
63   DiffuseSolarLostTransmittance = ttlTF;
64   DiffuseSolarLostReflectance = ttlRF;
65    integrateAbsorption(absFDif, agapRes, glassRes, out adF, out adB);
66   DiffuseSolarIncidentAbsorptance = adB;
67    integrateAbsorption(absBDif, agapRes, glassRes, out adF, out adB);
68   DiffuseSolarLostAbsorptance = adB;
69 }
```

プログラム 24.7 に光学特性の更新処理を示す。太陽のインスタンスを引数に取る。7~13 行は日の出前または日没後の場合の処理であり、この時は反射率を 100 %とする。太陽位置に変化があった場合（14 行で判定）には、21~42 行で直達日射の入射角特性を反映する。また、それぞれの日射遮蔽物に見かけの太陽高度を設定する（46 行）。49~66 行で日射遮蔽物の光学特性更新処理を行い、特性の変化があったか否かを sPropChanged に保存する。見かけの太陽高度と日射遮蔽物の特性がいずれも変化しない場合には、窓全体としても光学特性に変化は無いため、処理を終了する。ブラインドのスラット角変化のように、太陽位置に変化がなくても日射遮蔽物の光学特性に変化が生じる場合があることに注意する。見かけの太陽高度が変化した場合には 74~90 行で直達日射に対する光学特性を更新する。拡散日射に対するガラスの光学特性は見かけの太陽高度に依存しないため、日射遮蔽物の光学特性が変化した場合のみ更新処理を行う（93 行）。

プログラム 24.7　光学特性更新処理

Popolo.ThermalLoad.Window class

```
 1 /// <summary>光学特性を更新する</summary>
 2 /// <param name="sun">太陽</param>
 3 public void UpdateOpticalProperties(ImmutableSun sun)
 4 {
 5   //ガラス・日射遮蔽物単体の光学特性の更新処理//////////////////////////////
 6   double cos = OutsideIncline.GetDirectSolarRadiationRate(sun);
 7   if (sun.Altitude <= 0)
 8   {
 9     DirectSolarIncidentTransmittance = 0;
10     DirectSolarIncidentReflectance = 1;
11     DirectSolarIncidentAbsorptance = 0;
12     return;
13   }
14   bool sunMoved = (sun.Altitude != lstAlt || sun.Orientation != lstOri);
15   if (sunMoved)
16   {
17     lstAlt = sun.Altitude;
18     lstOri = sun.Orientation;
19
20     //ガラスの直達日射入射角特性を反映
21     if (0 < cos)
22     {
23       for (int i = 0; i < GlazingNumber; i++)
24       {
25         double tauF, tauB, rhoF, rhoB;
26         tauF = tauB = rhoF = rhoB = 0;
27         int lenth = tau_CF[i].Length - 1;
28         for (int j = lenth; 0 <= j; j--)
29         {
30           tauF = cos * (tauF + tau_CF[i][j]);
31           tauB = cos * (tauB + tau_CB[i][j]);
32           rhoF = cos * (rhoF + rho_CF[i][j]);
33           rhoB = cos * (rhoB + rho_CB[i][j]);
34         }
35         opFDir[2 * i + 1, 0] = tauF * taurhoF[i, 0];
36         opFDir[2 * i + 1, 1] = 1 - (1 - taurhoF[i, 1]) * rhoF;
37         opFDir[2 * i + 1, 2] = 1 - (opFDir[2 * i + 1, 0] + opFDir[2 * i + 1, 1]);
38         opBDir[2 * i + 1, 0] = tauB * taurhoB[i, 0];
39         opBDir[2 * i + 1, 1] = 1 - (1 - taurhoB[i, 1]) * rhoB;
40         opBDir[2 * i + 1, 2] = 1 - (opBDir[2 * i + 1, 0] + opBDir[2 * i + 1, 1]);
41       }
42     }
```

```
43
44    //日射遮蔽物のプロファイル角を更新
45    double pAngle = OutsideIncline.GetProfileAngle(sun);
46    for (int i = 0; i < sDevices.Length; i++) sDevices[i].ProfileAngle = pAngle;
47  }
48
49  //日射遮蔽物の光学特性を更新
50  bool sPropChanged = false;
51  for (int i = 0; i < sDevices.Length; i++)
52  {
53    int i2 = i * 2;
54    if (sDevices[i].HasPropertyChanged)
55    {
56      sDevices[i].ComputeOpticalProperties(false, true, out opFDir[i2, 0], out opFDir[i2, 1]);
57      sDevices[i].ComputeOpticalProperties(false, false, out opBDir[i2, 0], out opBDir[i2, 1]);
58      sDevices[i].ComputeOpticalProperties(true, true, out opFDif[i2, 0], out opFDif[i2, 1]);
59      sDevices[i].ComputeOpticalProperties(true, false, out opBDif[i2, 0], out opBDif[i2, 1]);
60      opFDir[i2, 2] = 1 - (opFDir[i2, 0] + opFDir[i2, 1]);
61      opBDir[i2, 2] = 1 - (opBDir[i2, 0] + opBDir[i2, 1]);
62      opFDif[i2, 2] = 1 - (opFDif[i2, 0] + opFDif[i2, 1]);
63      opBDif[i2, 2] = 1 - (opBDif[i2, 0] + opBDif[i2, 1]);
64      sPropChanged = true;
65    }
66  }
67
68  //太陽位置と日射遮蔽物光学特性不変の場合は終了
69  if (!sunMoved && !sPropChanged) return;
70
71  //総合光学特性の更新処理////////////////////////////////////////////////
72  double adF, adB;
73  double ttlTF, ttlRF, ttlTB, ttlRB;
74  //直達日射に関する特性を更新
75  if (0 < cos)
76  {
77    double[] bf = new double[absFDir.Length];
78    computeTotalOProperties
79      (opFDir, opBDir, out ttlTF, out ttlRF, ref absFDir, out ttlTB, out ttlRB, ref bf);
80    DirectSolarIncidentTransmittance = ttlTF;
81    DirectSolarIncidentReflectance = ttlRF;
82    integrateAbsorption(absFDir, agapRes, glassRes, out adF, out adB);
83    DirectSolarIncidentAbsorptance = adB;
84  }
85  else
86  {
87    DirectSolarIncidentTransmittance = 0;
88    DirectSolarIncidentReflectance = 1;
89    DirectSolarIncidentAbsorptance = 0;
90  }
91
92  //遮蔽物光学特性特性変化時には拡散日射に関する特性を更新
93  if (sPropChanged) updateDiffuseTotalProperties();
94 }
```

【例題 24.2】

南面に設置された二重ガラス窓について、7 月 20 日における日射熱取得の推移を計算せよ。また、スラット幅 22.5mm、スラット間隔 25mm のブラインドを外側、中空層、内側に設置した場合の差異を評価せよ。ただしスラット角は 30º で固定し、反射率は上側下側ともに 0.5 とする。屋内外の総合熱伝達率はそれぞれ 23W/(m^2·K)と 9W/(m^2·K)とする。

Low-E 6mm + 透明フロート 6mm

Low-E 3mm: 透過率 0.396, 反射率（表面）0.355, 反射率（裏面）0.427

透明フロート: 透過率 0.859, 反射率 0.077

中空層熱抵抗 : 8 W/(m^2·K)

【解】

計算処理をプログラム 24.8 に示す。出力の内、総合透過率と吸収日射熱取得率を合算した値の時系列グラフを作成すると図 24.6 が得られる。ガラスの入射角特性の影響により、入射角が浅くなる 8:00 や 16:00 頃の日射熱取得は小さく、逆に昼頃には大きな値となる。ブラインドを設置すると日射熱取得は低減されるが、設置箇所は外側に近いほど低減率が大きい。ブラインドに吸収された熱の室内側放熱成分が小さくなるためである。ピーク時間帯の日射熱取得は、ブラインド無に比較して内側ブラインド設置で約 8 割、外側ブラインド設置で約 3 割に減る。なお、入射角が浅い時間帯に逆転がみられるが、ブラインド材の吸収率を入射角によらず一定とみなすモデルとしているためである。

プログラム **24.8**　ガラス窓の日射熱取得の計算

```
 1 private static void windowTest()
 2 {
 3   Incline inc = new Incline(0.0, 0.5 * Math.PI);
 4   Window[] win = new Window[4];
 5   for (int i = 0; i < win.Length; i++)
 6   {
 7     win[i] = new Window(1,
 8       new double[] { 0.396, 0.859 }, new double[] { 0.355, 0.077 },
 9       new double[] { 0.396, 0.859 }, new double[] { 0.427, 0.077 }, inc);
10     for (int j = 0; j < win[i].GlazingNumber; j++)
11     {
12       if(j != win[i].GlazingNumber - 1)
13         win[i].SetAirGapResistance(j, 1 / 8d);
14       win[i].SetGlassResistance(j, 0.006 / 1d);
15     }
16     win[i].ConvectiveCoefficientF = 18.5;
17     win[i].RadiativeCoefficientF = 4.5;
18     win[i].ConvectiveCoefficientB = 4.5;
19     win[i].RadiativeCoefficientB = 4.5;
20   }
21
22   VenetianBlind[] blinds = new VenetianBlind[3];
23   for (int i = 0; i < blinds.Length; i++)
24   {
25     blinds[i] = new VenetianBlind(25, 22.5, 0, 0, 0.7, 0.7);
26     blinds[i].SlatAngle = 30 / 180d * Math.PI;
27     blinds[i].Pulldowned = true;
28   }
29   win[1].SetShadingDevice(2, blinds[0]);
30   win[2].SetShadingDevice(1, blinds[1]);
31   win[3].SetShadingDevice(0, blinds[2]);
32
33   Sun sun = new Sun(Sun.City.Tokyo);
34   DateTime dTime = new DateTime(2001, 7, 20, 7, 0, 0);
35   using (StreamWriter sWriter = new StreamWriter
36     ("WindowTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
37   {
38     sWriter.WriteLine
39       ("時刻,無透,無反,無吸,内透,内反,内吸,中透,中反,中吸,外透,外反,外吸");
40
41     while (true)
42     {
43       if (dTime.Hour == 16) break;
44
45       sWriter.Write(dTime.ToShortTimeString());
46       sun.Update(dTime);
47       for (int i = 0; i < win.Length; i++)
48       {
49         win[i].UpdateOpticalProperties(sun);
50         sWriter.Write(
51           "," + win[i].DirectSolarIncidentTransmittance.ToString("F3") +
52           "," + win[i].DirectSolarIncidentReflectance.ToString("F3") +
53           "," + win[i].DirectSolarIncidentAbsorptance.ToString("F3"));
54       }
55       sWriter.WriteLine();
56       dTime = dTime.AddMinutes(5);
57     }
58   }
59 }
```

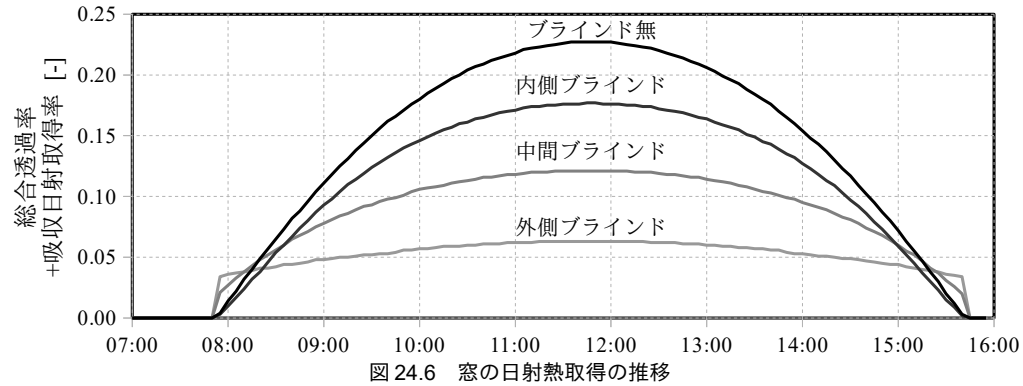


図 24.6　窓の日射熱取得の推移

【第 24 章 記号表】

A_W	：窓面積 [m²]	R_W	：窓の熱抵抗 [m²·K/W]
d	：厚み [m]	T	：温度 [K]
F_S	：天空への形態係数 [-]	T_{Sol}	：相当温度 [K]
I_D	：窓面に入射する屋外直達日射 [W/m²]	α	：吸収率 [-]
I_{SR}	：窓面に入射する屋外拡散日射 [W/m²]	α_T	：総合吸収率 [-]
I_{BSW}	：窓面に入射する室内拡散日射 [W/m²]	$\alpha_{(c)}$	：対流熱伝達率 [W/(m²·K)]
Q_{WO}	：内外温度差による熱取得 [W]	$\alpha_{(r)}$	：放射熱伝達率 [W/(m²·K)]
Q_α	：吸収日射熱取得 [W]	$\alpha_{(c+r)}$	：総合熱伝達率 [W/(m²·K)]
Q_τ	：透過日射熱取得 [W]	ε_{LW}	：長波長放射率 [-]
R_A	：中空層の熱抵抗 [m²·K/W]	θ	：入射角 [radian]
$R_{\alpha F}$	：吸収熱の表側放熱比 [-]	λ	：熱伝導率 [W/(m·K)]
$R_{\alpha B}$	：吸収熱の裏側放熱比 [-]	ρ	：反射率 [-]
R_G	：ガラスの熱抵抗 [m²·K/W]	τ	：透過率 [-]
R_N	：夜間放射 [W/m²]		
添字：			
B	：裏側	o	：屋外側
F	：正面側	W	：窓
G	：ガラス	ZN	：ゾーン
i	：室内側		

【第 24 章 参考文献】

24.1） Tallis, John: Tallis's history and description of the Crystal palace, and the Exhibition of the world's industry in 1851, London and New York, 1852

24.2） JIS 3106 1998: 板ガラス類の透過率・反射率・放射率・日射熱取得率の試験方法

24.3） JIS 3107 1998: 板ガラス類の熱抵抗及び建築における熱貫流率の算定方法

24.4） 窓の遮熱性能計算・試験方法の JIS・ISO 化 成果報告書: 経済産業省委託 平成 24 年度国際標準開発事業, 一般社団法人 日本建材・住宅設備産業協会, 平成 25 年 3 月

24.5） 旭硝子 総合カタログ技術資料編 2015 年 1 月版

24.6） 木下泰斗, 赤坂裕, 二宮秀興: 板ガラスとベネシャンブラインドとを組み合わせた光学特性の計算法, 日本建築学会環境系論文集, No.639, Vol.74, pp.569-577, 2009

第25章　熱負荷計算 4: 多数室連成計算
(Heat Load Calculation 4: Multi-Room Heat Balance Analysis)

25.1　概要

空調設備は内壁で隔てられた室単位で分割・計画されることが多い。住宅であれば、室毎に設けられた家庭用ルームエアコンがこれにあたる。また、同一の室であっても熱的な特徴が異なる場合には適当なゾーンに区切り、当該ゾーンに対して必要な容量の設備を計画することもある。例えば事務所建築であれば外乱の影響を受けやすいペリメータ部分と、主に内部発熱の処理が求められるインテリア部分とに分けることが通常である。

機器の容量決定にあたっては、通常は室単位あるいはゾーン単位で独立したピーク負荷計算を行う。しかし厳密にはこれらの室あるいはゾーンは熱的に独立していない。ゾーンとゾーンの間では空気の交換や壁表面相互の熱放射がある。また、内壁で分断された室と室であっても壁自体を通して熱の移動がある。従って、建物内の各部屋の空気温度や壁の温度は相互に影響しあっており、より正確に状態を予測するためにはこれらを連成させてモデルを解く必要がある†1)。本章では、第 22 章~第 24 章で開発した壁体および窓の計算モデルを用い、多数室の連成計算†2)を行う方法を示す。

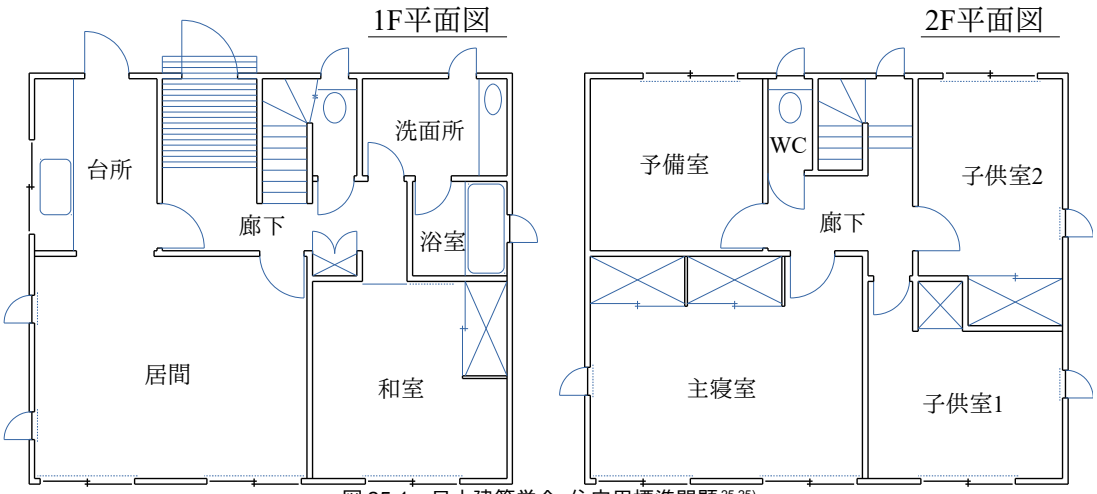


図 25.1　日本建築学会 住宅用標準問題[25.25)]

（1980 年代に多くの建築熱負荷計算ソフトウェアが多数室連成問題に対応し、
これらの計算結果の相互比較を目的に標準的な計算対象建物が提案された）

†1　全館空調のような運用方法の場合には各室を独立させた計算であっても大きな誤差は生じないと予想できるが、例えば非空調室に囲まれた空調室であったり、マンションで隣接住戸の生活スタイルが計算対象の住戸と異なる場合など、複数室の連成が必要となる場面は多い。

†2　本書で解説するモデルは、石田らによって提案された計算法[25.1)]を基礎に、ゾーンの概念を加えた計算モデルである。宇田川による文献[25.2)]には石田の計算法を 2 室に具体化させた例や、単室の場合の簡易計算に関しても解説がある。

25.2　理論

25.2.1　モデルの構造

本書のモデルでは建築を「室」と「ゾーン」の概念で分割して捉える。室は壁などで区切られた空間であり、ある室から別の室は直接に見ることができない。従って、室と室とで放射による直接的な熱移動は生じない。ただし、室と室とで空気の移動はあるため、空気を介した熱流は発生しうる。また、内壁を介した熱流も発生しうる。室を熱的な特性によって再分割した概念がゾーンであり、その代表例はインテリアゾーンとペリメータゾーンである。前者は外乱の影響が相対的に小さい空間であり、後者は窓や外壁の存在により外乱の影響が相対的に大きい空間である†1)。モデル上はそれぞれのゾーンに質点を設定して計算を行う。図 25.2 に室内の熱移動を示す。

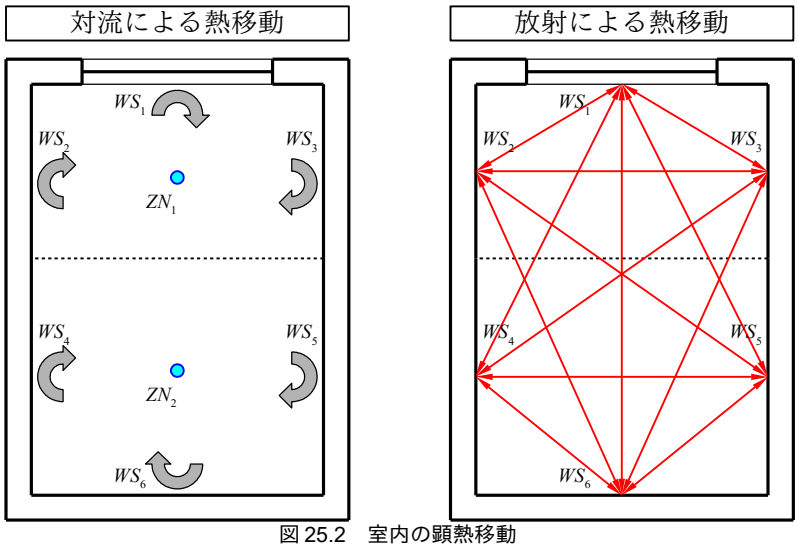


図 25.2　室内の顕熱移動

ZN_1 と ZN_2 という 2 つのゾーンに分割されており、それぞれが質点を持つ。室には WS_1～WS_6 までの 6 つの壁（窓）表面が存在しており、WS_1～WS_3 は ZN_1 に、WS_4～WS_6 は ZN_2 に面している。図 25.2 左に示すように、それぞれの壁表面は、面するゾーンとの間で対流による熱交換を行う。一方、図 25.2 右に示すように、放射による熱交換は同じ室に属する他のすべての壁表面との間で行われる。

計算対象には NRM 個の室が含まれるとする。i 番目の室に属するゾーンの数を NZN_i 個と表現すれば、i 番目の室に属する j 番目のゾーンに対して式 25.1 で通し番号 $q_{(i-j)}$を振ることができる。また、このゾーンを添字の$(i\text{-}j)$で表現することとする。総てのゾーン数を NQ と表現する。同様に、$q_{(i-j)}$番目のゾーンに面する壁表面の数を $NWS_{q(i-j)}$個と表現すれば、$q_{(i-j)}$番目のゾーンに面する k 番目の壁に対して、式 25.2 で通し番号 $s_{(i-j-k)}$を振ることができる。また、この壁表面を添字の$(i\text{-}j\text{-}k)$で表現することとする。すべての壁表面の数を NS と表現する。以降、$q_{(i-j)}$と $s_{(i-j-k)}$の添字は適宜省略して q および s と表現する。また、s 番目の壁表面の裏側の表面を sR と表現し、裏面は iR 番目の室に属する jR 番目のゾーンに面する kR 番目の壁表面であり、その通し番号は qR とする。

$$q_{(i-j)}=\sum_{rmn=0}^{i-1} NZN_{rmn}+j \tag{25.1}$$

†1　ペリメータゾーンとインテリアゾーンの捉え方は、最大熱負荷計算と機器容量選定に特に大きな影響を与える。石野らや中原らによる実測調査と数値計算にもとづく考察がある[25.15) 25.21)]。

$$s_{(i-j-k)}=\sum_{rmn=0}^{i-1}\sum_{znn=0}^{NZN_{rmn}} NWS_{q(rmn-znn)}+\sum_{znn=0}^{j-1} NWS_{q(i-znn)}+k \tag{25.2}$$

25.2.2 乾球温度と顕熱負荷

1）壁と窓の表面温度

第 22 章の式 22.31、式 22.32 や第 24 章の式 24.4 を用いると s 番目の壁表面温度 $T_{WS,s}$ [K]は式 25.3 で表現される。$T_{SolF,s}$ [K]および $T_{SolB,s}$ [K]はそれぞれ表側と裏側の相当温度であり、式 25.4 および式 25.5 で表現される。式 25.4 と式 25.5 右辺の各項はそれぞれ、ゾーンの空気の乾球温度、室に属する他の壁からの長波長放射、その他の放射成分、に関する項である。k_c [-]および k_r [-]は総合熱伝達率に対する対流熱伝達率および放射熱伝達率の比率であり、第 22 章の式 22.2 および式 22.3 に示したとおりである。第二項の ϕ [-]は基準化したゲブハルトの吸収係数であり、計算法については後述する。第三項の R_{SLW} [W/m^2]は壁表面への放射量であり、窓面を透過した日射（短波長放射）や室内の機器などが発する長波長放射を合算した値である（計算法については 25.2.2 節で述べる）。なお、式 25.3 は第 22 章の式 22.1 や第 24 章の式 24.2 と異なるように見えるが、式 25.3 で室温 T_{ZN} と壁表面の温度 T_{WS} を等しいと仮定すれば、式 22.1 および式 24.2 が得られる。第 22 章と第 24 章では壁や窓を取り囲む物体の温度が未知のため、このような仮定を行ったものである。

$$T_{WS,s}=IF_s+FF_sT_{SolF,s}+BF_sT_{SolB,sR} \tag{25.3}$$

$$T_{SolF,s}=k_{c,s}T_{ZN,s}+k_{r,s}\sum_{znn=0}^{NZN_i}\sum_{wsn=0}^{NWS_{q(i-znn)}}\phi_{s,(i-znn-wsn)}T_{WS,(i-znn-wsn)}+\frac{R_{SLW,s}}{\alpha_{(c+r),s}} \tag{25.4}$$

$$T_{SolB,sR}=k_{c,sR}T_{ZN,sR}+k_{r,sR}\sum_{znn=0}^{NZN_{iR}}\sum_{znn=0}^{NWS_{q(iR-znn)}}\phi_{sR,(i-znn-wsn)}T_{WS,(i-znn-wsn)}+\frac{R_{SLW,sR}}{\alpha_{(c+r),sR}} \tag{25.5}$$

裏側が屋外である場合など、相当温度が境界条件として与えられる場合には式 25.3 の相当温度 T_{SolB} に直接に相当温度 $T_{e,sR}$ を代入し、式 25.6 とする。また、裏側が機械室や倉庫などであり、あまり温度が重要でない場合には、式 25.7 で表現される隣室温度差係数（Adjacent space temperature difference factor）f_{td} を用いる。外気温度 T_{OA} とゾーンの温度 T_{ZN} を重み付けることで隣室の温度を予測するモデルであり、隣室温度差係数 f_{td} が 0.0 の場合には隣室温度=ゾーン温度 T_{ZN} となり、f_{td} が 1.0 の場合には隣室温度=外気温度 T_{OA} となる。室用途に応じた隣室温度差係数の例の表 25.1 に示す。

$$T_{SolB,sR}=T_{e,sR} \tag{25.6}$$

$$T_{SolB,sR}=f_{td}T_{OA}+(1-f_{td})T_{ZN,s} \tag{25.7}$$

表 25.1 隣室温度差係数の例[25.18) 25.19)]

建物用途	室用途	冷房時	暖房時
事務所	廊下	0.4	0.4
	便所（空調リターン空気により換気）	0.4	0.35~0.50
	便所（外気により換気）	0.75~0.85	0.75~0.85
	倉庫（換気あり）	0.8~0.9	0.8~0.9
	倉庫（換気なし）	0.7~0.8	0.8
集合住宅	南・北室	0.45	0.5
	東・西室	0.65	0.5
戸建住宅	小屋裏	2.2[†1)]	1.0
	高断熱非空調室	0.9	0.6
	低断熱非空調室	1.3	0.9
	床下	0.7	0.7

†1 日射の影響により隣室温度差係数が 1.0 を超えているとのことである。しかし、このような場合には熱せられた隣室による影響が大と推測されるため、隣室温度差係数を用いた簡易な計算ではなく、隣室に質点をとり、連成計算に含めるべきであるようにも思う。

式 25.3~式 25.6 を行列表示すると式 25.8 が得られる。[T_{WS}]と[T_{ZN}]は壁表面と室内空気の温度ベクトル、[*AT*]と[*BT*]はこれらに対する係数行列である。また、[*CT*]は現在時点の壁体内温度や床冷暖房配管への流入流量などによって記述され、将来時点の温度情報を含まない境界条件である。

[*AT*]は計算対象とする空間が持つすべての壁表面の数（*NS*）を行列数とする正方行列である。$s_1=s_{(i\text{-}j\text{-}k)}$、$s_2=s_{(l\text{-}m\text{-}n)}$とおくと、[*AT*]行列の s_1 行 s_2 列の要素は式 25.9 で表現される。ただし b.c は境界条件（boundary condition）である。[*BT*]は行数を *NS*、列数をすべてのゾーンの数（*NQ*）とする行列である。$q_1=q_{(i\text{-}j)}$、$q_2=q_{(l\text{-}m)}$とおくと、[*BT*]の s_1 行 q_2 列の要素は式 25.10 で表現される。[*CT*]は *NS* の要素数を持つベクトルであり、s_1 行の要素は式 25.11 で表現される。後述するように、熱平衡を解くためには[*AT*]の逆行列を計算する必要があるが、式 25.9 からわかるように壁および表面の熱抵抗と熱容量に変化がなければ値は変わらない。従って、これらの要素をプログラム中で変化させ続けるのでなければ、逆行列の計算処理は一度で済む。[*BT*]の値に関しても同様である。

$$[\boldsymbol{AT}][\boldsymbol{T_{WS}}]=[\boldsymbol{BT}][\boldsymbol{T_{ZN}}]+[\boldsymbol{CT}] \tag{25.8}$$

$$AT_{s_1,s_2}=\begin{cases}1 & (s_1=s_2)\\ -FF_{s1}k_{r,s1}\phi_{s1,s2} & ((i=l)\wedge(s_1\neq s_2))\\ -BF_{s1}k_{r,sR1}\phi_{sR1,s2} & ((iR=l)\wedge(sR_1\neq s_2)\wedge(sR_1\neq \mathrm{b.c}))\\ 0 & (\mathrm{others})\end{cases} \tag{25.9}$$

$$BT_{s_1,q_2}=\begin{cases}FF_{s1}k_{c,s1} & (q_1=q_2)\\ BF_{s1}k_{c,sR1} & ((q_{1R}=q_2)\wedge(sR_1\neq \mathrm{b.c}))\\ 0 & (\mathrm{others})\end{cases} \tag{25.10}$$

$$CT_{s_1}=\begin{cases}IF_{s1}+\dfrac{FF_{s1}R_{SLW,s1}}{\alpha_{(c+r),s1}}+\dfrac{BF_{s1}R_{SLW,sR1}}{\alpha_{(c+r),sR1}} & (sR_1\neq \mathrm{b.c})\\ IF_{s1}+\dfrac{FF_{s1}R_{SLW,s1}}{\alpha_{(c+r),s1}}+BF_{s1}T_{e,sR1} & (sR_1=\mathrm{b.c})\end{cases} \tag{25.11}$$

オフィスの基準階のように上下階に同じ部屋が繰り返し表れるモデルをつくりたいことがある。この場合には床スラブを挟んだ裏側のゾーンが自分自身であるとして式をたてる。従って、例えば床スラブの上側の表面温度であれば、25.3 に示すように下階に存在する自分自身の放射が床スラブを介して伝わることを表現する。この場合には式 25.9 の右辺第 1 行と第 3 行を組み合わせ、$AT_{s1,s1}$=1-$BF_{s1}k_{r,sR1}\phi_{s1,s2}$とする。同様に、式 25.10 でも $q_{1R}=q_1$（裏側が自分自身）となりうることを考慮したプログラムにする必要がある。

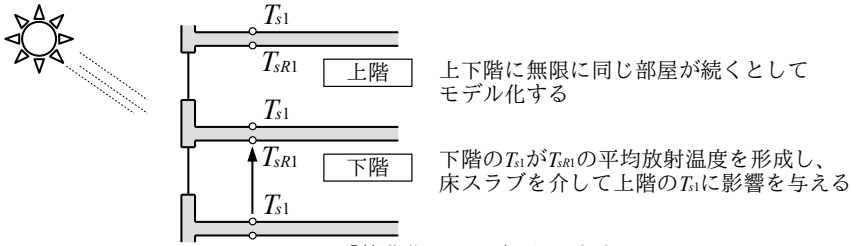


図 25.3　「基準階」のモデル化の方法

2）ゾーンの顕熱平衡

q_1 番目のゾーンの顕熱平衡は式 25.12 の後退差分式で表される。C_{SZN} [J/K]はゾーンの顕熱容量、Δt [sec]はタイムステップ、T_{ZN} [K]はゾーンのタイムステップ経過後の乾球温度、T^*_{ZN} [K]は現在時点の乾球温度である。

ゾーンの顕熱容量 C_{SZN} は式 25.13 に示すようにゾーンの湿り空気の熱容量と家具や資料などの熱容量 $C_{SFN,q1}$ [J/K]の合算値である。ただし M_{ZN} [kg]は室内空気の質量である。個別の建物について $C_{SFN,q1}$ を把握することは難しいが、事務所に関しては石野らの詳細な実測調査があり [25.13)]、室体積あたりで 15.2 kJ/(K·m^3)程度の値であると報告している。

右辺の 1 行目はゾーンに面する壁表面からの熱流を表している。それぞれの壁表面と室温との温度差に対して壁表面の面積 A [m^2]と対流熱伝達率 $\alpha_{(c)}$ [W/(m^2·K)]を乗じて合算する。

2 行目は他のゾーンとの空気交換による熱流を表している。$m_{a,(q2\text{-}q1)}$ [kg/s]は q_2 番目のゾーンから q_1 番目のゾーンへの空気の移動量、c_{pma} [J/(kg·K)]は湿り空気の定圧比熱である。このような空気の移動は一般にゾーン間換気または室間換気と呼ばれ、汎用的な推定法は無いが、石野らはインテリアゾーンとペリメータゾーンの間ではペリメータ容積基準で 20 回/h 程度という経験値を報告している [25.15) 25.16) 25.17) †1)]。また、ゾーン間の隣接長さあたりで設定する方法もある [25.20)]。単純な 2 ゾーン間の空気交換であれば $m_{a,(q2\text{-}q1)} = m_{a,(q1\text{-}q2)}$ となるが、複数のゾーンをまたぐ空気の移動がある場合（例えばインテリアでゾーンの空気が廊下にパスされた後に便所から排気されるような場合）には必ずしもこのようにならない。空気の収支が合うようにパラメータ設定を行うことが大切である。

3 行目は外気の流入による熱流を表しており、$m_{a,(OA\text{-}q1)}$ [kg/s]は換気量、T_{OA} [K]は外気乾球温度である。外気以外にも境界条件として与えられる外部からの流入空気がある場合には式 25.14 と 25.15 で補正する。m_{aBND} [kg/s]は外部からの流入空気流量、T_{BND} [K]は流入空気の乾球温度である。

4 行目は空調機などによる固定温度の給気による熱流を表しており、$m_{a,(SA\text{-}q1)}$ [kg/s]は換気量、T_{SA} [K]は外気乾球温度である。固定温度として取り扱う他のゾーンからのゾーン間換気についても本項で扱うため、任意の数が設定できるように Σ 記号を付している。

5 行目の第 1 項の $HG_{S(c)}$ [W]はゾーン内の人間、照明、機器などによる顕熱発生の対流成分である。放射成分に関しては日射とともに式 25.4 と式 25.5 の R_{SLW} を形成し、一旦、壁表面に入射して壁温上昇をもたらした後に式 25.12 の右辺 1 行目でゾーンの空気温度に影響を与える。5 行目の第 2 項の HL_S [W]は顕熱負荷であり、正の値の場合には暖房、負の値の場合には冷房である。

$$\begin{aligned} C_{SZN,q_1}\frac{T_{ZN,q_1}-T^{*}_{ZN,q_1}}{\Delta t} = & \sum_{k=0}^{NWS_{q1}} A_{s1}\alpha_{(c),s1}\left(T_{WS,s1}-T_{ZN,q_1}\right)+\sum_{l=0}^{NRM}\sum_{m=0}^{NZN_l} m_{a,(q2-q1)}c_{pma}\left(T_{ZN,q2}-T_{ZN,q1}\right) \\ & + m_{a,(OA-q_1)}c_{pma}\left(T_{OA}-T_{ZN,q1}\right)+\sum m_{a,(SA-q_1)}c_{pma}\left(T_{SA}-T_{ZN,q1}\right) \\ & + HG_{S(c),q1}+HL_{S,q1} \end{aligned} \tag{25.12}$$

$$C_{SZN,q_1} = c_{pma}M_{ZN,q1}+C_{SFN,q1} \tag{25.13}$$

$$m'_{a,(OA-q1)} = m_{a,(OA-q1)}+\sum_{n} m_{aBND,n} \tag{25.14}$$

$$T'_{OA} = \frac{m_{a,(OA-q1)}T_{OA}+\sum_{n} m_{aBND,n}T_{BND,n}}{m'_{a,(OA-q1)}} \tag{25.15}$$

式 25.12 を行列表示すると式 25.16 となる。[*DT*]はゾーンの数 *NQ* を行列数とする正方行列であり、式 25.17 で表される。[*ET*]はゾーンの数を行数とするベクトルとして式 25.18 で表され、将来時点の温度情報などを含まない境界条件である。[*FT*]はゾーンの数を行数、壁表面の数を列数とする行列で

†1 インテリアとペリメータで厳寒期においても非空調時明け方の温度差が 1ºC 程度という経験値から逆算して推定した値とのことである。いかにも建築分野的な発想で面白い。

あり、q_1行s_2列の要素は式 25.19 で表される。[HL_S]は各ゾーンの顕熱負荷を表すベクトルである。

$$[\boldsymbol{DT}][\boldsymbol{T_{ZN}}]=[\boldsymbol{ET}]+[\boldsymbol{FT}][\boldsymbol{T_{WS}}]+[\boldsymbol{HL_S}] \tag{25.16}$$

$$DT_{q_1,q_2}=\begin{cases}\dfrac{C_{SZN,q_1}}{\Delta t}+\sum_{k=0}^{NWS_{q1}} A_{s1}\alpha_{(c),s1}+c_{pma}\left(\sum_q m_{a,(q-q_1)}+m_{a,(OA-q_1)}+\sum m_{a,(SA-q_1)}\right) & (q_1=q_2)\\ -m_{a,(q_2-q_1)}c_{pma} & (q_1\neq q_2)\end{cases} \tag{25.17}$$

$$ET_{q_1}=\frac{C_{SZN,q_1}}{\Delta t}T^*_{ZN,q_1}+m_{a,(OA-q_1)}c_{pma}T_{OA}+\sum m_{a,(SA-q_1)}c_{pma}T_{SA}+HG_{S(c),q_1} \tag{25.18}$$

$$FT_{q_1,s_2}=\begin{cases}A_{s1}\alpha_{(c),s1} & (i=l\wedge j=m)\\ 0 & (\text{others})\end{cases} \tag{25.19}$$

式 25.16 の右辺には壁表面温度ベクトル[T_{WS}]が含まれる。式 25.8 の[AT]の逆行列[AT]$^{-1}$を求めれば、式 25.20 に示すように壁表面温度ベクトル[T_{WS}]を室温ベクトル[T_{ZN}]で表現できる。式 25.20 を式 25.16 に代入して壁表面温度ベクトル[T_{WS}]を消去すれば式 25.21 が得られる。ただし、[IT]および[JT]は式 25.22 と式 25.23 であり、ゾーンの数NQを要素数とする行列とベクトルである。

$$[\boldsymbol{T_{WS}}]=[\boldsymbol{AT}]^{-1}[\boldsymbol{BT}][\boldsymbol{T_{ZN}}]+[\boldsymbol{AT}]^{-1}[\boldsymbol{CT}] \tag{25.20}$$

$$[\boldsymbol{IT}][\boldsymbol{T_{ZN}}]=[\boldsymbol{JT}]+[\boldsymbol{HL_S}] \tag{25.21}$$

$$[\boldsymbol{IT}]=[\boldsymbol{DX}]-[\boldsymbol{FT}][\boldsymbol{AT}]^{-1}[\boldsymbol{BT}] \tag{25.22}$$

$$[\boldsymbol{JT}]=[\boldsymbol{ET}]+[\boldsymbol{FT}][\boldsymbol{AT}]^{-1}[\boldsymbol{CT}] \tag{25.23}$$

顕熱負荷[HL_S]を固定（境界条件）してゾーンの温度[TZ_N]を求めたい場合には式 25.21 の[IT]の逆行列[IT]$^{-1}$を求めて両辺に乗じればよい。しかし、実際の多くの問題では空調室と非空調室が入り混じるため、通常は[T_{ZN}]ベクトルの幾つかの要素と[HL_S]ベクトルの幾つかの要素がそれぞれ未知数となる。例えばゾーン数が 4 つの場合には式 25.21 は式 25.24 で表現される。ゾーン 0 と 2 のみを空調する場合には、下線の要素（ゾーン 1、3 の温度とゾーン 0、2 の空調負荷）が未知数となるため、式 25.24 を変形して式 25.25 に組み換えることで左辺に未知変数を集める。こうすれば左辺の係数行列の逆行列を計算して未知変数を一度に求めることができる。式 25.25 を一般化すると式 25.26 となる。ただし[KT]、[TH]、[LT]は式 25.27、式 25.28、式 25.29 である。[KT]はゾーンの数NQを行列数とする正方行列であり、この逆行列を求めて右辺の左から乗ずれば未知変数ベクトル[TH]が得られる。

$$\begin{bmatrix}IT_{0,0} & IT_{0,1} & IT_{0,2} & IT_{0,3}\\ IT_{1,0} & IT_{1,1} & IT_{1,2} & IT_{1,3}\\ IT_{2,0} & IT_{2,1} & IT_{2,2} & IT_{2,3}\\ IT_{3,0} & IT_{3,1} & IT_{3,2} & IT_{3,3}\end{bmatrix}\begin{bmatrix}T_{ZN0}\\ \underline{T_{ZN1}}\\ T_{ZN2}\\ \underline{T_{ZN3}}\end{bmatrix}=\begin{bmatrix}JT_0\\ JT_1\\ JT_2\\ JT_3\end{bmatrix}+\begin{bmatrix}\underline{HL_{S0}}\\ HL_{S1}\\ \underline{HL_{S2}}\\ HL_{S3}\end{bmatrix} \tag{25.24}$$

$$\begin{bmatrix}-1 & IT_{0,1} & 0 & IT_{0,3}\\ 0 & IT_{1,1} & 0 & IT_{1,3}\\ 0 & IT_{2,1} & -1 & IT_{2,3}\\ 0 & IT_{3,1} & 0 & IT_{3,3}\end{bmatrix}\begin{bmatrix}\underline{HL_{S0}}\\ \underline{T_{ZN1}}\\ \underline{HL_{S2}}\\ \underline{T_{ZN3}}\end{bmatrix}=\begin{bmatrix}JT_0\\ JT_1\\ JT_2\\ JT_3\end{bmatrix}+\begin{bmatrix}-(IT_{0,0}T_{ZN0}+IT_{0,2}T_{ZN2})\\ HL_{S1}-(IT_{1,0}T_{ZN0}+IT_{1,2}T_{ZN2})\\ -(IT_{2,0}T_{ZN0}+IT_{2,2}T_{ZN2})\\ HL_{S3}-(IT_{3,0}T_{ZN0}+IT_{3,2}T_{ZN2})\end{bmatrix} \tag{25.25}$$

$$[\boldsymbol{KT}][\boldsymbol{TH}]=[\boldsymbol{LT}] \tag{25.26}$$

$$KT_{q_1,q_2}=\begin{cases}IT_{q_1,q_2} & (T_{ZN,q_2}\neq \text{b.c})\\ -1 & ((T_{ZN,q_2}=\text{b.c})\wedge(q_1=q_2))\\ 0 & (\text{others})\end{cases} \tag{25.27}$$

$$TH_{q_1}=\begin{cases} T_{ZN,q_1} & (T_{ZN,q_1}\neq \mathrm{b.c}) \\ HL_{S,q_1} & (HL_{S,q_1}\neq \mathrm{b.c}) \end{cases} \tag{25.28}$$

$$LT_{q_1}=\begin{cases} JT_{q1}-\sum_{nq}^{NQ}\delta_{nq}IT_{q_1,nq}T_{ZN,nq} & (T_{ZN,q_1}\neq \mathrm{b.c}) \\ JT_{q1}-\sum_{nq}^{NQ}\delta_{nq}IT_{q_1,nq}T_{ZN,nq}+HL_{S,q_1} & (HL_{S,q_1}\neq \mathrm{b.c}) \end{cases} \tag{25.29}$$

$$\delta_{nq}=\begin{cases} 0 & (T_{ZN,nq}\neq \mathrm{b.c}) \\ 1 & (T_{ZN,nq}=\mathrm{b.c}) \end{cases} \tag{25.30}$$

熱負荷が空調設備容量を上回る場合には設定室温を満足できないため、熱負荷=設備容量とした上で、室温を未知変数として解く。しかし、熱負荷と設備容量の大小関係は予めわかるものではなく、熱負荷を計算して初めてわかる。従って、式 25.26 を繰り返し解く[†1]ことで熱負荷を固定するか、室温を固定するかを判定する必要がある。[*KT*]の逆行列を繰り返し計算することになるが、ゾーンの数 *NQ* を行列数とする行列であるため、その計算量は比較的小さい。式 25.16 から表面温度ベクトルを消去して未知変数を減らした狙いはここにある。

以上により各ゾーンの乾球温度が求まるため、式 25.20 の[T_{ZN}]に値を代入して表面温度ベクトル[T_{WS}]を計算する。さらにこれを式 25.4 と式 25.5 に代入して相当温度 T_{Sol} を計算すれば、以降は壁単体または窓単体の計算となり、第 22 章と第 24 章で解説したとおりである。

3）ゲブハルトの吸収係数

表面 *s*1 から表面 *s*2 への基準化したゲブハルトの吸収係数 $\phi_{s1,s2}$ はゲブハルトの吸収係数 $G_{s1,s2}$ [-]を用いて式 25.31 で計算する。なお、表面 *s*1 の放射熱伝達率 $\alpha_{(r),s1}$ [W/(m²·K)]はゲブハルトの吸収係数を用いて式 25.32 で近似する[†2]。ただし $\varepsilon_{LW,s1}$ は表面 *s*1 の長波長放射率である。T_m [K]は相互に放射熱交換を行う表面の平均温度であり、本書のモデルでは 297.15 K (=24 ºC)で一定とする。

$$\varphi_{s1,s2}=\begin{cases} \dfrac{G_{s1,s2}}{1-G_{s1,s1}} & (s1\neq s2) \\ 0 & (s1=s2) \end{cases} \tag{25.31}$$

$$\alpha_{(r),s1}=4\varepsilon_{LW,s1}\sigma T_m^3\left(1-G_{s1,s1}\right) \tag{25.32}$$

ゲブハルトの吸収係数 $G_{s1,s2}$ は、閉空間において表面 *s*1 から射出した放射の内、最終的に表面 *s*2 に吸収される放射量の比率である。閉空間に存在する各面のゲブハルトの吸収係数は、$G_{s1,s2}$ を成分とする *s*1 行 *s*2 列の行列[*G*]として式 25.33 で計算できる[25.3)]。[*F*]は *s*1 から *s*2 をみた形態係数 $F_{s1,s2}$ を成分とする行列である。また、[*I*]は単位行列であり、[ρ]と[ε]は式 25.34 と 25.35 に示すように対角成分を *s*1 番目の表面の長波長反射率 $\rho_{LW,s1}$ (=1-$\varepsilon_{LW,s1}$)または長波長放射率 $\varepsilon_{LW,s1}$ とする対角行列である。短波長放射について計算を行う場合には、長波長放射率 $\varepsilon_{LW,s1}$ を日射吸収率 α_{s1}（窓面の場合には(1-日射反射率 ρ_{s1})）に置き換えて計算する。

†1　すべてのゾーンの熱負荷を一回解き、設備容量を上回るゾーンのみを最大設備容量で固定すれば良いのだから、繰り返し計算は不要と思うかもしれないが、これは誤りである。ゾーンは相互に熱的に影響しているため、たとえ一回目の計算で 0 番目のゾーンの負荷が設備容量を下回ったとしても、0 番目のゾーンと熱的に関係のあるいくつかのゾーンが過負荷になったために室温が設定温度に維持できなくなり、当該ゾーン群からの熱流が増加した結果、0 番目のゾーンの負荷が増加して設備容量を上回るという可能性もある。従って、すべてのゾーンの負荷が設備容量以下になることを保証するためには、過負荷になるゾーンを順番に特定して取り除いていくという繰り返し計算が必要になるのである。

†2　線形近似の方法については第 22 章の 22.2.1 を参照。

$$[\boldsymbol{G}]=[\boldsymbol{F}]([\boldsymbol{I}]-[\boldsymbol{\rho}][\boldsymbol{F}])^{-1}[\boldsymbol{\varepsilon}_{\boldsymbol{LW}}] \tag{25.33}$$

$$[\boldsymbol{\rho}]=diag\left|(1-\varepsilon_{LW,0}),(1-\varepsilon_{LW,1}),...(1-\varepsilon_{LW,s1}),...(1-\varepsilon_{LW,NS})\right| \tag{25.34}$$

$$[\boldsymbol{\varepsilon}]=diag\left|\varepsilon_{LW,0},\varepsilon_{LW,1},...\varepsilon_{LW,s1},...\varepsilon_{LW,NS}\right| \tag{25.35}$$

形態係数は、特定の点や面などから対象となる面の見える比率である [†1)]。U 個の表面で構成される閉空間について、表面 t から表面 u をみた形態係数を $F_{t,u}$ と表現すると、式 25.36 の総和則が成立する。本書のモデルの場合には、同一の室に含まれる壁表面が放射熱交換を行うため、s_1 番目の壁表面について式 25.37 が成立することになる。また、表面 s_1 と表面 s_2 の表面積を A_{s1} と A_{s2} とすれば式 25.38 の相反法則が成立する。

$$\sum_{u=0}^{U} F_{t,u}=1 \tag{25.36}$$

$$\sum_{znn=0}^{NZN_i}\sum_{wsn=0}^{NWS_{znn}} F_{s1,s(i-znn-wsn)}=1 \tag{25.37}$$

$$A_{s1}F_{s1,s2}=A_{s2}F_{s2,s1} \tag{25.38}$$

面と点、面と面との相対的な位置関係が極々単純な場合には、形態係数を簡単に求めることができる計算式がある [25.4)]。また、熱的快適性を評価する際には平均放射温度を計算する必要があり、このために必要となる壁や窓表面と人体との形態係数に関しても簡易に近似する方法が提案されている [25.5) †2)]。比較的単純な多角形の場合には山崎らによる計算プログラムが提案されており [25.6)]、さらに複雑な形状の場合にはモンテカルロ法などを用いて数値計算的に近似値を求める方法もある [25.7)]。室内の壁や窓表面相互の形態係数を簡易に近似する方法としては、室に含まれる全壁表面積に対する比率とする方法が松尾によって提案されている [25.10)]。式 25.39 に i 番目の室に含まれる壁表面の形態係数の近似式を示す。

$$F_{s(i-j-k),s(i-m-n)}=A_{s(i-m-n)}/\sum_{znn=0}^{NZN_i}\sum_{wsn=0}^{NWS_{znn}} A_{s(i-znn-wsn)} \tag{25.39}$$

式 25.39 は計算が簡易で、総和則と相反法則を満足するが、自分自身をみる形態係数が 0 とはならないため、平面壁で囲まれることが一般的な建築の室空間においてはやや不合理な点もある。そこで松尾は本手法を発展させ、壁の表面積を円に内接する多角形の一辺とみなすことで形態係数を近似する手法を提案している [25.11) †3)]。図 25.4 に松尾による全形態係数の近似法の概念を示す。

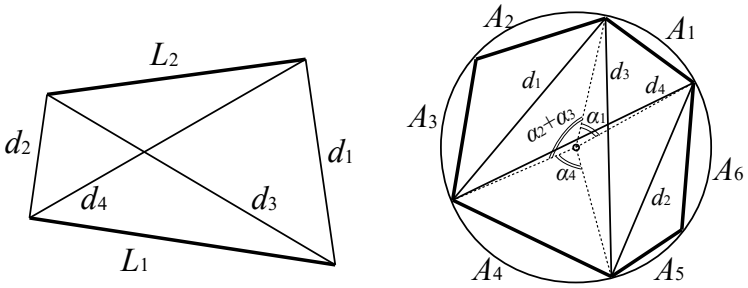


図 25.4　松尾による全形態係数の近似法 [25.11)]

†1　面対面の形態係数を特に「全形態係数」と呼ぶ。

†2　豚の形態係数などの提案もある [25.9)]。豚はストレスを感じやすく、人間と同様に熱的な快適性に関しての検討が必要な分野のようだ。

†3　松尾は本手法には解の一意性が無く、面の並べ順に規則が必要ではないかと指摘しているが、表面の数を n とした時に、$(n-1)$次元超球面に内接する n 次元多胞体と捉えれば自由度を調整できる可能性がある。検討が必要。

図 25.4 左に示すように二つの線分 L_1 と L_2 が与えられた場合、L_1 からみた L_2 の形態係数は式 25.40 によって計算できる[25.12) †4)]。ここで、閉空間に含まれる各表面の面積 A_n の値を線分の長さとして扱い、図 25.4 右に示すように円に内接する多角形として表現する。円の半径の求め方が問題となるが、まず、それぞれの辺と円の中心を結ぶ二等辺三角形に関して余弦定理を適用すると、式 25.41 に示すように頂角 α_n [radian]を半径 R [m]と一辺の長さ A_n [m]を用いて表現することができる。一方で、すべての二等辺三角形の頂角の和として式 25.42 が成立する。従って、初期値として適当な半径 R を与えて式 25.41 によって頂角を求め、これらを式 25.42 に設定して誤差を評価することにより、収束計算で半径 R を推定することができる。図 25.4 左の d_1~d_4 は円に内接する多角形の中で図 25.4 右のように表現されるため、式 25.40 は頂角の角度 α_n を用いて式 25.43 で表現できる。

$$F_{1,2}=0.5\left(d_3+d_4-d_1-d_2\right)/L_1 \tag{25.40}$$

$$\alpha_n=\arccos\left\{1-\left(A_n/2R\right)^2\right\} \tag{25.41}$$

$$\sum\alpha_n=2\pi \tag{25.42}$$

$$F_{n,m}=\frac{\left[\sin\left(\frac{1-\sum_{i=n+1}^{m}\alpha_i}{2}\right)+\sin\left(\frac{\sum_{i=n}^{m-1}\alpha_i}{2}\right)-\sin\left(\frac{\sum_{i=n+1}^{m-1}\alpha_i}{2}\right)-\sin\left(\frac{1-\sum_{i=n}^{m}\alpha_i}{2}\right)\right]}{A_n} \tag{25.43}$$

4）窓面の吸収日射熱取得

窓面の吸収日射熱取得の内、屋外からの日射による部分 Q_{WaF} [W]は第 24 章で解説したとおりである。屋内からの日射による部分 Q_{WaB} [W]を計算するためには、多重反射後に屋外に放出される日射 I_{BSW} [W/m^2]を計算する必要がある。このためには式 25.44 に示すように、ゲブハルトの吸収係数を用いて各窓面を透過して室内に入射した日射の内、どれだけが窓の内表面に分配されるかを積算する。

$Q_{SW,s}$ [W]は s 番目の表面から発せられる短波長であり、本書では図 25.5 右のようにモデル化する。$Q_{SW,s}$ を正確に計算するためには、窓面からの透過日射が床面のどの範囲に落ちるかを計算し、その範囲からの反射を計算する必要があり、図 25.5 左はこれを示している。しかし、日射の落ちる範囲は時々刻々変化するため、このような処理を行うと計算負荷が大きくなる。また、計算のためには床と窓面との三次元的な位置関係に関する情報が必要となるため、モデル作成作業も煩雑になる。そこで図 25.5 右に示すように、窓面からの透過日射の内、R_p [-]をペリメータ部の床面に配分し、残余（1-R_p）が窓面から放射されるとみなす。床面に配分された日射に床面の日射吸収率 a_s [-]を乗じた部分（$R_p \cdot a_s$）は床に吸収され、残余（$R_p \cdot (1-a_s)$）は反射される。このようにすれば毎時の再計算が不要となり、モデル作成上も特定の床を指定するだけで三次元形状の設定は不要となる。

問題は R_p の値であるが、筆者が二次元モデルでケーススタディをした結果、天井高 2.7m、室奥行 15.0m、窓高 2.2m、ペリメータ幅 5.0m、床面日射吸収率 0.7、壁面日射吸収率 0.5 の条件では R_p=0.7 で年間のペリメータ入射日射が詳細モデルとほぼ一致した[25.30)]。三次元形状が多少変化しても R_p の値は大きな影響を受けなかったため、0.7 を固定値として用いても誤差は小さいであろう。

R_p を用いると各面が発する短波長 Q_{SW} は式 25.45 で表される。A_W [m^2]は窓の表面積、I_D [W/m^2]は窓

†4　Cross string formula と呼ばれる。

面に入射する直達日射、$\tau_{TF,Dir}$ [-]は直達日射に対する総合透過率、I_{SR} [W/m²]は窓面に入射する拡散日射、$\tau_{TF,dif}$ [-]は拡散日射に対する総合透過率、$a_{s,flr}$ [-]は床面の日射吸収率である（式 24.18 を参照）。

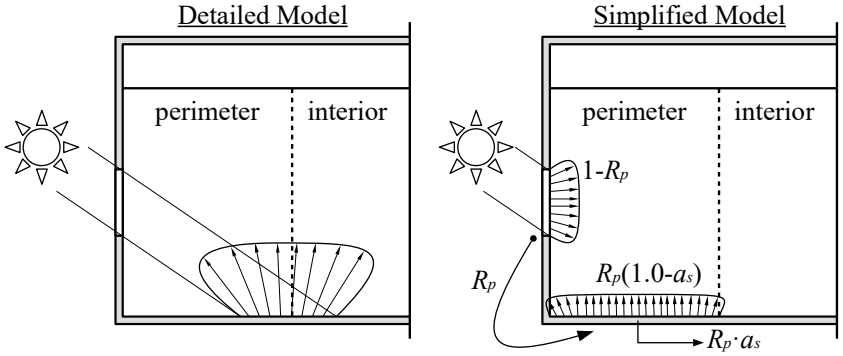


図 25.5　直達透過日射の分配方法

第 24 章の式 24.21 を適用し、式 25.44 で求められた値に対して屋内からの日射に対する吸収日射取得率を乗じることで Q_{WaB} を計算する。先に求めた屋外からの日射に対する部分 Q_{WaF} と合算すれば、対象となる窓の全体の吸収日射熱取得が得られる。

$$I_{BSW,s1}=\frac{\sum_{znn=0}^{NZN_i}\sum_{wsn=0}^{NWS_{znn}} G_{s(i-znn-wsn),s1}\,Q_{SW,s(i-znn-wsn)}}{A_{s1}} \tag{25.44}$$

$$Q_{SW}=\begin{cases} A_W\left(I_D\tau_{TF,Dir}(1-R_p)+I_{SR}\tau_{TF,Dif}\right) & (\text{window}) \\ A_W I_D\tau_{TF,Dir}R_p(1-a_{s,flr}) & (\text{perimeter floor}) \\ 0 & (\text{others}) \end{cases} \tag{25.45}$$

5）壁表面の放射熱取得

内部発熱 HG は顕熱発熱 HG_S と潜熱発熱 HG_L に分けられ、顕熱はさらに対流成分 $HG_{S(c)}$ と放射成分 $HG_{S(r)}$ に分けられる（式 25.46）。これらの内、潜熱発熱に関しては 25.2.3 節の潜熱平衡で取り扱い、対流成分に関しては式 25.12 で示したようにゾーンの熱平衡式の中で取り扱う。残る放射成分は式 25.47 で示すように室内の壁表面積比で各壁表面に割り振る。さらに多重反射後に壁に入射する日射 I_{BSW} を加算すれば、式 25.4 と式 25.5 における壁表面への入射 R_{SLW} が式 25.48 で計算できる。窓面の場合には式 25.45 に示したように吸収日射熱取得で考慮するため I_{BSW} は加算しないことに注意する。

$$HG=HG_{S(c)}+HG_{S(r)}+HG_L \tag{25.46}$$

$$I_{BLW,s1}=\frac{\sum_{znn=0}^{NZN_i} HG_{S(r),(i-znn)}}{A_{s1}}\cdot\frac{A_{s1}}{\sum_{znn=0}^{NZN_i}\sum_{wsn=0}^{NWS_{znn}} A_{s(i-znn-wsn)}}=\frac{\sum_{znn=0}^{NZN_i} HG_{S(r),(i-znn)}}{\sum_{znn=0}^{NZN_i}\sum_{wsn=0}^{NWS_{znn}} A_{s(i-znn-wsn)}} \tag{25.47}$$

$$R_{SLW}=\begin{cases} I_{BLW} & (\text{window}) \\ I_{BLW}+I_{BSW}+A_W I_D\tau_{TF,Dir}R_p a_{s,flr} & (\text{perimeter floor}) \\ I_{BLW}+I_{BSW} & (\text{others}) \end{cases} \tag{25.48}$$

25.2.3　絶対湿度と潜熱負荷

壁体を介した水分移動を考慮しなければ、潜熱平衡のモデルは顕熱平衡のモデルに比較して遥かに簡単である。

q_1 番目のゾーンの潜熱平衡は式 25.49 の後退差分式で表される。C_{LZN} [kg/(kg/kg)]はゾーンの水分容量であり、式 25.50 で計算する。式 25.50 の第 2 項の C_{LFN} [kg/(kg/kg)]は壁および家具の水分容量であ

り、慣用値としては室の容積あたりで 16.7 kg/((kg/kg)·m^3)程度を設定する[22.14]。W_{ZN} [kg/kg]はゾーンの絶対湿度であり、右辺第 1 項は壁体からの水分移動（吸放湿）を表現している。ただし、熱水分同時移動を解く方法については次節で解説することとし、本節では湿気伝達率 α_L [(kg/s)/((kg/kg)·m^2)]は 0 として扱い、本項は 0 となる。第 2 項はゾーン間換気による水分移動を表現している。第 3 項は換気による水分移動を表しており、W_{OA} [kg/kg]は外気絶対湿度である。第 4 項は空調機などの給気による水分移動を表しており、W_{SA} [kg/kg]は給気絶対湿度である。第 5 項の HG_L [kg/s]は人体などによる水分発生量、第 6 項の HL_L [kg/s]は水分供給量であり、正の場合には加湿、負の場合には除湿である。人体からの水分発生量は活動内容や温湿度条件によって変化するが（第 27 章 例題 27.1 参照）、熱負荷計算上は 40 W/人（= 1.6×10^{-5} (kg/s)/人）程度で一定として扱うことが多い。

$$C_{LZN,q_1}\frac{W_{ZN,q_1}-W^*_{ZN,q_1}}{\Delta t}=\sum_{k=0}^{NWS_{q1}}A_{s1}\alpha_{L,s1}(W_{WS,s1}-W_{ZN,q_1})+\sum_{l=0}^{NRM}\sum_{m=0}^{NZN_l}m_{a,(q2-q1)}(W_{ZN,q2}-W_{ZN,q1})$$
$$+m_{a,(OA-q_1)}(W_{OA}-W_{ZN,q1})+\sum m_{a,(SA-q_1)}(W_{SA}-W_{ZN,q1})+HG_{L,q1}+HL_{L,q1} \tag{25.49}$$

$$C_{LZN,q_1}=M_{ZN,q1}+C_{LFN,q1} \tag{25.50}$$

式 25.49 を行列で表現すると式 25.51 が得られる。[*AW*]はゾーンの数 *NQ* を行列数とする正方行列であり、式 25.52 で表される。[*BW*]はゾーンの数 *NQ* を行数とするベクトルとして式 25.53 で表される。[W_{ZN}]と[HL_L]はそれぞれ各ゾーンの絶対湿度と潜熱負荷を表すベクトルである。

$$[\boldsymbol{AW}][\boldsymbol{W_{ZN}}]=[\boldsymbol{BW}]+[\boldsymbol{HL_L}] \tag{25.51}$$

$$AW_{q_1,q_2}=\begin{cases}\dfrac{C_{LZN,q_1}}{\Delta t}+\sum_q m_{a,(q-q_1)}+m_{a,(OA-q_1)}+\sum m_{a,(SA-q_1)} & (q_1=q_2)\\ -m_{a,(q_2-q_1)} & (q_1\neq q_3)\end{cases} \tag{25.52}$$

$$BW_{q_1}=\frac{C_{LZN,q_1}}{\Delta t}W^*_{ZN,q_1}+m_{a,(OA-q_1)}W_{OA}+\sum m_{a,(SA-q_1)}W_{SA}+HG_{L,q_1} \tag{25.53}$$

顕熱平衡式の場合と同様に、湿度制御を行うか否かによって式 25.49 の W_{ZN} と HL_L のいずれが状態変数となるかが異なる。これを一般化して行列で表現すると式 25.54 が得られる。行列[*CW*]、[*WH*]、[*DW*]は絶対湿度 W_{ZN} が境界条件であるか否かに応じて式 25.55~25.57で計算する。

$$[\boldsymbol{CW}][\boldsymbol{WH}]=[\boldsymbol{DW}] \tag{25.54}$$

$$CW_{q_1,q_2}=\begin{cases}AW_{q_1,q_2} & (W_{ZN,q_2}\neq \mathrm{b.c})\\ -1 & ((W_{ZN,q_2}=\mathrm{b.c})\wedge(q_1=q_2))\\ 0 & (\mathrm{others})\end{cases} \tag{25.55}$$

$$WH_{q_1}=\begin{cases}W_{ZN,q_1} & (W_{ZN,q_1}\neq \mathrm{b.c})\\ HL_{L,q_1} & (HL_{L,q_1}\neq \mathrm{b.c})\end{cases} \tag{25.56}$$

$$DW_{q_1}=\begin{cases}BW_{q1}-\sum_{nq}^{NQ}\delta_{nq}AW_{q_1nq}W_{ZN,nq} & (W_{ZN,q_1}\neq \mathrm{b.c})\\ BW_{q1}-\sum_{nq}^{NQ}\delta_{nq}AW_{q_1nq}W_{ZN,nq}+HL_{L,q_1} & (HL_{L,q_1}\neq \mathrm{b.c})\end{cases} \tag{25.57}$$

25.2.4 熱水分同時移動

壁体の吸放湿を考慮する場合には、蒸発や凝縮により温度と湿度が相互に影響しあうため、25.2.2 節と 25.2.3 節で解説した顕熱平衡式と潜熱平衡式を連成させる必要がある。状態変数の数が増えてや

や煩雑になるが、基本的な方針は顕熱平衡の計算と同じである。式変形を工夫することにより壁表面の温湿度を消去して、ゾーンの温湿度のみで熱平衡式をたてる。

1）壁と窓の表面温湿度

第 22 章の 22.2.5 節に記したように、壁表面温度および絶対湿度は両側の相当温度 T_{Sol} と絶対湿度 W_{ZN} を用いて式 25.58、25.59 で表現できる。

$$T_{WS,s}=IF\,2_s+FFS\,2_s T_{SolF,s}+BFS\,2_s T_{SolB,s}+FFL\,2_s W_{ZNF,s}+BFL\,2_s W_{ZNB,s} \tag{25.58}$$

$$W_{WS,s}=IF\,3_s+FFS\,3_s T_{SolF,s}+BFS\,3_s T_{SolB,s}+FFL\,3_s W_{ZNF,s}+BFL\,3_s W_{ZNB,s} \tag{25.59}$$

式 25.4 と式 25.5 に示したように相当温度は壁表面温度で表現される。また、壁表面の裏側が境界条件の場合には式 25.6 に加え、絶対湿度を式 25.60 で表現する。式 25.4~25.6と式 25.60 を式 25.58 と 25.59 に代入して行列表示すると式 25.61 が得られる。$[TW_{WS}]$は要素数 $2NS$ のベクトルであり[†1]、0~NS が壁表面の温度、(NS+1)~$2NS$ が壁表面の絶対湿度である。$[TW_{ZN}]$は要素数 $2NQ$ のベクトルであり、0~NQ がゾーンの温度、(NQ+1)~$2NQ$ がゾーンの絶対湿度である。

$$W_{ZNB,sR}=W_{e,sR} \tag{25.60}$$

$$[\boldsymbol{A}][\boldsymbol{TW_{WS}}]=[\boldsymbol{B}][\boldsymbol{TW_{ZN}}]+[\boldsymbol{C}] \tag{25.61}$$

$[A]$は $2NS\times 2NS$ の係数行列であり、$s_1=s_{(i\text{-}j\text{-}k)}$、$s_2=s_{(l\text{-}m\text{-}n)}$、$r_1=s_1+NS$、$r_2=s_2+NS$ とおくと、各行列要素は式 25.62 で表現できる。$[B]$はそれぞれゾーンの温度と絶対湿度に対する係数行列であり、そのサイズは $2NS\times 2NQ$ である。$q_1=q_{(i\text{-}j)}$、$q_2=q_{(l\text{-}m)}$、$p_1=q_1+NQ$、$p_2=q_2+NQ$ とおくと、各行列要素は式 25.63 で表現できる。$[C]$は $2NS$ の要素数を持つベクトルであり、各要素は式 25.64 で表される。

$$A_{s_1,s_2}=\begin{cases}1 & (s_1=s_2)\\ -FFS\,2_{s1}k_{r,s1}\phi_{s1,s2} & ((i=l)\wedge(s_1\neq s_2))\\ -BFS\,2_{s1}k_{r,sR1}\phi_{sR1,s2} & ((iR=l)\wedge(sR_1\neq s_2)\wedge(sR_1\neq \mathrm{b.c}))\\ 0 & (\text{others})\end{cases}$$

$$A_{r_1,s_2}=\begin{cases}-FFS\,3_{s1}k_{r,s1}\phi_{s1,s2} & ((i=l)\wedge(s_1\neq s_2))\\ -BFS\,3_{s1}k_{r,sR1}\phi_{sR1,s2} & ((iR=l)\wedge(sR_1\neq s_2)\wedge(sR_1\neq \mathrm{b.c}))\\ 0 & (\text{others})\end{cases} \tag{25.62}$$

$$A_{r_1,r_2}=\begin{cases}1 & (s_1=s_2)\\ 0 & (\text{others})\end{cases},\qquad A_{s_1,r_2}=0$$

$$B_{s_1,q_2}=\begin{cases}FFS\,2_{s1}k_{c,s1} & (q_1=q_2)\\ BFS\,2_{s1}k_{c,sR1} & ((q_{1R}=q_2)\wedge(sR_1\neq \mathrm{b.c}))\\ 0 & (\text{others})\end{cases}$$

$$B_{s_1,p_2}=\begin{cases}FFL\,2_{s1} & (q_1=q_2)\\ BFL\,2_{s1} & ((q_{1R}=q_2)\wedge(sR_1\neq \mathrm{b.c}))\\ 0 & (\text{others})\end{cases}$$

$$B_{r_1,q_2}=\begin{cases}FFS\,3_{s1}k_{c,s1} & (q_1=q_2)\\ BFS\,3_{s1}k_{c,sR1} & ((q_{1R}=q_2)\wedge(sR_1\neq \mathrm{b.c}))\\ 0 & (\text{others})\end{cases} \tag{25.63}$$

$$B_{r_1,p_2}=\begin{cases}FFL\,3_{s1} & (q_1=q_2)\\ BFL\,3_{s1} & ((q_{1R}=q_2)\wedge(sR_1\neq \mathrm{b.c}))\\ 0 & (\text{others})\end{cases}$$

†1　実際には窓や水分移動が生じない壁体が存在するため、これらを除去して行列を作れば逆行列の計算負荷が小さくなる。しかし、通し番号の付与が非常に煩雑でありコーディングが困難になるため、単純に全ての壁表面で絶対湿度を解くという前提で式をたてている。

$$C_{s_1}=\begin{cases} IF2_{s1}+\dfrac{FFS2_{s1}R_{SLW,s1}}{\alpha_{(c+r),s1}}+\dfrac{BFS2_{s1}R_{SLW,sR1}}{\alpha_{(c+r),sR1}} & (sR_1\neq \mathrm{b.c}) \\ IF2_{s1}+\dfrac{FFS2_{s1}R_{SLW,s1}}{\alpha_{(c+r),s1}}+BFS2_{s1}T_{e,sR1}+BFL2_{s1}W_{e,sR1} & (sR_1=\mathrm{b.c}) \end{cases}$$
$$C_{r_1}=\begin{cases} IF3_{s1}+\dfrac{FFS3_{s1}R_{SLW,s1}}{\alpha_{(c+r),s1}}+\dfrac{BFS3_{s1}R_{SLW,sR1}}{\alpha_{(c+r),sR1}} & (sR_1\neq \mathrm{b.c}) \\ IF3_{s1}+\dfrac{FFS3_{s1}R_{SLW,s1}}{\alpha_{(c+r),s1}}+BFS3_{s1}T_{e,sR1}+BFL3_{s1}W_{e,sR1} & (sR_1=\mathrm{b.c}) \end{cases} \tag{25.64}$$

2）ゾーンの熱平衡

式 25.12 および式 25.49 を用いて、ゾーンの顕熱と潜熱を統合した平衡式は式 25.65 で表現できる。

$$[\boldsymbol{D}][\boldsymbol{TW_{ZN}}]=[\boldsymbol{E}]+[\boldsymbol{F}][\boldsymbol{TW_{WS}}]+[\boldsymbol{HL_{SL}}] \tag{25.65}$$

[*D*]は 2*NQ*×2*NQ* の行列であり、各要素は式 25.66 である。

$$D_{q_1,q_2}=\begin{cases} \dfrac{C_{SZN,q_1}}{\Delta t}+\sum_{k=0}^{NWS_{q1}}A_{s1}\alpha_{C,s1}+c_{pma}\left(\sum_q m_{a,(q-q_1)}+m_{a,(OA-q_1)}+\sum m_{a,(SA-q_1)}\right) & (q_1=q_2) \\ -m_{a,(q_2-q_1)}c_{pma} & (q_1\neq q_3) \end{cases}$$
$$D_{p_1,p_2}=\begin{cases} \dfrac{C_{LZN,q_1}}{\Delta t}+\sum_{k=0}^{NWS_{q1}}A_{s1}\alpha_{L,s1}+\sum_q m_{a,(q-q_1)}+m_{a,(OA-q_1)}+\sum m_{a,(SA-q_1)} & (q_1=q_2) \\ -m_{a,(q_2-q_1)} & (q_1\neq q_3) \end{cases} \tag{25.66}$$
$$D_{q_1,p_2}=D_{p_1,q_2}=0$$

[*E*]は長さ 2*NQ* のベクトルであり、各要素は式 25.67 である。

$$E_{q_1}=\frac{C_{SZN,q_1}}{\Delta t}T^*_{ZN,q_1}+m_{a,(OA-q_1)}c_{pma}T_{OA}+\sum m_{a,(SA-q_1)}c_{pma}T_{SA}+HG_{S(c),q_1}$$
$$E_{p_1}=\frac{C_{LZN,q_1}}{\Delta t}W^*_{ZN,q_1}+m_{a,(OA-q_1)}W_{OA}+\sum m_{a,(SA-q_1)}W_{SA}+HG_{L,q_1} \tag{25.67}$$

[*F*]は 2*NQ*×2*NS* の行列であり、各要素は式 25.68 である。

$$F_{q_1,s_2}=\begin{cases} A_{s1}\alpha_{(c),s1} & (i=l\wedge j=m) \\ 0 & (\mathrm{others}) \end{cases}$$
$$F_{p_1,r_2}=\begin{cases} A_{s1}\alpha_{L,s1} & (i=l\wedge j=m) \\ 0 & (\mathrm{others}) \end{cases} \tag{25.68}$$
$$F_{q_1,r_2}=F_{p_1,s_2}=0$$

式 25.61 で[*A*]の逆行列を求めて移項すると式 25.69 が得られる。これを式 25.65 に代入し、式 25.70 を得る。ただし、[*I*]と[*J*]は式 25.71 と式 25.72 であり、要素数が 2*NQ* の正方行列とベクトルである。

$$[\boldsymbol{TW_{WS}}]=[\boldsymbol{A}]^{-1}[\boldsymbol{B}][\boldsymbol{TW_{ZN}}]+[\boldsymbol{A}]^{-1}[\boldsymbol{C}] \tag{25.69}$$

$$[\boldsymbol{I}][\boldsymbol{TW_{ZN}}]=[\boldsymbol{J}]+[\boldsymbol{HL_{SL}}] \tag{25.70}$$

$$[\boldsymbol{I}]=[\boldsymbol{D}]-[\boldsymbol{F}][\boldsymbol{A}]^{-1}[\boldsymbol{B}] \tag{25.71}$$

$$[\boldsymbol{J}]=[\boldsymbol{E}]+[\boldsymbol{F}][\boldsymbol{A}]^{-1}[\boldsymbol{C}] \tag{25.72}$$

温湿度制御を行うか否かにより、T_{ZN} と HL_S、W_{ZN} と HL_L のいずれが状態変数となるかが異なる。一般化して行列で表現すると式 25.73~25.77が得られる。

$$[\boldsymbol{K}][\boldsymbol{TWH}]=[\boldsymbol{L}] \tag{25.73}$$

$$K_{m,n}=\begin{cases} I_{m,n} & (TW_{ZN,n}\neq \mathrm{b.c}) \\ -1 & ((TW_{ZN,n}=\mathrm{b.c})\wedge(m=n)) \\ 0 & (\mathrm{others}) \end{cases} \tag{25.74}$$

$$TWH_m=\begin{cases} TW_{ZN,m} & (TW_{ZN,m}\neq \mathrm{b.c}) \\ HL_{SL,m} & (HL_{SL,m}\neq \mathrm{b.c}) \end{cases} \tag{25.75}$$

$$L_m=\begin{cases} J_m-\sum_{nq}^{2NQ}\delta_{nq}I_{m,nq}TW_{ZN,nq} & (TW_{ZN,m}\neq \mathrm{b.c}) \\ J_m-\sum_{nq}^{2NQ}\delta_{nq}I_{m,nq}TW_{ZN,nq}+HL_{SL,m} & (HL_{SL,m}\neq \mathrm{b.c}) \end{cases} \tag{25.76}$$

$$\delta_{nq}=\begin{cases} 0 & (TW_{ZN,nq}\neq \mathrm{b.c}) \\ 1 & (TW_{ZN,nq}= \mathrm{b.c}) \end{cases} \tag{25.77}$$

25.3　計算法

25.3.1　壁・窓表面クラスの作成

顕熱平衡モデルあるいは熱水分同時移動モデルを解くためには、ゾーンの空気と壁・窓表面との間の顕熱流あるいは水分移動を計算する必要がある。このためにはゾーンからみて壁・窓表面が表（F 側）なのか裏（B 側）なのかを判別する必要がある。プログラムの可読性を高めるために、この条件分岐を司る wallWindowSurface クラスを開発する。プログラム 25.1 に wallWindowSuraface クラスのコンストラクタおよび関連するインスタンスとプロパティを示す。16~36 行に示すようにコンストラクタの第 1 引数は壁または窓のインスタンスであり、第 2 引数で F 側表面か B 側表面かを判定する。壁か窓か、表か裏かの判定フラグを用意する。14 行の Index は壁表面の通し番号を保持するプロパティであり、-1 で初期化しておく。

プログラム 25.1　コンストラクタおよび関連するインスタンスとプロパティ

Popolo.ThermalLoad.wallWindowSurface class

```
 1 /// <summary>F 側か否か</summary>
 2 private bool isSideF;
 3
 4 /// <summary>壁表面か否かを取得する</summary>
 5 public bool IsWall { get; private set; }
 6
 7 /// <summary>壁を取得する</summary>
 8 public Wall Wall { get; private set; }
 9
10 /// <summary>窓を取得する</summary>
11 public Window Window { get; private set; }
12
13 /// <summary>番号を設定・取得する</summary>
14 public int Index { get; set; }
15
16 /// <summary>インスタンスを初期化する</summary>
17 /// <param name="wall">壁</param>
18 /// <param name="isSideF">F 側表面か否か</param>
19 public wallWindowSurface(Wall wall, bool isSideF)
20 {
21   IsWall = true;
22   Wall = wall;
23   this.isSideF = isSideF;
24   Index = -1;
25 }
26
27 /// <summary>インスタンスを初期化する</summary>
28 /// <param name="window">窓</param>
29 /// <param name="isSideF">F 側表面か否か</param>
30 public wallWindowSurface(Window window, bool isSideF)
31 {
32   IsWall = false;
33   Window = window;
34   this.isSideF = isSideF;
35   Index = -1;
36 }
```

プログラム 25.2 に wallWindowSurface クラスのプロパティ定義を示す。1~11 行は壁・窓表面に関連

するパラメータを保存するためのプロパティである。wallWindowSuraface の主たる機能は 13 行以降のプロパティにある。例えば 13~23 行は裏側の表面を取得するためのプロパティであるが、壁か窓かの判定、F 側か B 側かの判定をプロパティ内で行い、適切なインスタンスを出力する。このような構成にすることで、wallWindowSurface インスタンスのプロパティを呼び出す側は、その表面の実体が壁なのか窓なのか、あるいは F 側なのか B 側なのかを気にせずに計算を進めることができるようになる。25~42 行は相当温度に関するプロパティであるが、設定処理（set アクセッサ）に関してもプロパティ内で判別処理を行い、適切なインスタンスに値を設定する。同様に、44~53 行は式 22.78、22.79 または式 24.3 にもとづき表面温度を計算する処理である。55~85 行も同様であり、具体的な処理は省略する。94 行以降は表面温湿度を両側の温湿度から計算するための係数（式 22.82~22.86）であるが、これらに関しても壁・窓の別、F 側 B 側の別の判定はプロパティ内で行う。

プログラム 25.2　プロパティの定義

Popolo.ThermalLoad.wallWindowSurface class

```
 1 /// <summary>地中壁か否か</summary>
 2 public bool IsGroundWall { get; set; } = false;
 3
 4 /// <summary>隣室温度差係数[-]を取得する</summary>
 5 public double AdjacentSpaceFactor { get; set; } = -1.0;
 6
 7 /// <summary>傾斜面を設定・取得する</summary>
 8 public ImmutableIncline Incline { get; set; }
 9
10 /// <summary>属するゾーン番号を設定・取得する</summary>
11 public int ZoneIndex { get; set; }
12
13 /// <summary>裏側表面を取得する</summary>
14 public wallWindowSurface ReverseSideSurface
15 {
16   get
17   {
18     if (IsWall && isSideF) return Wall.SurfaceB;
19     else if (IsWall && !isSideF) return Wall.SurfaceF;
20     else if (!IsWall && isSideF) return Window.InsideSurface;
21     else return Window.OutsideSurface;
22   }
23 }
24
25 /// <summary>相当温度[C]を設定・取得する</summary>
26 public double SolAirTemperature
27 {
28   set
29   {
30     if (IsWall && isSideF) Wall.SolAirTemperatureF = value;
31     else if (IsWall && !isSideF) Wall.SolAirTemperatureB = value;
32     else if (!IsWall && isSideF) Window.SolAirTemperatureF = value;
33     else Window.SolAirTemperatureB = value;
34   }
35   get
36   {
37     if (IsWall && isSideF) return Wall.SolAirTemperatureF;
38     else if (IsWall && !isSideF) return Wall.SolAirTemperatureB;
39     else if (!IsWall && isSideF) return Window.SolAirTemperatureF;
40     else return Window.SolAirTemperatureB;
41   }
42 }
43
44 /// <summary>表面温度[C]を取得する</summary>
45 public double SurfaceTemperature
46 {
47   get
48   {
49     if (IsWall) return IF2 + FFS2 * SolAirTemperature + BFS2 * ReverseSideSurface.SolAirTemperature
50         + FFL2 * HumidityRatio + BFL2 * ReverseSideSurface.HumidityRatio;
51     else return FFS2 * SolAirTemperature + BFS2 * ReverseSideSurface.SolAirTemperature;
52   }
53 }
54
```

```
 55 /// <summary>絶対湿度[kg/kg]を設定・取得する</summary>
 56 public double HumidityRatio
 57 {  //省略//get,set を実装
 58
 59 /// <summary>表面積[m2]を取得する</summary>
 60 public double Area
 61 {  //省略//get を実装
 62
 63 /// <summary>総合熱伝達率[W/(m2K)]を取得する</summary>
 64 public double FilmCoefficient
 65 {  //省略//get を実装
 66
 67 /// <summary>放射熱伝達率[W/(m2K)]を設定・取得する</summary>
 68 public double RadiativeCoefficient
 69 {  //省略//get,set を実装
 70
 71 /// <summary>対流熱伝達率[W/(m2K)]を設定・取得する</summary>
 72 public double ConvectiveCoefficient
 73 {  //省略//get,set を実装
 74
 75 /// <summary>湿気伝達率[(kg/s)/((kg/kg)m2)]を取得する</summary>
 76 public double MoistureCoefficient
 77 {  //省略//get を実装
 78
 79 /// <summary>短波長放射率[-]を取得する</summary>
 80 public double ShortWaveEmissivity
 81 {  //省略//get を実装
 82
 83 /// <summary>長波長放射率[-]を取得する</summary>
 84 public double LongWaveEmissivity
 85 {  //省略//get を実装
 86
 87 /// <summary>総合熱伝達率に占める対流熱伝達率の比率[-]を取得する</summary>
 88 public double ConvectiveRate
 89 { get { return ConvectiveCoefficient / FilmCoefficient; } }
 90
 91 /// <summary>総合熱伝達率に占める放射熱伝達率の比率[-]を取得する</summary>
 92 public double RadiativeRate { get { return 1 - ConvectiveRate; } }
 93
 94 /// <summary>表側の相当温度の係数（温度）を取得する</summary>
 95 public double FFS2
 96 {
 97   get
 98   {
 99     if (IsWall && isSideF) return Wall.FFS2_F;
100     else if (IsWall && !isSideF) return Wall.BFS2_B;
101     else if (!IsWall && isSideF) throw new Exception("Not Implemented");
102     else return 1 - 1d / (Window.GetResistance() * Window.FilmCoefficientB);
103   }
104 }
105
106 //省略//同様の方法で FFS3, FFL2, FFL3, BFS2, BFS3, BFL2, BFL3, IF2, IF3 を実装
```

25.3.2　ゾーンクラスの作成

1 つの質点を持つ空間を表わす Zone クラスを開発する。

ゾーンには発熱要素があり、これを表現するためにプログラム 25.3 に示すように発熱要素インターフェースを定義する。式 25.46 に示したように発熱は顕熱対流成分 $HG_{S(c)}$、顕熱放射成分 $HG_{S(r)}$、潜熱発熱 HG_L に分けられるため、これらを計算するためのメソッドを 7, 12, 17 行で定義する。引数の ImmutableZone は後に定義する Zone クラスの読み取り専用インターフェースである。人体のようにゾーンの温湿度に応じて発熱量が変化するような発熱体もあるため、引数で現在のゾーンの情報を得られるようにする。

プログラム 25.3　発熱要素インターフェースの定義

Popolo.ThermalLoad.IHeatGain interface

```
 1 /// <summary>発熱要素インターフェース</summary>
 2 public interface IHeatGain
 3 {
 4   /// <summary>顕熱取得[W]の内、対流成分を取得する</summary>
 5   /// <param name="zone">発熱要素が属するゾーン</param>
 6   /// <returns>顕熱取得の対流成分[kW]</returns>
 7   double GetConvectiveHeatGain(ImmutableZone zone);
```

```
 8
 9   /// <summary>顕熱取得[W]の内、放射成分を取得する</summary>
10   /// <param name="zone">発熱要素が属するゾーン</param>
11   /// <returns>顕熱取得の放射成分[kW]</returns>
12   double GetRadiativeHeatGain(ImmutableZone zone);
13
14   /// <summary>発生水分[kg/s]を取得する</summary>
15   /// <param name="zone">発熱要素が属するゾーン</param>
16   /// <returns>発生水分[kg/s]</returns>
17   double GetWaterGain(ImmutableZone zone);
18 }
```

IHeatGain の実装例をプログラム 25.4 に示す。$HG_{S(c)}$、$HG_{S(r)}$、HG_L をプロパティで設定できるようにした簡単な発熱要素であり、境界条件ファイルなどから固定値で発熱量を与えたいような場合にはこの程度の処理で対応できる。例えば、任意の計算時間間隔に対応した発熱要素とするために日時データを参照しながらスケジュールで発熱量を計算したい場合や、確率的に発熱量を発生させたい場合などには、IHeatGain インターフェースを実装してメソッドの中身を作りこめば良い。

プログラム 25.4　発熱要素インターフェースの実装例

Popolo.ThermalLoad.SimpleHeatGain class

```
 1 /// <summary>発熱要素</summary>
 2 public class SimpleHeatGain : IHeatGain
 3 {
 4   /// <summary>顕熱対流熱取得[W]を設定・取得する</summary>
 5   public double ConvectiveHeatGain { get; set; }
 6
 7   /// <summary>顕熱放射熱取得[W]を設定・取得する</summary>
 8   public double RadiativeHeatGain { get; set; }
 9
10   /// <summary>発生水分[kg/s]を設定・取得する</summary>
11   public double WaterGain { get; set; }
12
13   /// <summary>顕熱取得[W]の内、対流成分を取得する</summary>
14   /// <param name="zone">発熱要素が属するゾーン</param>
15   /// <returns>顕熱取得の対流成分[kW]</returns>
16   public double GetConvectiveHeatGain(ImmutableZone zone){ return ConvectiveHeatGain; }
17
18   /// <summary>顕熱取得[W]の内、放射成分を取得する</summary>
19   /// <param name="zone">発熱要素が属するゾーン</param>
20   /// <returns>顕熱取得の放射成分[kW]</returns>
21   public double GetRadiativeHeatGain(ImmutableZone zone){ return RadiativeHeatGain; }
22
23   /// <summary>発生水分[kg/s]を取得する</summary>
24   /// <param name="zone">発熱要素が属するゾーン</param>
25   /// <returns>発生水分[kg/s]</returns>
26   public double GetWaterGain(ImmutableZone zone){ return WaterGain; }
27
28   /// <summary>インスタンスを初期化する</summary>
29   /// <param name="convectiveHeatGain">顕熱対流熱取得[W]</param>
30   /// <param name="radiativeHeatGain">顕熱放射熱取得[W]</param>
31   /// <param name="waterGain">発生水分[kg/s]</param>
32   public SimpleHeatGain (double convectiveHeatGain, double radiativeHeatGain, double waterGain)
33   {
34     ConvectiveHeatGain = convectiveHeatGain;
35     RadiativeHeatGain = radiativeHeatGain;
36     WaterGain = waterGain;
37   }
38 }
```

プログラム 25.5 に Zone クラスのインスタンス・プロパティ・コンストラクタを示す。2 行は先に解説した発熱要素のリストである。インターフェースである IHeatGain のリストとしているため、ユーザーは任意の発熱要素クラスを開発することができる。5~72 行はプロパティの定義であり、ゾーンに関係する換気量 $m_{a,(OA\text{-}q)}$、空気の質量 M_{ZN}、顕熱容量 C_{SFN}、水分容量 C_{LFN}、乾球温度 T_{ZN}、顕熱供給 HL_S、絶対湿度 W_{ZN}、水分供給 HL_L などである。75~84 行は internal プロパティである。75 行は 25.3.1 節で作成した wallWindowSurface クラスであり、ゾーンに面する壁窓表面を保持する。77~84 行は他のゾーンからの空気流入がある場合の流入空気の温湿度管理用の変数である。

プログラム **25.5**　**Zone** クラスのインスタンス変数・プロパティ・コンストラクタ

Popolo.ThermalLoad.Zone class

```
/// <summary>発熱要素リスト</summary>
private List<IHeatGain> heatGains = new List<IHeatGain>();

/// <summary>名称を設定・取得する</summary>
public string Name { get; private set; }

/// <summary>換気量[kg/s]を設定・取得する</summary>
public double VentilationRate { get; set; }

/// <summary>給気量[kg/s]を設定・取得する</summary>
public double SupplyAirFlowRate { get; set; }

/// <summary>空気の質量[kg]を取得する</summary>
public double AirMass { get; private set; }

/// <summary>所属する室番号を取得する</summary>
/// <remarks>いずれにも所属しない場合には-1</remarks>
public int RoomIndex { get; internal set; } = -1;

/// <summary>ゾーン番号を取得する</summary>
public int Index { get; internal set; }

/// <summary>所属する多数室を取得する</summary>
public ImmutableMultiRooms MultiRoom { get; internal set; }

/// <summary>空気以外の顕熱容量[J/K]を設定・取得する</summary>
public double HeatCapacity { get; set; }

/// <summary>空気以外の水分容量[kg]を設定・取得する</summary>
public double WaterCapacity { get; set; }

/// <summary>乾球温度[C]を取得する</summary>
public double Temperature { get; internal set; } = 24;

/// <summary>乾球温度設定値[C]を取得する</summary>
public double TemperatureSetpoint { get; private set; } = 24;

/// <summary>供給顕熱[W]を取得する</summary>
public double HeatSupply { get; internal set; }

/// <summary>乾球温度[C]を制御するか否か</summary>
public bool TemperatureControlled { get; internal set; }

/// <summary>給気温度[C]を設定・取得する</summary>
public double SupplyAirTemperature { get; set; }

/// <summary>絶対湿度[kg/kg]を取得する</summary>
public double HumidityRatio { get; internal set; } = 0.0095;

/// <summary>絶対湿度設定値[kg/kg]を取得する</summary>
public double HumidityRatioSetpoint { get; private set; } = 0.0095;

/// <summary>供給水分[kg/s]を取得する</summary>
public double WaterSupply { get; internal set; }

/// <summary>絶対湿度[kg/kg]を制御するか否か</summary>
public bool HumidityControlled { get; internal set; }

/// <summary>給気絶対湿度[kg/kg]を設定・取得する</summary>
public double SupplyAirHumidityRatio { get; set; }

/// <summary>加熱能力[W]を設定・取得する</summary>
public double HeatingCapacity { get; set; } = double.PositiveInfinity;

/// <summary>冷却能力[W]を設定・取得する</summary>
public double CoolingCapacity { get; set; } = double.PositiveInfinity;

/// <summary>加湿能力[kg/s]を設定・取得する</summary>
public double HumidifyingCapacity { get; set; } = double.PositiveInfinity;

/// <summary>除湿能力[kg/s]を設定・取得する</summary>
public double DehumidifyingCapacity { get; set; } = double.PositiveInfinity;

/// <summary>ゾーンに面する壁・窓表面リスト</summary>
internal List<wallWindowSurface> Surfaces { get; set; }

/// <summary>他のゾーンからの給気量[kg/s]を設定・取得する</summary>
internal double supplyAirFlowRate2 { get; set; }
```

```
79
80 /// <summary>他のゾーンからの給気温度[C]を設定・取得する</summary>
81 internal double supplyAirTemperature2 { get; set; }
82
83 /// <summary>他のゾーンからの給気絶対湿度[kg/kg]を設定・取得する</summary>
84 internal double supplyAirHumidityRatio2 { get; set; }
85
86 /// <summary>新しいインスタンスを初期化する</summary>
87 /// <param name="name">名称</param>
88 /// <param name="airMass">空気の質量[kg]</param>
89 public Zone(string name, double airMass)
90 {
91   Name = name;
92   AirMass = airMass;
93   Surfaces = new List<wallWindowSurface>();
94 }
```

プログラム 25.6 に温湿度設定・制御に関する処理を示す。1~8 行は温湿度の初期化処理である。10~24 行が顕熱の制御に関する処理であり、式 25.21 において T_{ZN} と HL_S のいずれを未知変数にするかを設定する。10~16 行は T_{ZN} を制御する場合、18~24 行は HL_S を制御する場合である。同様に 26~40 行は湿度の制御に関する処理であり、式 25.51 において W_{ZN} と HL_L のいずれを未知変数にするかを設定する。42~54 行はゾーンに含まれる壁や窓表面の面積重み付け平均温度を計算する処理である。ある点における平均放射温度は、正確には壁や窓との位置関係に依存するが、近似値としてはこのような重み付け平均値を用いても良い。

プログラム 25.6　温湿度設定・制御に関する処理

Popolo.ThermalLoad.Zone class

```
 1 /// <summary>温湿度を初期化する</summary>
 2 /// <param name="temperature">乾球温度[C]</param>
 3 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
 4 public void InitializeAirState(double temperature, double humidityRatio)
 5 {
 6   Temperature = temperature;
 7   HumidityRatio = humidityRatio;
 8 }
 9
10 /// <summary>乾球温度[C]を制御する</summary>
11 /// <param name="setpoint">乾球温度設定値[C]</param>
12 public void ControlDrybulbTemperature(double setpoint)
13 {
14   TemperatureControlled = true;
15   TemperatureSetpoint = setpoint;
16 }
17
18 /// <summary>顕熱供給[W]を制御する</summary>
19 /// <param name="heatSupply">顕熱供給[W]</param>
20 public void ControlHeatSupply(double heatSupply)
21 {
22   TemperatureControlled = false;
23   HeatSupply = heatSupply;
24 }
25
26 /// <summary>絶対湿度[kg/kg]を制御する</summary>
27 /// <param name="setpoint">絶対湿度設定値[kg/kg]</param>
28 public void ControlHumidityRatio(double setpoint)
29 {
30   HumidityControlled = true;
31   HumidityRatioSetpoint = setpoint;
32 }
33
34 /// <summary>水分供給[kg/s]を制御する</summary>
35 /// <param name="waterSupply">水分供給[kg/s]</param>
36 public void ControlWaterSupply(double waterSupply)
37 {
38   HumidityControlled = false;
39   WaterSupply = waterSupply;
40 }
41
42 /// <summary>平均表面温度を計算する</summary>
43 /// <returns>平均表面温度[C]</returns>
44 public double GetMeanSurfaceTemperature()
45 {
```

```
46   double tSum = 0;
47   double sSum = 0;
48   foreach (wallWindowSurface ws in Surfaces)
49   {
50     tSum += ws.SurfaceTemperature * ws.Area;
51     sSum += ws.Area;
52   }
53   return tSum / sSum;
54 }
```

プログラム 25.7 に発熱要素の設定・集計処理を示す。1~7 行は追加と削除に関する処理である。9~16 行、18~25 行、27~34 行では発熱要素を積み上げ、それぞれ顕熱対流成分、顕熱放射成分、水分の合計値を計算する。

プログラム 25.7　発熱要素の設定・集計処理

Popolo.ThermalLoad.Zone class

```
 1 /// <summary>発熱要素を追加する</summary>
 2 /// <param name="heatGain">発熱要素</param>
 3 public void AddHeatGain(IHeatGain heatGain){ heatGains.Add(heatGain); }
 4
 5 /// <summary>発熱要素を削除する</summary>
 6 /// <param name="heatGain">発熱要素</param>
 7 public void RemoveHeatGain(IHeatGain heatGain){ heatGains.Remove(heatGain); }
 8
 9 /// <summary>顕熱取得の対流成分[W]を積算する</summary>
10 /// <returns>顕熱取得の対流成分[W]</returns>
11 public double IntegrateConvectiveHeatgains()
12 {
13   double sum = 0;
14   foreach (IHeatGain hg in heatGains) sum += hg.GetConvectiveHeatGain(this);
15   return sum;
16 }
17
18 /// <summary>顕熱取得の放射成分[W]を積算する</summary>
19 /// <returns>顕熱取得の放射成分[W]</returns>
20 public double IntegrateRadiativeHeatGains()
21 {
22   double sum = 0;
23   foreach (IHeatGain hg in heatGains) sum += hg.GetRadiativeHeatGain(this);
24   return sum;
25 }
26
27 /// <summary>発生水分[kg/s]を積算する</summary>
28 /// <returns>発生水分[kg/s]</returns>
29 public double IntegrateWaterGains()
30 {
31   double sum = 0;
32   foreach (IHeatGain hg in heatGains) sum += hg.GetWaterGain(this);
33   return sum;
34 }
```

25.3.3　多数室クラスの作成

複数のゾーンが属する室の温湿度と、壁や窓の相互放射を考慮しながら多数室連成計算を行う多数室クラス MultiRooms を作成する。

1）インスタンスメンバの定義と初期化処理

プログラム 25.8 に定数とインスタンス変数の宣言を示す。

6 行は式 25.32 で放射熱伝達率を計算するための係数である。8~15 行に示すようにゾーン、壁、窓のリストを持ち、これらを操作して連成計算を行う。壁や窓が追加された場合には室の形態係数を更新する必要があり、18 行はこのフラグである。21 行はゾーン間換気 $m_{a,(q1\text{-}q2)}$、28 行はゲブハルトの吸収率行列（室の数×(長波長放射用と短波長放射用)）である。

壁は両側が連成計算の対象となる場合と、片側のみが連成計算の対象となる場合（裏側のゾーンが他の MultiRooms クラスのインスタンスによって扱われる場合や境界条件となっている場合など）があり、31 行は連成計算の対象とする表面のリストである。

34 行は窓面からの短波長を優先して落とすペリメータ床（式 25.45 の perimeter floor）を保持する変数であり、37 行はその比率 R_p である。44 行と 48 行は壁と窓表面に入射する長波長と短波長（式 25.48）である。

ブラインドがある場合などには、窓の日射吸収率は一定ではない。日射吸収率が変化すると短波長に関してのゲブハルトの吸収率行列を再計算する必要がある。このため 55 行で窓の短波長吸収率を保存しておき、変化の有無を確認して再計算の必要性を判断する。

MultiRooms クラスには 1 以上の室があり、ゾーンはそのいずれかの室に属する。58 行は各室にどのゾーンが属しているかを管理する変数である。61 行は室番号 i、ゾーン番号 j、表面番号 k から壁表面の通し番号 $s(i\text{-}j\text{-}k)$を得るための変数である。

一旦、式 25.22 と式 25.23 の[IT]と[JT]求めれば、式 25.26 を用いて T_{ZN} と HL_S のいずれを変数として取り扱うかを切り替えることができる。64 行は[IT]および[JT]が計算済みか否かを判定するフラグである。潜熱平衡の場合も同様に式 25.51 の[AW]および[BW]が計算済みか否かを 67 行のフラグで判定する。72~78 行は計算に用いる行列やベクトルの記憶領域である。

プログラム 25.8　定数とインスタンス変数の宣言

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>湿り空気の定圧比熱[J/(kgK)]</summary>
 2 /// <remarks>24C, 50%, 9.5g/kg</remarks>
 3 private const double AIR_SPECIFICHEAT = 1005 + 1846 * 0.0095;
 4
 5 /// <summary>放射熱伝達率計算用係数[W/(m2K)]</summary>
 6 private const double RAD_COEF = 4.0 * 5.67e-8 * 297.15 * 297.15 * 297.15;
 7
 8 /// <summary>ゾーンリスト</summary>
 9 private Zone[] zones;
10
11 /// <summary>壁リスト</summary>
12 private Wall[] walls;
13
14 /// <summary>窓リスト</summary>
15 private Window[] windows;
16
17 /// <summary>形態係数更新の要否</summary>
18 private bool needInitialize = true;
19
20 /// <summary>ゾーン間換気[kg/s]</summary>
21 private double[,] zoneVent;
22
23 /// <summary>各室の形態係数</summary>
24 private IMatrix[] formFactor;
25
26 /// <summary>ゲブハルトの吸収率行列（長波長は規準化）</summary>
27 /// <remarks>surfaces と同サイズの行列数</remarks>
28 private double[,] gMatL, gMatS;
29
30 /// <summary>室内に面する壁・窓表面リスト</summary>
31 private wallWindowSurface[] surfaces;
32
33 /// <summary>窓面からの短波長を優先的に割り振る床表面リスト</summary>
34 private Dictionary<Window, wallWindowSurface> swDistFloor;
35
36 /// <summary>窓面からの短波長を床に割り振る割合</summary>
37 private Dictionary<Window, double> swDistRate;
38
39 /// <summary>壁・窓表面裏側が境界条件か否か</summary>
40 private bool[] isSFboundary;
41
42 /// <summary>室内に面する壁・窓表面に入射する短波長放射[W/m2]リスト</summary>
43 /// <remarks>surfaces と同サイズ。窓に関しては吸収日射熱取得の値を保持</remarks>
44 private double[] radToSurf_S;
45
46 /// <summary>各室の壁表面に入射する長波長放射[W/m2]リスト</summary>
47 /// <remarks>RoomNumber と同サイズ</remarks>
48 private double[] radToSurf_L;
```

```
49
50 /// <summary>境界条件表面リスト</summary>
51 private List<wallWindowSurface> bndSurfaces = new List<wallWindowSurface>();
52
53 /// <summary>ガラスの短波長放射率リスト</summary>
54 /// <remarks>windows と同サイズ</remarks>
55 private double[] wSWEmissivity;
56
57 /// <summary>各室に属するゾーンの番号</summary>
58 private List<int>[] rZones;
59
60 /// <summary>壁・窓の通し番号。[室][ゾーン][壁・窓]</summary>
61 private int[][][] wsNumber;
62
63 /// <summary>顕熱平衡の予測計算中か否か</summary>
64 private bool forecastingHeatTransfer = false;
65
66 /// <summary>水分平衡の予測計算中か否か</summary>
67 private bool forecastingWaterTransfer = false;
68
69 /// <summary>ゾーン温湿度リスト（一時保存）</summary>
70 private double[] zoneTemp, zoneHumid;
71
72 /// <summary>顕熱平衡計算用行列</summary>
73 private IMatrix matA, matAInv, matB, matD, matF, matI, matK, matKInv, matBf;
74 private IVector vecC, vecEJ, vecTH, vecTWS;
75
76 /// <summary>水分平衡計算用行列</summary>
77 private IMatrix matAW, matCW, matCWInv;
78 private IVector vecWH;
```

プログラム 25.9 に MultiRooms クラスのプロパティ定義を示す。1~13 行は計算時間間隔であり、クラスに属する壁のタイムステップを同じ値に変更する。

プログラム 25.9　MultiRooms クラスのプロパティ定義

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>計算時間間隔[sec]</summary>
 2 private double timeStep = 3600;
 3
 4 /// <summary>計算時間間隔[sec]を設定・取得する</summary>
 5 public double TimeStep
 6 {
 7   get { return timeStep; }
 8   set
 9   {
10     for (int i = 0; i < walls.Length; i++) walls[i].TimeStep = value;
11     timeStep = value;
12   }
13 }
14
15 /// <summary>熱・水分同時移動モデルか否か</summary>
16 public bool SolveMoistureTransferSimultaneously { get; private set; }
17
18 /// <summary>現在の日時を設定する</summary>
19 public DateTime CurrentDateTime { get; private set; }
20
21 /// <summary>太陽を設定する</summary>
22 public ImmutableSun Sun { get; set; }
23
24 /// <summary>室数を取得する</summary>
25 public int RoomNumber { get; private set; }
26
27 /// <summary>ゾーン数を取得する</summary>
28 public int ZoneNumber { get; private set; }
29
30 /// <summary>ゾーンを取得する</summary>
31 public ImmutableZone[] Zones { get { return zones; } }
32
33 /// <summary>壁を取得する</summary>
34 public ImmutableWall[] Walls { get { return walls; } }
35
36 /// <summary>窓を取得する</summary>
37 public ImmutableWindow[] Windows { get { return windows; } }
38
39 /// <summary>外気乾球温度[C]を取得する</summary>
40 public double OutdoorTemperature { get; private set; }
41
42 /// <summary>外気絶対湿度[kg/kg]を取得する</summary>
43 public double OutdoorHumidityRatio { get; private set; }
```

```
44
45 /// <summary>夜間放射[W/m2]を取得する</summary>
46 public double NocturnalRadiation { get; private set; }
47
48 /// <summary>地表面反射率[-]を設定・取得する</summary>
49 public double Albedo { get; set; }
```

プログラム 25.10 に MultiRooms クラスのコンストラクタを示す。引数は部屋の数、ゾーン・壁・窓のリストであり、これらの数に合わせて各種のインスタンス変数の記憶領域を初期化する。ゾーン・壁・窓の番号はここで与えられた配列の順序とする。31~39 行は熱水分同時移動計算を行うか否かの判定であり、壁の一つでも熱水分同時移動に対応している場合には多数室連成計算も熱水分同時移動に対応させる。40~51 行は計算用行列の記憶領域初期化処理であるが、41 行に示すように、熱水分同時移動の場合には同時に解く変数が倍増する（温度のみ→温度+湿度）ことに注意する。

プログラム 25.10　コンストラクタ

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>新しいインスタンスを初期化する</summary>
 2 /// <param name="rmNumber">部屋の数</param>
 3 /// <param name="zones">ゾーンリスト</param>
 4 /// <param name="walls">壁リスト</param>
 5 /// <param name="windows">窓リスト</param>
 6 public MultiRooms(int rmNumber, Zone[] zones, Wall[] walls, Window[] windows)
 7 {
 8   RoomNumber = rmNumber;
 9   ZoneNumber = zones.Length;
10   this.walls = walls;
11   this.windows = windows;
12   this.zones = zones;
13   formFactor = new IMatrix[RoomNumber];
14   rZones = new List<int>[RoomNumber];
15   radToSurf_L = new double[RoomNumber];
16   wsNumber = new int[RoomNumber][];
17   zoneVent = new double[ZoneNumber, ZoneNumber];
18   wSWEmissivity = new double[windows.Length];
19   zoneTemp = new double[ZoneNumber];
20   zoneHumid = new double[ZoneNumber];
21   swDistFloor = new Dictionary<Window, wallWindowSurface>();
22   swDistRate = new Dictionary<Window, double>();
23   for (int i = 0; i < RoomNumber; i++) rZones[i] = new List<int>();
24   for (int i = 0; i < ZoneNumber; i++)
25   {
26     AddZone(0, i);
27     zones[i].MultiRoom = this;
28     zones[i].Index = i;
29   }
30
31   //熱水分同時移動判定//水分透過壁が１つでもあれば
32   foreach (Wall wl in walls)
33   {
34     if (wl.ComputeMoistureTransfer)
35     {
36       SolveMoistureTransferSimultaneously = true;
37       break;
38     }
39   }
40   int num = ZoneNumber;
41   if (SolveMoistureTransferSimultaneously) num *= 2;
42   matD = new Matrix(num, num);
43   vecEJ = new Vector(num);
44   matI = new Matrix(num, num);
45   matK = new Matrix(num, num);
46   matKInv = new Matrix(num, num);
47   vecTH = new Vector(num);
48   matAW = new Matrix(num, num);
49   matCW = new Matrix(num, num);
50   matCWInv = new Matrix(num, num);
51   vecWH = new Vector(num);
52 }
```

2）モデル構築・入力条件設定に関する処理

プログラム 25.11 にゾーン・壁・窓のモデル化に関する処理を示す。

1~10 行は室へのゾーン登録処理である。ゾーンは 1 つの室にしか属さないため、7 行で他の室への登録を解除した後、8 行で室への登録を行う。

12~27 行はゾーンへの壁表面の登録処理である。ゾーン番号、壁番号に加え、F 側か否かのフラグを引数とする。19~21 行で壁表面のインスタンスを取得し、23, 24 行で他のゾーンまたは境界条件リストへの登録を解除した後、25 行でゾーンへの登録を行う。29~37 行は壁の両側のゾーンにまとめて登録する場合の処理である。片側のゾーンが他の MultiRooms インスタンスに委ねられている場合や境界条件の場合以外には、このメソッドを使った方が良い。

39~56 行は壁表面を外壁表面として設定する処理である。外表面の場合には太陽との位置関係を計算する必要があるため、外表面の情報を持つ ImmutableIncline インスタンスを引数にとる。その他の処理はゾーンへの壁表面の登録と同様であり、55 行で境界条件壁表面リストに追加する。

58~76 行は壁表面を地中壁とする場合の処理である。一般の外表面と同様に境界条件として取り扱われることになるが、日射や外気温度に影響を受けず、地中温度と熱コンダクタンスから熱流を計算する。壁表面インスタンスの IsGroundWall プロパティを使って一般の外表面と区別する。

78~94 行は隣室温度差係数を用いる場合の設定である。相当温度は式 25.7 を用いて与えられるため、この場合にも壁表面は境界条件となる。

96~106 行はゾーンへの窓表面登録処理である。窓の場合は常に B 側を室内側とするため、F 側と B 側の判定フラグは引数としない。108~115 行は多数室に含まれる温湿度の初期化処理である。

プログラム 25.11　ゾーン・壁・窓のモデル化に関する処理

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>室にゾーンを追加する</summary>
 2 /// <param name="roomIndex">室番号</param>
 3 /// <param name="zoneIndex">ゾーン番号</param>
 4 public void AddZone(int roomIndex, int zoneIndex)
 5 {
 6   needInitialize = true;
 7   for (int i = 0; i < RoomNumber; i++) rZones[i].Remove(zoneIndex);
 8   rZones[roomIndex].Add(zoneIndex);
 9   zones[zoneIndex].RoomIndex = roomIndex;
10 }
11
12 /// <summary>ゾーンに壁を追加する</summary>
13 /// <param name="zoneIndex">ゾーン番号</param>
14 /// <param name="wallIndex">壁番号</param>
15 /// <param name="isSideF">F 側の壁表面か否か</param>
16 public void AddWall(int zoneIndex, int wallIndex, bool isSideF)
17 {
18   needInitialize = true;
19   wallWindowSurface sf;
20   if (isSideF) sf = walls[wallIndex].SurfaceF;
21   else sf = walls[wallIndex].SurfaceB;
22
23   bndSurfaces.Remove(sf);
24   for (int i = 0; i < ZoneNumber; i++) zones[i].Surfaces.Remove(sf);
25   zones[zoneIndex].Surfaces.Add(sf);
26   sf.ZoneIndex = zoneIndex;
27 }
28
29 /// <summary>ゾーンに壁を追加する</summary>
30 /// <param name="zoneFIndex">F 側ゾーン番号</param>
31 /// <param name="zoneBIndex">B 側ゾーン番号</param>
32 /// <param name="wallIndex">壁番号</param>
33 public void AddWall(int zoneFIndex, int zoneBIndex, int wallIndex)
34 {
35   AddWall(zoneFIndex, wallIndex, true);
36   AddWall(zoneBIndex, wallIndex, false);
37 }
38
39 /// <summary>屋外に壁を追加する</summary>
```

```
 40 /// <param name="wallIndex">壁番号</param>
 41 /// <param name="isSideF">F 側か否か</param>
 42 /// <param name="incline">外表面傾斜面</param>
 43 public void SetOutsideWall(int wallIndex, bool isSideF, ImmutableIncline incline)
 44 {
 45   needInitialize = true;
 46   wallWindowSurface ws;
 47   if (isSideF) ws = walls[wallIndex].SurfaceF;
 48   else ws = walls[wallIndex].SurfaceB;
 49
 50   ws.AdjacentSpaceFactor = -1.0;
 51   ws.Incline = incline;
 52   ws.ZoneIndex = -1;
 53   ws.IsGroundWall = false;
 54   for (int i = 0; i < ZoneNumber; i++) zones[i].Surfaces.Remove(ws);
 55   bndSurfaces.Add(ws);
 56 }
 57
 58 /// <summary>地中壁を追加する</summary>
 59 /// <param name="wallIndex">壁番号</param>
 60 /// <param name="isSideF">F 側か否か</param>
 61 /// <param name="groundWallConductance">土壌-壁表面間の熱コンダクタンス[W/(m2K)]</param>
 62 /// <remarks>地中壁表面を境界条件とする場合には熱コンダクタンスを大きくする</remarks>
 63 public void SetGroundWall(int wallIndex, bool isSideF, double groundWallConductance)
 64 {
 65   needInitialize = true;
 66   wallWindowSurface ws;
 67   if (isSideF) ws = walls[wallIndex].SurfaceF;
 68   else ws = walls[wallIndex].SurfaceB;
 69
 70   ws.ZoneIndex = -1;
 71   ws.IsGroundWall = true;
 72   ws.ConvectiveCoefficient = groundWallConductance;
 73   ws.RadiativeCoefficient = 0;
 74   for (int i = 0; i < ZoneNumber; i++) zones[i].Surfaces.Remove(ws);
 75   bndSurfaces.Add(ws);
 76 }
 77
 78 /// <summary>隣室温度差係数[-]を設定する</summary>
 79 /// <param name="wallIndex">壁番号</param>
 80 /// <param name="isSideF">F 側か否か</param>
 81 /// <param name="adjacentSpaceFactor">隣室温度差係数[-]</param>
 82 public void UseAdjacentSpaceFactor(int wallIndex, bool isSideF, double adjacentSpaceFactor)
 83 {
 84   needInitialize = true;
 85   wallWindowSurface ws;
 86   if (isSideF) ws = walls[wallIndex].SurfaceF;
 87   else ws = walls[wallIndex].SurfaceB;
 88
 89   ws.AdjacentSpaceFactor = adjacentSpaceFactor;
 90   ws.ZoneIndex = -1;
 91   ws.IsGroundWall = false;
 92   for (int i = 0; i < ZoneNumber; i++) zones[i].Surfaces.Remove(ws);
 93   bndSurfaces.Add(ws);
 94 }
 95
 96 /// <summary>ゾーンに窓を追加する</summary>
 97 /// <param name="zoneIndex">ゾーン番号</param>
 98 /// <param name="windowIndex">窓番号</param>
 99 public void AddWindow(int zoneIndex, int windowIndex)
100 {
101   needInitialize = true;
102   Window win = windows[windowIndex];
103   for (int i = 0; i < ZoneNumber; i++) zones[i].Surfaces.Remove(win.InsideSurface);
104   zones[zoneIndex].Surfaces.Add(win.InsideSurface);
105   win.InsideSurface.ZoneIndex = zoneIndex;
106 }
107
108 /// <summary>壁とゾーンの温湿度を初期化する</summary>
109 /// <param name="temperature">乾球温度[C]</param>
110 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
111 public void InitializeAirState(double temperature, double humidityRatio)
112 {
113   foreach (Zone zn in zones) zn.InitializeAirState(temperature, humidityRatio);
114   foreach (Wall wl in walls) wl.Initialize(temperature);
115 }
```

プログラム 25.12 に境界条件の設定処理を示す。

1~15 行は外気条件設定処理であり、時刻、太陽（法線面直達日射、水平面全天日射、水平面天空

日射）、外気温度 T_{OA}、外気湿度 W_{OA}、夜間放射 R_N を設定する。17~27 行は地中温度の設定処理である。地中温度は壁体の埋設深さによって異なるため、壁表面毎に設定可能とする。

29~34 行はゾーン間換気の設定処理である。2 ゾーン間の交換のみを考慮し、$m_{a,(q1\text{-}q2)} = m_{a,(q2\text{-}q1)}$ とする場合にはこのメソッドを用いる。複数のゾーン間で空気が順に流れていくことなどを計算する場合には 36~42 行のメソッドを用いて個別に設定を行う。44~56 行は窓からの短波長を落とすペリメータ床の設定処理である。窓番号、ペリメータ床の番号、F 側か B 側かの判別フラグ、係数 R_p の値、が引数である。58~64 行は埋設配管がある場合の送水条件（流量と温度）設定処理である。

プログラム 25.12　境界条件の設定処理

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>外気条件を更新する</summary>
 2 /// <param name="dTime">現在の日時</param>
 3 /// <param name="sun">太陽</param>
 4 /// <param name="temperature">外気乾球温度[C]</param>
 5 /// <param name="humidityRatio">外気絶対湿度[kg/kg]</param>
 6 /// <param name="nocRadiation">夜間放射[W/m2]</param>
 7 internal void UpdateOutdoorCondition
 8   (DateTime dTime, ImmutableSun sun, double temperature, double humidityRatio, double nocRadiation)
 9 {
10   CurrentDateTime = dTime;
11   Sun = sun;
12   OutdoorTemperature = temperature;
13   OutdoorHumidityRatio = humidityRatio;
14   NocturnalRadiation = nocRadiation;
15 }
16
17 /// <summary>地中温度[C]を設定する</summary>
18 /// <param name="wallIndex">壁番号</param>
19 /// <param name="isSideF">F 側か否か</param>
20 /// <param name="groundTemperature">地中温度[C]</param>
21 public void SetGroundTemperature(int wallIndex, bool isSideF, double groundTemperature)
22 {
23   wallWindowSurface ws;
24   if (isSideF) ws = walls[wallIndex].SurfaceF;
25   else ws = walls[wallIndex].SurfaceB;
26   if (bndSurfaces.Contains(ws)) ws.SolAirTemperature = groundTemperature;
27 }
28
29 /// <summary>ゾーン間換気量[kg/s]を設定する</summary>
30 /// <param name="zoneIndex1">ゾーン番号 1</param>
31 /// <param name="zoneIndex2">ゾーン番号 2</param>
32 /// <param name="airMassFlowRate">ゾーン間換気量[kg/s]</param>
33 public void SetCrossVentilation(int zoneIndex1, int zoneIndex2, double airMassFlowRate)
34 { zoneVent[zoneIndex1, zoneIndex2] = zoneVent[zoneIndex2, zoneIndex1] = airMassFlowRate; }
35
36 /// <summary>ゾーン 1 からゾーン 2 へ向かう換気量[kg/s]を設定する</summary>
37 /// <param name="zoneIndex1">ゾーン番号 1</param>
38 /// <param name="zoneIndex2">ゾーン番号 2</param>
39 /// <param name="airMassFlowRate">換気量[kg/s]</param>
40 /// <remarks>他のゾーンを含め風量収支を合わせる必要あり</remarks>
41 public void SetAirFlow(int zoneIndex1, int zoneIndex2, double airMassFlowRate)
42 { zoneVent[zoneIndex1, zoneIndex2] = airMassFlowRate; }
43
44 /// <summary>窓からの短波長（直達）を優先的に割り振る床を設定する</summary>
45 /// <param name="windowIndex">窓番号</param>
46 /// <param name="wallIndex">壁（床）番号</param>
47 /// <param name="isSideF">F 側か否か</param>
48 /// <param name="distRate">短波長（直達）分配比率[-]</param>
49 public void SetSWDistributionRateToFloor(int windowIndex, int wallIndex, bool isSideF, double distRate)
50 {
51   wallWindowSurface ws;
52   if (isSideF) ws = walls[wallIndex].SurfaceF;
53   else ws = walls[wallIndex].SurfaceB;
54   swDistFloor[windows[windowIndex]] = ws;
55   swDistRate[windows[windowIndex]] = distRate;
56 }
57
58 /// <summary>送水条件を設定する</summary>
59 /// <param name="wallIndex">壁（床）番号</param>
60 /// <param name="mNumber">質点番号</param>
61 /// <param name="flowRate">通水量[kg/s]</param>
62 /// <param name="temperature">水温[C]</param>
```

```
63 public void SetBuriedPipeWaterState(int wallIndex, int mNumber, double flowRate, double temperature)
64 { walls[wallIndex].SetInletWater(mNumber, flowRate, temperature); }
```

プログラム 25.13 に温湿度制御関連の処理を示す。引数で与えられるゾーン番号を手がかりに、Zone クラスの制御関連のメソッドを呼び出す。25~47 行はゾーンに供給される空気の温湿度設定処理である。他のゾーンからの流入空気の状態はゾーン間換気の設定値にもとづいてプログラムで自動計算するため、internal プロパティとする（37~47 行）。

プログラム 25.13　温湿度制御関連の処理

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>乾球温度[C]を制御する</summary>
 2 /// <param name="zoneIndex">ゾーン番号</param>
 3 /// <param name="setpoint">乾球温度設定値[C]</param>
 4 public void ControlDrybulbTemperature(int zoneIndex, double setpoint)
 5 { zones[zoneIndex].ControlDrybulbTemperature(setpoint); }
 6
 7 /// <summary>顕熱供給[W]を制御する</summary>
 8 /// <param name="zoneIndex">ゾーン番号</param>
 9 /// <param name="heatSupply">顕熱供給[W]</param>
10 public void ControlHeatSupply(int zoneIndex, double heatSupply)
11 { zones[zoneIndex].ControlHeatSupply(heatSupply); }
12
13 /// <summary>絶対湿度[kg/kg]を制御する</summary>
14 /// <param name="zoneIndex">ゾーン番号</param>
15 /// <param name="setpoint">絶対湿度設定値[kg/kg]</param>
16 public void ControlHumidityRatio(int zoneIndex, double setpoint)
17 { zones[zoneIndex].ControlHumidityRatio(setpoint); }
18
19 /// <summary>水分供給[kg/s]を制御する</summary>
20 /// <param name="zoneIndex">ゾーン番号</param>
21 /// <param name="waterSupply">水分供給[kg/s]</param>
22 public void ControlWaterSupply(int zoneIndex, double waterSupply)
23 { zones[zoneIndex].ControlWaterSupply(waterSupply); }
24
25 /// <summary>給気条件を設定する</summary>
26 /// <param name="zoneIndex">ゾーン番号</param>
27 /// <param name="saTemperature">給気温度[C]</param>
28 /// <param name="saHumidityRatio">給気絶対湿度[kg/kg]</param>
29 /// <param name="saFlowRate">給気量[kg/s]</param>
30 public void SetSupplyAir(int zoneIndex, double saTemperature, double saHumidityRatio, double saFlowRate)
31 {
32   zones[zoneIndex].SupplyAirTemperature = saTemperature;
33   zones[zoneIndex].SupplyAirHumidityRatio = saHumidityRatio;
34   zones[zoneIndex].SupplyAirFlowRate = saFlowRate;
35 }
36
37 /// <summary>給気条件 2 を設定する</summary>
38 /// <param name="zoneIndex">ゾーン番号</param>
39 /// <param name="saTemperature">給気温度[C]</param>
40 /// <param name="saHumidityRatio">給気絶対湿度[kg/kg]</param>
41 /// <param name="saFlowRate">給気量[kg/s]</param>
42 internal void setSupplyAir(int zoneIndex, double saTemperature, double saHumidityRatio, double saFlowRate)
43 {
44   zones[zoneIndex].supplyAirTemperature2 = saTemperature;
45   zones[zoneIndex].supplyAirHumidityRatio2 = saHumidityRatio;
46   zones[zoneIndex].supplyAirFlowRate2 = saFlowRate;
47 }
```

プログラム 25.14 に初期化処理を示す。壁の追加・削除など、初期化処理が必要と判定される場合には 6~21 行を実行する。

6 行で壁窓表面に通し番号を付与する。通し番号付与の詳細は 25~68 行である。30 行、33 行、37 行で室番号 i、ゾーン番号 j、壁表面番号 k について繰り返し計算を行い、式 25.2 を用いて 39 行で通し番号を付与する。通し番号を計算すれば壁窓表面の総数 NS が明らかになるため、これを用いて 45~64 行で計算用行列・ベクトルの記憶領域を初期化する。また、計算を行う壁窓表面の裏側表面が計算対象ではない場合には、当該表面を境界条件として扱う（式 25.9~25.11での扱いが変わる（49 行））。

16 行で形態係数の計算を行うため、12~14 行で各室に含まれる壁窓表面の面積配列を作成し、引数としてこれを与える。さらに 20 行で形態係数を用いてゲブハルトの吸収率行列を計算する。

プログラム 25.14　初期化処理

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>初期化処理</summary>
 2 private void initialize()
 3 {
 4   if (needInitialize)
 5   {
 6     makeSerialNumber(); //通し番号付与
 7
 8     //形態係数の計算（自動推定）
 9     for (int i = 0; i < RoomNumber; i++)
10     {
11       List<double> area = new List<double>();
12       for (int j = 0; j < wsNumber[i].Length; j++)
13         for (int k = 0; k < wsNumber[i][j].Length; k++)
14           area.Add(surfaces[wsNumber[i][j][k]].Area);
15
16       formFactor[i] = computeFormFactor(area.ToArray());
17       int wsn = area.Count;
18     }
19     //ゲブハルトの吸収率行列更新
20     computeGebhartMatrix();
21     needInitialize = false;
22   }
23 }
24
25 /// <summary>壁・窓表面に通し番号を振る</summary>
26 private void makeSerialNumber()
27 {
28   List<wallWindowSurface> sfs = new List<wallWindowSurface>();
29   int wsIndex = 0;
30   for (int i = 0; i < RoomNumber; i++)
31   {
32     wsNumber[i] = new int[rZones[i].Count][];
33     for (int j = 0; j < wsNumber[i].Length; j++)
34     {
35       Zone zn = zones[rZones[i][j]];
36       wsNumber[i][j] = new int[zn.Surfaces.Count];
37       for (int k = 0; k < wsNumber[i][j].Length; k++)
38       {
39         sfs.Add(zn.Surfaces[k]);
40         zn.Surfaces[k].Index = wsNumber[i][j][k] = wsIndex;
41         wsIndex++;
42       }
43     }
44   }
45   surfaces = sfs.ToArray();
46   int sNum = surfaces.Length;
47   int zNum = ZoneNumber;
48   isSFboundary = new bool[sNum];
49   for (int i = 0; i < sfs.Count; i++) isSFboundary[i] = !sfs.Contains(sfs[i].ReverseSideSurface);
50   radToSurf_S = new double[sNum];
51   gMatL = new double[sNum, sNum];
52   gMatS = new double[sNum, sNum];
53   if (SolveMoistureTransferSimultaneously)
54   {
55     sNum *= 2;
56     zNum *= 2;
57   }
58   matA = new Matrix(sNum, sNum);
59   matAInv = new Matrix(sNum, sNum);
60   matB = new Matrix(sNum, zNum);
61   vecC = new Vector(sNum);
62   matF = new Matrix(zNum, sNum);
63   vecTWS = new Vector(sNum);
64   matBf = new Matrix(zNum, sNum);
65 }
```

3）多重放射に関する処理

プログラム 25.15 に形態係数およびゲブハルトの吸収率行列の計算を示す。

1~82 行が形態係数の計算である。6~12 行は 2 つしか面が無い特殊なケースであり、この場合には

$F_{0,1}$=$F_{1,0}$=1.0 となる。図 25.4 で解説した松尾の方法により自動計算を行う。まず、円に内接する多角形を一意に定めるため、14~17 行で表面積を昇順に並べ替える。

23~33 行は特殊なケースであり、1 つの表面積が他のすべての表面積の和よりも大きい場合である。この場合には多角形を形成することができないため、29 行に示すようにすべての面から最も大きな表面への形態係数を 1.0 とし、最も大きな表面からその他の面への形態係数は式 25.38 に示した相反法則により求める（30 行）。さらに最も大きな面について式 25.37 の総和速が成立するように自分自身への形態係数を計算する（31 行）。

36~53 行は円の半径 R の収束計算処理であり、38~48 行で式 25.41、式 25.42 を用いた誤差関数を定義する。得られた半径 R と式 25.43 を用いて 55~72 行で形態係数を計算する。75~79 行で昇順に並べた配列をもとの順番に戻す。

84~136 行はゲブハルトの吸収率行列の計算である。短波長放射用と長波長放射用と 2 種類の計算が必要である。99~112 行で式 25.33 の（[I]-[ρ][F]）を作成し、113~116 行で逆行列を求めて形態係数[F]を乗じる。さらに 120~125 行で放射率対角行列[ε_{LW}]を乗じて吸収率行列を計算する。長波長放射に関しては 128 行で式 25.32 により放射熱伝達率を計算する。また、式 25.31 を適用して 131 行で基準化したゲブハルトの吸収係数 $\phi_{s1,s2}$ に変換する。

プログラム 25.15　形態係数およびゲブハルトの吸収率行列の計算

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>表面積から形態係数を推定する</summary>
 2 /// <param name="area">表面積リスト</param>
 3 /// <returns>形態係数行列</returns>
 4 private static IMatrix computeFormFactor(double[] area)
 5 {
 6   if (area.Length == 2)
 7   {
 8     IMatrix mat = new Matrix(2, 2);
 9     mat[0, 0] = mat[1, 1] = 0.0;
10     mat[0, 1] = mat[1, 0] = 1.0;
11     return mat;
12   }
13
14   //昇順に並べ替え
15   int[] index = new int[area.Length];
16   for (int i = 0; i < index.Length; i++) index[i] = i;
17   Array.Sort(area, index);
18
19   double[,] ff = new double[area.Length, area.Length];
20   double sA = 0;
21   int last = area.Length - 1;
22   for (int i = 0; i < last; i++) sA += area[i];
23   if (sA < area[last])
24   {
25     //最も大きい面積が他の合算よりも大きい場合
26     ff[last, last] = 1.0;
27     for (int i = 0; i < last; i++)
28     {
29       ff[i, last] = 1.0;
30       ff[last, i] = area[i] / area[last];
31       ff[last, last] -= ff[last, i];
32     }
33   }
34   else
35   {
36     //仮想的な半径を収束計算
37     double[] alpha = new double[area.Length];
38     Roots.ErrorFunction eFnc = delegate (double r)
39     {
40       double sum = 0;
41       for (int i = 0; i < area.Length; i++)
42       {
43         double bf = (area[i] / r);
```

```
44          alpha[i] = Math.Acos(Math.Max(-1, 1 - 0.5 * bf * bf));
45          sum += alpha[i];
46        }
47        return sum - 2 * Math.PI;
48      };
49      //面積和=円周と見立てて収束計算の初期値を決定
50      double sumA = 0;
51      for (int i = 0; i < area.Length; i++) sumA += area[i];
52      double rad = Roots.NewtonBisection (eFnc, 0.5 * (sumA / Math.PI), 0.0001, 0.01, 0.01, 10);
53      rad = Roots.Newton(eFnc, rad, 0.00001, 0.00001, 0.00001, 10);
54
55      //形態係数を計算
56      for (int i = 0; i < area.Length - 1; i++)
57      {
58        ff[i, i] = 0;
59        for (int j = i + 1; j < area.Length; j++)
60        {
61          double a1 = alpha[i];
62          double a2 = alpha[j];
63          double a3 = 0;
64          for (int k = i + 1; k < j; k++) a3 += alpha[k];
65          double d1 = 2 * rad * Math.Sin(a3 / 2);
66          double d2 = 2 * rad * Math.Sin(Math.PI - (a1 + a2 + a3) / 2);
67          double d3 = 2 * rad * Math.Sin((a2 + a3) / 2);
68          double d4 = 2 * rad * Math.Sin((a1 + a3) / 2);
69          ff[i, j] = (d3 + d4 - d1 - d2) / (2 * area[i]);
70          ff[j, i] = ff[i, j] * area[i] / area[j];
71        }
72      }
73    }
74
75    //順序を戻す
76    IMatrix ff2 = new Matrix(area.Length, area.Length);
77    for (int i = 0; i < area.Length; i++)
78      for (int j = 0; j < area.Length; j++)
79        ff2[index[i], index[j]] = ff[i, j];
80
81    return ff2;
82  }
83
84  /// <summary>ゲブハルト吸収率行列を計算する</summary>
85  private void computeGebhartMatrix()
86  {
87    for (int i = 0; i < RoomNumber; i++)
88    {
89      //壁番号を保存
90      List<int> wInd = new List<int>();
91      for (int j = 0; j < rZones[i].Count; j++)
92        for (int k = 0; k < wsNumber[i][j].Length; k++)
93          wInd.Add(wsNumber[i][j][k]);
94
95      int wsn = wInd.Count;
96      IMatrix ffRhoL = new Matrix(wsn, wsn);
97      IMatrix ffRhoS = new Matrix(wsn, wsn);
98      IMatrix ffRhoInv = new Matrix(wsn, wsn);
99      for (int j = 0; j < wsn; j++)
100     {
101       wallWindowSurface ws = surfaces[wInd[j]];
102       for (int k = 0; k < wsn; k++)
103       {
104         ffRhoL[j, k] = -(1 - ws.LongWaveEmissivity) * formFactor[i][j, k];
105         ffRhoS[j, k] = -(1 - ws.ShortWaveEmissivity) * formFactor[i][j, k];
106         if (j == k)
107         {
108           ffRhoL[j, k]++;
109           ffRhoS[j, k]++;
110         }
111       }
112     }
113     LinearAlgebra.GetInverse(ref ffRhoL, ref ffRhoInv);
114     LinearAlgebra.Multiplicate(formFactor[i], ffRhoInv, ref ffRhoL);
115     LinearAlgebra.GetInverse(ref ffRhoS, ref ffRhoInv);
116     LinearAlgebra.Multiplicate(formFactor[i], ffRhoInv, ref ffRhoS);
117     for (int j = 0; j < wsn; j++)
118     {
119       wallWindowSurface ws1 = surfaces[wInd[j]];
120       for (int k = 0; k < wsn; k++)
121       {
122         wallWindowSurface ws2 = surfaces[wInd[k]];
123         gMatL[wInd[j], wInd[k]] = ffRhoL[j, k] * ws2.LongWaveEmissivity;
```

```
124         gMatS[wInd[j], wInd[k]] = ffRhoS[j, k] * ws2.ShortWaveEmissivity;
125       }
126       //長波長は基準化して放射熱伝達率を計算
127       double bf = 1 - gMatL[wInd[j], wInd[j]];
128       ws1.RadiativeCoefficient = bf * ws1.LongWaveEmissivity * RAD_COEF;
129       for (int k = 0; k < wsn; k++)
130       {
131         if (j != k) gMatL[wInd[j], wInd[k]] /= bf;
132         else gMatL[wInd[j], wInd[k]] = 0;
133       }
134     }
135   }
136 }
```

プログラム 25.16 に短波長・長波長分配関連の計算を示す。

1~15 行は長波長の分配であり、各室の顕熱発熱放射成分を 8~12 行で積算し、式 25.47 を用いて 13 行で面積按分する。

17~31 行は短波長分配に先立って行う、窓面光学特性の更新処理である。21~29 行ですべての窓について光学特性を再計算し、wSWEmissivity に保存していた短波長吸収率と異なる値となった場合には、30 行でゲブハルトの吸収率行列を更新する。

33~93 行は短波長の分配処理である。37 行、39 行、41 行の繰り返し処理ですべての窓面を洗い出し、式 25.45 と式 25.48 を用いて透過日射と吸収日射熱取得を計算する。48~51 行で窓面に入射する直達日射 I_D と拡散日射 I_{SR} を計算する。52 行は屋外から窓面に入射する日射に対する吸収日射熱取得の計算である（式 24.20 右辺第 1 項）。57 行で直達成分の透過日射を計算し、優先的に割り振るペリメータ床が設定されている場合には 59~66 行で、R_p の比率を床に割り振る（式 25.45、式 25.48）。以上により窓面からの短波長放射が rad に、床面で反射する短波長放射が radFromFloor に代入されるため、69~88 行でこれらの短波長放射を同一室に属する他の壁窓表面に分配する。78 行と 80 行で窓面およびペリメータ床面からの放射にゲブハルトの吸収率行列を乗じ、多重反射後に各面に入射する短波長放射 I_{BSW} を計算する。入射する面が窓表面であった場合には、式 24.21 を適用し（83 行、84 行）、室内側からの日射に対する吸収日射熱取得 Q_{WaB} を計算する。

プログラム 25.16　短波長・長波長分配関連の計算

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>長波長放射を分配する</summary>
 2 private void distributeLongwaveRad()
 3 {
 4   for (int i = 0; i < RoomNumber; i++)
 5   {
 6     radToSurf_L[i] = 0;
 7     double saSum = 0;
 8     for (int j = 0; j < rZones[i].Count; j++)
 9     {
10       radToSurf_L[i] += zones[rZones[i][j]].IntegrateRadiativeHeatGains();
11       for (int k = 0; k < wsNumber[i][j].Length; k++)  saSum += surfaces[wsNumber[i][j][k]].Area;
12     }
13     radToSurf_L[i] /= saSum;
14   }
15 }
16
17 /// <summary>窓の光学特性を更新</summary>
18 private void updateWindowOpticalProperties()
19 {
20   bool needUpdateGebhartMatrix = false;
21   for (int i = 0; i < windows.Length; i++)
22   {
23     windows[i].UpdateOpticalProperties(Sun);
24     if (wSWEmissivity[i] != windows[i].ShortWaveEmissivityB)
25     {
26       wSWEmissivity[i] = windows[i].ShortWaveEmissivityB;
27       needUpdateGebhartMatrix = true;
```

```
28     }
29   }
30   if (needUpdateGebhartMatrix) computeGebhartMatrix();
31 }
32
33 /// <summary>短波長放射を分配する</summary>
34 private void distributeShortwaveRad()
35 {
36   //窓面からの日射を各表面に分配
37   for (int i = 0; i < RoomNumber; i++)
38   {
39     for (int j = 0; j < wsNumber[i].Length; j++)
40     {
41       for (int k = 0; k < wsNumber[i][j].Length; k++)
42       {
43         wallWindowSurface ws1 = surfaces[wsNumber[i][j][k]];
44         if (!ws1.IsWall)
45         {
46           int indx1 = wsNumber[i][j][k];
47           //窓からの透過・吸収日射を計算
48           Window win = ws1.Window;
49           ImmutableIncline inc = win.OutsideIncline;
50           double dir = inc.GetDirectSolarIrradiance(Sun) * (1 - win.SunShade.GetShadowRate(Sun));
51           double dif = inc.GetDiffuseSolarIrradiance(Sun, Albedo);
52           radToSurf_S[indx1] +=
53             dir * win.DirectSolarIncidentAbsorptance + dif * win.DiffuseSolarIncidentAbsorptance;
54           //床に優先配分される短波長を計算
55           wallWindowSurface flr = null;
56           double flRate = 0.0;
57           double dir2 = dir * win.DirectSolarIncidentTransmittance * win.Area;
58           double radFromFloor = 0.0;
59           if (swDistFloor.ContainsKey(win))
60           {
61             flr = swDistFloor[win];
62             flRate = swDistRate[win];
63             radToSurf_S[flr.Index] += dir2 * flRate * flr.ShortWaveEmissivity / flr.Area;
64             radFromFloor = dir2 * flRate * (1.0 - flr.ShortWaveEmissivity);
65             dir2 *= (1 - flRate);
66           }
67           double rad = dir2 + dif * win.DiffuseSolarIncidentTransmittance * win.Area;
68
69           //同じ室に属する壁表面に分配
70           if (0 < rad)
71           {
72             for (int j2 = 0; j2 < wsNumber[i].Length; j2++)
73             {
74               for (int k2 = 0; k2 < wsNumber[i][j2].Length; k2++)
75               {
76                 int indx2 = wsNumber[i][j2][k2];
77                 wallWindowSurface wsf2 = surfaces[indx2];
78                 double ibsw = gMatS[indx1, indx2] * rad;
79                 //床面からの放射を加算
80                 if (flr != null) ibsw += gMatS[flr.Index, indx2] * radFromFloor;
81                 ibsw /= wsf2.Area;
82                 if (!wsf2.IsWall)
83                   ibsw *= wsf2.Window.DiffuseSolarIncidentAbsorptance
84                     / (1 - wsf2.Window.DiffuseSolarIncidentReflectance);
85                 radToSurf_S[indx2] += ibsw;
86               }
87             }
88           }
89         }
90       }
91     }
92   }
93 }
```

4）顕熱平衡（熱水分同時移動を含む）の計算

顕熱平衡計算の全体フローを図 25.6 に示す。

まず、入力条件から[*AT*]~[*JT*]を計算し、壁窓表面温度 T_{WS} を消去する。温度 T_{ZN} か顕熱負荷 HL_S のいずれを境界条件にするかを指定する。ForecastHeatTransfer メソッドを呼び、タイムステップ経過後の顕熱平衡を予測する。T_{ZN} と HL_S の値を確認し、問題なければ FixHeatTranfer メソッドを呼び、現在の温度 T_{ZN} を確定する。温度 T_{ZN} が上下限温度に達してしまったり、顕熱負荷 HL_S が設備容量を上回っ

てしまったりして、境界条件を変更する必要がある場合には、変更後に再度 ForecastHeatTransfer を呼び出す。無限大の設備容量を持つことを仮定するなど、問答無用に温度と顕熱負荷に更新をかける場合には、これらの全処理をまとめて実行する UpdateHeatTransfer メソッドを呼び出す。

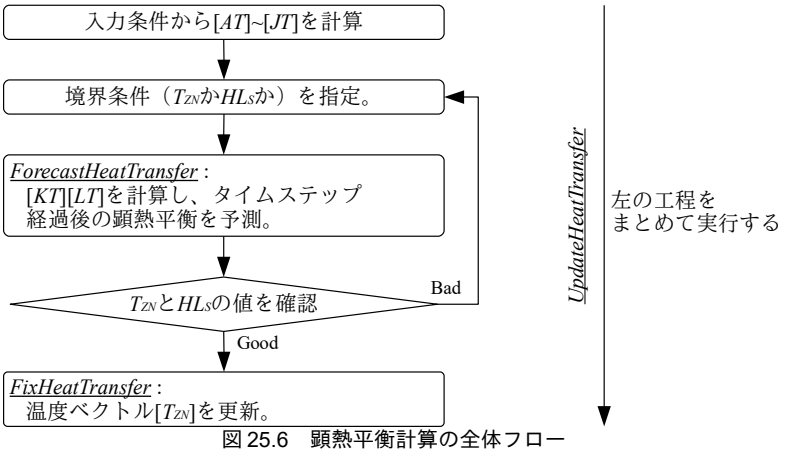


図 25.6　顕熱平衡計算の全体フロー

プログラム 25.17 に顕熱平衡予測のための準備計算を示す。すでに準備計算が実行されており、[*AT*]~[*JT*]が計算済の場合には 5 行で 37~48 行のメソッドを呼び出して温湿度を元に戻す。準備計算未了の場合には 6~34 行を実行する。8~13 行で現在の温湿度を保存し、計算後の取り消しに備える。15~19 行は初期化処理、窓の光学特性更新処理、長波長・短波長分配処理であり、既にプログラム 25.14~25.16で解説した。28 行は屋外温度などの境界条件設定処理であり、詳細は 50~85 行である。30~32 行は[*AT*][*BT*][*CT*]の計算処理であり、後述する。54~61 行は窓面の境界条件設定であり、式 24.2 である。窓面の相当温度の計算の際には、吸収日射熱取得による補正も行うため、24 行の短波長放射分配処理後に呼び出す必要があることに注意する。64~84 行は壁の境界条件設定である。67~74 行は一般外壁の相当温度計算であり式 22.1 である。76~82 行は隣室温度差係数を用いる場合の処理であり、式 25.7 である。

プログラム 25.17　顕熱平衡予測のための準備計算

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>熱平衡予測のための準備計算を行う</summary>
 2 private void prepareForHeatTransfer()
 3 {
 4   //温湿度を復旧
 5   if (forecastingHeatTransfer) ResetAirState();
 6   else
 7   {
 8     //温湿度を一時保存
 9     for (int i = 0; i < ZoneNumber; i++)
10     {
11       zoneTemp[i] = zones[i].Temperature;
12       zoneHumid[i] = zones[i].HumidityRatio;
13     }
14
15     //ゾーン・表面通し番号付与、形態係数行列初期化
16     initialize();
17
18     //窓の光学特性を更新、ゲブハルト吸収率行列初期化
19     updateWindowOpticalProperties();
20
21     //長波長・短波長放射を分配
22     for (int i = 0; i < radToSurf_S.Length; i++) radToSurf_S[i] = 0;
23     for (int i = 0; i < RoomNumber; i++) radToSurf_L[i] = 0;
24     distributeShortwaveRad();
25     distributeLongwaveRad();
26
```

```
27     //屋外側相当温度を設定
28     setOutdoorAirState();
29
30     //ABC 行列を計算
31     makeABMatrix();
32     makeCVector();
33     forecastingHeatTransfer = true;
34   }
35 }
36
37 /// <summary>ゾーンの温湿度を将来予測値から現在値に戻す</summary>
38 internal void ResetAirState()
39 {
40   if (forecastingHeatTransfer)
41   {
42     for (int i = 0; i < ZoneNumber; i++)
43     {
44       zones[i].Temperature = zoneTemp[i];
45       zones[i].HumidityRatio = zoneHumid[i];
46     }
47   }
48 }
49
50 /// <summary>外気条件を設定する</summary>
51 /// <remarks>窓面吸収日射計算後に呼び出す必要あり</remarks>
52 private void setOutdoorAirState()
53 {
54   foreach (Window win in windows)
55   {
56     wallWindowSurface ws = win.OutsideSurface;
57     double fs = win.OutsideIncline.ConfigurationFactorToSky;
58     ws.SolAirTemperature = OutdoorTemperature
59       + radToSurf_S[win.InsideSurface.Index] * win.GetResistance()
60       - ws.LongWaveEmissivity * fs * NocturnalRadiation / ws.FilmCoefficient;
61   }
62
63   foreach (wallWindowSurface ws in bndSurfaces)
64   {
65     if (!ws.IsGroundWall)
66     {
67       if (ws.AdjacentSpaceFactor < 0)
68       {
69         double fs = ws.Incline.ConfigurationFactorToSky;
70         ws.SolAirTemperature = OutdoorTemperature
71           + (ws.ShortWaveEmissivity * ws.Incline.GetSolarIrradiance(Sun, Albedo)
72           - ws.LongWaveEmissivity * fs * NocturnalRadiation) / ws.FilmCoefficient;
73         ws.HumidityRatio = OutdoorHumidityRatio;
74       }
75       else
76       {
77         //隣室温度差係数を用いる場合
78         double ftd = ws.AdjacentSpaceFactor;
79         double tmp = zones[ws.ReverseSideSurface.ZoneIndex].Temperature;
80         ws.SolAirTemperature = (1 - ftd) * tmp + ftd * OutdoorTemperature;
81         ws.HumidityRatio = (1 - ftd) * tmp + ftd * OutdoorHumidityRatio;
82       }
83     }
84   }
85 }
```

プログラム 25.18 に行列[*AT*][*BT*]の計算処理を示す。4~14 行は[*AT*][*BT*]の再計算の要否判断処理である。[*AT*]の逆行列計算負荷は大きいため、壁と壁表面の物性が更新されていなければ更新処理を飛ばして無駄な再計算を防ぐ。19, 21, 23 行で室番号 *i*、ゾーン番号 *j*、壁窓表面番号 *k* についてループをかけ、すべての壁窓表面について計算を行う。

29~66 行は行列[*AT*]の計算であり、式 25.9 および式 25.62 の実装である。31~48 行は同一の室に属する壁表面との間の係数であり、式 25.9 の 1 行、2 行にあたる。49~66 行は裏側の室に属する壁表面との間の係数であり、式 25.9 の 3 行にあたる。

68~92 行は行列[*BT*]の計算であり、式 25.10 および式 25.63 の実装である。72~81 行で同一ゾーンとの間の係数を設定し、82~91 行で裏側ゾーンとの間の係数を設定する。

プログラム **25.18**　行列[*AT*]|[*BT*]の計算

Popolo.ThermalLoad.MultiRooms class

```
/// <summary>熱平衡計算用 A,B 行列を計算する</summary>
private void makeABMatrix()
{
  //AB 行列の再計算の要否を確認
  bool needUpdateAB = false;
  for (int i = 0; i < walls.Length; i++)
  {
    if (walls[i].invMatrixUpdated)
    {
      needUpdateAB = true;
      break;
    }
  }
  if (!needUpdateAB) return;

  int nS = surfaces.Length;
  matA.Initialize(0);
  matB.Initialize(0);
  for (int i = 0; i < RoomNumber; i++)
  {
    for (int j = 0; j < wsNumber[i].Length; j++)
    {
      for (int k = 0; k < wsNumber[i][j].Length; k++)
      {
        int s1 = wsNumber[i][j][k];
        wallWindowSurface ws1 = surfaces[s1];
        wallWindowSurface ws1R = ws1.ReverseSideSurface;

        //行列 A を作成
        //同一室
        for (int m = 0; m < wsNumber[i].Length; m++)
        {
          for (int n = 0; n < wsNumber[i][m].Length; n++)
          {
            int s2 = wsNumber[i][m][n];
            if (s1 == s2)
            {
              matA[s1, s2] += 1;
              if (SolveMoistureTransferSimultaneously) matA[s1 + nS, s2 + nS] += 1;
            }
            else
            {
              double bf = ws1.RadiativeRate * gMatL[s1, s2];
              matA[s1, s2] += -ws1.FFS2 * bf;
              if (SolveMoistureTransferSimultaneously) matA[s1 + nS, s2] += -ws1.FFS3 * bf;
            }
          }
        }
        //裏側室
        if (!isSFboundary[s1])
        {
          int rvRm = zones[ws1R.ZoneIndex].RoomIndex;
          for (int m = 0; m < wsNumber[rvRm].Length; m++)
          {
            for (int n = 0; n < wsNumber[rvRm][m].Length; n++)
            {
              int s2 = wsNumber[rvRm][m][n];
              if (ws1R.Index != s2)
              {
                double bf = ws1R.RadiativeRate * gMatL[ws1R.Index, s2];
                matA[s1, s2] += -ws1.BFS2 * bf;
                if (SolveMoistureTransferSimultaneously) matA[s1 + nS, s2] += -ws1.BFS3 * bf;
              }
            }
          }
        }

        //行列 B を作成
        for (int q = 0; q < zones.Length; q++)
        {
          int nQ = ZoneNumber;
          if (rZones[i][j] == q)
          {
            matB[s1, q] += ws1.FFS2 * ws1.ConvectiveRate;
            if (SolveMoistureTransferSimultaneously)
            {
              matB[s1, q + nQ] += ws1.FFL2;
              matB[s1 + nS, q] += ws1.FFS3 * ws1.ConvectiveRate;
```

```
79              matB[s1 + nS, q + nQ] += ws1.FFL3;
80            }
81          }
82          if (ws1R.ZoneIndex == q && !isSFboundary[s1])
83          {
84            matB[s1, q] += ws1.BFS2 * ws1R.ConvectiveRate;
85            if (SolveMoistureTransferSimultaneously)
86            {
87              matB[s1, q + nQ] += ws1.BFL2;
88              matB[s1 + nS, q] += ws1.BFS3 * ws1R.ConvectiveRate;
89              matB[s1 + nS, q + nQ] += ws1.BFL3;
90            }
91          }
92        }
93      }
94    }
95  }
96
97  //A の逆行列計算
98  LinearAlgebra.GetInverse(ref matA, ref matAInv);
99 }
```

プログラム 25.19 にベクトル[*CT*]の計算処理を示す。式 25.11 および式 25.64 の実装である。

プログラム 25.19　ベクトル[*CT*]の計算

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>熱平衡計算用 C ベクトルを計算する</summary>
 2 private void makeCVector()
 3 {
 4   int nS = surfaces.Length;
 5   for (int i = 0; i < RoomNumber; i++)
 6   {
 7     for (int j = 0; j < wsNumber[i].Length; j++)
 8     {
 9       for (int k = 0; k < wsNumber[i][j].Length; k++)
10       {
11         int s1 = wsNumber[i][j][k];
12         wallWindowSurface ws1 = surfaces[s1];
13         wallWindowSurface ws1R = ws1.ReverseSideSurface;
14
15         //ベクトル C を作成
16         double rdsl;
17         if (ws1.IsWall) rdsl = (radToSurf_L[i] + radToSurf_S[s1]) / ws1.FilmCoefficient;
18         else rdsl = radToSurf_L[i] / ws1.FilmCoefficient;
19         vecC[s1] = ws1.IF2 + ws1.FFS2 * rdsl;
20         if (SolveMoistureTransferSimultaneously) vecC[s1 + nS] = ws1.IF3 + ws1.FFS3 * rdsl;
21         if (!isSFboundary[s1])
22         {
23           rdsl=(radToSurf_L[zones[ws1R.ZoneIndex].RoomIndex] + radToSurf_S[ws1R.Index]) /
24 ws1R.FilmCoefficient;
25           vecC[s1] += ws1.BFS2 * rdsl;
26           if (SolveMoistureTransferSimultaneously) vecC[s1 + nS] += ws1.BFS3 * rdsl;
27         }
28         else
29         {
30           vecC[s1] += ws1.BFS2 * ws1R.SolAirTemperature + ws1.BFL2 * ws1R.HumidityRatio;
31           if (SolveMoistureTransferSimultaneously)
32             vecC[s1 + nS] += ws1.BFS3 * ws1R.SolAirTemperature + ws1.BFL3 * ws1R.HumidityRatio;
33         }
34       }
35     }
36   }
   }
```

プログラム 25.20 に行列[*DT*]~[*JT*]の計算処理を示す。14~56 行で、式 25.17~25.19および式 25.66~25.68を用いて[*DT*]、[*ET*]、[*FT*]を計算した後、59~63 行で式 25.22 と 25.23 または式 25.71 と 25.72 を用いて行列[*IT*]と[*JT*]を計算する。

プログラム 25.20　行列[*DT*]~[*JT*]の計算

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>熱平衡計算用 IJ 行列を計算する</summary>
 2 private void makeIJMatrix()
 3 {
 4   int nS = surfaces.Length;
 5   int nQ = zones.Length;
 6   matD.Initialize(0);
```

```
 7   matF.Initialize(0);
 8   for (int q1 = 0; q1 < ZoneNumber; q1++)
 9   {
10     Zone zq1 = zones[q1];
11     double capSZN = (zq1.HeatCapacity + zq1.AirMass * AIR_SPECIFICHEAT) / TimeStep;
12     double capLZN = (zq1.WaterCapacity + zq1.AirMass) / TimeStep;
13
14     //行列Dを作成
15     for (int q2 = 0; q2 < ZoneNumber; q2++)
16     {
17       if (q1 == q2)
18       {
19         double bf = zq1.VentilationRate + zq1.SupplyAirFlowRate + zq1.supplyAirFlowRate2;
20         for (int q3 = 0; q3 < ZoneNumber; q3++)
21           if (zoneVent[q1, q3] != 0) bf += zoneVent[q1, q3];
22         matD[q1, q1] = bf * AIR_SPECIFICHEAT + capSZN;
23         if (SolveMoistureTransferSimultaneously) matD[q1 + nQ, q1 + nQ] = bf + capLZN;
24         for (int k = 0; k < zq1.Surfaces.Count; k++)
25         {
26           wallWindowSurface ws = zq1.Surfaces[k];
27           matD[q1, q1] += ws.Area * ws.ConvectiveCoefficient;
28           if (SolveMoistureTransferSimultaneously) matD[q1 + nQ, q1 + nQ] += ws.Area * ws.MoistureCoefficient;
29         }
30       }
31       else
32       {
33         if (zoneVent[q1, q2] != 0)
34         {
35           matD[q1, q2] = -zoneVent[q1, q2] * AIR_SPECIFICHEAT;
36           if (SolveMoistureTransferSimultaneously) matD[q1 + nQ, q2 + nQ] = -zoneVent[q1, q2];
37         }
38       }
39     }
40
41     //ベクトルEを作成
42     vecEJ[q1] = capSZN * zq1.Temperature + AIR_SPECIFICHEAT
43       * (zq1.VentilationRate * OutdoorTemperature + zq1.SupplyAirFlowRate * zq1.SupplyAirTemperature
44       + zq1.supplyAirFlowRate2 * zq1.supplyAirTemperature2) + zq1.IntegrateConvectiveHeatgains();
45     if (SolveMoistureTransferSimultaneously)
46       vecEJ[q1 + nQ] = capLZN * zq1.HumidityRatio
47         + zq1.VentilationRate * OutdoorHumidityRatio + zq1.SupplyAirFlowRate * zq1.SupplyAirHumidityRatio
48         + zq1.supplyAirFlowRate2 * zq1.supplyAirHumidityRatio2 + zq1.IntegrateWaterGains();
49
50     //行列Fを作成
51     for (int k = 0; k < zq1.Surfaces.Count; k++)
52     {
53       wallWindowSurface ws = zq1.Surfaces[k];
54       matF[q1, ws.Index] = ws.Area * ws.ConvectiveCoefficient;
55       if (SolveMoistureTransferSimultaneously) matF[q1 + nQ, ws.Index + nS] = ws.Area *
56 ws.MoistureCoefficient;
57     }
58   }
59
60   //IJ行列作成処理
61   LinearAlgebra.Multiplicate(matF, matAInv, ref matBf);
62   LinearAlgebra.Multiplicate(matBf, matB, ref matI);
63   LinearAlgebra.Add(matD, ref matI, 1, -1);
64   LinearAlgebra.Multiplicate(matBf, vecC, ref vecEJ, 1, 1);
  }
```

プログラム 25.21 に将来の熱平衡状態予測計算を示す。5 行と 8 行の準備計算で用意した行列[*IT*]と[*JT*]を用いて、境界条件の設定に応じて[*KT*][*LT*]を作成して解くメソッドである。10~15 行で空調制御を行うゾーンを特定する。この情報を用いて 20~25 行で[*KT*]を作成（式 25.27、式 25.74）、27~41 行で行列[*LT*]を作成する（式 25.29、式 25.76）。45 行で連立方程式を解き、46~63 行でその計算結果をゾーンの温湿度および顕熱潜熱負荷に代入する。

プログラム 25.21　将来の熱平衡状態予測

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>将来の熱平衡状態を予測する</summary>
 2 internal void ForecastHeatTransfer()
 3 {
 4   //準備計算実行
 5   prepareForHeatTransfer();
 6
 7   //IJ行列作成処理
```

```
 8   makeIJMatrix();
 9
10   bool[] ctrl = new bool[matK.Columns];
11   for (int i = 0; i < ctrl.Length; i++)
12   {
13     if (i < ZoneNumber) ctrl[i] = zones[i].TemperatureControlled;
14     else ctrl[i] = zones[i - ZoneNumber].HumidityControlled;
15   }
16
17   matK.Initialize(0);
18   for (int i = 0; i < matI.Rows; i++)
19   {
20     //K 行列を作成
21     for (int j = 0; j < matI.Columns; j++)
22     {
23       if (!ctrl[j]) matK[i, j] = matI[i, j];
24       else if (ctrl[j] && i == j) matK[i, j] = -1;
25     }
26
27     //L ベクトルを作成
28     vecTH[i] = vecEJ[i];
29     for (int j = 0; j < matI.Columns; j++)
30     {
31       if (ctrl[j])
32       {
33         if (j < ZoneNumber) vecTH[i] -= matI[i, j] * zones[j].TemperatureSetpoint;
34         else vecTH[i] -= matI[i, j] * zones[j - ZoneNumber].HumidityRatioSetpoint;
35       }
36     }
37     if (!ctrl[i])
38     {
39       if (i < ZoneNumber) vecTH[i] += zones[i].HeatSupply;
40       else vecTH[i] += zones[i - ZoneNumber].WaterSupply;
41     }
42   }
43
44   //連立方程式を解く (温湿度を計算)
45   LinearAlgebra.SolveLinearEquations(ref matK, ref vecTH);
46   for (int i = 0; i < ZoneNumber; i++)
47   {
48     if (zones[i].TemperatureControlled)
49     {
50       zones[i].HeatSupply = vecTH[i];
51       zones[i].Temperature = zones[i].TemperatureSetpoint;
52     }
53     else zones[i].Temperature = vecTH[i];
54     if (SolveMoistureTransferSimultaneously)
55     {
56       if (zones[i].HumidityControlled)
57       {
58         zones[i].WaterSupply = vecTH[i + ZoneNumber];
59         zones[i].HumidityRatio = zones[i].HumidityRatioSetpoint;
60       }
61       else zones[i].HumidityRatio = vecTH[i + ZoneNumber];
62     }
63   }
64 }
```

プログラム 25.22 に熱平衡状態確定処理を示す。4 行で将来予測フラグを解除する。これにより、次に ForecastHeatTransfer メソッドが呼び出された際に、[*AT*]~[*JT*]の準備計算処理が実行されることになる。各ゾーンの温度 T_{ZN} は計算済であるため、式 25.20 または式 25.69 を用いて 6~13 行で壁の表面温度を計算する。すべての壁の表面温度が得られれば式 25.4、式 25.5 により相当温度 T_{Sol} が計算できるため、15~42 行でそれぞれの壁表面に相当温度を設定する。

プログラム 25.22　熱平衡状態確定処理

Popolo.ThermalLoad.MultiRooms class

```
 1 /// <summary>熱平衡状態を確定する</summary>
 2 internal void FixHeatTransfer()
 3 {
 4   forecastingHeatTransfer = false;
 5
 6   //壁表面温度を計算
 7   for (int i = 0; i < ZoneNumber; i++)
 8   {
```

```
 9     vecTH[i] = zones[i].Temperature;
10     if (SolveMoistureTransferSimultaneously) vecTH[i + ZoneNumber] = zones[i].HumidityRatio;
11   }
12   LinearAlgebra.Multiplicate(matB, vecTH, ref vecC, 1, 1);
13   LinearAlgebra.Multiplicate(matAInv, vecC, ref vecTWS, 1, 0);
14
15   //温湿度条件を設定
16   for (int i = 0; i < RoomNumber; i++)
17   {
18     for (int j = 0; j < wsNumber[i].Length; j++)
19     {
20       for (int k = 0; k < wsNumber[i][j].Length; k++)
21       {
22         //相当温度
23         wallWindowSurface ws1 = surfaces[wsNumber[i][j][k]];
24         ws1.SolAirTemperature = 0;
25         for (int m = 0; m < wsNumber[i].Length; m++)
26         {
27           for (int n = 0; n < wsNumber[i][m].Length; n++)
28           {
29             int s2 = surfaces[wsNumber[i][m][n]].Index;
30             ws1.SolAirTemperature += gMatL[ws1.Index, s2] * vecTWS[s2];
31           }
32         }
33         ws1.SolAirTemperature *= ws1.RadiativeCoefficient;
34         ws1.SolAirTemperature += zones[rZones[i][j]].Temperature * ws1.ConvectiveCoefficient;
35         ws1.SolAirTemperature += radToSurf_L[i];
36         if (ws1.IsWall) ws1.SolAirTemperature += radToSurf_S[ws1.Index];
37         ws1.SolAirTemperature /= ws1.FilmCoefficient;
38
39         //絶対湿度
40         if (SolveMoistureTransferSimultaneously) ws1.HumidityRatio = zones[rZones[i][j]].HumidityRatio;
41       }
42     }
43   }
44 }
```

プログラム 25.23 に熱平衡状態の更新確定処理を示す。ForecastHeatTransfer により将来予測を行い、続けて FixHeatTransfer を呼び出して状態を確定させる。

プログラム 25.23　熱平衡状態の更新確定処理

Popolo.ThermalLoad.MultiRooms class

```
1 /// <summary>熱平衡状態を更新確定する</summary>
2 internal void UpdateHeatTransfer()
3 {
4   ForecastHeatTransfer();
5   FixHeatTransfer();
6 }
```

5）水分平衡の計算

プログラム 25.24 に水分平衡の計算処理を示す。大きな流れは顕熱平衡の計算と同様であり、ForecastWaterTransfer で繰り返し将来予測を行い、問題のない絶対湿度 W_{ZN} および潜熱負荷 HL_L が得られたら FixWaterTransfer で状態を確定させる。4~10 行は予測計算中の絶対湿度保存および復旧処理である。12~31 行で式 25.52 と式 25.53 を用いて行列[*AW*]と[*BW*]を作成する。さらに式 25.55 と式 25.57 を用いて 36~49 行で[*CW*]と[*DW*]を作成し、52 行で連立方程式を解く。53~61 行で各ゾーンに絶対湿度 W_{ZN} と潜熱負荷 HL_L を設定する。

プログラム 25.24　水分平衡の計算処理

Popolo.ThermalLoad.MultiRooms class

```
1 /// <summary>将来の水分衡状態を予測する</summary>
2 internal void ForecastWaterTransfer()
3 {
4   if (forecastingWaterTransfer)
5     for (int i = 0; i < ZoneNumber; i++) zones[i].HumidityRatio = zoneHumid[i];
6   else
7   {
8     for (int i = 0; i < ZoneNumber; i++) zoneHumid[i] = zones[i].HumidityRatio;
9     forecastingWaterTransfer = true;
10  }
```

```
11
12   //熱平衡を計算
13   for (int q1 = 0; q1 < ZoneNumber; q1++)
14   {
15     Zone zq1 = zones[q1];
16     double capLZN = (zq1.WaterCapacity + zq1.AirMass) / TimeStep;
17
18     //行列 AW を作成
19     for (int q2 = 0; q2 < ZoneNumber; q2++)
20     {
21       if (q1 == q2)
22       {
23         matAW[q1, q1] = zq1.VentilationRate + zq1.SupplyAirFlowRate + zq1.supplyAirFlowRate2 + capLZN;
24         for (int q3 = 0; q3 < ZoneNumber; q3++) matAW[q1, q1] += zoneVent[q1, q3];
25       }
26       else matAW[q1, q2] = -zoneVent[q1, q2];
27     }
28     //ベクトル BW を作成
29     vecWH[q1] = capLZN * zq1.HumidityRatio
30       + zq1.VentilationRate * OutdoorHumidityRatio + zq1.SupplyAirFlowRate * zq1.SupplyAirHumidityRatio
31       + zq1.supplyAirFlowRate2 * zq1.supplyAirHumidityRatio2 + zq1.IntegrateWaterGains();
32   }
33
34   //潜熱平衡を計算
35   matCW.Initialize(0);
36   for (int i = 0; i < ZoneNumber; i++)
37   {
38     //CW 行列を作成
39     for (int j = 0; j < ZoneNumber; j++)
40     {
41       if (!zones[j].HumidityControlled) matCW[i, j] = matAW[i, j];
42       else if (zones[j].HumidityControlled && (i == j)) matCW[i, j] = -1;
43     }
44
45     //DW ベクトルを作成
46     for (int j = 0; j < ZoneNumber; j++)
47       if (zones[j].HumidityControlled) vecWH[i] -= matAW[i, j] * zones[j].HumidityRatioSetpoint;
48     if (!zones[i].HumidityControlled) vecWH[i] += zones[i].WaterSupply;
49   }
50
51   //連立方程式を解く (湿度を計算)
52   LinearAlgebra.SolveLinearEquations(ref matCW, ref vecWH);
53   for (int i = 0; i < ZoneNumber; i++)
54   {
55     if (zones[i].HumidityControlled)
56     {
57       zones[i].WaterSupply = vecWH[i];
58       zones[i].HumidityRatio = zones[i].HumidityRatioSetpoint;
59     }
60     else zones[i].HumidityRatio = vecWH[i];
61   }
62 }
63
64 /// <summary>水分平衡状態を確定する</summary>
65 internal void FixWaterTransfer()
66 { forecastingWaterTransfer = false; }
67
68 /// <summary>水分平衡状態を更新確定する</summary>
69 internal void UpdateWaterTransfer()
70 {
71   ForecastWaterTransfer();
72   FixWaterTransfer();
73 }
```

25.3.4　建物熱負荷計算クラスの作成

建物全体の熱負荷計算を行うクラスである BuildingThermalModel を開発する。BuildingThermalModel クラスは複数の MultiRooms インスタンスを扱い、例えば 10 階建ての建物について、10 の MultiRooms インスタンスでそれぞれのフロアを表現し、これらを束ねて BuildingThermalModel を構築するような使い方をする。

プログラム 25.25 に定数、インスタンス変数、プロパティの定義を示す。2 行は並列計算を行うか否かのフラグである。BuildingThermalModel クラスでは、一定時間間隔で MultiRooms インスタンスの状態変数を受け渡すだけで連成計算は行わないため、複数の CPU を用いた並列処理に向いてい

る。上記のように 10 のフロアがあれば、それぞれの計算を別の CPU で実行することで計算速度を向上させることができる。

多数室クラスと同様に ForecastHeatTransfer メソッドで将来予測を行い、FixState メソッドで確定するという構成とする。このため、境界条件などの変更があり将来の状態が変化しうるか否かを 5 行と 8 行の変数に保存する。

プログラム 25.25　定数、インスタンス変数、プロパティの定義

Popolo.ThermalLoad.BuildingThermalModel class

```
 1 /// <summary>初回の予測計算か否か</summary>
 2 private bool isFirstForecast = true;
 3
 4 /// <summary>顕熱流に関する境界条件変更の真偽</summary>
 5 private Dictionary<MultiRooms, bool> hasHTChgd = new Dictionary<MultiRooms, bool>();
 6
 7 /// <summary>水分流に関する境界条件変更の真偽</summary>
 8 private Dictionary<MultiRooms, bool> hasWTChgd = new Dictionary<MultiRooms, bool>();
 9
10 /// <summary>壁リスト</summary>
11 private List<Wall> walls;
12
13 /// <summary>多数室</summary>
14 private MultiRooms[] mRooms;
15
16 /// <summary>多数室を取得する</summary>
17 public ImmutableMultiRooms[] MultiRoom { get { return mRooms; } }
18
19 /// <summary>現在の日時を取得する</summary>
20 public DateTime CurrentDateTime { get; private set; }
21
22 /// <summary>計算時間間隔[sec]</summary>
23 private double timeStep = 3600;
24
25 /// <summary>計算時間間隔[sec]を設定・取得する</summary>
26 public double TimeStep
27 {
28   get { return timeStep; }
29   set
30   {
31     for (int i = 0; i < mRooms.Length; i++) mRooms[i].TimeStep = value;
32     timeStep = value;
33   }
34 }
35
36 /// <summary>ゾーン間換気リスト</summary>
37 private interZoneAirFlowCollection[][] zoneVent;
```

プログラム 25.26 にコンストラクタを示す。引数は MultiRooms クラスのインスタンスの配列である。壁の熱流更新は BuildingThermalModel クラスで扱うため、17, 18 行でそれぞれの MultiRooms に含まれる Wall を保存する。20~29 行で、同一のゾーンあるいは窓が複数の MultiRooms クラスで計算されていないかを確認する。なお、壁に関しては F 側と B 側があるため、複数の MultiRooms クラスで計算対象となっている可能性がある。

プログラム 25.26　コンストラクタ

Popolo.ThermalLoad.BuildingThermalModel class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="mRooms">多数室</param>
 3 public BuildingThermalModel(MultiRooms[] mRooms)
 4 {
 5   this.mRooms = mRooms;
 6   zoneVent = new interZoneAirFlowCollection[mRooms.Length][];
 7   walls = new List<Wall>();
 8   List<ImmutableZone> zns = new List<ImmutableZone>();
 9   List<ImmutableWindow> wins = new List<ImmutableWindow>();
10   for (int i = 0; i < mRooms.Length; i++)
11   {
12     zoneVent[i] = new interZoneAirFlowCollection[mRooms[i].ZoneNumber];
13     for (int j = 0; j < zoneVent[i].Length; j++) zoneVent[i][j] = new interZoneAirFlowCollection();
14     hasHTChgd.Add(mRooms[i], true);
15     hasWTChgd.Add(mRooms[i], true);
```

```
16
17     foreach (Wall wl in mRooms[i].Walls)
18       if (!walls.Contains(wl)) walls.Add(wl);
19
20     //ゾーンの重複確認
21     foreach (ImmutableZone zn in mRooms[i].Zones)
22       if (zns.Contains(zn)) throw new Exception("Zone belongs to more than 1 MultiRooms");
23         zns.AddRange(mRooms[i].Zones);
24
25     //窓の重複確認
26     foreach (ImmutableWindow win in mRooms[i].Windows)
27       if (wins.Contains(win)) throw new Exception("Window belongs to more than 1 MultiRooms");
28     wins.AddRange(mRooms[i].Windows);
29   }
30 }
```

プログラム 25.27 に境界条件の設定処理を示す。

1~16 行は外気条件の設定である。日時情報、太陽位置情報、温湿度、夜間放射量を設定する。これらの情報はすべての多数室で共通とする。

18~27 行は地中温度の設定である。壁の番号と面の向きを指定して地中温度を設定する。

29~82 行はゾーン間換気の設定である。ゾーンで相互に空気が混合する場合には 29~56 行、あるゾーンから別のあるゾーンに一方的に空気が流れる場合には 58~82 行のメソッドを用いる。

84~95 行は埋設配管の流体条件設定、97~108 行はゾーンに給気する空気条件の設定である。

プログラム 25.27　境界条件設定処理

Popolo.ThermalLoad.BuildingThermalModel class

```
 1 /// <summary>外気条件を更新する</summary>
 2 /// <param name="dTime">現在の日時</param>
 3 /// <param name="sun">太陽</param>
 4 /// <param name="temperature">外気乾球温度[C]</param>
 5 /// <param name="humidityRatio">外気絶対湿度[kg/kg]</param>
 6 /// <param name="nocRadiation">夜間放射[W/m2]</param>
 7 public void UpdateOutdoorCondition
 8   (DateTime dTime, ImmutableSun sun, double temperature, double humidityRatio, double nocRadiation)
 9 {
10   CurrentDateTime = dTime;
11   foreach (MultiRooms mr in mRooms)
12   {
13     mr.UpdateOutdoorCondition(dTime, sun, temperature, humidityRatio, nocRadiation);
14     hasHTChgd[mr] = hasWTChgd[mr] = true;
15   }
16 }
17
18 /// <summary>地中温度[C]を設定する</summary>
19 /// <param name="mRoomIndex">多数室番号</param>
20 /// <param name="wallIndex">壁番号</param>
21 /// <param name="isSideF">F 側か否か</param>
22 /// <param name="groundTemperature">地中温度[C]</param>
23 public void SetGroundTemperature(int mRoomIndex, int wallIndex, bool isSideF, double groundTemperature)
24 {
25   mRooms[mRoomIndex].SetGroundTemperature(wallIndex, isSideF, groundTemperature);
26   hasHTChgd[mRooms[mRoomIndex]] = hasWTChgd[mRooms[mRoomIndex]] = true;
27 }
28
29 /// <summary>ゾーン間換気量[kg/s]を設定する</summary>
30 /// <param name="rmIndex">多数室番号</param>
31 /// <param name="znIndex1">ゾーン番号 1</param>
32 /// <param name="znIndex2">ゾーン番号 2</param>
33 /// <param name="airFlowRate">ゾーン間換気量[kg/s]</param>
34 public void SetCrossVentilation(int rmIndex, int znIndex1, int znIndex2, double airFlowRate)
35 {
36   mRooms[rmIndex].SetCrossVentilation (znIndex1, znIndex2, airFlowRate);
37   hasHTChgd[mRooms[rmIndex]] = hasWTChgd[mRooms[rmIndex]] = true;
38 }
39
40 /// <summary>ゾーン間換気量[kg/s]を設定する</summary>
41 /// <param name="rmIndex1">多数室番号 1</param>
42 /// <param name="znIndex1">ゾーン番号 1</param>
43 /// <param name="rmIndex2">多数室番号 2</param>
44 /// <param name="znIndex2">ゾーン番号 2</param>
45 /// <param name="airFlowRate">ゾーン間換気量[kg/s]</param>
46 public void SetCrossVentilation(int rmIndex1, int znIndex1, int rmIndex2, int znIndex2, double airFlowRate)
```

```
 47 {
 48   //同一の多数室の場合には連成計算
 49   if (rmIndex1 == rmIndex2) SetCrossVentilation(rmIndex1, znIndex1, znIndex2, airFlowRate);
 50   //他の多数室の場合には境界条件として計算
 51   else
 52   {
 53     zoneVent[rmIndex2][znIndex2].AddAirFlow(rmIndex1, znIndex1, airFlowRate);
 54     zoneVent[rmIndex1][znIndex1].AddAirFlow(rmIndex2, znIndex2, airFlowRate);
 55   }
 56 }
 57
 58 /// <summary>ゾーン１からゾーン２へ向かう換気量[kg/s]を設定する</summary>
 59 /// <param name="rmIndex1">多数室番号 1</param>
 60 /// <param name="znIndex1">ゾーン番号 1</param>
 61 /// <param name="rmIndex2">多数室番号 2</param>
 62 /// <param name="znIndex2">ゾーン番号 2</param>
 63 /// <param name="airFlowRate">換気量[kg/s]</param>
 64 public void SetAirFlow(int rmIndex1, int znIndex1, int rmIndex2, int znIndex2, double airFlowRate)
 65 {
 66   //同一の多数室の場合には連成計算
 67   if (rmIndex1 == rmIndex2) SetAirFlow(rmIndex1, znIndex1, znIndex2, airFlowRate);
 68   //他の多数室の場合には境界条件として計算
 69   else zoneVent[rmIndex2][znIndex2].AddAirFlow(rmIndex1, znIndex1, airFlowRate);
 70 }
 71
 72 /// <summary>ゾーン１からゾーン２へ向かう換気量[kg/s]を設定する</summary>
 73 /// <param name="mRoomIndex">多数室番号</param>
 74 /// <param name="zoneIndex1">ゾーン番号 1</param>
 75 /// <param name="zoneIndex2">ゾーン番号 2</param>
 76 /// <param name="airFlowRate">換気量[kg/s]</param>
 77 /// <remarks>他のゾーンを含め風量収支を合わせる必要あり</remarks>
 78 public void SetAirFlow(int mRoomIndex, int zoneIndex1, int zoneIndex2, double airFlowRate)
 79 {
 80   mRooms[mRoomIndex].SetAirFlow(zoneIndex1, zoneIndex2, airFlowRate);
 81   hasHTChgd[mRooms[mRoomIndex]] = hasWTChgd[mRooms[mRoomIndex]] = true;
 82 }
 83
 84 /// <summary>送水条件を設定する</summary>
 85 /// <param name="mRoomIndex">多数室番号</param>
 86 /// <param name="wallIndex">壁（床）番号</param>
 87 /// <param name="mNumber">質点番号</param>
 88 /// <param name="flowRate">通水量[kg/s]</param>
 89 /// <param name="temperature">水温[C]</param>
 90 public void SetBuriedPipeWaterState
 91   (int mRoomIndex, int wallIndex, int mNumber, double flowRate, double temperature)
 92 {
 93   mRooms[mRoomIndex].SetBuriedPipeWaterState(wallIndex, mNumber, flowRate, temperature);
 94   hasHTChgd[mRooms[mRoomIndex]] = true;
 95 }
 96
 97 /// <summary>給気条件を設定する</summary>
 98 /// <param name="mRoomIndex">多数室番号</param>
 99 /// <param name="zoneIndex">ゾーン番号</param>
100 /// <param name="saTemperature">給気温度[C]</param>
101 /// <param name="saHumidityRatio">給気絶対湿度[kg/kg]</param>
102 /// <param name="saFlowRate">給気量[kg/s]</param>
103 public void SetSupplyAir
104   (int mRoomIndex, int zoneIndex, double saTemperature, double saHumidityRatio, double saFlowRate)
105 {
106   mRooms[mRoomIndex].SetSupplyAir(zoneIndex, saTemperature, saHumidityRatio, saFlowRate);
107   hasHTChgd[mRooms[mRoomIndex]] = hasWTChgd[mRooms[mRoomIndex]] = true;
108 }
109
110 /// <summary>壁とゾーンの温湿度を初期化する</summary>
111 /// <param name="temperature">乾球温度[C]</param>
112 /// <param name="humidityRatio">絶対湿度[kg/kg]</param>
113 public void InitializeAirState(double temperature, double humidityRatio)
114 { foreach (MultiRooms ml in mRooms) ml.InitializeAirState(temperature, humidityRatio); }
```

プログラム 25.28 に温湿度制御関連の処理を示す。1 番目の引数で与えられる多数室の番号を手がかりに MultiRooms インスタンスに処理を受け渡す。

プログラム 25.28　温湿度制御関連の処理

Popolo.ThermalLoad.BuildingThermalModel class

```
  1 /// <summary>乾球温度[C]を制御する</summary>
  2 /// <param name="mRoomIndex">多数室番号</param>
  3 /// <param name="zoneIndex">ゾーン番号</param>
  4 /// <param name="setpoint">乾球温度設定値[C]</param>
```

```
 5 public void ControlDrybulbTemperature(int mRoomIndex, int zoneIndex, double setpoint)
 6 {
 7   mRooms[mRoomIndex].ControlDrybulbTemperature(zoneIndex, setpoint);
 8   hasHTChgd[mRooms[mRoomIndex]] = true;
 9   if (mRooms[mRoomIndex].SolveMoistureTransferSimultaneously) hasWTChgd[mRooms[mRoomIndex]] = true;
10 }
11
12 /// <summary>顕熱供給[W]を制御する</summary>
13 /// <param name="mRoomIndex">多数室番号</param>
14 /// <param name="zoneIndex">ゾーン番号</param>
15 /// <param name="heatSupply">顕熱供給[W]</param>
16 public void ControlHeatSupply(int mRoomIndex, int zoneIndex, double heatSupply)
17 {
18   mRooms[mRoomIndex].ControlHeatSupply(zoneIndex, heatSupply);
19   hasHTChgd[mRooms[mRoomIndex]] = true;
20   if (mRooms[mRoomIndex].SolveMoistureTransferSimultaneously) hasWTChgd[mRooms[mRoomIndex]] = true;
21 }
22
23 /// <summary>絶対湿度[kg/kg]を制御する</summary>
24 /// <param name="mRoomIndex">多数室番号</param>
25 /// <param name="zoneIndex">ゾーン番号</param>
26 /// <param name="setpoint">絶対湿度設定値[kg/kg]</param>
27 public void ControlHumidityRatio(int mRoomIndex, int zoneIndex, double setpoint)
28 {
29   mRooms[mRoomIndex].ControlHumidityRatio(zoneIndex, setpoint);
30   hasWTChgd[mRooms[mRoomIndex]] = true;
31   if (mRooms[mRoomIndex].SolveMoistureTransferSimultaneously) hasHTChgd[mRooms[mRoomIndex]] = true;
32 }
33
34 /// <summary>水分供給[kg/s]を制御する</summary>
35 /// <param name="mRoomIndex">多数室番号</param>
36 /// <param name="zoneIndex">ゾーン番号</param>
37 /// <param name="waterSupply">水分供給[kg/s]</param>
38 public void ControlWaterSupply(int mRoomIndex, int zoneIndex, double waterSupply)
39 {
40   mRooms[mRoomIndex].ControlWaterSupply(zoneIndex, waterSupply);
41   hasWTChgd[mRooms[mRoomIndex]] = true;
42   if (mRooms[mRoomIndex].SolveMoistureTransferSimultaneously) hasHTChgd[mRooms[mRoomIndex]] = true;
43 }
```

プログラム 25.29 に顕熱・水分平衡計算処理を示す。

多数室をまたぐゾーン間換気がある場合には、他の多数室からの流入空気の温湿度が境界条件となるため、初回の計算時には他の給気温湿度を初期化する（8~13 行、46~51 行）。90~114 行がこの処理であり、各ゾーンからの流入空気の温湿度を流量で重み付け平均する。以降の処理は MultiRooms クラスに受け渡すだけであるが、ForecastHeatTransfer メソッドに関しては並列計算に対応させており、MultiRooms クラスの ForecastHeatTransfer メソッドを並列に実行させる（17~24 行）。並列計算を実行する際に気をつけるべき大きな点は、複数のスレッド（1 つ 1 つのプログラムの処理の流れ）から同一のインスタンス変数の値を書き換えてデータを破損させてはならないという点である。このためには、インスタンス変数の操作に先立ち、インスタンスをロックし、他のスレッドからの操作を防止するという方法がある。またはそもそも他のインスタンス変数の書き換えを行わず、読み取りのみの処理とするという方法もある。前者はロックおよび解除自体の処理に時間がかかるため、この時間を費やしてまで並列化をするべきか否かについて検討が必要になる。MultiRooms クラスの ForecastHeatTransfer メソッドは、他の MultiRooms と共有するインスタンス変数の値を読みこむだけであるため、データ破壊の恐れがなく、並列化が容易である。一方、FixHeatTransfer メソッドではゾーンの温度を確定させて壁の相当温度を更新するため、他の MultiRooms と共有する Wall インスタンスの値を書き換える。従って、並列化にあたっては Wall インスタンスのロック作業が発生するため、本書のモデルでは並列化処理の対象としていない。近年ではマルチコア CPU が一般化したため、並列計算技術に関しても多くの情報が手に入る環境が整っている。詳細については参考文献を参

照されたい[25.22) 25.23)]。

プログラム **25.29**　顕熱・水分平衡計算

Popolo.ThermalLoad.BuildingThermalModel class

```
/// <summary>将来の熱平衡状態を予測する</summary>
/// <remarks>
/// 繰り返し呼び出すことが可能。
/// 最後にFixHeatTransferを呼び出して確定させる
/// </remarks>
public void ForecastHeatTransfer()
{
  //初回計算時に他のゾーン温湿度を境界条件に設定
  if (isFirstForecast)
  {
    setInterZoneAirFlow();
    isFirstForecast = false;
  }

  if (PARALLEL_COMPUTING)
  {
    Parallel.ForEach(mRooms, mr =>
    {
      if (hasHTChgd[mr])
      {
        mr.ForecastHeatTransfer();
        hasHTChgd[mr] = false;
      }
    });
  }
  else
  {
    foreach (MultiRooms mr in mRooms)
    {
      if (hasHTChgd[mr])
      {
        mr.ForecastHeatTransfer();
        hasHTChgd[mr] = false;
      }
    }
  }
}

/// <summary>将来の水分衡状態を予測する</summary>
/// <remarks>
/// 繰り返し呼び出すことが可能。
/// 最後にFixWaterTransferを呼び出して確定させる
/// </remarks>
public void ForecastWaterTransfer()
{
  //初回計算時に他のゾーン温湿度を境界条件に設定
  if (isFirstForecast)
  {
    setInterZoneAirFlow();
    isFirstForecast = false;
  }

  foreach (MultiRooms mr in mRooms)
  {
    if (!mr.SolveMoistureTransferSimultaneously)
    {
      if (hasWTChgd[mr])
      {
        mr.ForecastWaterTransfer();
        hasWTChgd[mr] = false;
      }
    }
  }
}

/// <summary>状態を確定させる</summary>
public void FixState()
{
  foreach (MultiRooms mr in mRooms)
  {
    mr.FixHeatTransfer();
    mr.FixWaterTransfer();
    hasHTChgd[mr] = hasWTChgd[mr] = true;
  }
  //壁の熱流を更新
```

```
 76   foreach (Wall wl in walls) wl.Update();
 77   isFirstForecast = true;
 78 }
 79
 80 /// <summary>ゾーンの温湿度を将来予測値から現在値に戻す</summary>
 81 public void ResetAirState()
 82 {
 83   foreach (MultiRooms mr in mRooms)
 84   {
 85     mr.ResetAirState();
 86     hasHTChgd[mr] = hasWTChgd[mr] = true;
 87   }
 88 }
 89
 90 /// <summary>ゾーン間換気条件を設定する</summary>
 91 private void setInterZoneAirFlow()
 92 {
 93   for (int i = 0; i < zoneVent.Length; i++)
 94   {
 95     for (int j = 0; j < zoneVent[i].Length; j++)
 96     {
 97       double afsum, tsa, wsa;
 98       afsum = tsa = wsa = 0;
 99       foreach (interZoneAirFlow zaf in zoneVent[i][j])
100       {
101         afsum += zaf.aFlow;
102         ImmutableZone zn = mRooms[zaf.rmIndex].Zones[zaf.znIndex];
103         tsa += zn.Temperature * zaf.aFlow;
104         wsa += zn.HumidityRatio * zaf.aFlow;
105       }
106       if (afsum != 0)
107       {
108         tsa /= afsum;
109         wsa /= afsum;
110       }
111       mRooms[i].setSupplyAir(j, tsa, wsa, afsum);
112     }
113   }
114 }
```

【例題 25.1】

図 25.7 に示す平面および断面を持つ 2 室の建物について、熱負荷計算モデルを作成せよ。また、表 25.2 の外気条件（夜間放射の影響は無視）を前提に熱負荷の周期定常計算を実行せよ。壁の構成と厚みは表 25.3 に示す通りとし、窓ガラスは透明フロート二重ガラスとし、ガラス 1 枚の透過率は 0.7、反射率は 0.04 とする。床は完全に断熱されており、地中からの熱流は考慮しないこととする。長波長放射率は内外表面とも 0.9、短波長放射率は 0.8、アルベドは 0.2 とする。家具類の熱容量は見込まない。空調運転時間は 8:00~18:00 とし、温湿度設定値は夏季が 26 ºC, 50 %（10.5g/kg）、冬季が 22 ºC, 40 %（6.6g/kg）とする。漏気は室容積あたりで 0.2 回/h とする。空調機の容量は各室ともに暖房 2 kW、冷房 2 kW とする。

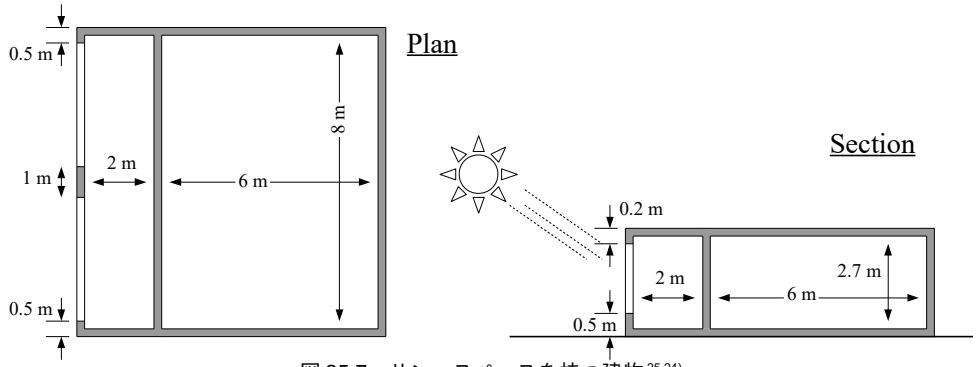


図 25.7　サン・スペースを持つ建物[25.24)]

表 25.2　気象条件

	夏季（7 月 20 日）			冬季（1 月 20 日）		
時刻	乾球温度 [C]	絶対湿度 [g/kg]	水平面全天日射 [W/m²]	乾球温度 [C]	絶対湿度 [g/kg]	水平面全天日射 [W/m²]
0:00	24.9	12.9	0	4.1	4.2	0
1:00	24.7	12.8	0	4.2	4.0	0
2:00	23.8	13.9	0	4.6	4.0	0
3:00	24.2	14.6	0	5.0	3.6	0
4:00	24.2	14.7	0	4.4	4.0	0
5:00	25.0	14.6	0	3.8	3.7	0
6:00	25.0	13.5	93	4.8	3.9	0
7:00	24.4	13.7	288	4.4	3.9	1
8:00	24.1	14.8	465	4.9	3.5	146
9:00	23.7	15.4	629	6.5	3.2	318
10:00	24.6	16.1	781	7.7	3.4	438
11:00	25.0	15.5	860	8.1	3.5	506
12:00	25.3	15.6	870	9.0	3.3	532
13:00	25.2	15.6	827	10.1	3.2	467
14:00	24.9	16.0	725	10.7	3.0	391
15:00	24.9	16.5	598	10.2	3.1	232
16:00	25.3	16.6	403	10.1	3.1	83
17:00	25.9	16.4	217	9.8	2.8	0
18:00	25.8	16.8	49	8.6	2.8	0
19:00	25.1	16.0	0	7.9	3.0	0
20:00	24.2	16.0	0	7.6	3.5	0
21:00	23.5	16.1	0	7.1	3.1	0
22:00	23.6	16.2	0	5.4	3.0	0
23:00	23.5	16.6	0	4.1	3.1	0

表 25.3　壁構成と厚み

部位	材料	熱伝導率 [W/(m·K)]	容積比熱 [kJ/(m³·K)]	厚み [m]
外壁	コンクリートブロック（内）	0.51	1,400	0.1
	断熱材	0.04	14	0.0615
	木質系サイディング（外）	0.14	477	0.009
床	コンクリート	1.13	1,400	0.08
屋根	プラスターボード（内）	0.16	798	0.01
	断熱材	0.04	10	0.1118
	木質系屋根材（外）	0.14	477	0.019
内壁	コンクリートブロック	0.51	1,400	0.2

【解】

モデル化をする前に図 25.8 に示すように壁に番号を振り、整理をすると良い。本モデルには 11 の壁がある。また、窓のあるサンスペースをゾーン 0（室 0）、裏側のバックスペースをゾーン 1（室 1）とする。

建物モデルの作成処理をプログラム 25.30 に示す。3~8 行で垂直 4 方位と水平の傾斜面を作成する。10~23 行は表 25.3 の壁構成作成処理である。原則として室側が F 側、屋外側が B 側となるようにした。17 行に示すように地中からの熱流を無視するために、床材には極めて小さい熱伝導率と熱容量を持つ 1m の断熱層を設ける。25~27 行で表面積を指定して壁を作成する。南面の壁は 2 箇所が窓によって繰り抜かれているが、本モデルでは壁の 3 次元的な熱流は解かないため、南面の壁は全体面積から単純に窓面積分を差し引き 28 行のように作成する。38~42 行は短波長放射および長波長放射の設定処理である。44~47 行で窓を、49~55 行でゾーンを作成し、58 行で 2 つの室からなる多数室を作成する。60~62 行はゾーンの設定であり、室 0 にはゾーン 0 のみ、室 1 にはゾーン 1 のみが属する。63~74 行は壁の登録処理であり、0~9 番の壁は片面が屋内に、片面が屋外に面する。10 番の内壁のみ両側が屋内に面しており、74 行で示すように両ゾーンの番号を与えて登録を行う。80~82 行は地中温度の設定であり、本計算では常に 20 ºC として扱う。84 行で建物モデルを初期化して出力する。

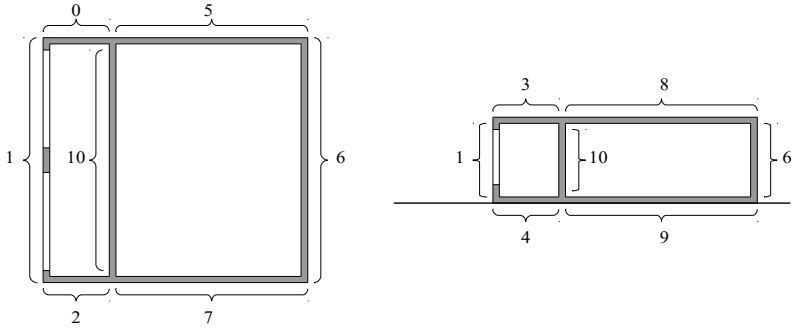


図 25.8　壁番号の設定

プログラム **25.30**　建物モデルの作成

```
 1 private static BuildingThermalModel makeSunSpaceBuildingModel()
 2 {
 3   //傾斜面を作成
 4   Incline incN = new Incline(Incline.Orientation.N, 0.5 * Math.PI);
 5   Incline incE = new Incline(Incline.Orientation.E, 0.5 * Math.PI);
 6   Incline incW = new Incline(Incline.Orientation.W, 0.5 * Math.PI);
 7   Incline incS = new Incline(Incline.Orientation.S, 0.5 * Math.PI);
 8   Incline incH = new Incline(Incline.Orientation.N, 0);
 9
10   //壁構成を作成
11   WallLayer[] exWL = new WallLayer[3];  //外壁
12   exWL[0] = new WallLayer("コンクリートブロック", 0.51, 1400, 0.1);
13   exWL[1] = new WallLayer("断熱材", 0.04, 14, 0.0615);
14   exWL[2] = new WallLayer("木質系サイディング", 0.14, 477, 0.009);
15   WallLayer[] flWL = new WallLayer[2];  //床
16   flWL[0] = new WallLayer("コンクリート", 1.13, 1400, 0.08);
17   flWL[1] = new WallLayer("断熱材（無限大）", 0.00001, 0.00001, 1.0);
18   WallLayer[] rfWL = new WallLayer[3];  //屋根
19   rfWL[0] = new WallLayer("プラスターボード", 0.16, 798, 0.01);
20   rfWL[1] = new WallLayer("断熱材", 0.04, 10, 0.1118);
21   rfWL[2] = new WallLayer("木質系屋根", 0.14, 477, 0.019);
22   WallLayer[] inWL = new WallLayer[1];  //内壁
23   inWL[0] = new WallLayer("コンクリートブロック", 0.51, 1400, 0.2);
24
25   //壁を作成
26   Wall[] walls = new Wall[11];
27   walls[0] = new Wall(2 * 2.7, exWL);
28   walls[1] = new Wall(8 * 2.7 - 6 * 2, exWL);
29   walls[2] = new Wall(2 * 2.7, exWL);
30   walls[3] = new Wall(2 * 8, rfWL);
31   walls[4] = new Wall(2 * 8, flWL);
32   walls[5] = new Wall(6 * 2.7, exWL);
33   walls[6] = new Wall(8 * 2.7, exWL);
34   walls[7] = new Wall(6 * 2.7, exWL);
35   walls[8] = new Wall(6 * 8, rfWL);
36   walls[9] = new Wall(6 * 8, flWL);
37   walls[10] = new Wall(8 * 2.7, inWL);
38   for (int i = 0; i < walls.Length; i++)
39   {
40     walls[i].LongWaveEmissivityF = walls[i].LongWaveEmissivityB = 0.9;
41     walls[i].ShortWaveAbsorptanceF = walls[i].ShortWaveAbsorptanceB = 0.8;
42   }
43
44   //窓を作成
45   Window[] windows = new Window[1];
46   windows[0] = new Window
47     (6 * 2, new double[] { 0.7, 0.7 }, new double[] { 0.04, 0.04 }, incS);
48
49   //ゾーンを作成
50   Zone[] zones = new Zone[2];
51   zones[0] = new Zone("SunSpace", 2 * 8 * 2.7 * 1.2);
52   zones[1] = new Zone("BackSpace", 6 * 8 * 2.7 * 1.2);
53   //0.2 回/h の漏気
54   zones[0].VentilationRate = zones[0].AirMass / 3600d * 0.2;
55   zones[1].VentilationRate = zones[1].AirMass / 3600d * 0.2;
56
57   //MultiRooms を作成
58   MultiRooms mRoom = new MultiRooms(2, zones, walls, windows);
59
60   //ゾーン設定
61   mRoom.AddZone(0, 0);
62   mRoom.AddZone(1, 1);
63   //壁設定
64   mRoom.AddWall(0, 0, true); mRoom.SetOutsideWall(0, false, incW);
65   mRoom.AddWall(0, 1, true); mRoom.SetOutsideWall(1, false, incS);
66   mRoom.AddWall(0, 2, true); mRoom.SetOutsideWall(2, false, incE);
67   mRoom.AddWall(0, 3, true); mRoom.SetOutsideWall(3, false, incH);
68   mRoom.AddWall(0, 4, true); mRoom.SetOutsideWall(4, false, incH);
69   mRoom.AddWall(1, 5, true); mRoom.SetOutsideWall(5, false, incW);
70   mRoom.AddWall(1, 6, true); mRoom.SetOutsideWall(6, false, incN);
71   mRoom.AddWall(1, 7, true); mRoom.SetOutsideWall(7, false, incE);
72   mRoom.AddWall(1, 8, true); mRoom.SetOutsideWall(8, false, incH);
73   mRoom.AddWall(1, 9, true); mRoom.SetOutsideWall(9, false, incH);
74   mRoom.AddWall(0, 1, 10);
75   //窓設定
76   mRoom.AddWindow(0, 0);
77   //アルベド
78   mRoom.Albedo = 0.2;
79
```

```
80   //地中温度は20Cで固定
81   mRoom.SetGroundTemperature(4, false, 20);
82   mRoom.SetGroundTemperature(9, false, 20);
83
84   return new BuildingThermalModel(new MultiRooms[] { mRoom });
85 }
```

周期的（通常は24時間）な境界条件を与えて非定常計算を繰り返すと、出力である乾球温度や顕熱負荷なども境界条件と同じ周期で変動するようになる。このような方法で算出された熱負荷を、一般の非定常計算による熱負荷や定常計算による熱負荷と区別して「周期定常熱負荷」と呼ぶ。

プログラム25.31に周期定常熱負荷計算処理を示す。

9~18行は表25.2の周期的境界条件（乾球温度・絶対湿度・水平面全天日射）である。23~35行は出力が周期定常となったか否かを判定するために用いる、過去24時間分の乾球温度と顕熱負荷を保存する配列である。39~99行で、夏と冬の2回分の計算を行う。44~98行が周期定常に至るまで繰り返される処理である。46~50行で境界条件を設定する。水平面全天日射から直散分離を行うが、過去1時間のデータであるため、太陽位置を計算する時刻を30分シフトさせる[†1)]。

52~67行は温湿度制御の処理であり、8:00~18:00は52~59行で温湿度を固定、その他の時間は61~67行で自然室温とする。69行で将来時点の熱平衡を予測するが、空調機容量が不足する場合には、71~76行で熱供給量を最大能力にした上で自然室温を計算する。その後、77行で再度将来予測を行い、79行で熱平衡を確定させる[†2)]。

81~96行は収束計算判定である。24時間前の同ゾーンの乾球温度および顕熱負荷との差を誤差として評価して積算する[†3)]。24時間の計算が終わる度に91~96行で誤差の積算値を確認し、これが十分に小さくなり定常状態に近づいたと判定したところで収束計算を打ち切る。

プログラム 25.31　周期定常熱負荷計算

```
 1 /// <summary>周期定常熱負荷計算テスト</summary>
 2 private static void periodicUnsteadyStateTest()
 3 {
 4   double CAP_CL = -2000;  //冷房能力
 5   double CAP_HT = 2000;   //暖房能力
 6   double[] DBTSET = new double[] { 26, 22 };  //設定温度
 7   double[] HRTSET = new double[] { 0.0105, 0.0066 };  //設定湿度
 8
 9   //気象データ
10   double[][] dbt = new double[2][];
11   double[][] hrt = new double[2][];
12   double[][] rad = new double[2][];
13   dbt[0] = new double[] { 24.9, 24.7, 23.8, 24.2, 24.2,...//以下略
14   dbt[1] = new double[] { 4.1, 4.2, 4.6, 5, 4.4, 3.8, 4.8,...//以下略
15   hrt[0] = new double[] { 12.9, 12.8, 13.9, 14.6, 14.7, 14.6,...//以下略
16   hrt[1] = new double[] { 4.2, 4, 4, 3.6, 4, 3.7, 3.9,...//以下略
17   rad[0] = new double[] { 0, 0, 0, 0, 0, 0, 93, 288, 465,...//以下略
18   rad[1] = new double[] { 0, 0, 0, 0, 0, 0, 0, 1, 146, 318,...//以下略
19
20   //建物データを作成
21   BuildingThermalModel bMdl = makeSunSpaceBuildingModel();
22
23   //収束判定配列
24   double[][][] temp = new double[2][][];
25   double[][][] load = new double[2][][];
26   for (int i = 0; i < temp.Length; i++)
27   {
28     temp[i] = new double[2][];
29     load[i] = new double[2][];
30     for (int j = 0; j < temp[i].Length; j++)
31     {
32       temp[i][j] = new double[24];
33       load[i][j] = new double[24];
34     }
35   }
36
37   Sun sun = new Sun(Sun.City.Tokyo);
38   DateTime dTime;
```

†1　日射量の記録方法はデータソースによってマチマチで、過去1時間の日射量を毎正時のデータとするもの、将来1時間の日射量を毎正時のデータするもの、前後30分の日射量を毎正時のデータとするものなどがある。データソースの記録方式を確認せずに直散分離を行うと大きな誤差が生じるため、注意が必要である。

†2　2室の単純なモデルのため、ここでは簡易化して1回の判定で終わりにしたが、実際には25.2.2節で述べたように、空調室は繰り返し計算によって解除していく必要がある。

†3　乾球温度と顕熱負荷では絶対値に大きな違いがあるため、本来であれば誤差率として積算すべきであるが、0割の判定などが煩雑なため、ここでは簡略化した。結果に大差はない。

```
39  for (int ssn = 0; ssn < 2; ssn++) //ssn=0 で夏季、ssn=1 で冬季の計算
40  {
41    if (ssn == 0) dTime = new DateTime(2001, 7, 20, 0, 0, 0);
42    else dTime = new DateTime(2001, 1, 20, 0, 0, 0);
43    double err = 0;
44    while (true)
45    {
46      int hr = dTime.Hour;
47      //日射は過去 1 時間の積算データのため、太陽位置を 30 分ずらす
48      sun.Update(dTime.AddMinutes(30));
49      sun.SeparateGlobalHorizontalRadiation(rad[ssn][hr], Sun.SeparationMethod.Erbs);
50      bMdl.UpdateOutdoorCondition(dTime, sun, dbt[ssn][hr], hrt[ssn][hr], 0);
51
52      if (8 < dTime.Hour && dTime.Hour < 18)
53      {
54        //空調時間帯
55        bMdl.ControlDrybulbTemperature(0, 0, DBTSET[ssn]);
56        bMdl.ControlDrybulbTemperature(0, 1, DBTSET[ssn]);
57        bMdl.ControlHumidityRatio(0, 0, HRTSET[ssn]);
58        bMdl.ControlHumidityRatio(0, 1, HRTSET[ssn]);
59      }
60      else
61      {
62        //非空調時間帯 (自然室温)
63        bMdl.ControlHeatSupply(0, 0, 0);
64        bMdl.ControlHeatSupply(0, 1, 0);
65        bMdl.ControlWaterSupply(0, 0, 0);
66        bMdl.ControlWaterSupply(0, 1, 0);
67      }
68      //熱平衡予測
69      bMdl.ForecastHeatTransfer();
70      //過負荷の場合には最大容量で成り行き計算
71      for (int i = 0; i < 2; i++)
72      {
73        double hs = bMdl.MultiRoom[0].Zones[i].HeatSupply;
74        if (hs < CAP_CL) bMdl.ControlHeatSupply(0, i, CAP_CL);
75        else if (CAP_HT < hs) bMdl.ControlHeatSupply(0, i, CAP_HT);
76      }
77      bMdl.ForecastHeatTransfer();
78      //熱平衡を確定
79      bMdl.FixHeatTransfer();
80
81      //収束誤差
82      ImmutableZone[] zn = bMdl.MultiRoom[0].Zones;
83      for (int i = 0; i < zn.Length; i++)
84      {
85        err += Math.Abs(temp[ssn][i][dTime.Hour] - zn[i].Temperature);
86        err += Math.Abs(load[ssn][i][dTime.Hour] - zn[i].HeatSupply);
87        temp[ssn][i][dTime.Hour] = zn[i].Temperature;
88        load[ssn][i][dTime.Hour] = zn[i].HeatSupply;
89      }
90      //収束判定
91      if (dTime.Hour == 23)
92      {
93        dTime = dTime.AddHours(-23);
94        if (err < 0.0001) break;
95        else err = 0;
96      }
97      else dTime = dTime.AddHours(1);
98    }
99  }
100
101 //出力処理
102 using (StreamWriter sWriter = new StreamWriter
103 ("heatLoadTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
104 {
105   sWriter.WriteLine
106     ("時刻,Z0_DB,Z0_HLS,Z1_DB,Z1_HLS,Z0_DB,Z0_HLS,Z1_DB,Z1_HLS");
107
108   for (int i = 0; i < 24; i++)
109   {
110     sWriter.Write(i);
111     for (int ssn = 0; ssn < 2; ssn++)
112     {
113       sWriter.Write("," + load[ssn][0][i] + "," + load[ssn][1][i]
114        + "," + temp[ssn][0][i] + "," + temp[ssn][1][i]);
115     }
116     sWriter.WriteLine();
117   }
118 }
```

計算結果をグラフ化すると図 25.9 が得られる。上段は夏季、下段は冬季である。

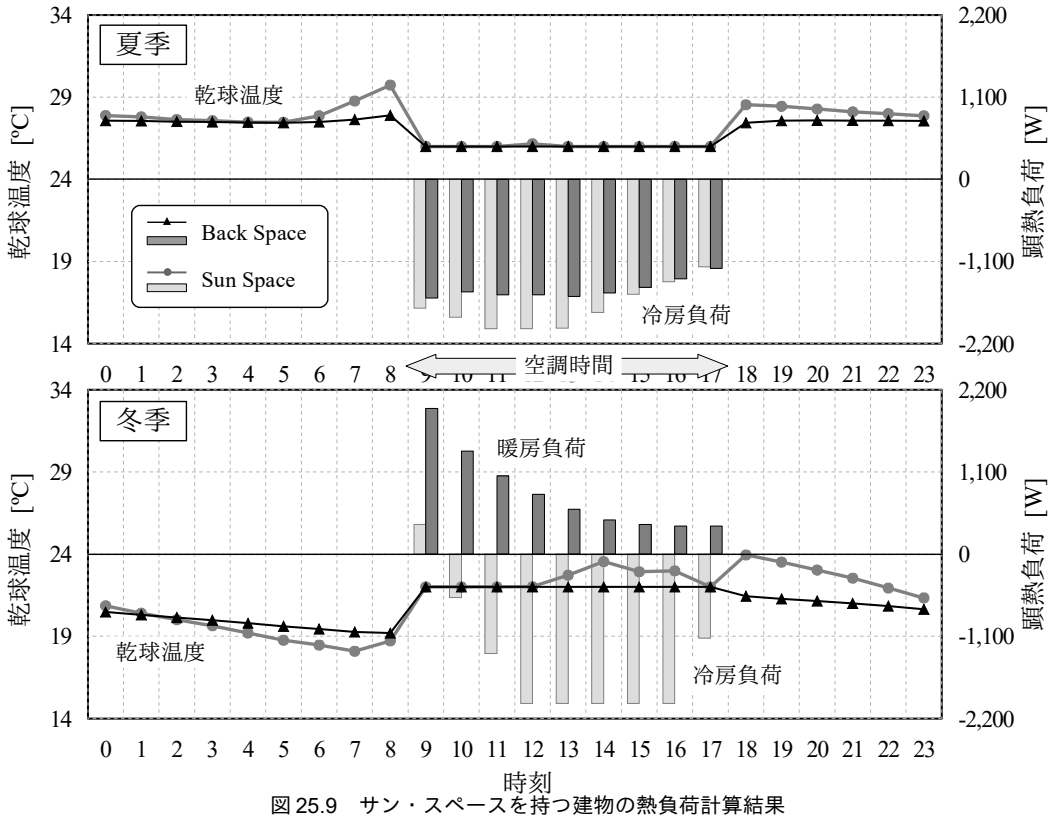


図 25.9　サン・スペースを持つ建物の熱負荷計算結果

夏季の非空調時間帯をみると、7~8 時に日が昇ると、サンスペースの自然室温がバックスペースよりも早く上昇することがわかる。外壁表面積自体はバックスペースのほうが大きいが、窓面から入射する日射負荷が大きいため、冷房負荷はサンスペースの方が大きいことがわかる。12 時に少しサンスペースで過負荷が発生するが、他の時間帯に関しては設備容量が不足することはない。

冬季においてもサンスペースの日射負荷は大きく、立ち上がり時に少し暖房負荷が発生する以外は、日中はすべて冷房負荷である。また、日中には設備容量が不足し、22 ºC が維持できなくなる。一方、バックスペースは直接的に日射熱取得が無いため、終日、暖房負荷となる。夏季とは異なり、夜間に蓄冷された効果により、最大負荷は明け方に発生する。コンクリートなどの熱容量の大きい建築物ではこのような形でピーク暖房負荷が発生することが多い。従って、正月休みなどの長期休業明けの暖房に注意する必要がある。なお、このように熱容量の大きい躯体によって立ち上がり時に発生する大きな負荷を特に蓄熱負荷と呼ぶ。

ところで本問の計算対象は、建物の熱負荷計算の検証法を定めた BESTEST[25.24)]と呼ばれる基準に記載されている建物である。熱負荷計算やエネルギーシミュレーションの検証や相互比較に関しては、この他にもいくつかの報告があるため[25.25) 25.26) 25.27) 25.28)]、デバッグ作業の一貫としてこれらの問題への適合性を確認すると良い。

【第 25 章 記号表】

A	：壁表面積 [m²]	NWS_q	：q 番目のゾーンに属する壁表面の数
a_s	：日射吸収率 [-]	NZN_i	：i 番目の室に属するゾーンの数
BF	：表面温度計算のための係数（B 側）	$\alpha_{(c+r)}$	：総合熱伝達率 [W/(m²·K)]
C	：熱容量 [J/K]	$\alpha_{(r)}$	：放射熱伝達率 [W/(m²·K)]
c_{pma}	：湿り空気の定圧比熱 [kJ/(kg·K)]	$\alpha_{(c)}$	：対流熱伝達率 [W/(m²·K)]
F	：形態係数 [-]	Q_{SW}	：短波長放射 [W]
FF	：表面温度計算のための係数（F 側）	$Q_{W\tau}$	：透過日射熱取得 [W]
G	：ゲブハルトの吸収係数 [-]	$Q_{W\alpha}$	：吸収日射熱取得 [W]
HG	：熱取得 [W]	R_P	：ペリメータ床への短波長按分比 [-]
HL	：熱負荷 [W]	R_{SLW}	：壁表面への入射 [W/m²]
I_{BSW}	：多重反射後の屋外放出日射 [W]	T	：絶対温度 [K]
IF	：表面温度計算のための係数（壁内部）	T_{Sol}	：相当温度 [K]
k_c	：対流熱伝達率比[-]	W_{ZN}	：ゾーンの絶対湿度 [kg/kg]
k_r	：放射熱伝達率比[-]	ρ	：反射率 [-]
M	：空気の質量 [kg]	ε_{LW}	：長波長放射率 [-]
m_a	：換気量 [kg/s]	ϕ	：基準化したゲブハルトの吸収係数
NQ	：すべてのゾーンの数	Δt	：タイムステップ [sec]
NRM	：室の数	σ	：ステファン・ボルツマン係数
NS	：すべての壁表面の数		=5.67×10-8 W/(m²·K⁴)

添字：

B	：裏側	q	：ゾーンの通し番号
c	：対流成分	r	：放射成分
F	：表側	S	：顕熱
FN	：家具など	s	：壁表面の通し番号
L	：潜熱	WS	：壁表面
OA	：外気	ZN	：ゾーン

【第 25 章 参考文献】

25.1） 石田建一, 宇田川光弘: 換気および壁面相互ふく射を考慮した多数室室温・熱負荷計算法, 日本建築学会計画系論文報告集 (381), 46-55, 1987

25.2） 宇田川光弘: パソコンによる空気調和計算法, 第 8 章, 第 10 章, オーム社, 1986

25.3） 空気調和衛生工学便覧, 第 14 版, 基礎編, p.88

25.4） 宿谷昌則 : 数値計算で学ぶ光と熱の建築環境学, 第 3 章 建築光伝達の基礎, 丸善, 1993.7

25.5） 中村泰人 : 建築都市空間内の人体に対する熱放射場の表現方法について , 日本建築学会計画系論文報告集, Vol.376, pp.29-35, 1987

25.6） 山崎均: 多角形の形態係数計算プログラム, 日本建築学会研究報告 九州支部. 2, 計画系, (26), 41-44, 1982

25.7） 幾島毅, 鈴木邦彦, 吉田一: モンテカルロ法による熱放射形態係数の算出-計算精度および計算時間について面積積分法との比較, 日本原子力学会誌, 30(6), pp.548-556, 1988

25.8） 長谷川房雄, 石川善美: 形態係数を用いた室内相互輻射計算の誤差の程について, 日本建築学会東北支部研究報告集, (24), pp.61-64, 1974

25.9） 蓑輪雅好: 矩形面に対する立位豚の全方位形態係数, 農業施設, Vol. 34, 2003-2004, No.2, pp.101-112

25.10） 松尾陽: 熱, 新建築学体系 10, 彰国社, pp.67-71, 1984

25.11） 松尾陽: 熱負荷計算に使用する擬似的全形態係数について, 日本建築学会大会学術講演梗概集, pp.25-26, 2003

25.12） G.H. Derricka: A three-dimensional analogue of the Hottel string construction for radiation transfer, International Journal of Optics, Vol.32,Issue 1, pp.39-60,1985

25.13） 石野久彌, 郡公子: 事務所建築における家具類の熱的影響に関する実測・実験研究, 日本建築学会計画系論文報告集, Vol.372, pp.59-66, 1987

25.14） 谷本潤: 建築室内環境における調湿作用に関する研究, 早稲田大学博士学位請求論文, 1992

25.15） 石野久彌: 最大熱負荷の精度に関する考察, 日本建築学会大会学術講演梗概集, pp.709-710, 1982.9

25.16） 郡公子, 石野久彌: 温熱環境とエネルギー消費量の同時評価法に関する研究, 日本建築学会計画系論文報告集, 第 365 号、pp.40-48, 1986.7

25.17） 久保木真俊, 村上周三, 石野久彌, 郡公子: 外皮・躯体と設備・機器の総合エネルギーシミュレーションツール「BEST」の開発（その 82） ゾーン間換気と相互影響の解析, 空気調和・衛生工学会大会学術講演論文集, pp.1691-1694, 2011

25.18） 郡公子, 石野久彌: 設計用非空調隣室温度に関する研究, 日本建築学会大会学術講演梗概集, pp.761-762, 1989

25.19） 佐藤誠, 石野久彌 他: 試して学ぶ熱負荷 HASPEE ~新最大熱負荷計算法~, p.130 表 6.2, 丸善出版, 2012

25.20） 空気調和・衛生工学会 SHASE-G 1008-2016 建物エネルギーシミュレーションツールの評価手法に関するガイドライン, 2016

25.21） 中原信生, 島田謙児, 湯澤秀樹: 暖房予熱時の装置容量と予熱負荷係数の推定法に関する研究, 空気調和・衛生工学会論文集, pp.33-43, No.38, 1988
25.22） 結城浩: Java 言語で学ぶデザインパターン入門 マルチスレッド編, SB クリエイティブ, 2006
25.23） Clay Bresherars: 平行コンピューティング技法-実践マルチコア/マルチスレッドプログラミング, 2009
25.24） R. Judkoff, J. Neymark: International Energy Agency Building Energy Simulation Test (BESTEST) and Diagnostic Method, 1995, National Renewable Energy Laboratory, U.S. Department of Energy
25.25） 宇田川光弘: 標準問題の提案 住宅用標準問題, 日本建築学会環境工学委員会熱環境運営委員会 第 15 回熱シンポジウムテキスト, 1985
25.26） 滝沢博: 標準問題の提案 オフィス用標準問題, 日本建築学会環境工学委員会熱環境運営委員会 第 15 回熱シンポジウムテキスト, 1985
25.27） 日本建築設備士協会 空気調和設備シミュレーション研究会: 国内の空調装置シミュレーションプログラムの比較, 空気調和・衛生工学 第 58 巻, 第 7 号, pp.65-78
25.28） SHASE G 1008-2016 建物のエネルギーシミュレーションツールの評価手順ガイドライン , 平成 28 年 3 月, 空気調和・衛生工学会標準化委員会, 空調システムのエネルギーシミュレーションツール評価法ガイドライン作成小委員会
25.29） 宇田川光弘: 標準問題の提案（住宅用標準問題）, 日本建築学会環境工学委員会 熱分科会第 15 回熱シンポジウム, pp.23-33, 1985
25.30） 富樫英介: 日射のペリメータ配分率に関する研究, 日本建築学会大会 学術講演梗概集, pp.27-28, 2016

第26章　空気調和機
(Air Conditioner)

26.1　概要

空気調和機（以降、空調機）は、温度調整（冷却と加熱）、湿度調整（加湿と除湿）、清浄度調整（除塵）を行った上で空気を搬送する機器である。これらの調整[†1)]を行うために空調機内にはいくつかの細かな機器が組み込まれており、第 9 章のプレートフィン付管熱交換器（コイル）、第 16 章のファン、第 11 章の空気対空気熱交換器、加湿器、フィルタなどが代表的な構成要素である。

調整した空気の搬送方法としては単一のダクトを用いる単一ダクト方式と、2 本のダクトを用いて冷風と温風をそれぞれ独立して供給できる二重ダクト方式がある。二重ダクト方式は冷風と温風を混合させることができるため、供給できる空気状態の自由度は大きいが、スペース・コストともに必要になるため、今日では殆どの施設では単一ダクト方式が採用されている。

風量制御に関しては定風量方式と可変風量方式とがある。定風量方式は予め設計した一定量を各ゾーンに供給する方式であり、人間の在不在の状況などに対してきめ細やかな制御はできず、温湿度状態も完全に一定に維持することは難しい。一方、可変風量方式は適当なグループごとに風量を調整するための機構が設けられており、給気する室の負荷傾向に多少の違いがあっても風量で調整ができるという利点がある。また、少ない風量しか求められない場合には、ファンの回転数制御を行うことで空気搬送動力を大きく削減できるため、消費エネルギーの観点からも優れている。

本書では単一ダクト定風量方式と単一ダクト可変風量方式の空調機についてモデル化を行う。また、近年、採用事例の多い、最小外気取入制御と外気冷房制御を行った場合と全熱交換器を導入した場合についても計算法を示す。

写真 26.1　日本で初めて空調が行われた富士瓦斯紡績保土ヶ谷工場（横浜開港資料館所蔵）

†1　英語では Air Conditioning であり、この用語は W. H. Carrier による空調システムが導入された紡績工場の技師である Stuart W. Cramer が 1906 年に初めて用いた[26.1)]。我が国では直訳である「空気調整」という言葉が使われたこともあったが、今日では「空気調和」という表現が主流である。再度英訳すると Air Harmonizing であり、むしろこの方が空調設計者が目指す室内空間を表現できているようにも思える。

26.2 理論

26.2.1 湿り空気線図上の動き

1）冷却・除湿・再熱運転

図 26.1 に冷却・除湿・再熱運転時の湿り空気状態を示す[†1)]。以降、添字の *OA*, *RA*, *CC*, *HC, HM, AHU, SF, RF*, *i*, *o*, *set* はそれぞれ外気、還気、冷水コイル、温水コイル、加湿器、空調機、給気ファン、還気（排気）ファン、入口、出口、設定値、を表わすものとする。

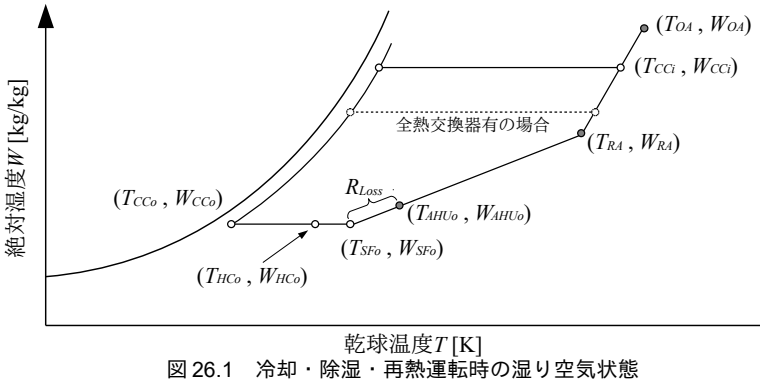


図 26.1 冷却・除湿・再熱運転時の湿り空気状態

還気 *RA* と外気 *OA* が混合され冷水コイルの入口状態 *CCi* となる。ただし全熱交換器があれば熱回収効果により、図の破線で示すように湿り空気状態が還気 *RA* 側に近づく。冷水コイル入口 *CCi* から冷水コイル出口 *CCo* に至るまでの状態変化については既に第 9 章において解説した。室内の潜熱負荷を除去するために必要な除湿を行うと、空気の乾球温度が必要以上に低下する可能性がある。温度と湿度の両者を完全に制御するためには温水コイル（再熱コイル）などを用いて再熱を行い、T_{HCo} まで乾球温度を上げる必要がある。ただし、この方法はエネルギー消費量が大きくなるため、厳密な温湿度制御が求められない多くの施設では、再熱コイルを設けずに湿度は成り行きとすることが多い[†2)]。この場合には T_{CCo}=T_{HCo} である。本書では再熱コイルの無い空調機を前提とする。

温水コイル出口空気 *HCo* はファンによって圧力を上昇させられて室に供給されるが、ファンが与えたエネルギーは最終的に熱に転化するため、温度が上昇する。従って給気ファン出口空気の温度 T_{SFo} は温水コイル出口空気 T_{HCo} よりも高い温度となる[†3)]。

空調機出口空気はダクトによって室に運ばれるが、この際に熱損失とリークによる空気自体の損失が発生する。質量と温度の両者を複合的に扱うと計算が煩雑になるため、本書のモデルでは質量損失を温度損失に合算して全体で比率 R_{Loss} の熱損失が発生するとして計算を行う。また厳密にはそれぞれのゾーンに向かうダクトの長さや種類、ダクト周辺の温湿度環境にも依存するが、計算上は式 26.1 と式 26.2 に示すように還気温湿度との差に対して一定の比率を乗じることで熱ロスを表現する。

$$T_{AHUo}=T_{SFo}+(T_{RA}-T_{SFo})R_{Loss}=T_{SFo}(1-R_{Loss})+T_{RA}R_{Loss} \tag{26.1}$$

$$W_{AHUo}=W_{SFo}+(W_{RA}-W_{SFo})R_{Loss}=W_{SFo}(1-R_{Loss})+W_{RA}R_{Loss} \tag{26.2}$$

図 26.1 の灰色に着色したプロットが空調機にとっての出入口条件である。これらの空気状態とそ

†1 説明のためにかなりデフォルメした湿り空気線図であるため注意されたい。

†2 顕熱交換器を設けることで還気 *RA* と冷水コイル出口空気 *CCo* との間で熱交換を行い、再熱に必要なエネルギーを抑制する機種がある。また、近年、採用事例が増えているデシカント空調機も温度と湿度の両方を独立に制御することができる。

†3 給気ファンと排気（還気）ファンの発熱があるが、この発熱による温度上昇をどこでどのように見込むかは、ファンの配置に依存する。詳細は後述の全熱交換器の項で述べる。

の風量は、定風量方式か否かの別やその他の省エネ手法の有無によって変化する。これら（入口空気状態 T_{OA}, W_{OA}, T_{RA}, W_{RA}、出口空気設定値 $T_{AHUo,set}$, $W_{AHUo,set}$、風量 m_{OA}, m_{RA}）の計算法は 26.2.2 節~26.2.4節で解説する。

全熱交換器、冷水コイル、温水コイルについて入口空気条件から出口状態を計算する方法については第 9 章および第 11 章で解説しており、図 26.1 の外気 *OA* および還気 *RA* から温水コイル出口空気 *HCo* までの変化は既に計算できる。*HCo* から *SFo* までの湿り空気の変化は、ファンのエネルギー供給による温度上昇である。ファンの出入口空気温度（T_{Fi} と T_{Fo}）の関係は式 26.3 で表現される。W_F [kW]はファンの理論動力、c_{pma} [kJ/(kg·K)]は湿り空気の定圧比熱、m_{SA} [kg/s]は給気風量である。ファンの理論動力 W_F は第 16 章で示したように、風量と圧力から計算できる。

$$T_{Fo} = T_{Fi} + \frac{W_F}{c_{pma} \cdot m_{SA}} \tag{26.3}$$

2）加熱・加湿運転

図 26.2 に加熱・加湿運転時の湿り空気状態を示す。還気 *RA* と外気 *OA* が混合され温水コイルの入口状態 *HCi* となる。冷却運転時と同様、全熱交換器があれば還気側に近づく。温水コイル出口状態 *HCo* から加湿器により加湿されて *HMo* となり、さらにファン発熱を加えるとファン出口状態 *SFo* となる。冷却除湿運転と同様に、空調機出口状態から室給気までの間ではダクトの熱損失 R_{Loss} を見込む。図 26.2 では給気絶対湿度設定値が室内絶対湿度よりも高いが、低い場合もある。冬季の潜熱負荷は室内の潜熱発生と隙間風負荷の大小関係によって正と負の値の両方をとり得るためである。

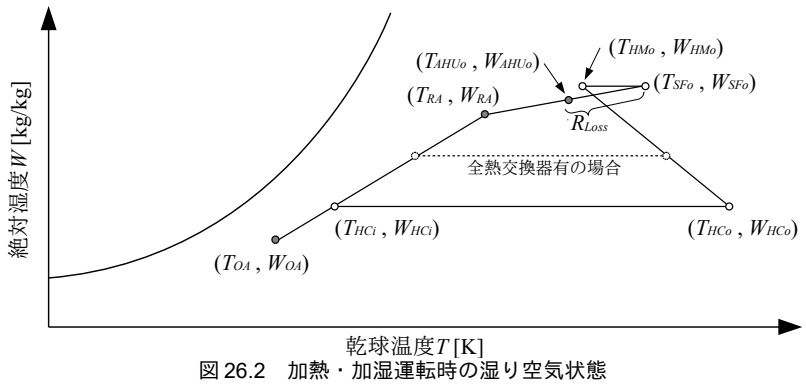


図 26.2　加熱・加湿運転時の湿り空気状態

・加湿器[26.2)]

加湿の方式は大きく水加湿と蒸気加湿に分けられ、図 26.2 は水加湿の例である。水加湿は水を浸透させた加湿材に空気を通過させて加湿する滴下浸透気化式加湿器と、超音波や高圧スプレーなどで水分を直接に霧化して加湿する水噴霧式加湿器に分けられる。蒸気加湿は蒸気を直接に外部から投入する方式と、電熱線などを利用して加湿器自体が蒸気を製造する方式があり、後者の場合には直接に電気エネルギーが必要となることに注意する必要がある[†1)]。

水加湿であれ蒸気加湿であれ、加湿は熱水分比（絶対湿度変化に対する比エンタルピーの変化量）一定で行われ、それぞれ湿り空気線図上では図 26.3 で示される。水加湿の場合には湿球温度一定の線上を動くが、反復計算を回避するために比エンタルピーが一定の線で近似することもある。蒸気の

†1　もちろん蒸気加湿であっても蒸気を製造するためにはエネルギーを投入している。しかし電熱線はジュール熱を用いた方式であるため、一次エネルギーでの効率はガスなどによる蒸気製造に大きく劣る。

場合にも厳密には蒸気圧と温度から蒸気の比エンタルピーを計算する必要があるが、計算を簡略化するため、湿り空気線図上を垂直移動する（乾球温度一定）とみなすことが多い。

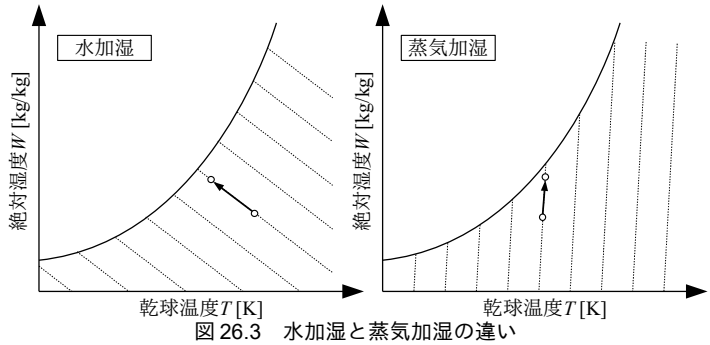


図 26.3　水加湿と蒸気加湿の違い

加湿器の能力は飽和効率 η_{hmS} [-]で表現され、飽和絶対湿度 W_S [kg/kg]と加湿器出入口絶対湿度（W_{HMi}と W_{HMo}）を用いて式 26.4 で計算する。飽和効率は給水量で制御できるが、加湿材や蒸発吸収距離の大きさの限界により、滴下浸透気化式加湿器では最大で 80 %程度、超音波式で 50 %程度、高圧スプレー式で 30 %程度である。一方、蒸気加湿の場合には 100 %と扱って良い[26.2)]。入口絶対湿度と飽和効率が与えられた場合の、出口絶対湿度の最大値は式 26.5 である。

$$\eta_{hmS}=\frac{W_{HMo}-W_{HMi}}{W_S-W_{HMi}} \tag{26.4}$$

$$W_{HMo}=(1-\eta_{hmS})W_{HMi}+\eta_{hmS}W_S \tag{26.5}$$

加湿器に供給される水はすべてが有効に加湿に使われるわけではなく、一部は排水される。有効に加湿に使われた比率を給水有効利用率 η_{hmW} [-]と呼び、式 26.6 で計算する。m_{hmW} [kg/s]は加湿量、m_{sW} [kg/s]は給水量である。給水有効利用率は、蒸気加湿で 75~95 %、滴下浸透気化式で 30~70 %程度、超音波式で 80~100 %程度、高圧スプレー式で 30~50 %程度である[26.2)]。

$$\eta_{hmW}=\frac{m_{hmW}}{m_{sW}} \tag{26.6}$$

26.2.2　単一ダクト定風量方式

1）システム系統図

図 26.4 に単一ダクト定風量方式によるダクト系統図の例を示す。

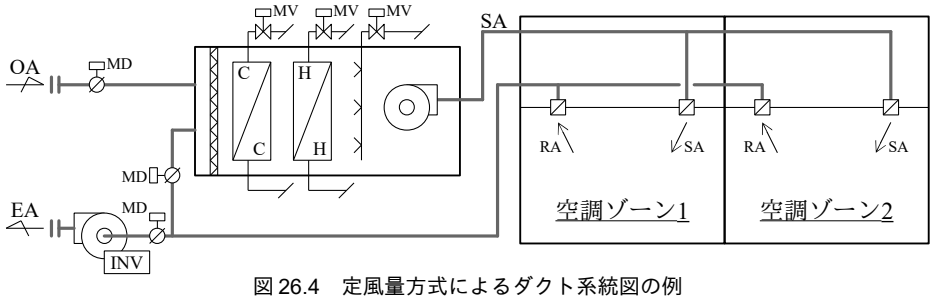


図 26.4　定風量方式によるダクト系統図の例

本例では、空調機からの給気（SA: Supply Air）は分岐して 2 つの空調ゾーンの給気口に接続される。室からの還気（RA: Return Air）は排気ファンに接続され、一部が排気（EA: Exaust Air）となり、一部が空調機に戻って外気（OA: Outdoor Air）と混合される。空調機にはエアフィルタ、冷水コイル、温水コイル、加湿器が設置されており、除塵、冷却（または加熱）、加湿が行われた後、給気ファンによって再び空調ゾーンに送風される。

上記は 1 つの典型的なシステムであるが、空調機およびダクトの構成は無数にある。例えば、空調機単体であれば、冷水コイルのみを配置した機種、冷水コイルと温水コイルを合わせて冷温水コイルとした機種、冷温水ではなく直接に冷媒を蒸発させる直膨コイルとした機種、ファンを 2 台設けて機器内で台数制御をする機種など、種類が多い。また、ダクトの構成としても、還気を取らずに全量外気とする場合、逆に外気を取らずに全循環とする場合、給気口はゾーン毎に設ける一方で還気口は 1 箇所で集中してとる場合、定風量ユニットを設けてゾーン毎の風量を保証する場合など、様々である。大切なことは、計算対象のシステム系統図をよく読み取ることで、空気の流れと設計の狙いを正しく掴み、計算モデルに反映することである。

2）給気温湿度の計算

空調機が給気するゾーンの総数を N とする。n 番目のゾーンの顕熱負荷を $HL_{S,n}$ [kW]、乾球温度を $T_{ZN,n}$ [K]、給気風量を $m_{SA,n}$ [kg/s]とすると、各ゾーンが必要とする給気温度 $T_{AHUo,n}$ [K]は式 26.7 で表現できる。ただし、顕熱負荷は暖房を正、冷房を負とする。同様に、潜熱負荷を $HL_{L,n}$ [kg/s][†1)]、絶対湿度を $W_{ZN,n}$ [kg/kg]とすれば、各ゾーンが必要とする給気絶対湿度 $W_{SA,n}$ [kg/kg]は式 26.8 となる。潜熱負荷は加湿を正、除湿を負とする。

$$T_{AHUo,n} = T_{ZN,n} + \frac{HL_{S,n}}{c_{pma} \cdot m_{SA,n}} \tag{26.7}$$

$$W_{AHUo,n} = W_{ZN,n} + \frac{HL_{L,n}}{m_{SA,n}} \tag{26.8}$$

定風量方式の場合にはゾーンの給気量は固定のため、必ずしもすべてのゾーンの温湿度設定値を同時に満足することはできない。制御の方法は大きく 2 つあり、1 つはすべてのゾーンの温湿度を代表する値として還気の温湿度を制御対象とする方法である。もう 1 つは、特定のゾーンの温湿度を制御対象とする方法である。いずれの制御方法をとるにせよ、定風量方式の場合の基本は、負荷の傾向が似通ったゾーンを 1 つの空調機の対象としてまとめるということである。逆に言えば、負荷傾向の異なったゾーンをまとめてしまうと、どこかのゾーンの温湿度制御を犠牲にせざるを得ない。

3）還気条件の計算

還気流量は各室の還気量を積算して式 26.9 で計算する。多くの場合、便所や喫煙室排気としての利用や屋外への漏洩によって還気量 m_{RA} は給気量 m_{SA} を下回る。特に全熱交換器を導入した場合には還気が持つ熱が省エネルギー効果に影響するため、給気と還気のバランスに注意が必要がある。

$$m_{RA} = \sum_{n=0}^{N} m_{RA,n} \tag{26.9}$$

空調機入口の還気温度 T_{RA} と還気湿度 W_{RA} は各ゾーンの温湿度を還気量で重み付けて式 26.10 と式 26.11 で計算する。

$$T_{RA} = \left(\sum_{n=0}^{N} m_{RA,n} T_{ZN,n} \right) / m_{RA} \tag{26.10}$$

$$W_{RA} = \left(\sum_{n=0}^{N} m_{RA,n} W_{ZN,n} \right) / m_{RA} \tag{26.11}$$

†1　潜熱負荷の単位としては W や kW を用いることが多いが、計算を簡単にするため、本書では以降の章も含めて水分量で表現する。kW に変換するためには水の蒸発潜熱（2500 kJ/(kg·K)）を乗ずれば良い。

26.2.3　単一ダクト可変風量方式

1）システム系統図

図 26.5 に単一ダクト変風量方式によるダクト系統図の例を示す。

定風量方式とは異なり、給気ダクトにゾーン別に可変風量（VAV: Variable Air Volume）ユニットが設けられる†1)。空調機のファンにはインバータを設けて回転数制御を可能とし、要求風量に応じて風量調整を行う。ファンの回転数は VAV コントローラから送られる開度情報と風量設定値により制御され、全開となる VAV が無いように絞られる。従って、ポンプにおける DDC 連携制御と同様に、必要静圧が最小化するように制御されるとみなせる。

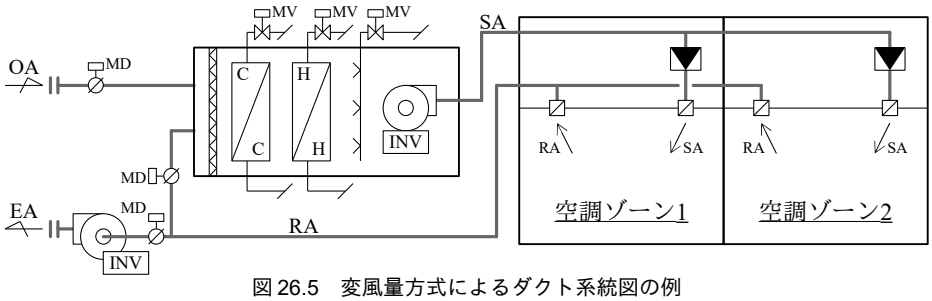


図 26.5　変風量方式によるダクト系統図の例

2）給気温湿度の計算

給気温湿度の計算式は定風量方式と同じ式 26.7 と式 26.8 であるが、可変風量方式の場合には給気風量 m_{SA} が未知である。従って、空調機が過負荷状態とならない範囲内で給気風量をできるだけ小さく抑えるための吹出温度を計算する必要がある。一方で、外気処理空調機の場合には換気のために必要な最小限度の外気の給気風量があり、過剰に低い出口温度としてはこの風量を下回ってしまう。このように空気搬送動力の削減や換気量の確保を目的に給気温度を再設定する制御を給気温度リセット制御と呼ぶ。なお、同一の負荷を処理するとすれば、風量を絞るとコイルの冷温水流量を増加させなければならないが、一般に水搬送動力に比較して空気搬送動力の方が大きいため、原則としては風量最小化を優先させた制御とすることが省エネルギー化には有効である。

計算の手順としては、まず、式 26.7 に VAV の最大風量 $m_{SAmax,n}$ [kg/s]を代入し、最大風量における給気温度 $T_{AHUomax,n}$ [K]を計算する。最も負荷条件の厳しいゾーンに対して給気温度を確保するため、AHU には式 26.12 の吹出温度が求められる。なお、空調機の能力は最大風量で最大化するため、この条件でも負荷が処理できなければ風量を絞る余地はない†2)。

$$T_{AHUomax,set}=\begin{cases} Min\left\{T_{AHUomax,n}\right\} & (\text{cooling}) \\ Max\left\{T_{AHUomax,n}\right\} & (\text{heating}) \end{cases} \tag{26.12}$$

負荷が処理可能であれば、式 26.7 に最小風量 $m_{SAmin,n}$ [kg/s]を代入し、最小風量における給気温度 $T_{AHUomin,n}$ [K]を計算する。空調機の出口温度の設定方法は 2 つあり、1 つは式 26.13 で吹出温度設定値 $T_{AHUomin,set}$ を求める方法である。最も負荷条件の厳しいゾーンを基準に吹出温度を設定する方法であるため、風量は最小化する。しかし，その他のゾーンで最適風量が最小風量を下回り、過冷却または過

†1　図 26.4 では記載を省略したが、定風量方式の場合には VD（Volume Dumper）または定風量（CAV :Constant Air Volume）ユニットを設置して風量が一定となるように調整する。

†2　厳密には風量を増加させることで給気と還気との温度差は減少するため、室温の絶対値によっては、むしろ風量を減少させることで負荷を取り除くことができるようになる可能性もある。しかし、空調機は能力不足に対しては風量を増大させるように制御されることが通常であるから、このような状態で均衡することは現実には無いだろう。

加熱が生じる可能性もある。もう 1 つの方法は式 26.14 で吹出温度設定値 $T_{AHUomin,set}$ を求める方法である。最も条件の緩やかなゾーンを基準に吹出温度を設定する方法であるため、すべてのゾーンで最適風量が最小風量以上となり、過冷却や過加熱は生じない。しかし、全体の風量は最小化しない。HASP/ACSS では前者を最小風量補償、後者を最大風量補償と呼んでいる[26.9)]。

$$T_{AHUomin,set}=\begin{cases} Min\left\{T_{AHUomin,n}\right\} & (\text{cooling}) \\ Max\left\{T_{AHUomin,n}\right\} & (\text{heating}) \end{cases} \tag{26.13}$$

$$T_{AHUomin,set}=\begin{cases} Max\left\{T_{AHUomin,n}\right\} & (\text{cooling}) \\ Min\left\{T_{AHUomin,n}\right\} & (\text{heating}) \end{cases} \tag{26.14}$$

式 26.13 または式 26.14 の設定温度で給気すれば、式 26.12 に比較してファン動力は小さくなるが、AHU がこの温度を実現できるか否かは別問題である。風量を下げるためには冷房時であれば吹出温度を下げ、暖房時であれば吹出温度を上げる必要があり、これは冷水コイルおよび温水コイルにとっては能力が出しづらい条件になるからである。そこで第 2 章 2.4.1 節で解説した一変数関数の最適化手法（本書では黄金探索法を採用）を用いて AHU が過負荷にならない範囲で給気温度を式 26.13 または式 26.14 の設定値 $T_{AHUomin,set}$ に近づける。具体的には図 26.6 に示すような評価関数を用いる（冷房時）。吹出温度を下げていくと、ある値で冷水コイルが過負荷状態になり出口温度設定値を満足できなくなる。この限界点が最適な吹出温度 $T_{AHUo,opt}$ [K]である。そこで、式 26.15 で評価関数を定義する。吹出温度が実現できない過負荷状態の場合には還気温度 T_{RA} を加える事で評価関数が同範囲において極小化しないようにする。吹出温度が変化すると各ゾーンへの給気量も変化するため、式 26.7 を風量 $m_{SA,n}$ について解いた式 26.16 で風量を求め、さらに式 26.10 を適用して AHU 還気温度を更新する。なお、図 26.6 は冷房時の評価関数であるが、暖房時には吹出温度および還気温度の正負を逆転させれば同じ方法を用いることができる。

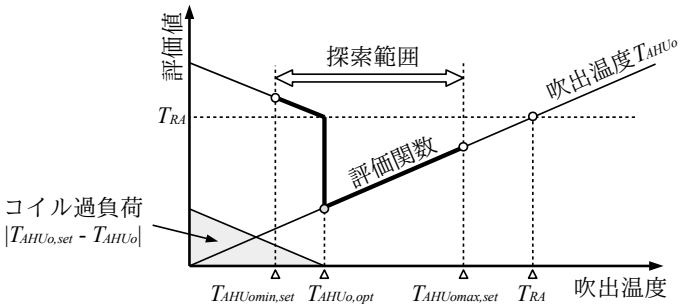


図 26.6　VAV 風量最小化のための評価関数

$$f_{err}(T_{AHUo})=\begin{cases} T_{AHUo} & (T_{AHUo,set}=T_{AHUo}) \\ T_{RA}+\left|T_{AHUo,set}-T_{AHUo}\right| & (T_{AHUo,set}\neq T_{AHUo}) \end{cases} \tag{26.15}$$

$$m_{SA,n}=\frac{HL_{S,n}}{c_{pma}(T_{AHUo}-T_{ZN,n})} \tag{26.16}$$

26.2.4　省エネ手法

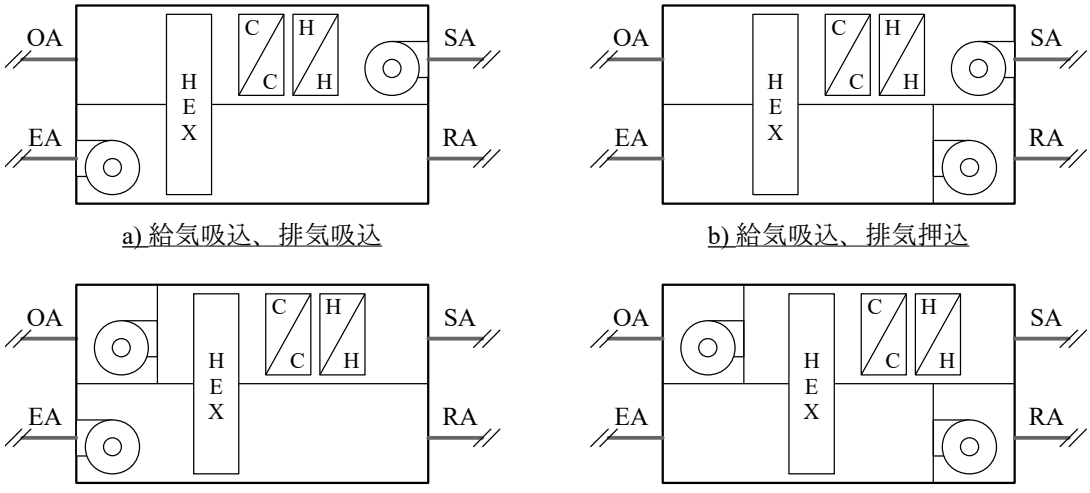
1）全熱交換器

全熱交換器の入口空気条件および風量が与えられれば、第 11 章で解説した方法で出口状態を計算することができる。入口空気条件の内、外気は直接に与えられ、また、還気温湿度は式 26.10 と式 26.11 で計算できる。風量収支としては式 26.17 が成立する。取り入れ外気量 m_{OA} [kg/s]と排気量 m_{EA}

[kg/s]は等しい場合もあるが、一部を便所排気や喫煙室排気などの局所排気に使ったり、外部への漏気があるために $m_{EA} < m_{OA}$ となる場合（$m_{RA} < m_{SA}$ となる）も多いことに注意する。

$$m_{SA} - m_{OA} = m_{RA} - m_{EA} \tag{26.17}$$

給気ファンと排気ファンを全熱交換器の前後のどちらに配置するかは設計者の裁量であり、図 26.7 に示すように 4 通りの構成が考えられる。計算上の問題はファン発熱による空気温度の上昇をどのように反映するかである。排気ファンが全熱交換器の後に配置される場合（図 26.7 の a)と c)）には、排気ファンの発熱は排気として屋外に放出されるが、全熱交換器の前に配置される場合（図 26.7 の b)と d)）には全熱交換器による回収熱量に影響を与える（冷房時は回収熱量減、暖房時は回収熱量増）。給気ファンの場合には配置の前後に関わりなくファン発熱が室へ供給されるが、配置によって冷水・温水コイルの入口条件が変わる。これらを厳密に場合分けした汎用的なモデルを作ることは非常に煩雑であるため、本書では a)の構成を前提とする。即ち、排気ファンによる顕熱供給は考慮せず、給気ファンによる温度上昇はコイル通過後に発生するものとする（図 26.1 および図 26.2 はこの前提で作成している）。なお、第 11 章で、全熱交換器における空気の逆流を防ぐ目的からパージセクターというじゃま板を設けることに触れた。このような逆流を防ぐという観点からは、給気側と排気側で全熱交換器出入口の圧力バランスの取りやすい a)または d)の構成が優れており、排気側が正圧となる b)の構成は原則としてとるべきではない。ただし、ファンが対角配置になるため、空調機のサイズは b)および c)に比較して a)および d)が大きくなる傾向となり、スペースは不利である。



a) 給気吸込、排気吸込　　b) 給気吸込、排気押込

c) 給気押込、排気吸込　　d) 給気押込、排気押込

図 26.7　給気ファンと排気ファンの位置

2）最小外気取入制御

室内空気の主たる汚染源は人間であり、粉塵、臭気、二酸化炭素（CO_2）などを発する。CO_2 は計測がしやすく、他の汚染と相関も高い。そこで CO_2 の濃度の計測値によって室内の汚染状況を評価し、外気量を必要最低限度に抑えて外気負荷を削減しようとする手法が、最小外気取入制御である。

室内の CO_2 濃度を計算するためには、人体の CO_2 発生量を予測する必要がある。大西らは実測調査をもとに式 26.18 に示す推定式を提案している。1 行目は実験から得られた回帰式であり決定係数 R^2 は 0.9792 である。また、M [W]は代謝量である。2 行目は代謝量 M [W]を体表面積 A_{Du} [m^2]、活動量

Met [met]、性別 C_s（女性 0、男性 1）の関数で表した式である。代謝量と体表面積の計算については第 27 章で解説するが、日本人男性がオフィス作業をしていることを前提とすると、体表面積 A_{Du} =1.7 m²、代謝量 *Met* = 1.1 程度であり、I_{CO2} = 0.02 m³/(h·人)となる。

$$\begin{aligned} I_{CO2} &= 1.575\times10^{-4} M + 3.693\times10^{-4} \\ &= 1.575\times10^{-4}(92.8\,A_{Du} + 83.9\,Met + 17.2\,C_s - 141.1) + 3.693\times10^{-4} \end{aligned} \tag{26.18}$$

室内 CO_2 濃度 C_{ZN} [m³/m³]、CO_2 発生速度が C_G [m³/s]、外気 CO_2 濃度が C_{OA} [m³/m³]、換気量が Q [m³/s]、室の気積が V [m³]のとき、物質収支は式 26.19 の微分方程式で表現できる。初期条件 $t = 0$ において室内 CO_2 濃度を $C_{ZN,0}$ [m³/m³]とし、式 26.19 を解くと式 26.20 に示す Seidel の式が得られる。式 26.20 は直接に換気量 Q については解くことはできない。そこで、t 秒経過後に CO_2 濃度 C_t を達成するために必要な換気量を求める場合には、式 26.19 を後退差分で近似して式 26.21 で計算する†1)。

CO_2 濃度は通常は m³/m³ ではなく百万分の 1 を表わす ppm（parts per million）という単位で表される。m³/m³ に換算するためには 10^{-6} を乗じれば良い。なお、ビル管理法上は CO_2 濃度の許容値は 1,000 ppm である。また、近年は都市部の CO_2 濃度が上昇傾向にあり、外気で 350~450 ppm 程度である。

$$V\frac{dC_{ZN}(t)}{dt} = Q(C_{OA} - C_{ZN}(t)) + C_G \tag{26.19}$$

$$C_{ZN}(t) = C_{OA} + (C_{ZN,0} - C_{OA})e^{-\frac{Q}{V}t} + \frac{C_G}{Q}\left(1 - e^{-\frac{Q}{V}t}\right) \tag{26.20}$$

$$Q = \frac{V(C_{ZN,t+1} - C_{ZN,t}) - C_G\Delta t}{(C_{OA} - C_{ZN,t+1})\Delta t} \tag{26.21}$$

【例題 26.1】

床面積 1,000 m²、天井高 2.7 m のオフィスについて、空調（換気）開始時点の CO_2 濃度が 500 ppm であったとする。オフィスの人員密度が 0.1 人/m²、外気の CO_2 濃度が 400 ppm であるとすると、1 時間後に CO_2 濃度が 1,000 ppm を下回るにはどれだけの換気量が必要か計算せよ。またその換気量を継続した場合、2 時間後の CO_2 濃度はいくらになるか計算せよ。

【解】

オフィスの人員は 0.1 人/m²×1,000 m² = 100 人であり、一人あたりの CO_2 発生量が 0.02 m³/(h·人)とすると、総発生量は 0.02×100 = 2 m³/h = 5.56×10⁻⁴ m³/s となる。また、オフィスの気積は 2.7 × 1,000 = 2,700 m³ である。以上を式 26.21 に代入すると、

$$Q = \frac{2700(1000\times10^{-6} - 500\times10^{-6}) - 5.56\times10^{-4}\times3600}{(400\times10^{-6} - 1000\times10^{-6})\times3600} = 0.301$$

となる。Q=0.301 を式 26.20 に代入すると

$$C_{ZN}(3600) = 400\times10^{-6} + (500-400)\times10^{-6}e^{-\frac{0.301}{2700}3600} + \frac{5.56\times10^{-4}}{0.301}\left(1 - e^{-\frac{0.301}{2700}3600}\right) = 0.001078\ \mathrm{m^3/m^3}$$

となり、差分化近似による誤差があるが、概ね 1,000 ppm である。さらにもう一時間、同一の換気量で時間を経過させると CO_2 濃度は、

$$C_{ZN}(3600) = 400\times10^{-6} + (1043-400)\times10^{-6}e^{-\frac{0.301}{2700}3600} + \frac{5.56\times10^{-4}}{0.301}\left(1 - e^{-\frac{0.301}{2700}3600}\right) = 0.001463\ \mathrm{m^3/m^3}$$

まで増加する。

3）外気冷房

中間期や冬季において冷房負荷が発生しており、室内空気に比較して外気のエネルギーが十分に低い場合がある。この場合には、外気を直接に室に供給することで室内空気を冷却することが可能であ

†1　室と室との間で空気の流れがあることを考慮するためには、後に第 25 章で示すように室間換気による CO_2 濃度の増減を含めて連立方程式で表現するという方法がある。しかし、熱や水分に比較して CO_2 を蓄積する物体が無い（と著者は理解している）ことを考えれば、それぞれの室において式 26.21 で完結した計算をしても結果に大きな誤差は生まれないだろう。

り、冷房に必要なエネルギーを削減することができる。これを外気冷房と呼ぶ。

図 26.8 に外気冷房有効範囲の例を示す。外気乾球温度が室温より低いとともに、比エンタルピーも室内空気を下回る必要がある。また、吹き出し空気が低くなり過ぎないように外気乾球温度の下限値も設ける。中間期においては外気が高湿度低温となることも多く、このような条件においては室内温湿度設定値を調整して外気冷房の条件拡大を図ることが有効である。熱的快適性を考慮した室内温湿度設定値の緩和については第 27 章の例題 27.2 で検討する。

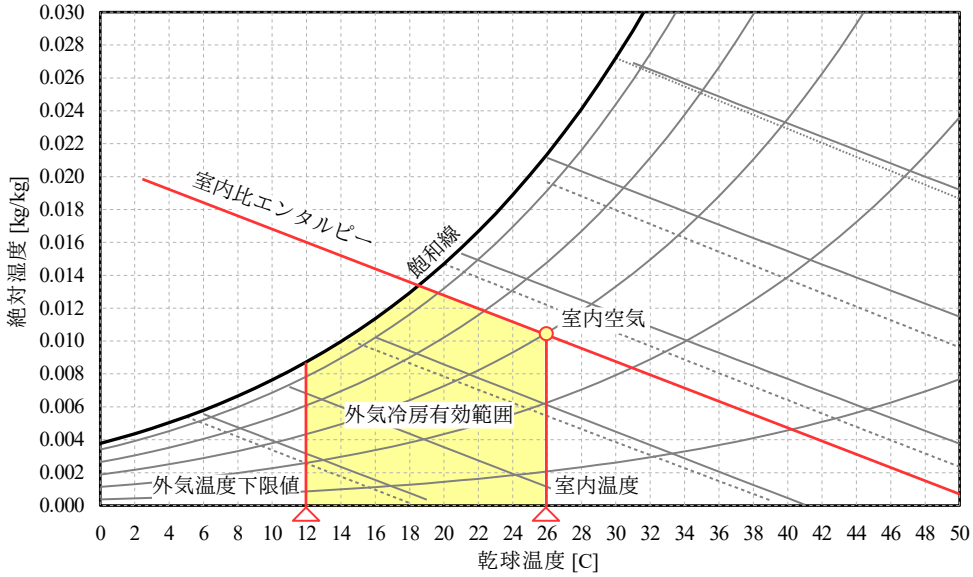


図 26.8　外気冷房有効範囲の例

図 26.8 は 1 例であり、比エンタルピーが下回っていても絶対湿度が高い場合には除湿が必要となるため、室内絶対湿度を基準に外気冷房有効範囲を限定するという考え方もある。また、サーバー対応の 24 時間空調においては外気温度下限値を設けないという考え方もある[26.6)]。設計や運用の方針を理解して有効範囲を判断する必要がある。外気冷房ではないが、山本らはアンケートと文献調査により日本における 42 物件の自然換気口自動解放条件について整理しており、採用件数としては図 26.8 に示すように上下限値と室内外比エンタルピー比較により判断するものが多いと報告している[26.8)]。

導入すべき外気量は、顕熱負荷、潜熱負荷、全熱負荷、のいずれを最小化するかによって異なる。顕熱負荷を最小化する場合には、給気ファン入口乾球温度 T_{SFi} [K]が所定の温度になるように式 26.22 により外気と還気を混合する。式 26.22 を外気量について解くと式 26.23 が得られる。

$$T_{SFi}=\frac{m_{OA}T_{OA}+m_{RA}T_{RA}}{m_{OA}+m_{RA}} \tag{26.22}$$

$$m_{OA}=m_{RA}\frac{T_{RA}-T_{SFi}}{T_{SFi}-T_{OA}} \tag{26.23}$$

同様に、潜熱負荷または全熱負荷を最小化する場合には式 26.24 または式 26.25 を用いて給気ファン入口絶対湿度 W_{SFi} [kg/kg]または比エンタルピー h_{SFi} [kJ/kg]を制御し、式 26.26 または式 26.27 を用いて外気量を計算する。

$$W_{SFi}=\frac{m_{OA}W_{OA}+m_{RA}W_{RA}}{m_{OA}+m_{RA}} \tag{26.24}$$

$$h_{SFi}=\frac{m_{OA}h_{OA}+m_{RA}h_{RA}}{m_{OA}+m_{RA}} \tag{26.25}$$

$$m_{OA}=m_{RA}\frac{W_{RA}-W_{SFi}}{W_{SFi}-W_{OA}} \tag{26.26}$$

$$m_{OA}=m_{RA}\frac{h_{RA}-h_{SFi}}{h_{SFi}-h_{OA}} \tag{26.27}$$

顕熱または全熱負荷を最小化するように外気量を制御した場合には、加湿負荷が増加する可能性もある。このような場面においては、冷水と温水の供給を行うそれぞれの熱源の効率についても注意する必要がある。例えば暖房熱源としてガスボイラ、冷房熱源としてターボ冷凍機を用いた設備があるとする。このとき、温水製造の一次 COP は 0.9 程度であるが、冷水製造の一次 COP は 2.3 程度である。従って、1 単位の暖房負荷が増加する一方で 1 単位以上の冷房負荷が削減されたからと言って、熱源効率によっては必ずしも全体のエネルギー消費量の削減につながらない可能性もある。上記の場合には、外気冷房によって 1 単位の冷熱負荷を減少させても、加湿のための温熱負荷が 0.39 = 0.9 / 2.3 以上増加してしまってはむしろ一次エネルギー消費量が増大する。

26.3　計算法

空調機は給気ファン、還気ファン、冷温水コイル、加湿器、全熱交換器などが組み合わされたシステムである。従って、これらの要素を表現したクラスのインスタンスを組み合わせてモデルを作ると表現が非常に簡単になる。そこで空調機の計算に関しては静的メソッドで対応するのではなく、空調機クラス（AirHandlingUnit）のインスタンスを生成することを前提とする。また、空調機単体ではなくゾーンとの相互影響が問題となる場合が多いため、空調機とゾーンを組み合わせたサブシステムの計算モデルについても解説する。

26.3.1　空調機クラスの初期化処理

プログラム 26.1 に外気導入制御方式と加湿方式の列挙型定義を示す。

プログラム 26.1　列挙型の定義

Popolo.HVAC.AirConditioner.AirHandlingUnit class

```
/// <summary>外気導入制御方式</summary>
public enum OutdoorAirCoolingControl
{
  /// <summary>無し</summary>
  None,
  /// <summary>乾球温度基準</summary>
  DrybulbTemperature,
  /// <summary>絶対湿度基準</summary>
  HumidityRatio,
  /// <summary>比エンタルピー基準</summary>
  Enthalpy
}

/// <summary>加湿方式</summary>
public enum HumidifierType
{
  /// <summary>無し</summary>
  None,
  /// <summary>蒸気加湿</summary>
  Steam,
  /// <summary>水滴下気化式</summary>
  DropPervaporation,
  /// <summary>水噴霧式</summary>
  Atomizing,
  /// <summary>超音波式</summary>
  UltraSonic,
}
```

プログラム 26.2 にインスタンス変数およびプロパティの定義を示す。給気ファン、還気ファン、全熱交換器、冷温水コイルとして、第 II 編で開発したクラスを用いる。外部にこれらのインスタンスの情報を渡すために 23~36 行に示すようにプロパティを用いるが、Immutable インターフェースによる読み取り専用インスタンスとする点に注意する。このような構成にすれば外部のプログラムによりコイルやファンが不適当に操作されてデータが破壊されることを防ぐことができる。同時に、コイルの水量やファンの消費電力などを外部に出力するために、空調機インスタンスで新たな同内容のプロパティ定義が不要となる点も利点である。

プログラム 26.2　インスタンスおよびプロパティの定義

Popolo.HVAC.AirConditioner.AirHandlingUnit class

```
 1 /// <summary>冷却コイル</summary>
 2 private CrossFinHeatExchanger cCoil;
 3
 4 /// <summary>加熱コイル</summary>
 5 private CrossFinHeatExchanger hCoil;
 6
 7 /// <summary>給気ファン</summary>
 8 private CentrifugalFan saFan;
 9
10 /// <summary>還気ファン</summary>
11 private CentrifugalFan raFan;
12
13 /// <summary>全熱交換器</summary>
14 private RotaryRegenerator regen = null;
15
16 /// <summary>外気冷房制御方式を設定・取得する</summary>
17 public OutdoorAirCoolingControl OutdoorAirCooling
18 { get; set; } = OutdoorAirCoolingControl.None;
19
20 /// <summary>加湿方式を取得する</summary>
21 public HumidifierType Humidifier { get; private set; }
22
23 /// <summary>冷水コイルを取得する</summary>
24 public ImmutableCrossFinHeatExchanger CoolingCoil { get { return cCoil; } }
25
26 /// <summary>温水コイルを取得する</summary>
27 public ImmutableCrossFinHeatExchanger HeatingCoil { get { return hCoil; } }
28
29 /// <summary>給気ファンを取得する</summary>
30 public ImmutableFluidMachinery SupplyAirFan { get { return saFan; } }
31
32 /// <summary>還気ファンを取得する</summary>
33 public ImmutableFluidMachinery ReturnAirFan { get { return raFan; } }
34
35 /// <summary>全熱交換器を取得する</summary>
36 public ImmutableRotaryRegenerator Regenerator { get { return regen; } }
37
38 /// <summary>ダクト熱損失率[-]を設定・取得する</summary>
39 public double DuctLossRate { get; set; } = 0.03;
40
41 /// <summary>全熱交換器をバイパスさせるか否かを設定・取得する</summary>
42 public bool BypassRegenerator { get; set; }
43
44 /// <summary>最大外気量[kg/s]を取得する</summary>
45 public double MaxOAFlowRate { get; private set; }
46
47 /// <summary>最小外気量[kg/s]を取得する</summary>
48 public double MinOAFlowRate { get; private set; }
49
50 /// <summary>外気量[kg/s]を取得する</summary>
51 public double OAFlowRate { get; private set; }
52
53 /// <summary>還気量[kg/s]を取得する</summary>
54 public double RAFlowRate { get; private set; }
55
56 /// <summary>給気量[kg/s]を取得する</summary>
57 public double SAFlowRate { get; private set; }
58
59 /// <summary>排気量[kg/s]を取得する</summary>
60 public double EAFlowRate { get; private set; }
61
62 /// <summary>還気乾球温度[C]を設定・取得する</summary>
63 public double RATemperature { get; set; }
```

```
64
65 /// <summary>還気絶対湿度[kg/kg]を設定・取得する</summary>
66 public double RAHumidityRatio { get; set; }
67
68 /// <summary>外気乾球温度[C]を設定・取得する</summary>
69 public double OATemperature { get; set; }
70
71 /// <summary>外気絶対湿度[kg/kg]を設定・取得する</summary>
72 public double OAHumidityRatio { get; set; }
73
74 /// <summary>給気乾球温度[C]を取得する</summary>
75 public double SATemperature { get; private set; }
76
77 /// <summary>給気絶対湿度[kg/kg]を取得する</summary>
78 public double SAHumidityRatio { get; private set; }
79
80 /// <summary>冷水入口温度[C]を設定・取得する</summary>
81 public double ChilledWaterInletTemperature { get; set; }
82
83 /// <summary>温水入口温度[C]を設定・取得する</summary>
84 public double HotWaterInletTemperature { get; set; }
85
86 /// <summary>加湿用給水量[kg/s]を取得する</summary>
87 public double WaterConsumption { get; private set; }
88
89 /// <summary>加湿用蒸気消費量[kg/s]を取得する</summary>
90 public double SteamConsumption { get; private set; }
91
92 /// <summary>加湿水の給水効率[-]を設定・取得する</summary>
93 public double WaterSupplyCoefficient { get; set; }
94
95 /// <summary>加湿器の最大飽和効率[-]を設定・取得する</summary>
96 public double MaxSaturationEfficiency { get; set; }
97
98 /// <summary>加湿器の飽和効率[-]を設定・取得する</summary>
99 public double SaturationEfficiency { get; set; }
100
101 /// <summary>過加熱あるいは過冷却が発生してでも風量を最小化するか否か</summary>
102 public bool MinimizeAirFlow { get; set; } = true;
103
104 /// <summary>吹き出し上限温度[C]を設定・取得する</summary>
105 public double UpperTemperatureLimit { get; set; } = 37;
106
107 /// <summary>吹き出し下限温度[C]を設定・取得する</summary>
108 public double LowerTemperatureLimit { get; set; } = 13;
```

プログラム 26.3 にコンストラクタを示す。コイルや給排気ファンなど、空調機を構成する機器が引数である。全熱交換器が無い場合には null を設定する。20~41 行で加湿器のタイプに応じて最大飽和効率 η_{hmS} と給水有効利用率 η_{hmW} を初期化する。これらはデフォルト値であり、プログラム 26.2 のプロパティを用いて直接に値を設定してもよい。

プログラム 26.3　コンストラクタ

Popolo.HVAC.AirConditioner.AirHandlingUnit class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="cCoil">冷却除湿コイル</param>
 3 /// <param name="hCoil">加熱コイル</param>
 4 /// <param name="humidType">加湿器のタイプ</param>
 5 /// <param name="saFan">給気ファン</param>
 6 /// <param name="raFan">還気ファン</param>
 7 /// <param name="regenerator">全熱交換器</param>
 8 public AirHandlingUnit
 9   (CrossFinHeatExchanger cCoil, CrossFinHeatExchanger hCoil, HumidifierType humidType,
10   CentrifugalFan saFan, CentrifugalFan raFan, RotaryRegenerator regenerator)
11 {
12   this.cCoil = cCoil;
13   this.hCoil = hCoil;
14   this.Humidifier = humidType;
15   this.saFan = saFan;
16   this.raFan = raFan;
17   this.regen = regenerator;
18
19   switch (humidType)
20   {
21     case HumidifierType.DropPervaporation:
22       WaterSupplyCoefficient = 0.5;
23       MaxSaturationEfficiency = 0.8;
```

```
24        WaterSupplyCoefficient = 0.5;
25        break;
26      case HumidifierType.Steam:
27        WaterSupplyCoefficient = 0.9;
28        MaxSaturationEfficiency = 1.0;
29        WaterSupplyCoefficient = 0.9;
30        break;
31      case HumidifierType.UltraSonic:
32        WaterSupplyCoefficient = 0.9;
33        MaxSaturationEfficiency = 0.5;
34        WaterSupplyCoefficient = 0.9;
35        break;
36      case HumidifierType.Atomizing:
37        WaterSupplyCoefficient = 0.4;
38        MaxSaturationEfficiency = 0.3;
39        WaterSupplyCoefficient = 0.4;
40        break;
41    }
42 }
```

プログラム 26.4 に風量設定（式 26.17 による）と機器の停止処理を示す。式 26.17 の風量収支が合うように外気、排気、給気、還気のバランスをとることが重要である。

プログラム 26.4　風量設定・停止処理

Popolo.HVAC.AirConditioner.AirHandlingUnit class

```
 1 /// <summary>風量[kg/s]を設定する</summary>
 2 /// <param name="raFlowRate">還気量[kg/s]</param>
 3 /// <param name="saFlowRate">給気量[kg/s]</param>
 4 public void SetAirFlowRate(double raFlowRate, double saFlowRate)
 5 {
 6   RAFlowRate = Math.Max(0, raFlowRate);
 7   SAFlowRate = Math.Max(0, saFlowRate);
 8   EAFlowRate = RAFlowRate + OAFlowRate - SAFlowRate;
 9 }
10
11 /// <summary>外気量範囲を設定する</summary>
12 /// <param name="minOAFlow">最小外気量[kg/s]</param>
13 /// <param name="maxOAFlow">最大外気量[kg/s]</param>
14 public void SetOutdoorAirFlowRange(double minOAFlow, double maxOAFlow)
15 {
16   MinOAFlowRate = minOAFlow;
17   MaxOAFlowRate = maxOAFlow;
18   OAFlowRate = Math.Min(Math.Max(OAFlowRate, MinOAFlowRate), MaxOAFlowRate);
19   EAFlowRate = RAFlowRate + OAFlowRate - SAFlowRate;
20 }
21
22 /// <summary>機器を停止させる</summary>
23 public void ShutOff()
24 {
25   cCoil.ShutOff();
26   hCoil.ShutOff();
27   saFan.ShutOff();
28   raFan.ShutOff();
29   if (regen != null) regen.ShutOff();
30   WaterConsumption = 0.0;
31   OAFlowRate = 0;
32   SetAirFlowRate(0, 0);
33 }
```

26.3.2　冷却運転と加熱運転の計算

空調機の計算も他の機器類と同じように、成り行き運転の計算と出口状態を制御した場合の計算の 2 つが必要となる。前者は風量、冷温水量、飽和効率を固定した場合に出口空気状態がどのようになるかの計算であり、後者は、吹出空気状態を制御値に合わせるために、風量、冷温水量、飽和効率をどのような値に制御すれば良いかの計算である。

1）冷却運転

プログラム 26.5 に冷却運転の計算処理を示す。1~97 行のメソッドは成り行き運転と出口温度制御運転の両方に対応した汎用の private メソッドである。外部に公開する public メソッドはこれを使って 100 行と 102~108 行で定義する。第 1 引数は出口空気温度を制御するか否かのフラグであり、これが false の場合には第 2、第 3 引数は意味を持たない。制御する場合には第 2 引数で与えられる吹出温

度に合致するように冷水量、外気冷房、全熱交換器を制御する。第 3 引数は絶対湿度の設定値であり、外気冷房を絶対湿度基準または比エンタルピー基準で行う際に用いる情報である。再熱コイルを持たないモデルとするため、出口湿度を制御するためのものではない点に注意する。

VAV 制御の場合には給気量も変動するが、これに関しては別のメソッドで計算することとし、本メソッドでは給気量 m_{SA} は所与とする。従って、17、18 行で給気および還気量にもとづいてファン動力を計算し、ファンによる昇温幅を 19 行で計算する（式 26.3）。さらに式 26.1 にもとづき、22 行で還気温度とファン昇温幅から出口温度設定を補正する。

25~54 行は外気冷房の計算である。出口温度を制御しない場合には外気量も与えられた風量で固定するため、計算は行わない。式 26.22~式 26.27 を適用し、外気条件と還気条件から最適な外気量を計算する。

56~78 行は全熱交換器の計算である。59 行に示すように、そもそも全熱交換器が無い場合、バイパス制御が有効な場合、熱交換対象の外気量または排気量が 0 の場合には適用しない。61, 63 行で外気と排気の比エンタルピーを求め、これらを比較することで全熱交換の是非を判断する。一旦、69 行で成り行き計算を行い、過剰に熱交換をしてしまった場合には 71~73 行で全熱交換器の出口空気状態を制御する。

80~89 行は冷水コイルの計算である。81~83 行で還気と外気を混合して冷水コイルの入口空気条件を確定させる。出口温度制御か成り行き運転かの別によって 84~87 行で処理を切り分ける。88, 89 行に示すように、冷却運転の場合には温水コイルと加湿を停止させる。

90~94 行でファン発熱による昇温とダクト熱ロスを反映させ、出口温度設定値に十分に近い場合には吹出温度制御成功として true を出力する（96 行）。ダクトの熱ロスは式 26.1 と式 26.2 にもとづき、110~116 行のメソッドで計算する。

プログラム 26.5　冷却運転の計算処理

Popolo.HVAC.AirConditioner.AirHandlingUnit class

```
 1 /// <summary>空気を冷却除湿する</summary>
 2 /// <param name="controlOutletTemp">AHU 出口温度を制御するか否か</param>
 3 /// <param name="spTemp">給気温度設定値[C]</param>
 4 /// <param name="spHumid">給気絶対湿度設定値[kg/kg]</param>
 5 /// <returns>制御成功の真偽</returns>
 6 /// <remarks>絶対湿度設定値は外気冷房制御用。温度基準の場合は不要。</remarks>
 7 private bool coolAir(bool controlOutletTemp, double spTemp, double spHumid)
 8 {
 9   //SA 風量が 0 の場合は停止処理（全外気で RA 風量=0 はありえる）
10   if (SAFlowRate <= 0)
11   {
12     ShutOff();
13     return false;
14   }
15
16   //ファン昇温を計算
17   saFan.UpdateState(SAFlowRate / AIR_DENSITY);
18   raFan.UpdateState(RAFlowRate / AIR_DENSITY);
19   double tRise = saFan.GetElectricConsumption() / (SAFlowRate * AIR_SPECIFICHEAT);
20
21   //ダクト熱損失とファン昇温を考慮して出口温度設定を補正
22   double sp2 = Math.Max(LowerTemperatureLimit, spTemp);
23   double tdCo = (sp2 - RATemperature * DuctHeatLossRate) / (1 - DuctHeatLossRate) - tRise;
24
25   //外気冷房を反映した外気量調整//成り行き計算の場合は外気量は所与
26   if (controlOutletTemp)
27   {
28     double mOA = MinOAFlowRate;
29     if (OutdoorAirCooling == OutdoorAirCoolingControl.DrybulbTemperature)
30     {
31       if (tdCo < OATemperature && OATemperature < RATemperature)
```

```
 32         mOA = Math.Min(MaxOAFlowRate, SAFlowRate);
 33       else mOA = RAFlowRate * (RATemperature - tdCo) / (tdCo - OATemperature);
 34     }
 35     else if (OutdoorAirCooling == OutdoorAirCoolingControl.HumidityRatio)
 36     {
 37       if (spHumid < OAHumidityRatio && OAHumidityRatio < RAHumidityRatio)
 38         mOA = Math.Min(MaxOAFlowRate, SAFlowRate);
 39       else mOA = RAFlowRate * (RAHumidityRatio - spHumid) / (spHumid - OAHumidityRatio);
 40     }
 41     else if (OutdoorAirCooling == OutdoorAirCoolingControl.Enthalpy)
 42     {
 43       double hAHUi =
 44         MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(RATemperature, RAHumidityRatio);
 45       double hAHUo =
 46         MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(tdCo, spHumid);
 47       double hOA =
 48         MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(OATemperature, OAHumidityRatio);
 49       if (hAHUo < hOA && hOA < hAHUi) mOA = Math.Min(MaxOAFlowRate, SAFlowRate);
 50       else mOA = RAFlowRate * (hAHUi - hAHUo) / (hAHUo - hOA);
 51     }
 52     OAFlowRate = Math.Max(MinOAFlowRate, Math.Min(MaxOAFlowRate, mOA));
 53     EAFlowRate = RAFlowRate + OAFlowRate - SAFlowRate;
 54   }
 55
 56   //全熱交換器による熱回収
 57   double tdOA = OATemperature;
 58   double hrOA = OAHumidityRatio;
 59   if (regen != null && !BypassRegenerator && 0 < OAFlowRate && 0 < EAFlowRate)
 60   {
 61     double hRtn =
 62       MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(RATemperature, RAHumidityRatio);
 63     double hOA =
 64       MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(OATemperature, OAHumidityRatio);
 65     if (hRtn < hOA)
 66     {
 67       const double cf = 3600 / AIR_DENSITY;
 68       //成り行き計算を試行
 69       regen.UpdateState(OAFlowRate * cf, EAFlowRate * cf, 1.0, tdOA, hrOA, RATemperature, RAHumidityRatio);
 70       //過剰処理の場合には給気温度を制御
 71       if (controlOutletTemp && regen.SupplyAirOutletDrybulbTemperature < tdCo)
 72         regen.ControlOutletTemperature
 73           (OAFlowRate * cf, EAFlowRate * cf, 1.0, tdOA, hrOA, RATemperature, RAHumidityRatio, tdCo);
 74       tdOA = regen.SupplyAirOutletDrybulbTemperature;
 75       hrOA = regen.SupplyAirOutletHumidityRatio;
 76     }
 77     else regen.ShutOff();
 78   }
 79
 80   //OA と RA を混合して冷却
 81   double mr = OAFlowRate / (OAFlowRate + RAFlowRate);
 82   double tdCi = tdOA * mr + RATemperature * (1 - mr);
 83   double hrCi = hrOA * mr + RAHumidityRatio * (1 - mr);
 84   if (controlOutletTemp)
 85     cCoil.ControlOutletAirTemperature(tdCi, hrCi, ChilledWaterInletTemperature, SAFlowRate, tdCo);
 86   else
 87     cCoil.UpdateOutletState(tdCi, hrCi, ChilledWaterInletTemperature, SAFlowRate, cCoil.WaterFlowRate);
 88   hCoil.ShutOff();
 89   WaterConsumption = SteamConsumption = 0.0;
 90
 91   //SA ファンによる昇温とダクト熱損失効果を反映
 92   SATemperature = cCoil.OutletAirTemperature + tRise;
 93   SAHumidityRatio = cCoil.OutletAirHumidityRatio;
 94   computeDuctLoss();
 95
 96   return -1e-4 < spTemp - SATemperature;
 97 }
 98
 99 /// <summary>空気を冷却する（成り行き計算）</summary>
100 public void CoolAir() { coolAir(false, 0, 0); }
101
102 /// <summary>空気を冷却する（吹出温度制御）</summary>
103 /// <param name="setpointTemperature">給気温度設定値[C]</param>
104 /// <param name="setpointHumidity">給気絶対湿度設定値[kg/kg]</param>
105 /// <returns>制御成功の真偽</returns>
106 /// <remarks>絶対湿度設定値は外気冷房制御用。温度基準の場合は不要。</remarks>
107 public bool CoolAir(double setpointTemperature, double setpointHumidity)
108 { return coolAir(true, setpointTemperature, setpointHumidity); }
109
110 /// <summary>吹き出し空気にダクトロスを反映する</summary>
111 private void computeDuctLoss()
```

```
112 {
113   double dl1 = 1 - DuctLossRate;
114   SATemperature = SATemperature * dl1 + RATemperature * DuctLossRate;
115   SAHumidityRatio = SAHumidityRatio * dl1 + RAHumidityRatio * DuctLossRate;
116 }
```

外気冷房による処理熱はプログラム 26.6 で計算する。

プログラム 26.6　外気冷房による処理熱計算処理

Popolo.HVAC.AirConditioner.AirHandlingUnit class

```
 1 /// <summary>外気冷房による処理熱[kW]を計算する</summary>
 2 /// <returns>外気冷房による処理熱[kW]</returns>
 3 public double GetOutdoorCoolingHeat()
 4 {
 5   double hSA = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
 6     (cCoil.InletAirTemperature, cCoil.InletAirHumidityRatio);
 7   double hOA = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
 8     (OATemperature, OAHumidityRatio);
 9   return (OAFlowRate - MinOAFlowRate) * (hSA - hOA);
10 }
```

2）加熱運転

加熱運転の計算処理をプログラム 26.7 に示す。冷却運転の大きな流れは同じであるが、湿度制御を考慮する点に違いがある。特に水加湿の場合には、加湿によって乾球温度が低下する効果を考慮して温水コイルの出口温度を制御する必要がある。

8~60 行は冷却運転と同様である。ただし、加熱運転の場合には外気冷房という概念は無いため、外気量は常に最小値とする（26~29 行）。

62~65 行に示すように、水加湿を行う場合には出口温湿度設定値から比エンタルピーを求め、温水出口空気が同じ比エンタルピーになるように加熱する。

75~112 行が加湿処理である。80~104 行が水加湿であり、飽和絶対湿度と飽和効率を用いて加湿量を計算する。105~112 行は蒸気加湿であり、この場合には飽和絶対湿度まで自由に制御可能とする。

プログラム 26.7　加熱運転の計算処理

Popolo.HVAC.AirConditioner.AirHandlingUnit class

```
 1 /// <summary>空気を加熱加湿する</summary>
 2 /// <param name="controlOutletState">給気温度を制御するか</param>
 3 /// <param name="spTemp">給気温度設定値</param>
 4 /// <param name="spHumid">給気絶対湿度設定値</param>
 5 /// <returns>制御成功の真偽</returns>
 6 private bool heatAir(bool controlOutletState, double spTemp, double spHumid)
 7 {
 8   //SA 風量が 0 の場合は停止処理（全外気で RA 風量=0 はありえる）
 9   if (SAFlowRate <= 0)
10   {
11     ShutOff();
12     return false;
13   }
14
15   //ファン昇温を計算
16   saFan.UpdateState(SAFlowRate / AIR_DENSITY);
17   raFan.UpdateState(RAFlowRate / AIR_DENSITY);
18   double tRise = saFan.GetElectricConsumption() / (SAFlowRate * AIR_SPECIFICHEAT);
19
20   //ダクト熱損失とファン昇温を考慮して出口温湿度設定を補正
21   double sp2 = Math.Min(UpperTemperatureLimit, spTemp);
22   double tdCo = (sp2 - RATemperature * DuctHeatLossRate) / (1 - DuctHeatLossRate) - tRise;
23   double wAHUo = (spHumid - RAHumidityRatio * DuctHeatLossRate) / (1 - DuctHeatLossRate);
24   double hCo = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(tdCo, wAHUo);
25
26   //外気量は最小値
27   double mOA = MinOAFlowRate;
28   OAFlowRate = MinOAFlowRate;
29   EAFlowRate = RAFlowRate + OAFlowRate - SAFlowRate;
30
31   //全熱交換器による熱回収
32   double tdOA = OATemperature;
33   double hrOA = OAHumidityRatio;
34   if (regen != null && !BypassRegenerator && 0 < OAFlowRate && 0 < EAFlowRate)
```

```
35  {
36    double hRtn =
37      MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(RATemperature, RAHumidityRatio);
38    double hOA =
39      MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(OATemperature, OAHumidityRatio);
40    if (hOA < hRtn)
41    {
42      const double cf = 3600 / AIR_DENSITY;
43      //成り行き計算実行
44      regen.UpdateState(OAFlowRate * cf, EAFlowRate * cf, 1.0, tdOA, hrOA, RATemperature, RAHumidityRatio);
45      //過剰処理の場合には給気比エンタルピーを制御
46      double hRego = MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio
47        (regen.SupplyAirOutletDrybulbTemperature, regen.SupplyAirOutletHumidityRatio);
48      if (controlOutletState && hCo < hRego)
49        regen.ControlOutletEnthalpy
50          (OAFlowRate * cf, EAFlowRate * cf, 1.0, tdOA, hrOA, RATemperature, RAHumidityRatio, hCo);
51      tdOA = regen.SupplyAirOutletDrybulbTemperature;
52      hrOA = regen.SupplyAirOutletHumidityRatio;
53    }
54    else regen.ShutOff();
55  }
56
57  //OA と RA を混合
58  double mr = OAFlowRate / (OAFlowRate + RAFlowRate);
59  double tdCi = tdOA * mr + RATemperature * (1 - mr);
60  double hrCi = hrOA * mr + RAHumidityRatio * (1 - mr);
61
62  //水加湿の場合には温水コイル出口温度を比エンタルピーで調整
63  if (hrCi < wAHUo &&
64    (Humidifier != HumidifierType.None && Humidifier != HumidifierType.Steam))
65    tdCo = MoistAir.GetDryBulbTemperatureFromHumidityRatioAndEnthalpy(hrCi, hCo);
66
67  //加熱
68  if (controlOutletState)
69    hCoil.ControlOutletAirTemperature(tdCi, hrCi, HotWaterInletTemperature, SAFlowRate, tdCo);
70  else
71    hCoil.UpdateOutletState(tdCi, hrCi, HotWaterInletTemperature, SAFlowRate, hCoil.WaterFlowRate);
72  tdCo = hCoil.OutletAirTemperature;
73  cCoil.ShutOff();
74
75  //加湿
76  double tSFi, wSFi;
77  tSFi = hCoil.OutletAirTemperature;
78  wSFi = hCoil.OutletAirHumidityRatio;
79  WaterConsumption = SteamConsumption = 0.0;
80  //水加湿
81  if (Humidifier != HumidifierType.None && Humidifier != HumidifierType.Steam)
82  {
83    double hSFi =
84      MoistAir.GetEnthalpyFromDryBulbTemperatureAndHumidityRatio(tdCo, hCoil.OutletAirHumidityRatio);
85    double satW = MoistAir.GetSaturationHumidityRatioFromEnthalpy(hSFi, ATMOSPHERIC_PRESSURE);
86    if (controlOutletState)
87    {
88      if (hCoil.OutletAirHumidityRatio < wAHUo)
89      {
90        double maxW = (1 - MaxSaturationEfficiency)
91          * hCoil.OutletAirHumidityRatio + MaxSaturationEfficiency * satW;
92        wAHUo = Math.Min(maxW, wAHUo);
93        SaturationEfficiency = (wAHUo - hCoil.OutletAirHumidityRatio) / (satW - hCoil.OutletAirHumidityRatio);
94      }
95      else
96      {
97        wAHUo = hCoil.OutletAirHumidityRatio;
98        SaturationEfficiency = 0.0;
99      }
100   }
101   else wAHUo = (1 - SaturationEfficiency) * hCoil.OutletAirHumidityRatio + SaturationEfficiency * satW;
102   tSFi = MoistAir.GetDryBulbTemperatureFromHumidityRatioAndEnthalpy(wAHUo, hSFi);
103   WaterConsumption = (wAHUo - hCoil.OutletAirHumidityRatio) * SAFlowRate / WaterSupplyCoefficient;
104 }
105 //蒸気加湿//最大飽和効率=100%
106 else if (Humidifier == HumidifierType.Steam)
107 {
108   double satW = MoistAir.GetSaturationHumidityRatioFromDryBulbTemperature
109     (hCoil.OutletAirTemperature, ATMOSPHERIC_PRESSURE);
110   wAHUo = Math.Min(satW, wAHUo);
111   SteamConsumption = (wAHUo - hCoil.OutletAirHumidityRatio) * SAFlowRate / WaterSupplyCoefficient;
112 }
113
114 //SA ファンによる昇温とダクト熱損失効果を反映
```

```
115   SATemperature = tSFi + tRise;
116   SAHumidityRatio = wAHUo;
117   computeDuctLoss();
118
119   return -1e-4 < SATemperature - spTemp;
120 }
121
122 /// <summary>空気を加熱加湿する（成り行き計算）</summary>
123 public void HeatAir() { heatAir(false, 0, 0); }
124
125 /// <summary>空気を加熱加湿する（吹出温湿度制御）</summary>
126 /// <param name="setpointTemperature">出口空気温度設定値[C]</param>
127 /// <param name="setpointHumidity">出口空気湿度設定値[kg/kg]</param>
128 /// <returns>加熱成功の真偽</returns>
129 public bool HeatAir(double setpointTemperature, double setpointHumidity)
130 { return heatAir(true, setpointTemperature, setpointHumidity); }
```

3）換気運転

換気運転の計算処理をプログラム 26.8 に示す。この場合には冷温水を通水しないため、ファンによる昇温とダクトによる熱ロスしか発生しない。

プログラム 26.8　換気運転の計算処理

Popolo.HVAC.AirConditioner.AirHandlingUnit class

```
 1 /// <summary>換気運転を行う</summary>
 2 public void Ventilate()
 3 {
 4   //SA 風量が 0 の場合は停止処理（全外気で RA 風量=0 はありえる）
 5   if (SAFlowRate <= 0)
 6   {
 7     ShutOff();
 8     return;
 9   }
10
11   //ファン昇温とダクト熱損失の計算
12   saFan.UpdateState(SAFlowRate / AIR_DENSITY);
13   raFan.UpdateState(SAFlowRate / AIR_DENSITY);
14   double tRise = saFan.GetElectricConsumption() / (SAFlowRate * CPMA);
15   SATemperature = RATemperature + tRise * (1 - DuctLossRate);
16   SAHumidityRatio = RAHumidityRatio;
17 }
```

26.3.3　VAV 制御の計算

VAV 制御で最適風量を求める処理をプログラム 26.9 に示す。

17~57 行で最大風量と最小風量での各ゾーンの要求吹出温度（$T_{AHUomax,n}$ と $T_{AHUomin,n}$）を計算する。式 26.12~式 26.14 の実装である。43~48 行は最小風量補償制御、49~54 行は最大風量補償制御の場合の処理である。

異常な吹出温度で執務者に不快感を与えないように、通常は空調機出口温度に上下限値を設ける。本メソッドの第 10、第 11 引数はこの上下限値の設定値であり、58~67 行でこの範囲に吹出温度を修正する。ただし、冷却運転の場合には上限温度は適用しない。上限温度を超える場合には現実には発停状態に入っているが、静的シミュレーションではこのような状態は表現できない。従ってシミュレーションは停止時間を含めた一定期間の期待値を出力するものであり、期待値が上限温度を超えたとしても問題ないとみなせるためである。同様に、加熱運転の場合には下限温度を適用しない。

75~126 行は図 26.6 と式 26.15 に示した誤差評価関数である。式 26.16 にもとづき、83~104 行で各ゾーンの給気、還気量を更新する。106~114 行で熱処理が可能か否かを判定する。暖房加湿運転の場合には、給気風量の変化に伴い吹出要求湿度も変化するため、要求湿度の再調整を行う必要がある点に注意する（112 行）。過負荷か否かに応じて 115~125 行で誤差を評価して出力する。

128~143 行は吹出温度 T_{AHUo} の最適化処理である。最大風量で計算を行い、そもそも冷暖要求と運転モードが逆転している場合には吹出風量を最小化して計算を終了する。最大風量で処理ができない

場合や最小風量で処理が可能な場合には最適化は不要であり、それぞれ最大風量と最小風量で計算を行えば良い。これ以外の場合には最適風量が存在するということであるから、上で定義した誤差関数を用いて黄金探索を行う（図 26.6）。

プログラム 26.9　VAV 風量の最適化処理

Popolo.HVAC.AirConditioner.AirHandlingUnit class

```
/// <summary>VAV 風量を計算する</summary>
/// <param name="isCooling">冷房運転か否か</param>
/// <param name="supplyHumiditySP">給気湿度設定値[kg/kg]（暖房時に使用）</param>
/// <param name="isVAVShutOff">VAV 停止状態</param>
/// <param name="zoneTemps">ゾーン温度[C]</param>
/// <param name="zoneHumids">ゾーン湿度[kg/kg]</param>
/// <param name="zoneSLoads">ゾーン顕熱負荷[kW]（暖房が正）</param>
/// <param name="minSAFlow">最小給気量[kg/s]</param>
/// <param name="maxSAFlow">最大給気量[kg/s]</param>
/// <param name="maxRAFlow">最大還気量[kg/s]</param>
/// <param name="success">制御成功の真偽</param>
/// <returns>VAV 風量リスト[kg/s]</returns>
public double[] OptimizeVAV
  (bool isCooling, double supplyHumiditySP, bool[] isVAVShutOff, double[] zoneTemps, double[] zoneHumids,
  double[] zoneSLoads, double[] minSAFlow, double[] maxSAFlow, double[] maxRAFlow, out bool success)
{
  //AHU 吹出温度の上下限値を計算
  double tAHUoMin, tAHUoMax;
  if (isCooling)
  {
    if (MinimizeAirFlow) tAHUoMin = 60;
    else tAHUoMin = -10;
    tAHUoMax = 60;
  }
  else
  {
    if (MinimizeAirFlow) tAHUoMin = -10;
    else tAHUoMin = 60;
    tAHUoMax = -10;
  }
  for (int i = 0; i < zoneTemps.Length; i++)
  {
    if (!isVAVShutOff[i])
    {
      //最大風量での給気温度と AHU 吹出温度
      double tMax = zoneTemps[i] + zoneSLoads[i] / (AIR_SPECIFICHEAT * maxSAFlow[i]);
      if (isCooling) tAHUoMax = Math.Min(tAHUoMax, tMax);
      else tAHUoMax = Math.Max(tAHUoMax, tMax);
      //最小風量での給気温度と AHU 吹出温度
      if (0 < minSAFlow[i])
      {
        double tMin = zoneTemps[i] + zoneSLoads[i] / (AIR_SPECIFICHEAT * minSAFlow[i]);
        //過冷却と過加熱が生じても風量を最小化する
        if (MinimizeAirFlow)
        {
          if (isCooling) tAHUoMin = Math.Min(tAHUoMin, tMin);
          else tAHUoMin = Math.Max(tAHUoMin, tMin);
        }
        //過冷却と過加熱を発生させないように風量を絞る
        else
        {
          if (isCooling) tAHUoMin = Math.Max(tAHUoMin, tMin);
          else tAHUoMin = Math.Min(tAHUoMin, tMin);
        }
      }
    }
  }
  if (isCooling)
  {
    tAHUoMin = Math.Max(LowerTemperatureLimit, tAHUoMin);
    tAHUoMax = Math.Max(LowerTemperatureLimit, tAHUoMax);
  }
  else
  {
    tAHUoMin = Math.Min(UpperTemperatureLimit, tAHUoMin);
    tAHUoMax = Math.Min(UpperTemperatureLimit, tAHUoMax);
  }

  //評価関数を定義
  double mSAMax = 0;
  for (int i = 0; i < maxSAFlow.Length; i++) mSAMax += maxSAFlow[i];
```

```
 72   double[] mSAn = new double[zoneTemps.Length];
 73   bool suc = true;
 74   bool overLoad = true;
 75   Minimization.MinimizeFunction mFnc = delegate (double tAHUo)
 76   {
 77     //還気条件を計算
 78     double mSAsum = 0;
 79     double mRAsum = 0;
 80     RATemperature = 0;
 81     RAHumidityRatio = 0;
 82     suc = true;
 83     for (int i = 0; i < zoneTemps.Length; i++)
 84     {
 85       if (!isVAVShutOff[i])
 86       {
 87         if (tAHUo == zoneTemps[i]) mSAn[i] = 0;
 88         else mSAn[i] = zoneSLoads[i] / (AIR_SPECIFICHEAT * (tAHUo - zoneTemps[i]));
 89         if (1e-10 < minSAFlow[i] - mSAn[i] || 1e-10 < mSAn[i] - maxSAFlow[i]) suc = false;
 90         mSAn[i] = Math.Min(maxSAFlow[i], Math.Max(minSAFlow[i], mSAn[i]));
 91         double mRAn = mSAn[i] - (maxSAFlow[i] - maxRAFlow[i]);
 92         mSAsum += mSAn[i];
 93         mRAsum += mRAn;
 94         RATemperature += mRAn * zoneTemps[i];
 95         RAHumidityRatio += mRAn * zoneHumids[i];
 96       }
 97       else mSAn[i] = 0;
 98     }
 99     if (1e-7 < mRAsum)
100     {
101       RATemperature /= mRAsum;
102       RAHumidityRatio /= mRAsum;
103     }
104     SetAirFlowRate(mRAsum, mSAsum);
105
106     //誤差を評価
107     if (isCooling) overLoad = !CoolAir(tAHUo, supplyHumiditySP);
108     else
109     {
110       //必要な湿度を補正
111       double bf = SAFlowRate / mSAMax;
112       double hsp = RAHumidityRatio * (1 - bf) + supplyHumiditySP * bf;
113       overLoad = !HeatAir(tAHUo, hsp);
114     }
115     if (overLoad)
116     {
117       double err = Math.Abs(SATemperature - tAHUo);
118       if (isCooling) return RATemperature + err;
119       else return -RATemperature + err;
120     }
121     else
122     {
123       if (isCooling) return SATemperature;
124       else return -SATemperature;
125     }
126   };
127
128   //最大風量で計算
129   mFnc(tAHUoMax);
130   //冷暖逆転の場合には最小風量
131   if ((isCooling && RATemperature < tAHUoMax) || (!isCooling && tAHUoMax < RATemperature)) mFnc(tAHUoMin);
132   //最大風量で過負荷の場合には最大風量
133   else if (!overLoad)
134   {
135     //最小風量で AHU が過負荷になるならば処理可能な風量まで修正
136     mFnc(tAHUoMin);
137     if (overLoad)
138     {
139       if (isCooling) Minimization.GoldenSection(ref tAHUoMin, tAHUoMax, mFnc);
140       else Minimization.GoldenSection(ref tAHUoMax, tAHUoMin, mFnc);
141     }
142   }
143   success = suc;
144   return mSAn;
145 }
```

【例題 26.2】

表 26.1 に示す空調機モデルを作成し、VAV 制御（最小風量補償）の場合に負荷率を下げていった場合の AHU の挙動を計算せよ。ただし、AHU は 3 つのゾーンに給気を行うこととし、それぞれのゾーンの最大負

荷と最大給気量は表 26.2 のとおりとし、VAV は 40%まで絞ることができるとする。また、外気量は 1,513 CMH とし、温湿度条件は冷却運転時が 33.4 ºC, 0.0194 kg/kg、加熱運転時が 2.0 ºC, 0.0014 kg/kg とする。冷水と温水の入口温度は 7 ºC と 50 ºC に固定する。

表 26.1　空調機の仕様

給気ファン	風量	7,476 CMH	還気ファン	風量	6,946 CMH
	全圧	800 Pa		全圧	300 Pa
	消費電力	3.44 kW		消費電力	1.31 kW
加湿器	水滴下気化式 15.5 kg/h				
冷水コイル	空気条件	入口：27.46 ºC / 12.06 g/kg 出口：14.30 ºC / 9.32 g/kg	温水コイル	空気条件	入口：17.46 ºC / 5.54 g/kg 出口：36.10 ºC / 5.54 g/kg
	冷水条件	7 ºC → 12.6 ºC		温水条件	50 ºC → 43.6 ºC
	冷水流量	127 L/min		温水流量	105 L/min
	冷却能力	49.6 kW		加熱能力	46.9 kW
	列数×段数	6 列×24 段		列数×段数	4 列×24 段
	正面面積	820 mm×910 mmH		正面面積	820 mm×910 mmH
	フロー	ハーフフロー		フロー	ハーフフロー

※フィンピッチ 2.9 mm、厚み 0.2 mm、熱伝導率 237 W/(m·K)、チューブ内外径 14.6 mm、15.8 mm

表 26.2　ゾーンの負荷、風量、温湿度条件

	ゾーン 1	ゾーン 2	ゾーン 3
最大冷房負荷 [kW]	16.0	7.0	6.0
最大暖房負荷 [kW]	12.1	5.3	4.6
最大給気量 [m³/h]	4,151	1,782	1,543
乾球温度設定値 [ºC]	22.0	24.0	26.0
絶対湿度設定値 [kg/kg]	0.0105	0.0105	0.0105

【解】

プログラムを 26.10 に示す。8~23 行でコイル、ファン、全熱交換器のインスタンスを生成し、これをコンストラクタに与えて 26 行で空調機を初期化する。50~69 行、70~88 行がそれぞれ冷却運転と加熱運転に関する感度計算であり、負荷率を 10%ずつ低下させて、必要風量、吹出温度、ファン動力を出力する。プログラムを実行すると表 26.3 および表 26.4 の結果が得られる。各ゾーンともに負荷率の低下によって風量が低下する様子が確認できる。負荷率 40 %程度で VAV が最小風量となるため、風量が一定となる。従って、これ以降の処理熱の制御は吹出温度によって行われるため、冷房であれば吹出温度が低下から上昇に転じ、暖房であれば上昇から低下に転じることがわかる。なお、最小風量補償制御のため、最小風量以下の範囲では過冷却または過加熱が生じる。参考に最大風量補償制御（30 行で MinimizeAirFlow プロパティを false に設定）とした場合の結果を表 26.5 と表 26.6 に示す。この場合には過冷却と過加熱が生じないように吹出温度が制御されるため低負荷領域において必ずしも風量が最小化せず、ファン動力が増加する。

プログラム 26.10　VAV 制御感度解析

```
 1 private static void AirHandlingUnitVAVTest()
 2 {
 3   //給気・還気・外気量[m3/s]
 4   double qsa = 7476d / 3600;
 5   double qra = 6946d / 3600;
 6   double qoa = 1513d / 3600;
 7
 8   //冷水・温水コイル作成
 9   CrossFinHeatExchanger cCoil = new CrossFinHeatExchanger
10     (0.82, 0.910, 6, 24, qsa * 1.2, 27.46, 0.01206, 95, 127d / 60, 127d / 60, 7,
11     CrossFinHeatExchanger.WaterFlowType.HalfFlow, 49.6, true);
12   CrossFinHeatExchanger hCoil = new CrossFinHeatExchanger
13     (0.82, 0.910, 4, 24, qsa * 1.2, 17.46, 0.00554, 95, 105d / 60, 105d / 60, 50,
14     CrossFinHeatExchanger.WaterFlowType.HalfFlow, 46.9, true);
15
16   //給気・還気ファン作成
17   CentrifugalFan saFan = new CentrifugalFan(0.8, qsa, 0.8, qsa, 3, true);
18   CentrifugalFan raFan = new CentrifugalFan(0.3, qra, 0.3, qra, 3, true);
19   saFan.MinimumRotationRatio = raFan.MinimumRotationRatio = 0.2;
20
21   //全熱交換器作成
22   RotaryRegenerator regenerator = new RotaryRegenerator
23     (0.34, qoa * 3600, (qoa + qra - qsa) * 3600, true, 34.4, 0.0194, 26, 0.0105);
24
25   //AHU 作成
```

```
26  AirHandlingUnit ahu = new AirHandlingUnit(cCoil, hCoil,
27    AirHandlingUnit.HumidifierType.DropPervaporation, saFan, raFan, regenerator);
28  ahu.SetOutdoorAirFlowRange(qoa * 1.2, qsa * 1.2);
29  ahu.SetAirFlowRate(qra, qsa * 1.2);
30  ahu.MinimizeAirFlow = true;
31  ahu.UpperTemperatureLimit = 40;
32  ahu.LowerTemperatureLimit = 10;
33
34  //運転状態設定
35  const double CNV = 1.2 / 3600d; //m3/h→kg/s 換算係数
36  bool[] off = new bool[] { false, false, false };
37  double[] zTemp = new double[] { 22, 24, 26 };
38  double[] zHumid = new double[] { 0.008, 0.008, 0.008 };
39  double[] zHLoad = new double[3];
40  double cf = 6946d / 7476d;
41  double[] maxSA = new double[] { 4151d * CNV, 1782d * CNV, 1543d * CNV };
42  double[] maxRA = new double[] { maxSA[0] * cf, maxSA[1] * cf, maxSA[2] * cf };
43  double[] minSA = new double[3];
44  for (int i = 0; i < minSA.Length; i++) minSA[i] = maxSA[i] * 0.4;
45
46  //冷温水入口温度設定
47  ahu.ChilledWaterInletTemperature = 7;
48  ahu.HotWaterInletTemperature = 50;
49
50  //冷却運転テスト
51  Console.WriteLine("--冷却運転---------------------------------");
52  ahu.OATemperature = 34.4;
53  ahu.OAHumidityRatio = 0.0194;
54  bool suc;
55  for (int i = 0; i < 8; i++)
56  {
57    zHLoad[0] = -16 * (1 - 0.1 * i) * 1000;
58    zHLoad[1] = -7 * (1 - 0.1 * i) * 1000;
59    zHLoad[2] = -6 * (1 - 0.1 * i) * 1000;
60    double[] af = ahu.OptimizeVAV(true, 0, off,
61  zTemp, zHumid, zHLoad, minSA, maxSA, maxRA, out suc);
62    Console.WriteLine(
63      (af[0] / CNV).ToString("F0") + "," +
64      (af[1] / CNV).ToString("F0") + "," +
65      (af[2] / CNV).ToString("F0") + "," +
66      (ahu.SAFlowRate / CNV).ToString("F0") + "," +
67      ahu.SATemperature.ToString("F1") + "," +
68      ahu.SupplyAirFan.GetElectricConsumption().ToString("F2"));
69  }
70  //加熱運転テスト
71  Console.WriteLine("--加熱運転---------------------------------");
72  ahu.OATemperature = 2.0;
73  ahu.OAHumidityRatio = 0.0014;
74  for (int i = 0; i < 8; i++)
75  {
76    zHLoad[0] = 12.1 * (1 - 0.1 * i) * 1000;
77    zHLoad[1] = 5.3 * (1 - 0.1 * i) * 1000;
78    zHLoad[2] = 4.6 * (1 - 0.1 * i) * 1000;
79    double[] af = ahu.OptimizeVAV(false, 0.0105, off,
80      zTemp, zHumid, zHLoad, minSA, maxSA, maxRA, out suc);
81    Console.WriteLine(
82      (af[0] / CNV).ToString("F0") + "," +
83      (af[1] / CNV).ToString("F0") + "," +
84      (af[2] / CNV).ToString("F0") + "," +
85      (ahu.SAFlowRate / CNV).ToString("F0") + "," +
86      ahu.SATemperature.ToString("F1") + "," +
87      ahu.SupplyAirFan.GetElectricConsumption().ToString("F2"));
88  }
89 }
```

表 26.3　VAV 風量感度解析結果（冷房：最小風量補償制御）

負荷率	風量 1 [CMH]	風量 2 [CMH]	風量 3 [CMH]	合計 [CMH]	吹出温度 [ºC]	ファン [kW]
100%	4,151	1,543	1,150	6,844	12.6	2.74
90%	4,151	1,518	1,117	6,786	12.6	2.67
80%	3,937	1,424	1,040	6,401	12.5	2.28
70%	3,306	1,204	884	5,395	12.1	1.52
60%	2,707	993	733	4,434	11.6	1.02
50%	2,152	796	617	3,565	11.1	0.67
40%	1,660	713	617	2,990	10.7	0.50
30%	1,660	713	617	2,990	13.5	0.49

表 26.4　VAV 風量感度解析結果（暖房：最小風量補償制御）

負荷率	風量 1 [CMH]	風量 2 [CMH]	風量 3 [CMH]	合計 [CMH]	吹出温度 [ºC]	ファン [kW]
100%	2,785	1,447	1,543	5,775	31.5	1.77
90%	2,691	1,418	1,543	5,652	31.7	1.69
80%	2,582	1,383	1,543	5,508	31.9	1.60
70%	2,318	1,248	1,405	4,971	32.7	1.28
60%	1,660	825	870	3,355	35.3	0.60
50%	1,660	713	692	3,066	35.7	0.52
40%	1,660	713	617	2,990	34.7	0.49
30%	1,660	713	617	2,990	32.6	0.49

表 26.5　VAV 風量感度解析結果（冷房：最大風量補償制御）

負荷率	風量 1 [CMH]	風量 2 [CMH]	風量 3 [CMH]	合計 [CMH]	吹出温度 [ºC]	ファン [kW]
100%	4,151	1,543	1,150	6,844	12.6	2.74
90%	4,151	1,518	1,117	6,786	12.6	2.67
80%	3,937	1,424	1,040	6,401	12.5	2.28
70%	3,306	1,204	884	5,395	12.1	1.52
60%	2,707	993	733	4,434	11.6	1.02
50%	2,287	838	617	3,742	11.7	0.74
40%	2,535	873	617	4,025	14.6	0.85
30%	3,091	940	617	4,648	17.4	1.12

表 26.6　VAV 風量感度解析結果（暖房：最大風量補償制御）

負荷率	風量 1 [CMH]	風量 2 [CMH]	風量 3 [CMH]	合計 [CMH]	吹出温度 [ºC]	ファン [kW]
100%	2,785	1,447	1,543	5,775	31.5	1.77
90%	2,691	1,418	1,543	5,652	31.7	1.69
80%	2,582	1,383	1,543	5,508	31.9	1.60
70%	2,318	1,248	1,405	4,971	32.7	1.28
60%	1,660	862	917	3,439	34.8	0.63
50%	1,660	895	1,009	3,564	32.7	0.67
40%	1,660	949	1,186	3,796	30.6	0.76
30%	1,660	1,057	1,543	4,260	28.4	0.94

26.3.4　「空調機システム」クラスの作成

前節までの検討では、ゾーンの顕熱負荷、潜熱負荷、乾球温度、絶対湿度が既知であるとして空調機の計算を行った。しかし、現実には空調機の吹出温度によってゾーンの温湿度は影響を受けるため、空調機モデルは第 25 章で開発したゾーンの熱負荷計算モデルと連成して解く必要がある。本節では、両者を連成させる「空調機システム」クラス（AHUSystem class）を開発する。

1）初期化と境界条件設定

モデルには多数の空調機と CAV, VAV が含まれるため、AHUSystem クラスの中にインナークラスを定義して制御情報を整理する。プログラム 26.11 に制御情報を保持するインナークラスの定義を示す。1~46 行が VAV および CAV の制御情報を保持するインナークラスである。給気量、還気量、発停、温度設定値に加え、建物熱負荷計算モデルにおける給気ゾーンを特定するために、多数室番号とゾーン番号を保持する。48~77 行が AHU の制御情報を保持するインナークラスである。CAV と VAV の別、温湿度設定値などを保持する。

プログラム 26.11　制御情報を保持するインナークラスの定義

Popolo.HVAC.SubSystem.AHUSystem class

```
1 /// <summary>VAV または CAV 制御クラス</summary>
2 public class VolumeController
3 {
4   /// <summary>給気量[kg/s]を取得する</summary>
5   public double SAFlow { get; internal set; }
6
```

```
 7   /// <summary>還気量[kg/s]を取得する</summary>
 8   public double RAFlow { get { return SAFlow - (MaxSAFlow - MaxRAFlow); } }
 9
10   /// <summary>最大給気量[kg/g]を取得する</summary>
11   public double MaxSAFlow { get; internal set; }
12
13   /// <summary>最大還気量[kg/g]を取得する</summary>
14   public double MaxRAFlow { get; internal set; }
15
16   /// <summary>最小給気量[kg/g]を取得する</summary>
17   public double MinSAFlow { get; internal set; }
18
19   /// <summary>給気する室番号を取得する</summary>
20   public int RoomIndex { get; internal set; }
21
22   /// <summary>給気するゾーン番号を取得する</summary>
23   public int ZoneIndex { get; internal set; }
24
25   /// <summary>温度設定値を取得する</summary>
26   public double SetPointTemperature { get; internal set; }
27
28   /// <summary>停止中か否かを取得する</summary>
29   public bool IsShutOff { get; internal set; }
30
31   /// <summary>インスタンスを初期化する</summary>
32   /// <param name="rmIndex">多数室番号</param>
33   /// <param name="znIndex">ゾーン番号</param>
34   /// <param name="maxSAFlow">最大給気量[kg/s]</param>
35   /// <param name="maxRAFlow">最大還気量[kg/s]</param>
36   /// <param name="minSAFlow">最小給気量[kg/s]</param>
37   public VolumeController
38     (int rmIndex, int znIndex, double maxSAFlow, double maxRAFlow, double minSAFlow)
39   {
40     RoomIndex = rmIndex;
41     ZoneIndex = znIndex;
42     MaxSAFlow = maxSAFlow;
43     MaxRAFlow = maxRAFlow;
44     MinSAFlow = minSAFlow;
45   }
46 }
47
48 /// <summary>AHU 制御クラス</summary>
49 public class AHUController
50 {
51   /// <summary>インスタンスを初期化する</summary>
52   internal AHUController() { }
53
54   /// <summary>運転モードを設定・取得する</summary>
55   public OperatingMode Mode { get; set; } = OperatingMode.ShutOff;
56
57   /// <summary>CAV 制御か否かを取得する</summary>
58   public bool IsCAVControl { get; internal set; } = true;
59
60   /// <summary>還気温度制御か否かを設定・取得する</summary>
61   public bool IsRATemperatureControl { get; set; } = true;
62
63   /// <summary>制御対象室番号を設定・取得する</summary>
64   public int TargetRoomIndex { get; set; }
65
66   /// <summary>制御対象ゾーン番号を設定・取得する</summary>
67   public int TargetZoneIndex { get; set; }
68
69   /// <summary>CAV 温度設定値[C]を設定・取得する</summary>
70   public double SetpointTemperature { get; set; } = 24.0;
71
72   /// <summary>最低湿度[kg/kg]を設定・取得する</summary>
73   public double MinimumHumidity { get; set; }
74
75   /// <summary>給気湿度設定値[kg/kg]を設定・取得する</summary>
76   internal double splyHumidSP { get; set; }
77 }
```

プログラム 26.12 に列挙型、インスタンス変数、プロパティ、コンストラクタの定義を示す。建物熱負荷計算クラスと空調機クラスのインスタンスの他、プログラム 26.11 で作成した制御情報配列を保持する。

プログラム **26.12**　列挙型、インスタンス変数、プロパティ、コンストラクタの定義

Popolo.HVAC.SubSystem.AHUSystem class

```
/// <summary>AHU 運転モード</summary>
public enum OperatingMode
{
  /// <summary>停止</summary>
  ShutOff,
  /// <summary>換気（温調無）</summary>
  Ventilation,
  /// <summary>冷房専用</summary>
  Cooling,
  /// <summary>暖房専用</summary>
  Heating
}

/// <summary>建物熱負荷計算モデル</summary>
private BuildingThermalModel bModel;

/// <summary>AHU リスト</summary>
private AirHandlingUnit[] ahu;

/// <summary>AHU 毎のゾーン制御リスト</summary>
private VolumeController[][] vlmCtrl;

/// <summary>AHU 制御点が確定済か否か</summary>
private bool[] ctrlFixed;

/// <summary>空調機リストを取得する</summary>
public ImmutableAirHandlingUnit[] AHUs { get { return ahu; } }

/// <summary>空調機制御リストを取得する</summary>
public AHUController[] Controllers { get; private set; }

/// <summary>収束計算の初回か否か</summary>
private bool isFirstCall = true;

/// <summary>AHU 入口絶対湿度リスト</summary>
private double[] hrAHUs;

/// <summary>インスタンスを初期化する</summary>
/// <param name="bModel">建物熱負荷計算モデル</param>
/// <param name="ahu">空気調和機リスト</param>
public AHUSystem(BuildingThermalModel bModel, AirHandlingUnit[] ahu)
{
  this.bModel = bModel;
  this.ahu = ahu;

  ctrlFixed = new bool[ahu.Length];
  Controllers = new AHUController[ahu.Length];
  vlmCtrl = new VolumeController[ahu.Length][];
  for (int i = 0; i < ahu.Length; i++)
  {
    Controllers[i] = new AHUController();
    vlmCtrl[i] = new VolumeController[0];
  }
}
```

プログラム 26.13 に境界条件およびゾーン空調条件設定処理を示す。

プログラム **26.13**　境界条件、ゾーン空調条件設定処理

Popolo.HVAC.SubSystem.AHUSystem class

```
/// <summary>外気条件を設定する</summary>
/// <param name="temperature">外気温度[C]</param>
/// <param name="humidityRatio">外気絶対湿度[kg/kg]</param>
public void SetOutdoorAirState(double temperature, double humidityRatio)
{
  for (int i = 0; i < ahu.Length; i++)
  {
    ahu[i].OATemperature = temperature;
    ahu[i].OAHumidityRatio = humidityRatio;
  }
}

/// <summary>AHU 外気量を設定する</summary>
/// <param name="ahuIndex">AHU 番号</param>
/// <param name="minOA">最小外気量[kg/s]</param>
/// <param name="maxOA">最大外気量[kg/s]</param>
public void SetOutdoorAirFlow(int ahuIndex, double minOA, double maxOA)
{ ahu[ahuIndex].SetOutdoorAirFlowRange(minOA, maxOA); }
```

```
19
20 /// <summary>空調機に VAV を登録する</summary>
21 /// <param name="ahuIndex">空調機番号</param>
22 /// <param name="vavs">VAV リスト</param>
23 public void SetVAV(int ahuIndex, VolumeController[] vavs)
24 {
25   vlmCtrl[ahuIndex] = vavs;
26   Controllers[ahuIndex].IsCAVControl = false;
27 }
28
29 /// <summary>空調機に CAV を登録する</summary>
30 /// <param name="ahuIndex">空調機番号</param>
31 /// <param name="cavs">CAV リスト</param>
32 public void SetCAV(int ahuIndex, VolumeController[] cavs)
33 {
34   vlmCtrl[ahuIndex] = cavs;
35   Controllers[ahuIndex].IsCAVControl = true;
36 }
37
38 /// <summary>ゾーン温度を制御する</summary>
39 /// <param name="ahuIndex">空調機番号</param>
40 /// <param name="controlZoneIndex">制御ゾーン番号</param>
41 /// <param name="setPointTemperature">設定温度[C]</param>
42 public void ControlZoneTemperature
43   (int ahuIndex, int controlZoneIndex, double setPointTemperature)
44 {
45   if (Controllers[ahuIndex].IsCAVControl)
46   {
47     Controllers[ahuIndex].SetpointTemperature = setPointTemperature;
48     Controllers[ahuIndex].IsRATemperatureControl = false;
49   }
50   else
51   {
52     VolumeController vc = vlmCtrl[ahuIndex][controlZoneIndex];
53     vc.IsShutOff = false;
54     vc.SetPointTemperature = setPointTemperature;
55   }
56 }
57
58 /// <summary>ゾーンの空調を停止する</summary>
59 /// <param name="ahuIndex">空調機番号</param>
60 /// <param name="controlZoneIndex">制御ゾーン番号</param>
61 public void ShutOff(int ahuIndex, int controlZoneIndex)
62 { vlmCtrl[ahuIndex][controlZoneIndex].IsShutOff = true; }
```

2）連成計算

ゾーンの温湿度は空調機の吹出空気によって決まるが、空調機の吹出空気はゾーンからの還気温湿度に依存する。従って、計算が循環してしまうため、収束計算が必要となる。図 26.9 に収束計算の全体フローを示す。

冷却コイルの計算や加湿時の計算において示したように温度と湿度は相互に影響があるため、正確には両者をまとめて連成させることが必要である。しかし、収束計算対象の変数の数が多くなり計算速度が非常に低下するため、本書のモデルでは空調機の入口湿度は連成計算の対象から除外する。即ち、現在時点の各ゾーンの絶対湿度から空調機還気絶対湿度を確定させる。

制御方式が VAV か CAV か、また、空調機が過負荷か否かによって、収束計算の対象となる状態変数が異なる。VAV 制御で軽負荷（過冷却・過加熱でも無い）の場合には各ゾーンの温度は風量調整によって設定温度に到達する。この場合には収束計算は必要ない。CAV 制御の場合には各ゾーンの温度は成り行きになるため、収束計算により、還気温度が設定温度になるような吹出温度を推定する必要がある。過負荷の場合にはゾーンや還気の温度を設定値に制御することができないため、設定値からどの程度まで乖離するのかを収束計算で求める必要がある。

計算の手順としては、まず、すべての空調機が無限大の能力を持つ軽負荷状態にある空調機であると仮定して計算を進める。VAV 制御の空調機に関しては、各ゾーンの乾球温度を設定温度に一致させることができる（能力無限大の仮定であるため）ため、建物熱負荷計算モデルで室温を境界条件と

して顕熱負荷を算出する。CAV 制御の空調機に関しては、吹出空気乾球温度を未知変数とし、還気または制御対象のゾーンの温度が設定値に一致する吹出温度を収束計算で求める。複数の吹出温度が未知変数となりうるため、多次元のニュートン法を用いる。十分に収束したら、それぞれの空調機の過負荷状態を判定する。VAV 制御の空調機に関しては算出された顕熱負荷から必要な吹出空気温度を逆算し、その温度まで冷却または加熱が可能か否かを判定する。同様に、CAV 制御の空調機に関しては、収束計算によって求められた必要吹出空気温度に対して過負荷判定を行う。空調機が過負荷状態にあることが確定したら、当該系統の空調機の冷温水量および風量を最大値に固定し、成り行き計算に切り替える。この結果、過負荷系統のゾーン温度が変化し、隣接ゾーンの空調機などが過負荷に転ずる可能性が発生するため、再度の収束計算を行う。過負荷に転ずる系統が無くなったら、軽負荷状態にある VAV 制御の空調機の風量を最適化し、計算を終了する。ただし、軽負荷状態にあってもゾーン間の負荷バランスが悪い場合には、過加熱あるいは過冷却の可能性がある。この場合には還気温度が未知となるため、収束計算を行う必要がある。

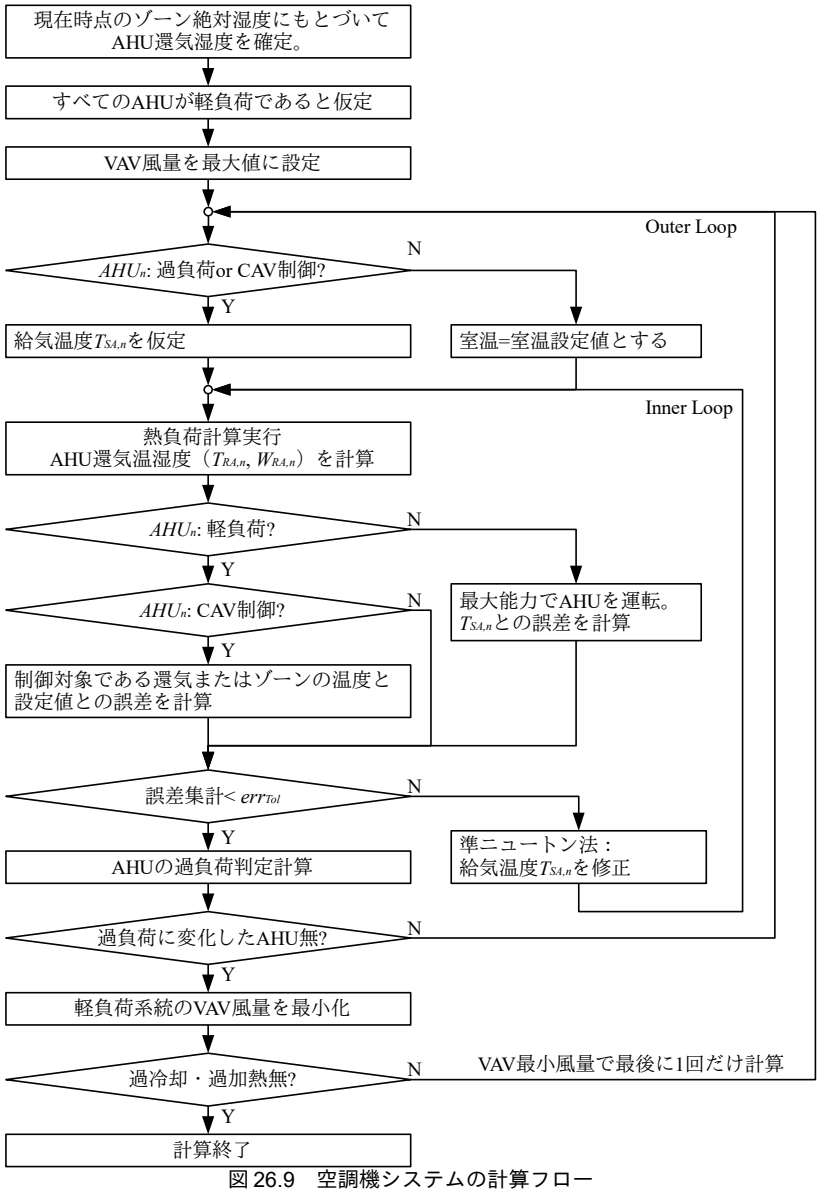


図 26.9　空調機システムの計算フロー

プログラム 26.14 に空調機システムの計算処理を示す。冷水と温水の往温度にもとづいて空調機と建物熱負荷の連成計算を行う。熱源システムモデルから本メソッドを呼び出し、冷温水還温度と流量を評価することを想定している。連成のための反復計算の過程で空調機の還気絶対湿度は不変のため、初回の計算では 10~25 行でゾーンからの平均還気絶対湿度を計算して保存する。26, 27 行でこの還気絶対湿度を空調機に設定する。isFirstCall は初期化のフラグであり、繰り返し計算が終了した段階で解除する（126 行）。29~65 行は空調機の冷温水および風量の初期化処理である。停止中の空調機と CAV, VAV を除き、すべての吹出口の風量を最大値とする。すべての空調機が停止している特殊な場合には、自然室温計算を行って計算を終了する（67~83 行）。86 行が顕熱平衡の収束計算処理であり、後述する。顕熱平衡状態が確定できたら 88~101 行で水分平衡を計算する。顕熱平衡計算によって確定した空調機吹出空気状態と風量を境界条件として建物熱負荷計算モデルを解く。104 行は冷温水還温度計算メソッド（107~123 行）であり、すべての空調機の還水温を流量で重み付け平均した値を算出する。

プログラム 26.14　空調機システムの計算処理

Popolo.HVAC.SubSystem.AHUSystem class

```
 1 /// <summary>冷温水還温度を予想する</summary>
 2 /// <param name="chilledWaterSupplyTemperature">冷水往温度[C]</param>
 3 /// <param name="hotWaterSupplyTemperature">温水往温度[C]</param>
 4 public void ForecastReturnWaterTemperature
 5   (double chilledWaterSupplyTemperature, double hotWaterSupplyTemperature)
 6 {
 7   ChilledWaterSupplyTemperature = chilledWaterSupplyTemperature;
 8   HotWaterSupplyTemperature = hotWaterSupplyTemperature;
 9
10   if (isFirstCall)
11   {
12     isFirstCall = false;
13     for (int i = 0; i < ahu.Length; i++)
14     {
15       hrAHUs[i] = 0;
16       double mRA = 0;
17       foreach (VolumeController vc in vlmCtrl[i])
18       {
19         ImmutableZone zn = bModel.MultiRoom[vc.RoomIndex].Zones[vc.ZoneIndex];
20         hrAHUs[i] += zn.HumidityRatio * vc.MaxRAFlow;
21         mRA += vc.MaxRAFlow;
22       }
23       hrAHUs[i] /= mRA;
24     }
25   }
26   for (int i = 0; i < ahu.Length; i++)
27     if (Controllers[i].Mode != OperatingMode.ShutOff) ahu[i].RAHumidityRatio = hrAHUs[i];
28
29   //AHU の冷温水と風量初期化
30   bool isAllAHUShutoff = true;
31   for (int i = 0; i < ahu.Length; i++)
32   {
33     //冷温水コイルに通水する
34     ahu[i].ChilledWaterInletTemperature = chilledWaterSupplyTemperature;
35     ahu[i].HotWaterInletTemperature = hotWaterSupplyTemperature;
36
37     //給気風量を初期化
38     double maxSA = 0;
39     double maxRA = 0;
40     if (Controllers[i].Mode != OperatingMode.ShutOff)
41     {
42       bool allClosed = true;
43       foreach (VolumeController ctrl in vlmCtrl[i])
44       {
45         if (!ctrl.IsShutOff) allClosed = false;
46         ctrl.SAFlow = ctrl.MaxSAFlow;
47         maxSA += ctrl.MaxSAFlow;
48         maxRA += ctrl.MaxRAFlow;
49       }
50       //全 VAV,CAV が停止の場合には AHU も停止させる
```

```
51        if (allClosed) Controllers[i].Mode = OperatingMode.ShutOff;
52        else ahu[i].SetAirFlowRate(maxRA, maxSA);
53      }
54
55      if (Controllers[i].Mode == OperatingMode.ShutOff)
56      {
57        ahu[i].ShutOff();
58        ctrlFixed[i] = true;
59      }
60      else
61      {
62        isAllAHUShutoff = false;
63        ctrlFixed[i] = false;
64      }
65    }
66
67    //すべての AHU が非稼働の場合には給気量 0 で成行計算を 1 回実行して終了
68    if (isAllAHUShutoff)
69    {
70      for (int i = 0; i < ahu.Length; i++)
71      {
72        foreach (VolumeController vc in vlmCtrl[i])
73        {
74          bModel.SetSupplyAir(vc.RoomIndex, vc.ZoneIndex, 0, 0, 0);
75          bModel.ControlHeatSupply(vc.RoomIndex, vc.ZoneIndex, 0);
76          bModel.ControlWaterSupply(vc.RoomIndex, vc.ZoneIndex, 0);
77        }
78      }
79      bModel.ForecastHeatTransfer();
80      bModel.ForecastWaterTransfer();
81      updateReturnWaterState();
82      return;
83    }
84
85    //顕熱平衡の収束計算
86    solveHeatTransfer();
87
88    //水分平衡の計算
89    for (int i = 0; i < ahu.Length; i++)
90    {
91      foreach (VolumeController vc in vlmCtrl[i])
92      {
93        if (Controllers[i].Mode == OperatingMode.ShutOff)
94          bModel.SetSupplyAir(vc.RoomIndex, vc.ZoneIndex, 0, 0, 0);
95        else bModel.SetSupplyAir
96            (vc.RoomIndex, vc.ZoneIndex, ahu[i].SATemperature, ahu[i].SAHumidityRatio, vc.SAFlow);
97        bModel.ControlHeatSupply(vc.RoomIndex, vc.ZoneIndex, 0);
98        bModel.ControlWaterSupply(vc.RoomIndex, vc.ZoneIndex, 0);
99      }
100   }
101   bModel.ForecastWaterTransfer();
102
103   //冷温水量を更新
104   updateReturnWaterState();
105 }
106
107 /// <summary>還冷温水状態を更新する</summary>
108 private void updateReturnWaterState()
109 {
110   double tcw, thw;
111   tcw = thw = ChilledWaterFlowRate = HotWaterFlowRate = 0;
112   foreach (AirHandlingUnit ah in ahu)
113   {
114     tcw += ah.CoolingCoil.OutletWaterTemperature * ah.CoolingCoil.WaterFlowRate;
115     thw += ah.HeatingCoil.OutletWaterTemperature * ah.HeatingCoil.WaterFlowRate;
116     ChilledWaterFlowRate += ah.CoolingCoil.WaterFlowRate;
117     HotWaterFlowRate += ah.HeatingCoil.WaterFlowRate;
118   }
119   if (ChilledWaterFlowRate == 0) ChilledWaterReturnTemperature = ChilledWaterSupplyTemperature;
120   else ChilledWaterReturnTemperature = tcw / ChilledWaterFlowRate;
121   if (HotWaterFlowRate == 0) HotWaterReturnTemperature = HotWaterSupplyTemperature;
122   else HotWaterReturnTemperature = thw / HotWaterFlowRate;
123 }
124
125 /// <summary>状態を確定する</summary>
126 public void FixState() { isFirstCall = true; }
```

プログラム 26.15 に顕熱平衡の計算処理を示す。第 2 章で解説したように、ニュートン法は初期値の良し悪しにより収束速度と安定性が大きく変わる。そこで、解に近い初期値を与えられるよう

に、4~53 行で、完全に室温が制御できたと仮定した場合の熱負荷から逆算で給気温度を計算し、これを初期値として用いる。55~89 行が図 26.9 の外側のループである。59~67 行で先に用意した初期値を用いて収束計算の状態変数を初期化し、69~80 行でニュートン法を適用する。83 行のメソッド（後述）で過負荷判定を行い、過負荷に変化する空調機が無くなるまで計算を続ける。すべての空調機の状態が確定したら 86 行のメソッド（後述）で VAV 風量の最適化を行い、過冷却あるいは過加熱があれば最後の計算を重ねて終了する。

プログラム 26.15　空調機システムの顕熱平衡計算処理

Popolo.HVAC.SubSystem.AHUSystem class

```
 1 /// <summary>顕熱平衡を解く</summary>
 2 private void solveHeatTransfer()
 3 {
 4   //完全に制御した場合の負荷にもとづいて収束計算初期値を作成
 5   double[] initVec = new double[ahu.Length];
 6   for (int i = 0; i < ahu.Length; i++)
 7   {
 8     if (Controllers[i].Mode == OperatingMode.Cooling
 9       || Controllers[i].Mode == OperatingMode.Heating)
10     {
11       foreach (VolumeController vc in vlmCtrl[i])
12       {
13         double sp;
14         if (Controllers[i].IsCAVControl) sp = Controllers[i].SetpointTemperature;
15         else sp = vc.SetPointTemperature;
16         bModel.ControlDrybulbTemperature(vc.RoomIndex, vc.ZoneIndex, sp);
17         bModel.SetSupplyAir(vc.RoomIndex, vc.ZoneIndex, 0, 0, 0); //DEBUG
18
19         //加湿系統は潜熱負荷を計算する
20         if (Controllers[i].Mode == OperatingMode.Heating
21           && ahu[i].Humidifier != AirHandlingUnit.HumidifierType.None)
22           bModel.ControlHumidityRatio(vc.RoomIndex, vc.ZoneIndex, Controllers[i].MinimumHumidity);
23       }
24     }
25   }
26   bModel.ForecastHeatTransfer();
27   bModel.ForecastWaterTransfer();
28
29   for (int i = 0; i < ahu.Length; i++)
30   {
31     initVec[i] = 0;
32     Controllers[i].splyHumidSP = 0;
33     double sHL = 0;
34     double mRA = 0;
35     double mSA = 0;
36     foreach (VolumeController ctrl in vlmCtrl[i])
37     {
38       ImmutableZone zn = bModel.MultiRoom[ctrl.RoomIndex].Zones[ctrl.ZoneIndex];
39       initVec[i] += zn.Temperature * ctrl.RAFlow;
40       sHL += zn.HeatSupply;
41       mRA += ctrl.MaxRAFlow;
42       mSA += ctrl.MaxSAFlow;
43
44       //加湿系統の AHU 出口湿度を計算する
45       if (Controllers[i].Mode == OperatingMode.Heating
46         && ahu[i].Humidifier != AirHandlingUnit.HumidifierType.None)
47       {
48         double wAHUo = zn.WaterSupply / ctrl.MaxSAFlow + zn.HumidityRatio;
49         Controllers[i].splyHumidSP = Math.Max(Controllers[i].splyHumidSP, wAHUo);
50       }
51     }
52     initVec[i] = initVec[i] / mRA + sHL / (mSA * AIR_SPECIFICHEAT);
53   }
54
55   //過負荷に変化する AHU がなくなるまで繰り返し計算
56   bool lastCalc = false;
57   while (true)
58   {
59     //収束計算対象の給気温度を初期化
60     List<double> vars = new List<double>();
61     for (int i = 0; i < ahu.Length; i++)
62     {
63       AHUController ctr = Controllers[i];
64       if (ctr.Mode != OperatingMode.ShutOff
```

```
65          && (ctrlFixed[i] || ctr.Mode == OperatingMode.Ventilation || ctr.IsCAVControl))
66          vars.Add(initVec[i]);
67      }
68
69      //収束計算実行
70      IVector tandh = new Vector(vars.Count);
71      IVector fx = new Vector(vars.Count);
72      if (vars.Count != 0)
73      {
74        int iter;
75        for (int i = 0; i < tandh.Length; i++) tandh[i] = vars[i];
76        //絶対誤差 0.01 度未満まで収束計算
77        double err;
78        MultiRoots.Newton(errorFnc, ref tandh, 1e-3, 1e-3, 50, out iter, out err);
79      }
80      else errorFnc(tandh, ref fx);
81
82      //過負荷に変化する系統がなくなれば VAV 風量を最小化して最終の計算処理
83      if (!overLoadAHUChanged(tandh))
84      {
85        if (lastCalc) return;
86        else if (minimizeVAV(initVec)) lastCalc = true;
87        else return;
88      }
89    }
90 }
```

プログラム 26.16 にニュートン法で求根する誤差関数を示す。空調機（軽負荷系統は除く）の吹出温度を収束計算対象とし、6~23 行でこれらを熱負荷計算モデルに設定する。25 行で顕熱平衡を更新し、32~42 行で空調機の還気温度を求める。過負荷状態にある空調機または換気運転の空調機の場合には、還気条件から成り行きでの吹出空気状態を計算し、当初に仮定した吹出温度との差を誤差として評価する（44~54 行）。軽負荷 CAV 系統の空調機に関しては制御目標である還気設定温度またはゾーンの設定温度との差を誤差として評価する（55~66 行）。

プログラム 26.16　空調機システムの顕熱平衡誤差関数

Popolo.HVAC.SubSystem.AHUSystem class

```
 1 /// <summary>誤差評価関数</summary>
 2 /// <param name="vecX">AHU 吹出温湿度ベクトル</param>
 3 /// <param name="vecF">誤差ベクトル</param>
 4 private void errorFnc(IVector vecX, ref IVector vecF)
 5 {
 6   //制御点確定済・換気・CAV 制御の場合には給気温度が収束計算対象
 7   int indx = 0;
 8   for (int i = 0; i < ahu.Length; i++)
 9   {
10     AHUController ctr = Controllers[i];
11     if (ctr.Mode != OperatingMode.ShutOff &&
12       (ctrlFixed[i] || ctr.Mode == OperatingMode.Ventilation || ctr.IsCAVControl))
13     {
14       foreach (VolumeController ctrl in vlmCtrl[i])
15       {
16         bModel.SetSupplyAir(ctrl.RoomIndex, ctrl.ZoneIndex, vecX[indx], 0, ctrl.SAFlow);
17         bModel.ControlHeatSupply(ctrl.RoomIndex, ctrl.ZoneIndex, 0);
18       }
19       indx++;
20     }
21   }
22   //顕熱平衡を更新
23   bModel.ForecastHeatTransfer();
24
25   //誤差集計
26   indx = 0;
27   for (int i = 0; i < ahu.Length; i++)
28   {
29     AHUController ctr = Controllers[i];
30     if (ctr.Mode != OperatingMode.ShutOff)
31     {
32       //還気温度を計算
33       double tra = 0;
34       double mra = 0;
35       foreach (VolumeController ctrl in vlmCtrl[i])
36       {
37         ImmutableZone zn = bModel.MultiRoom[ctrl.RoomIndex].Zones[ctrl.ZoneIndex];
```

```
38          tra += zn.Temperature * ctrl.RAFlow;
39          mra += ctrl.RAFlow;
40        }
41        tra /= mra;
42        ahu[i].RATemperature = tra;
43
44        //制御点確定済または換気のみの AHU の給気温度の誤差を計算
45        if (ctrlFixed[i] || ctr.Mode == OperatingMode.Ventilation)
46        {
47          //AHU 成行計算処理
48          if (ctr.Mode == OperatingMode.Cooling) ahu[i].CoolAir();
49          else if (ctr.Mode == OperatingMode.Heating) ahu[i].HeatAir();
50          else ahu[i].Ventilate();
51          //誤差評価
52          vecF[indx] = Math.Abs(ahu[i].SATemperature - vecX[indx]);
53          indx++;
54        }
55        //軽負荷 CAV 系統の制御誤差評価
56        else if (ctr.IsCAVControl)
57        {
58          double sp = ctr.SetpointTemperature;
59          if (ctr.IsRATemperatureControl) vecF[indx] = Math.Abs(tra - sp);
60          else
61          {
62            ImmutableZone zn = bModel.MultiRoom[ctr.TargetRoomIndex].Zones[ctr.TargetZoneIndex];
63            vecF[indx] = Math.Abs(zn.Temperature - sp);
64          }
65          indx++;
66        }
67      }
68    }
69 }
```

プログラム 26.17 に空調機の過負荷判定処理を示す。14~23 行で還気条件、27~41 行で吹出条件をそれぞれ求めた後、42~45 行で吹出温度が実現可能かを判定する。

プログラム 26.17　空調機システムの過負荷判定処理

Popolo.HVAC.SubSystem.AHUSystem class

```
 1 /// <summary>過負荷系統の AHU 台数に変化は無いか</summary>
 2 /// <param name="vec">状態変数リスト</param>
 3 /// <returns>変化の有無</returns>
 4 private bool overLoadAHUChanged(IVector vec)
 5 {
 6   bool incrsd = false;
 7   int indx = 0;
 8   for (int i = 0; i < ahu.Length; i++)
 9   {
10     AHUController ctr = Controllers[i];
11     if (ctr.Mode != OperatingMode.ShutOff && !ctrlFixed[i] &&
12       (ctr.Mode == OperatingMode.Cooling || ctr.Mode == OperatingMode.Heating))
13     {
14       //還気温度を計算
15       ahu[i].RATemperature = 0;
16       double mra = 0;
17       foreach (VolumeController ctrl in vlmCtrl[i])
18       {
19         ImmutableZone zn = bModel.MultiRoom[ctrl.RoomIndex].Zones[ctrl.ZoneIndex];
20         ahu[i].RATemperature += zn.Temperature * ctrl.RAFlow;
21         mra += ctrl.RAFlow;
22       }
23       ahu[i].RATemperature /= mra;
24
25       //過負荷判定
26       double tSP;
27       //CAV 制御の場合には収束計算対象の状態変数が出口温度設定値
28       if (ctr.IsCAVControl) tSP = vec[indx];
29       //VAV の場合には顕熱負荷にもとづき出口温度設定値を計算
30       else
31       {
32         if (ctr.Mode == OperatingMode.Cooling) tSP = 60;
33         else tSP = -10;
34         foreach (VolumeController ctrl in vlmCtrl[i])
35         {
36           ImmutableZone zn = bModel.MultiRoom[ctrl.RoomIndex].Zones[ctrl.ZoneIndex];
37           double bf = zn.Temperature + zn.HeatSupply / (CPMA * ctrl.SAFlow);
38           if (ctr.Mode == OperatingMode.Cooling) tSP = Math.Min(tSP, bf);
39           else tSP = Math.Max(tSP, bf);
40         }
```

```
41        }
42        //出口温度実現可能か
43        if (ctr.Mode == OperatingMode.Cooling) ctrlFixed[i] = !ahu[i].CoolAir(tSP, 0);
44        else ctrlFixed[i] = !ahu[i].HeatAir(tSP, ctr.splyHumidSP);
45        if (ctrlFixed[i]) incrsd = true;
46      }
47
48      if (ctr.Mode != OperatingMode.ShutOff &&
49        (ctrlFixed[i] || ctr.Mode == OperatingMode.Ventilation || ctr.IsCAVControl))
50        indx++;
51    }
52    return incrsd;
53 }
```

プログラム 26.18 に VAV 風量の最適化処理を示す。34 行でプログラム 26.9 を呼び出して最適化を行い、過加熱または過冷却がある場合には 38~44 行で制御状態を確定して最終の収束計算に備える。本クラスの適用例に関しては第 28 章において具体的な建物モデルを構築した後に解説する。

プログラム 26.18　空調機システムの VAV 風量最適化処理

Popolo.HVAC.SubSystem.AHUSystem class

```
 1 /// <summary>VAV 風量を最小化</summary>
 2 /// <remarks>VAV が無ければ false を出力</remarks>
 3 private bool minimizeVAV(double[] initVec)
 4 {
 5   bool hasVAV = false;
 6   for (int i = 0; i < ahu.Length; i++)
 7   {
 8     //制御点未確定で VAV 制御の AHU について計算
 9     if (Controllers[i].Mode != OperatingMode.ShutOff
10       && !Controllers[i].IsCAVControl && !ctrlFixed[i])
11     {
12       bool isCooling = (Controllers[i].Mode == OperatingMode.Cooling);
13       int vcNum = vlmCtrl[i].Length;
14       bool[] vavSt = new bool[vcNum];
15       double[] znT = new double[vcNum];
16       double[] znW = new double[vcNum];
17       double[] znHL = new double[vcNum];
18       double[] minSA = new double[vcNum];
19       double[] maxSA = new double[vcNum];
20       double[] maxRA = new double[vcNum];
21       for (int j = 0; j < vlmCtrl[i].Length; j++)
22       {
23         VolumeController vc = vlmCtrl[i][j];
24         ImmutableZone zn = bModel.MultiRoom[vc.RoomIndex].Zones[vc.ZoneIndex];
25         znT[j] = zn.Temperature;
26         znW[j] = zn.HumidityRatio;
27         znHL[j] = zn.HeatSupply;
28         vavSt[j] = vc.IsShutOff;
29         minSA[j] = vc.MinSAFlow;
30         maxSA[j] = vc.MaxSAFlow;
31         maxRA[j] = vc.MaxRAFlow;
32       }
33       bool sc;
34       double[] aFlow = ahu[i].OptimizeVAV(isCooling, Controllers[i].splyHumidSP,
35         vavSt, znT, znW, znHL, minSA, maxSA, maxRA, out sc);
36       for (int j = 0; j < vlmCtrl[i].Length; j++) vlmCtrl[i][j].SAFlow = aFlow[j];
37
38       //過剰処理系統があれば制御点を確定して収束計算実施
39       if (!sc)
40       {
41         hasVAV = true;
42         ctrlFixed[i] = true;
43         initVec[i] = ahu[i].SATemperature; //収束計算用給気温度仮定値を更新
44       }
45     }
46   }
47   return hasVAV;
48 }
```

【第 26 章 記号表】

A_{Du}	：体表面積 [m²]	m_{sW}	：給水量 [kg/s]
c_{pma}	：湿り空気の定圧比熱 [kJ/(kg·K)]	Met	：代謝量 [met]
C	：CO_2 濃度 [m³/m³]	Q	：換気量 [m³/s]
C_G	：CO_2 発生速度 [m³/s]	R_{Loss}	：熱損失率 [-]
h	：比エンタルピー [kJ/kg]	T	：温度 [K]
HL_L	：潜熱負荷 [kg/s]	V	：気積 [m³]
HL_S	：顕熱負荷 [kW]	W	：絶対湿度 [kg/kg]
I_{CO2}	：人体からの CO2 発生量 [m³/(h·人)]	W_S	：飽和絶対湿度 [kg/kg]
m	：質量流量 [kg/s]	η_{hmS}	：飽和効率 [-]
m_{hmW}	：加湿量 [kg/s]	η_{hmW}	：給水有効利用率 [-]

添字：

AHU	：空気調和機	min	：最小風量運転時
CC	：冷水コイル	o	：出口
EA	：排気	OA	：外気
F	：ファン	RA	：還気
HC	：温水コイル	set	：設定値
HM	：加湿器	SA	：給気
i	：入口	SF	：給気ファン
max	：最大風量運転時	ZN	：空調ゾーン

【第 26 章 参考文献】

26.1） 平野彦兵衛: Air Conditioning 50 周年について, 衛生工業協會誌, 第 26 巻, 第 7 号, pp37-40, 1952

26.2） ウェットマスター株式会社 技術資料 空気調和における加湿と加湿器, 2010.10

26.3） 松元忠雄, 田崎茂: 環境共生世代の建築設備の自動制御入門, 第 1 版, 日本工業出版, 2010

26.4） 大西裕治, 井上貴之, 田島昌樹: 人間の呼気に含まれる CO_2 を利用した居室の換気性能評価, 日本建築学会大会学術講演梗概集, pp.831-832, 2014.9

26.5） JIS A1406 室内換気量測定法（炭酸ガス法）, 2010

26.6） 高橋隆勇: 空調自動制御と省エネルギー, オーム社, 平成 23 年

26.7） 千葉孝男: 空調システムの自動制御, オーム社, 平成 16 年

26.8） 山本佳嗣, 田辺新一: 自然換気システムの換気口解放条件に関する研究, 日本建築学会環境系論文集, 第 81 集, 第 722 号, pp.375-384, 2016.04

26.9） 空調システム標準シミュレーションプログラム HASP/ACSS/8502 プログラム解説書, 日本建築設備士協会, p.101, 1986

第27章　熱的快適性
(Thermal Comfort)

27.1　概要

空調設備システムの目的の一つは、室内の熱環境を制御することで居住者にとっての熱的快適性を担保することにある。熱的快適性は温度、湿度、気流速度などを含むいくつもの要素に大きく影響を受けるため、これらの要素を考慮した快適性の予測法が必要になる。このために基礎となる概念が人体の熱収支を解く人体モデルである。

これまでの章で解説してきた設備機器や建物と比較すると、人間の熱的な感覚に影響を与える要素は非常に多く、また、個人差も大きい。従って、いくつかの限定的な熱的要素にもとづく熱平衡計算だけでは人間の感覚を完全に予測することはできない。そこで、熱平衡計算の結果と被験者実験などの実測データとを結びつけた統計的なモデルが提案されている。本書では Gagge と西らによる 2-Node モデル、P. O. Fanger による PMV、田辺による JOS について解説する。

人間の不確実性を相手にするという姿勢は、工業生産品を相手にするプラント設計には無く、建築熱環境設計の難しさと面白さの源であり、醍醐味と言える。後章では、本章で解説する PMV を用いて熱的快適性を制約条件にとりながら各種の省エネルギー化手法の導入を試みる。しかし、これはエネルギーと快適性を同時に評価するための 1 つの便法であり、両者の真の統合が容易に成し遂げられると考えてはならない。人体モデルを用いた熱的快適性の評価は、人間という不思議な存在に熱的な観点から迫る 1 つの切り口にすぎない。

写真 27.1　サーマルマネキンと研究者たち[27.1)]
左から James Bogart, Harwood E. Belding, Ralph F. Goldman, J. Robert Breckenridge
(着衣評価のためのサーマルマネキンは極寒地域における兵士の活動可能性検討を契機に大きく発達したという)

27.2　理論

本書では、まず、Gagge らによる 2-Node モデル[27.2]および P. O. Fanger による PMV[27.5]について解説を行う。2-Node モデルは人体を核（Core）と皮膚（Shell）の 2 つの Node として表現する単純なモデルであるが、世界的に知られた SET*を導出する際に使われるモデルであるとともに、後続の様々な研究で繰り返し参照される基礎的なモデルである。PMV も 2-Node モデルと同様に人体の熱収支を解くが、最終的に温冷感申告値に結びつける点に特徴がある。SET*と PMV は上下温度分布がある場合や空調立ち上がり時で熱環境が推移する場合など、不均一・非定常な場合には直接的には適用できない。この問題に対応させた人体モデルも多く開発されており[27.9) 27.10) 27.11) 27.12]、本書では田辺らによる JOS（Joint System Thermoregulation Model）[27.8]について解説する。

27.2.1　2-Node モデル

人体が定常状態にあるとすれば、人体内部で発生する熱と人体から外部へ散逸する熱は釣り合うはずである。これを式で表現すると式 27.1 が得られる。

$$M_r - W_{exr} - E_{resr} - C_{resr} - E_{swr} - E_{skr} = R_{clr} + C_{clr} \tag{27.1}$$

左辺の M_r は代謝量[W/m²]、W_{exr} は外部仕事[W/m²]である。E_{resr} [W/m²]と C_{resr} [W/m²]は呼吸による熱損失量を示しており、前者は蒸発による潜熱損失、後者は対流による顕熱損失である。E_{swr} [W/m²]は発汗による潜熱損失量である。E_{skr} [W/m²]は人体表面から外界への潜熱の移動を表現しており、式 27.2 で計算できる。w は皮膚の濡れ率[-]、h_e'は潜熱伝達率[W/(m²·kPa)]、$P_{Ws,sk}$ は皮膚温度に対する飽和水蒸気圧[kPa]、$P_{W,a}$ は外界の水蒸気分圧[kPa]である。

$$E_{skr} = w h'_e \left(P_{Ws,sk} - P_{W,a} \right) \tag{27.2}$$

右辺の R_{clr} [W/m²]と C_{clr} [W/m²]はそれぞれ放射と対流による人体表面から外界への顕熱の移動を表現しており、式 27.3 で計算できる。h'は顕熱伝達率[W/(m²·K)]、T_{sk} は皮膚温度[K]、T_o は外界の作用温度[K]である。

$$R_{clr} + C_{clr} = h' \left(T_{sk} - T_o \right) \tag{27.3}$$

2-Node モデルは人体を核（Core）と皮膚（Skin）の 2 つの節点（Node）として表現するモデルである。核から皮膚への熱流を $Q_{cr\text{-}sk}$ [W/m²]とすれば、核と皮膚のそれぞれに流入する熱は式 27.4 と 27.5 で表現できる。

$$Q_{cr} = M_r - W_{exr} - E_{resr} - C_{resr} - Q_{cr-sk} \tag{27.4}$$

$$Q_{sk} = Q_{cr-sk} - h' \left(T_{sk} - T_o \right) - w h'_e \left(P_{Ws,sk} - P_{W,a} \right) - E_{swr} \tag{27.5}$$

定常状態においては各々の節点で出入りする熱が釣り合い、$Q_{cr} = Q_{sk} = 0$ となるため、式 27.1 が成立する。2-Node モデルの概念を図 27.1 に示す。式 27.1~27.5に表れる各要素の計算方法は下記のとおりである。

1）顕熱伝達率 h'と潜熱伝達率 h_e' の計算

・顕熱伝達率 h'

顕熱伝達率 h'の計算式を式 27.6 に示す[27.2]。I_{clo} は衣服の熱抵抗を表す着衣量であり単位は clo である。表 27.2 に代表的な衣服の clo 値を示す。着衣に応じて表 27.2 から積み上げて計算をすればよい

が、事務室であれば夏季は 0.6 程度、冬季は 1.0 程度としてもよい。f_{cl} は人体表面に対する衣服の表面積の比率[-][†1]を意味しており、clo 値の関数として式 27.7 で計算する。h_r と h_c はそれぞれ放射熱伝達率[W/(m²·K)]と対流熱伝達率[W/(m²·K)]である。放射熱伝達率は衣服の温度 T_{cl} [°C]と平均放射温度 T_{mr} [°C]の関数として式 27.8 で計算する。ただし σ は黒体の放射定数で 5.67×10^{-8} W/(m²·K⁴)である。また有効放射定数 εf_{eff} は 0.72[-]とする。対流熱伝達率は式 27.9 で計算する。1 つ目の項は相対気流速度 v [m/s]に基づく強制対流による対流熱伝達率の計算式[27.29]であり、2 つ目の項は自然対流による対流熱伝達率の計算式である。ただし v は安静時の気流速度である 0.15 m/s を下限値とする。

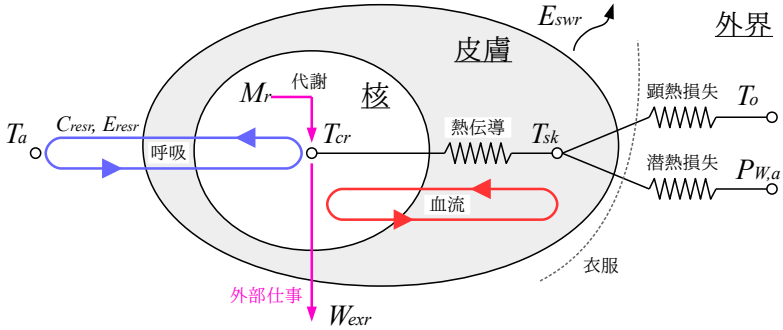


図 27.1　2-Node モデルの概念

$$\frac{1}{h'}=\frac{1}{f_{cl}(h_r+h_c)}+0.155\,I_{clo} \tag{27.6}$$

$$f_{cl}=1+0.25\,I_{clo} \tag{27.7}$$

$$h_r=4\,\sigma\,\varepsilon\,f_{eff}\left(\frac{T_{cl}+T_{mr}}{2}\right)^3 \tag{27.8}$$

$$h_c=Max\left(8.6\,v^{0.53},5.66\left(Met-0.85\right)^{0.39}\right) \tag{27.9}$$

Met は活動量を表す数値であり、椅座安静時の代謝量である 58.15 W/m² を基準として無次元化した値である。表 27.1 に活動内容と Met 値、代謝量との関係を示す。なお、個体差が表現できるように、代謝量は体表面積で除した単位面積あたりの数値である W/m² として表現している。

・潜熱伝達率 h_e'

潜熱伝達率 h_e'の計算式を式 27.10 に示す[27.2]。i_{cl} は透湿係数[kPa/K]であり、0.45 で固定値とする[27.14]。L_e はルイスの係数[K/kPa]であり、皮膚温度の関数として式 27.11 で計算する。

$$\frac{1}{h_e'}=\frac{1}{L_e}\left(\frac{1}{f_{cl}h_c}+\frac{0.155\,I_{clo}}{i_{cl}}\right) \tag{27.10}$$

$$L_e=15.15\frac{T_{sk}}{273.15}=0.0555\,T_{sk} \tag{27.11}$$

・体表面積 A_{du} [m²]

上記の顕熱伝達率 h'と潜熱伝達率 h_e'はいずれも単位体表面積あたりの数値（W/(m²·K)と W/(m²·kPa)）として計算されるため、これに乗ずる体表面積を計算する必要がある。Dubois によれば人体の表面積は体重 Wt [kg]と身長 H [m]の関数として式 27.12 で計算できる[27.15]。2-node モデルで SET*を計算する際には標準体躯として体重 70 kg、身長 1.7 m、体表面積 1.8m² の人間を前提とする。

†1　「着膨れ」により外界との熱交換面積が増加することの影響を評価するためである。

$$A_{du} = 0.202\, Wt^{0.425}\, H^{0.725} \tag{27.12}$$

表 27.1　活動と Met 値、代謝量の関係[27.6)]

活動	Met 値 [-]	代謝量 [W/m²]	活動	Met 値 [-]	代謝量 [W/m²]
Resting			Miscellaneous Occupational Activities		
Sleeping	0.7	40	Cooking	1.6~2.0	95~115
Reclining	0.8	45	House cleaning	2.0~3.4	115~200
Seated, quiet	1.0	60	Seated, heavy limb movement	2.2	130
Standing, relaxed	1.2	70	Machine work		
Walking (on level surface)			sawing (table saw)	1.8	105
0.9 m/s, 3.2 km/h, 2.0 mph	2.0	115	light (electrical industry)	2.0~2.4	115~140
1.2 m/s, 4.3 km/h, 2.7 mph	2.6	150	heavy	4.0	235
1.8 m/s, 6.8 km/h, 4.2 mph	3.8	220	Handling 50 kg (100 lb) bags	4.0	235
Office Activities			Pick and shovel work	4.0~4.8	235~280
Seated, reading, or writing	1.0	60	Miscellaneous Leisure Activities		
Typing	1.1	65	Dancing, social	2.4~4.4	140~255
Filing, seated	1.2	70	Calisthenics/exercise	3.0~4.0	175~235
Filing, standing	1.4	80	Tennis, single	3.6~4.0	210~270
Walking about	1.7	100	Basketball	5.0~7.6	290~440
Lifting/packing	2.1	120	Wrestling, competitive	7.0~8.7	410~505
Driving/Flying					
Automobile	1.0~2.0	60~115			
Aircraft, routine	1.2	70			
Aircraft, instrument landing	1.8	105			
Aircraft, combat	2.4	140			
Heavy vehicle	3.2	185			

表 27.2　代表的な衣服の clo 値[27.6)]

衣服	着衣量	衣服	着衣量
Underwear		Dress and Skirts	
Bra	0.01	Skirt (thin)	0.14
Panties	0.03	Skirt (thick)	0.23
Men's briefs	0.04	Sleeveless, scoop neck (thin)	0.23
T-shirt	0.08	Sleeveless, scoop neck (thick), i.e.,jumper	0.27
Half-slip	0.14	Short-sleeve shirtdress (thin)	0.29
Long underwear bottoms	0.15	Long-sleeve shirtdress (thin)	0.33
Full slip	0.16	Long-sleeve shirtdress (thick)	0.47
Long underwear top	0.20	Sweaters	
Footwear		Sleeveless vest (thin)	0.13
Ankle-length athletic socks	0.02	Sleeveless vest (thick)	0.22
Pantyhose/stockings	0.02	Long-sleeve (thin)	0.25
Sandals/thongs	0.02	Long-sleeve (thick)	0.36
Shoes	0.02	Suit Jackets and Vestsd	
Slippers (quilted, pile lined)	0.03	Sleeveless vest (thin)	0.10
Calf-length socks	0.03	Sleeveless vest (thick)	0.17
Knee socks (thick)	0.06	Single-breasted (thin)	0.36
Boots	0.10	Single-breasted (thick)	0.42
Shirts and Blouses		Double-breasted (thin)	0.44
Sleeveless/scoop-neck blouse	0.13	Double-breasted (thick)	0.48
Short-sleeve knit sport shirt	0.17	Sleepwear and Robes	
Short-sleeve dress shirt	0.19	Sleeveless short gown (thin)	0.18
Long-sleeve dress shirt	0.25	Sleeveless long gown (thin)	0.20
Long-sleeve flannel shirt	0.34	Short-sleeve hospital gown	0.31
Long-sleeve sweatshirt	0.34	Short-sleeve short robe (thin)	0.34
Trousers and Coveralls		Short-sleeve pajamas (thin)	0.42
Short shorts	0.06	Long-sleeve long gown (thick)	0.46
Walking shorts	0.08	Long-sleeve short wrap robe (thick)	0.48
Straight trousers (thin)	0.15	Long-sleeve pajamas (thick)	0.57
Straight trousers (thick)	0.24	Long-sleeve long wrap robe (thick)	0.69
Sweatpants	0.28		
Overalls	0.30		
Coveralls	0.49		

2）体温制御機構の計算

人体は深部の温度を安定させるために、各種の制御を行うことで外界への熱移動量を調整する。 2-

Node モデルでは、1）皮膚の血管の拡張と収縮によるコアと皮膚間の熱移動量の調整、2）発汗による蒸発熱損失量の拡大、3）ふるえによる代謝量の拡大という 3 種類の体温制御機構についてモデル化を行っている。

・血管運動

皮膚の血流 BF_{skr} [mL/(m²·s)]は式 27.13 で計算する。$BF_{skr,b}$[mL/(m²·s)]は熱的に中立な状態における血流（基礎血流）であり 1.75 mL/(m²·s)とする。SIG_{dilr} と SIG_{strr} は血管拡張と血管収縮に関する制御信号であり、式 27.14 と式 27.15 で計算する。$T_{cr,b}$ と $T_{sk,b}$ はそれぞれ熱的中立状態におけるコア温度と皮膚温度であり、$T_{cr,b}$=36.8 °C、$T_{sk,b}$=33.7 °C である。また、C_{dil} と C_{str} は制御信号に対する係数であり、C_{dil}=55.6 mL/(m²·s·K)、C_{str}=0.1 K^{-1} とする。制御信号によって血流値は変化するが、その範囲は 0.139 ~ 25 mL/(m²·s)とする。

$$BF_{skr}=\frac{BF_{skr,b}+C_{dil}SIG_{dilr}}{1+C_{str}\cdot SIG_{strr}} \tag{27.13}$$

$$SIG_{dilr}=Max(T_{cr}-T_{cr,b},0) \tag{27.14}$$

$$SIG_{strr}=Max(T_{sk,b}-T_{sk},0) \tag{27.15}$$

・発汗

発汗量 m_{swr} [mg/(m²·s)]は式 27.16 で計算する。SIG_{swr1} と SIG_{swr2} は発汗に関する制御信号であり、式 27.17 と式 27.18 で計算する。C_{sw} は制御信号に対する係数であり、C_{sw}=47.2 mg/(m²·s·K)とする。また α はコアと皮膚との重量比であるため、式 27.17 の第一項は平均体温を示している。α は式 27.19 で計算し、血流量が多いほど α が小さい（皮膚部分の重量比が小さい）傾向にある。発汗量に 35°C における水の蒸発潜熱 γ_{35} (2,418 J/g)を乗じ、発汗による蒸発熱損失 E_{swr}[W/m²]は式 27.20 で計算できる。

$$m_{swr}=C_{sw}SIG_{swr1}\cdot\exp(SIG_{swr2}/10.7) \tag{27.16}$$

$$\begin{aligned}SIG_{swr1}&=Max((\alpha T_{sk}+(1-\alpha)T_{cr})-(0.1T_{sk,b}+0.9T_{cr,b}),0)\\&=Max((\alpha T_{sk}+(1-\alpha)T_{cr})-36.49,0)\end{aligned} \tag{27.17}$$

$$SIG_{swr2}=Max(T_{sk}-T_{sk,b},0) \tag{27.18}$$

$$\alpha=0.0417737+0.2069953/(BF_{skr}+0.1626158) \tag{27.19}$$

$$E_{swr}=m_{swr}\gamma_{35}/1000 \tag{27.20}$$

・ふるえ

ふるえによる熱産出 M_{shvr} [W/m²]は式 27.21 で計算する。SIG_{shvr1} と SIG_{shvr2} はふるえ熱産生に関する制御信号であり、式 27.22 と式 27.23 で計算する。C_{shv} は制御信号に対する係数であり、C_{shv}=19.4 W/(m²·K²)とする。代謝量 M_r [W/m²]はふるえによる熱産出 M_{shvr} [W/m²]に基礎代謝 M_{br} [W/m²]を加算して式 27.24 で計算する。

$$M_{shvr}=C_{shv}\cdot SIG_{shvr1}\cdot SIG_{shvr2} \tag{27.21}$$

$$SIG_{shvr1}=Max(T_{sk,b}-T_{sk},0) \tag{27.22}$$

$$SIG_{shvr2}=Max(T_{cr,b}-T_{cr},0) \tag{27.23}$$

$$M_r=M_{br}+M_{shvr} \tag{27.24}$$

3）呼吸に伴う熱損失の計算

呼吸に伴う潜熱損失量および顕熱損失量は式 27.25 と 27.26 で計算できる。

$$E_{resr}=V_{resr}\gamma_{35}(W_{res}-W_a) \tag{27.25}$$

$$C_{resr}=V_{resr}Cp_a(T_{res}-T_a) \tag{27.26}$$

V_{resr}は呼吸量[(g/s)/m^2]であり代謝量の関数として式 27.27 で計算する[27.16) 27.17)]。

$$V_{resr}=0.0014M_r \tag{27.27}$$

W_{res}とT_{res}はそれぞれ呼気の絶対湿度 [kg/kg]と乾球温度 [°C]である。

吸気と呼気の湿度差は吸気絶対湿度の関数として式 27.28 で計算できる[27.18)]。水蒸気分圧で表現するために、絶対湿度の計算式 27.29 を式 27.28 に代入し、式 27.30 が得られる。

$$W_{res}-W_a=0.029-0.8W_a \tag{27.28}$$

$$W_a=0.62198\frac{P_{W,a}}{P-P_{W,a}}\approx 0.0061786P_{W,a} \tag{27.29}$$

$$W_{res}-W_a=0.029-0.0049429P_{W,a} \tag{27.30}$$

式 27.27 と式 27.30 を式 27.25 に代入すると、潜熱損失量の計算式 27.31 が得られる。なお、35 °C の水の蒸発潜熱λ_{35}は 2,400 J/g である。

$$E_{resr}=0.017251M_r(5.8662-P_{W,a}) \tag{27.31}$$

呼気の乾球温度T_{res}を 34°C で一定、乾き空気の比熱Cp_aを 1.0 kJ/(kg·K)とすると、呼吸に伴う顕熱損失量C_{res}は式 27.32 で計算できる。

$$C_{resr}=0.0014M_r((34+273.15)-T_a)=0.0014M_r(307.15-T_a) \tag{27.32}$$

4）皮膚の濡れ率の計算

皮膚表面での蒸発熱損失は、皮膚表面が 100%濡れている場合に最大となり、その値E_{max} [W/m^2]は式 27.33 で計算できる。また、発汗による蒸発熱損失は式 27.20 で計算したとおりであるため、発汗に起因する皮膚表面の濡れ率w_{sw} [-]は式 27.34 で表現することができる。実際には発汗がない場合においても人体の皮膚表面からは蒸発が生じており、このような蒸発を不感蒸泄と呼ぶ。これに対して前述の体温制御を目的とした積極的な発汗を調節発汗と呼ぶ。不感蒸泄による熱損失量E_{br} [W/m^2]は式 27.35 で計算できる。即ち、最大蒸発熱損失量から発汗による熱損失量を減じた値に対して 6%を乗じた値である。不感蒸泄による熱損失量を皮膚の濡れ率w_b [-]に換算すると式 27.36 となる。

$$E_{max}=h'_e(P_{Ws,sk}-P_{W,a}) \tag{27.33}$$

$$w_{sw}=E_{swr}/E_{max} \tag{27.34}$$

$$E_{br}=0.06(E_{max}-E_{swr}) \tag{27.35}$$

$$w_b=E_{br}/E_{max} \tag{27.36}$$

一方で、非常に発汗量が多い場合には、全ての汗が皮膚表面で有効に熱移動を伴って蒸発せず、雫として滴下することになる。この臨界値をw_{crit}=85 %とすれば、皮膚の濡れ率w [-]は式 27.37 で計算できる。

$$w=Min(w_{crit},w_{sw}+w_b) \tag{27.37}$$

5）コアから皮膚への熱流の計算

コアから皮膚への熱流$Q_{cr\text{-}sk}$ [W/m^2]は熱伝導と血流によって生じ、式 27.38 で計算できる。$K_{cr\text{-}sk}$はコアと皮膚との間の熱コンダクタンス、ρCp_bは血液の体積比熱であり、それぞれ 5.28 W/(m^2·K)と 3.842 J/(mL·K)とする。

$$Q_{cr-sk}=\left(K_{cr-sk}+\rho Cp_b\cdot BF_{skr}\right)\left(T_{cr}-T_{sk}\right) \tag{27.38}$$

6）新標準有効温度（SET* : Standard Effective Temperature）の計算

前述の式を連立させることで、ある条件下における人体の皮膚表面の濡れ率や人体から外界への熱流を解くことができる（解き方は後述する）。相対湿度と相対気流速度を固定した上で、ある条件と同等の熱流と皮膚濡れ率を実現する乾球温度を SET*と呼ぶ。相対湿度は 50%、相対気流速度は 0 m/s、着衣量は活動量に応じて標準化された値とする。

標準化された着衣量 I_{cls} [clo]は式 27.39 で計算する。ただし、WK は met 単位で表現した外部仕事量であり、$W_{exr}=58.15\times WK$ である。

$$I_{cls}=1.3264/\left(Met-WK+0.7383\right)-0.0953 \tag{27.39}$$

SET*の定義により式 27.40 が成立する。ただし、標準条件における顕熱伝達率 h_s'と潜熱伝達率 h_{es}'は式 27.39 の標準化された着衣量と式 27.6~27.10を用いることで計算する。また、対流熱伝達率 h_cの下限値は 3 W/(m²·K)とする。2-Node モデルを解けば式 27.40 の左辺の値が得られる。右辺の $P_{Ws,SET}$ [kPa]は温度 SET*における飽和蒸気圧であり、相対湿度 50 %であるため 0.5 を乗じている。$P_{Ws,SET}$ は SET*の関数であり、式 27.40 を SET*について解析的に解くことはできないため、反復計算が必要となる。

$$h'\left(T_{sk}-T_o\right)+w\,h'_e\left(P_{Ws,sk}-P_a\right)=h_s^{'}\left(T_{sk}-SET^*\right)+w\,h_{es}^{'}\left(P_{Ws,sk}-0.5\,P_{Ws,SET}\right) \tag{27.40}$$

7）温熱六要素

以上のモデルにも表現されているように、人体の熱収支を検討する際には、乾球温度、相対湿度、平均放射温度、相対気流速度という 4 つの外界条件と、着衣量と活動量という 2 つの人間側の条件が計算結果に大きな影響を与える。これらの 6 つの要素を温熱六要素と呼ぶ†1)。

27.2.2　予測平均温冷感申告 (PMV : Predicted Mean Vote)

1）PMV

予測平均温冷感申告（PMV）は P. O. Fanger により提唱された人間の温冷感に関する指標であり、2-Node モデルと同様に人体の熱平衡式である式 27.1 を基礎におく。2-Node モデルの節において説明したように、人体には体温制御機構があるため、様々な温熱環境下において式 27.1 を満足し、熱平衡を保つことができる。しかし、実際に人間が快適を感じる領域は狭い。そこで、人間が快適を感じる皮膚温および発汗量を予め求めておき、この皮膚温と発汗量を人体モデルに代入した場合の「熱平衡の崩れ」に基づいて熱的快適性を予測しようとするものが PMV である。この熱平衡の崩れを「人体への負荷 L」と呼び†2)、式 27.41 で計算する。

$$L=\left(M_r-W_{exr}\right)-E_{skr}-E_{swr}-E_{resr}-C_{resr}-R_{clr}-C_{clr} \tag{27.41}$$

人体への負荷 L を用いると、PMV は式 27.42 で計算できる。

$$PMV=\left(0.303\exp\left(-0.036\,M_r\right)+0.028\right)L \tag{27.42}$$

式 27.41 の各要素の計算法†3)は、ほとんどが 2-Node モデルと同じであるが、いくつか異なるものがある。まず、放射による熱伝達は、2-Node モデルにおいては線形化した放射熱伝達率 h_rを用いた簡

†1　乾球温度以外の要素も人間の温熱感に影響を与えうるという説は、18 世紀初頭に William Heberden が初めて唱えたようだ[27.33]。
†2　建物の熱負荷計算における「負荷」とは異なり、人体の体温制御機構の稼働状況という意味での「負荷」である。
†3　ISO による計算方法[27.7]を前提とする。ASHRAE Standard[27.6]に示された計算法と同一である。

易計算としていたが、PMV ではステファンボルツマンの法則に従い、式 27.43 で計算を行う。f_{eff}は有効放射面積率[-]であり、座位では 0.696、立位では 0.725 となり、計算上は両者の平均である 0.71 を用いる。放射率 ε は 0.97 とする。また人体表面積に対する着衣面積率 f_{cl} [-]は式 27.44 に示す通り、場合分けして計算を行う。ただし R_{clo} は着衣抵抗[(m²·K)/W]であり、式 27.45 に示す通り着衣量 I_{clo} [clo]を用いて計算する。

$$R_{clr}=f_{eff}\,f_{cl}\,\varepsilon\,\sigma\left(T_{cl}^4-T_{mr}^4\right)=3.96\cdot10^{-8}\,f_{cl}\left(T_{cl}^4-T_{mr}^4\right) \tag{27.43}$$

$$f_{cl}=\begin{cases}1.00+1.290\,R_{clo} & (R_{clo}\le 0.078)\\ 1.05+0.645\,R_{clo} & (R_{clo}>0.078)\end{cases} \tag{27.44}$$

$$R_{clo}=0.155\,I_{clo} \tag{27.45}$$

対流熱伝達率は式 27.46 で計算する。強制対流と自然対流で場合分けをする点は 2-Node モデルと同様であるが、計算式がやや異なり、特に自然対流に関しては代謝量ではなく着衣表面温度 T_{cl} と外界温度 T_a との温度差の関数として表現している。

$$h_c=Max\left(12.1\sqrt{v},\,2.38\left(T_{cl}-T_a\right)^{0.25}\right) \tag{27.46}$$

人体表面からの潜熱損失は式 27.47 で計算する。K_W は皮膚の水分コンダクタンスであり 1.271×10^{-6} kg/(s·kPa·m²)とする。P_a [kPa]は空気の水蒸気分圧である。皮膚温が 27~37°C の範囲であれば飽和水蒸気圧 P_{Ws} [kPa]と乾球温度 T [K]の関係は式 27.48 で誤差 3 %未満で近似可能であるため、これを式 27.47 に代入して式 27.49 が得られる。

$$E_{skr}=\gamma_{35}K_W\left(P_{Ws,sk}-P_a\right) \tag{27.47}$$

$$P_{Ws}=0.25598\,T-73.293 \tag{27.48}$$

$$E_{skr}=3.05\left(0.25598\,T_{sk}-73.293-P_a\right) \tag{27.49}$$

以上を式 27.41 に反映させると式 27.50 となる[†1)]。

$$\begin{aligned}L=&\left(M_r-W_{ex}\right)-3.05\left(0.24964\,T_{sk}-73.293-P_a\right)-E_{swr}-0.017\,M_r\left(5.867-P_a\right)\\&-0.0014\,M_r\left(307.15-T_a\right)-3.96\cdot10^{-8}\,f_{cl}\left(T_{cl}^4-T_{mrt}^4\right)-f_{cl}\,h_c\left(T_{cl}-T_a\right)\end{aligned} \tag{27.50}$$

Fanger の被験者実験によれば、人間が熱的快適性を感じる皮膚温度 $\bar{T}_{sk}$ [K]および発汗による熱損失量 $\bar{E}_{swr}$ [W/m²]は式 27.51 と 27.52 で表現される。

$$\bar{T}_{sk}=308.85-0.028\left(M_r-W_{ex}\right) \tag{27.51}$$

$$\bar{E}_{swr}=Max\left(0,0.42\left(M_r-W_{ex}-58.15\right)\right) \tag{27.52}$$

これらを式 27.50 に代入すると式 27.53 が得られる。

$$\begin{aligned}L=&\left(M_r-W_{exr}\right)-3.05\left(5.733-0.00699\left(M_r-W_{exr}\right)-P_a\right)\\&-0.42\left(M_r-W_{exr}-58.15\right)-0.017\,M_r\left(5.867-P_a\right)\\&-0.0014\,M_r\left(307.15-T_a\right)-3.96\cdot10^{-8}\,f_{cl}\left(T_{cl}^4-T_{mr}^4\right)-f_{cl}\,h_c\left(T_{cl}-T_a\right)\end{aligned} \tag{27.53}$$

式 27.53 を計算するためには着衣表面の温度 T_{cl} [K]を求める必要がある。人体表面から着衣表面までの熱流 Q_{clr} は$(T_{sk}-T_{cl})$ / R_{clo} であり、式 27.51 を代入すると式 27.54 となる。着衣表面から外界への熱流である R_{clr} と C_{clr} の合算値と等式で結び、T_{cl} について解けば式 27.55 が得られる。式 27.55 は左辺と右辺の両方に T_{cl} を含むため、適当な初期値を設定して反復計算を行う。

†1　呼吸による線熱損失 E_{res} は 2-Node モデルとわずかに係数の桁数が異なる。

$$Q_{clr}=\frac{T_{sk}-T_{cl}}{R_{clo}}=\frac{308.85-0.028(M_r-W_{exr})-T_{cl}}{R_{clo}} \tag{27.54}$$

$$T_{cl}=308.85-0.028(M_r-W_{exr})-R_{clo}\left\{3.96\cdot10^{-8}f_{cl}\left(T_{cl}^4-T_{mr}^4\right)+f_{cl}h_c(T_{cl}-T_a)\right\} \tag{27.55}$$

式 27.55 を解き T_{cl} を求め、式 27.50 に T_{cl} を代入して人体への負荷 L を計算すれば、式 27.42 によって PMV を計算することができる。

2）予測不満足者率（PPD: Predicted Percentage of Dissatisfied)

予測不満足者率（PPD）は、ある温熱環境において不満を感じる人の割合を予測するための指標である。PPD は PMV の関数であり、式 27.56 で計算する。PMV と PPD の関係は図 27.2 で示される[†1)]。

$$PPD=100-95\exp(-0.03353\,PMV^4-0.2179\,PMV^2) \tag{27.56}$$

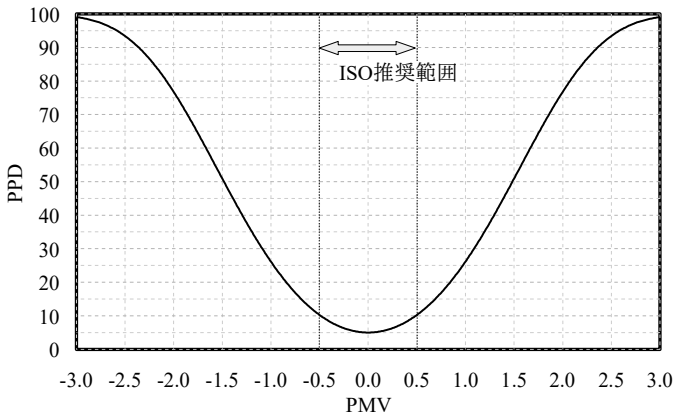


図 27.2　PMV と PPD の関係

27.2.3　非定常・不均一モデル

2-Node モデルは人体を核と皮膚の 2 つの節点で捉える単純なモデルであり、下記のような条件には対応できない。

・温熱環境が不均一で、体の部位によって暴露される環境が異なる場合

・温熱環境が時間変化する非定常な場合

非定常かつ不均一な条件に対応するためには、人体の部位ごとに節点を設けて熱移動をより詳細に表現するという方法がある[†2)]。このような計算法の古典は Stolwijk による人体モデルであり[27.10)]、同モデルでは人体を頭（Head）、胴（Trunk）、左右の腕（Arm）、手（Hand）、脚（Legs）、足（Feet）の 10 の部位に分割して熱流を解いている。また、各部位はさらに核（Core）、筋肉（Muscle）、脂肪（Fat）、皮膚（Skin）の 4 つの層に分割される。その後、多くの者が同様の多節点モデルを提案している[27.8) 27.9) 27.11) 27.12)]。本書では田辺等による JOS（Joint System Thermoregulation Model）を解説する[†3)]。本モデルは Stolwijk モデルに比較して、1）部位間の血流を正確にモデル化、2）手掌部・足底部の AVA 血流をモデル化、3）年齢・性別の変更に対応、4）胴を旨と背中に細分割、脚を大腿部と下腿部に細分割、としている点が特徴である。

†1　PPD のミソは、たとえ PMV が 0 となる条件であっても、平均的には 5%の居住者が不満を感じるという点にある。このように人間が不完全でままならない様子が表現されたモデルは多くの人に「ウケる」。タスクアンビエント空調の後ろ盾ともなるモデルである。

†2　ただし、歴史的には Stolwijk による非定常不均一モデルが先にあり[27.3)]、これを簡易化する形で Two Node Model[27.4)]が提案されたことには注意が必要である。

†3　田辺らによるオリジナルの JOS では、各人体パーツは核と皮膚の 2-Node で表現されているが、本書では、同じく田辺らによる人体モデルである 65MN（Multi-Node）モデルの成果を反映し、各パーツを核、筋肉、脂肪、皮膚の 4-Node で表現する。

1）人体の分割

図 27.3 に人体の分割と血液の流れを示す。表 27.3 に各部位の表面積比[-]と重量比[-]を示す。各部位は血液を介して熱的に連結される。血液は、胸にある中央血液溜まりを起点として頭、手、脚、背中などに向かって動脈を流れていき、静脈を流れて中央血液溜まりへ戻る。特に四肢部位においては表在静脈と深部静脈があり[27.11]、手または脚に到達した血液は熱環境に応じて表在静脈と深部静脈に分配される†1)。また、各部位には静脈と動脈の他、核（骨+内蔵等）、筋肉、脂肪、皮膚の組織があり、これらの間でも熱交換が行われる。以下、それぞれの組織を添字の *ve*（Vein：静脈）、*sv*（Superficial Vein：表在静脈）、*ar*（Artery：動脈）、*cr*（Core：核）、*ms*（Muscle：筋肉）、*ft*（Fat：脂肪）、*sk*（Skin：皮膚）で表現する。

体表面積 A_{du} [m^2]は DuBois による式 27.12 により求め、これに表 27.3 の表面積比 $R_{A,i}$ [%]を乗じることで各部位の表面積 $A_{du,i}$ [m^2]を求める。同様に全身の体重に重量比 $R_{W,i}$ [%]を乗じることで各部位の重量 Wt_i [kg]を求める。

表 27.3　部位別の表面積比 [%]・重量比 [%]

i	部位	面積比 $R_{A,i}$ [%]	重量比 $R_{W,i}$ [%]
0	頭	5.59	4.27
1	首	1.51	1.13
2	胸	9.00	16.66
3	背中	8.21	14.82
4	下腹部	12.75	23.62
5	左肩	5.15	2.90
6	左腕	3.27	1.84
7	左手	2.43	0.46
8	右肩	5.15	2.90
9	右腕	3.27	1.84
10	右手	2.43	0.46
11	左大腿部	11.69	9.42
12	左下腿部	5.66	4.49
13	左脚	3.27	0.64
14	右大腿部	11.69	9.42
15	右下腿部	5.66	4.49
16	右脚	3.27	0.64
全身		100.00	100.00

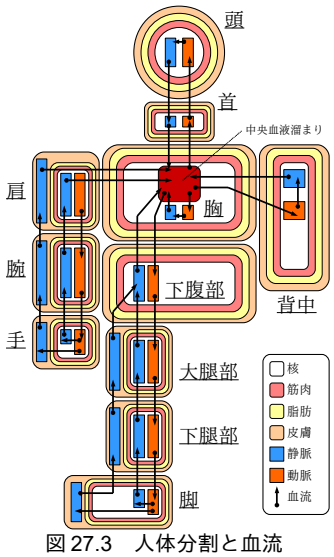


図 27.3　人体分割と血流

表 27.4 に各部位における組織別の重量比 $R_{Wst,i,j}$ [%]を示す[27.10) 27.19)]。

表 27.4 の比率は Stolvijk モデルを細分化した基準値であり、体脂肪率（体重に対する脂肪重量の割合）は 15 %である。体脂肪率の個体差を評価するためには、式 27.57 と式 27.58 を用いて各組織の重量比を調整する。ここで $R_{fat,st}$ は体脂肪率の基準値（=15 %）、R_{fat} は計算対象となる個体の体脂肪率である。

$$R_{W,i,3}=\begin{cases}\left(R_{Wst,i,3}\left(1-R_{fat}\right)+R_{fat}-R_{fat,st}\right)/\left(1-R_{fat,st}\right) & \left(R_{fat,st}<R_{fat}\right)\\ R_{Wst,i,3}R_{fat}/R_{fat,st} & \left(R_{fat}\le R_{fat,st}\right)\end{cases} \tag{27.57}$$

$$R_{W,i,j(j\neq 3)}=R_{Wst,i,j}\left(1-R_{W,i,3}\right)/\left(1-R_{Wst,i,3}\right) \tag{27.58}$$

各組織の重量は各部位の重量 Wt_i に重量比 $R_{W,i,j}$ を乗じて式 27.59 で計算する。また、熱容量 $C_{i,j}$ [kJ/K]は、重量に比熱を乗じて式 27.60 で計算する。ただし、骨（j=0）と脂肪（j=3）の比熱 Cp_j はそれぞれ 2.092 kJ/(kg·K)と 2.510 kJ/(kg·K)とし、その他の組織（j=1,2,4,5,6,7,8）は 3.766 kJ/(kg·K)とする

†1　毛細血管を通らずに動脈から表在静脈へと短絡する回路を動静脈吻合（AVA: arteriovenous anastomoses）と呼ぶ

27.10)。

$$Wt_{i,j}=Wt_i R_{W,i,j} \tag{27.59}$$

$$C_{i,j}=Wt_{i,j} Cp_j \tag{27.60}$$

表 27.4　各部位における組織別の重量比 $R_{W,i,j,st}$ [%]

j	組織	頭, 首 i = 0,1	胸, 背中 i = 2,3	下腹部 i = 4	肩 i = 5, 8	腕 i = 6, 9	手 i = 7, 10	大腿 i = 11, 14	下腿 i = 12, 15	脚 i = 13, 16
0	骨	30.3	6.9	6.9	21.4	21.4	34.3	24.2	24.2	38.5
1	内蔵等	34.1	17.0	26.6	10.0	10.1	3.3	8.7	8.6	3.1
2	筋肉	7.1	40.7	40.1	45.3	45.8	7.6	45.9	45.9	3.6
3	脂肪	9.2	17.2	17.2	13.7	13.7	22.4	11.5	11.5	22.9
4	皮膚	6.7	3.3	3.3	5.7	5.7	25.3	4.8	4.4	20.8
5	動脈	2.9	0.9	1.4	0.8	0.6	1.2	1.1	1.1	2.1
6	静脈	9.7	3.3	4.5	2.0	1.7	2.8	2.8	2.9	4.8
7	表在静脈	-	-	-	1.1	1.0	3.1	1.0	1.4	4.2
8	中央血液	-	10.7	-	-	-	-	-	-	-
合計		100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

2）産熱量の計算

式 27.61 に各部位・各組織の産熱量 $M_{i,j}$ [W]を示す。筋肉層（j=2）においては基礎代謝 $M_{b,i,j}$ [W]、ふるえによる産熱 $M_{shv,i}$ [W]、外部仕事による産熱 $W_{ex,i}$ [W]の合計値であり、その他の組織においては基礎代謝 $M_{b,i,j}$ [W]である。

$$M_{i,j}=\begin{cases} M_{b,i,j} & (j\neq 2) \\ M_{b,i,j}+M_{shv,i}+W_{ex,i} & (j=2) \end{cases} \tag{27.61}$$

部位別・組織別の基礎代謝を計算するために、まず全身の基礎代謝 M_b [W]を求める。全身の基礎代謝 M_b [W]は、Ganpule らによる式 27.62 を用いて身長、体重、年齢、性別にもとづき計算する 27.20)。dm_g は性別ダミー変数であり、男性は 1、女性は 2 を設定する。式 27.62 は日本人男女 137 人のデータから回帰した結果であり、外国人の場合には式 27.63 に示す Harris の式を用いる 27.21)。

$$M_b=(0.1238+2.34\,H+0.0481\,Wt-0.0138\,Age-0.5473\,dm_g)/0.0864 \tag{27.62}$$

$$M_b=\begin{cases} 66.473+500.33\,H+13.7516\,Wt-6.755\,Age & (male) \\ 655.0955+1.8496\,H+9.5634\,Wt-4.6756\,Age & (female) \end{cases} \tag{27.63}$$

表 27.5 に各部位・各組織の基礎代謝配分比 $R_{Mb,i,j}$ [%]を示す 27.10) 27.19)。式 27.62 などを用いて計算した全身の基礎代謝 M_b に比率を乗じて部位・組織別の基礎代謝 $M_{b,i,j}$ [W]を計算する。

表 27.5　各部位・各組織の基礎代謝配分比 $R_{Mb,i,j}$ [%]

	頭	首	胸	背中	下腹部	肩	腕	手	大腿	下腿	脚	合計
核	19.580	0.318	25.023	22.090	9.509	0.214	0.111	0.053	0.405	0.120	0.144	78.614
筋肉	0.252	0.004	2.997	2.997	4.804	0.500	0.260	0.026	0.973	0.260	0.041	15.174
脂肪	0.127	0.002	0.671	0.592	0.950	0.721	0.037	0.027	0.178	0.041	0.066	4.482
皮膚	0.123	0.033	0.211	0.187	0.300	0.059	0.031	0.059	0.144	0.027	0.118	1.730
合計	20.082	0.357	28.902	25.866	15.563	1.494	0.439	0.165	1.700	0.448	0.369	100.00

※肩, 腕, 手, 大腿, 下腿, 脚は 1 本あたりの値

外部仕事による産熱 $W_{ex,i}$ [W]は式 27.64 で計算する。ただし、$R_{met,i}$ は外部仕事およびふるえによる熱産生 $M_{shv,i}$ [W]の配分比であり、式 27.65 で示すように各部位の筋肉の重量比とする†1)。

$$W_{ex,i}=Max(0,\,58.15\,WK\cdot A_{du}-M_b)\,R_{met,i} \tag{27.64}$$

†1　原理的には運動の種別によって異なる値を設定すべきである。Stolvijk モデルおよび JOS モデルでは自転車エルゴメータによる実験値を採用しているため、やや脚部の配分が大きい。

$$R_{met,i} = Wt_{i,ms} / \sum_{i=0}^{16} Wt_{i,ms} \tag{27.65}$$

ふるえによる熱産生（後述）の配分比も式 27.65 の $R_{met,i}$ を準用する。

3）各部位の熱収支

図 27.4 に各部位における熱移動を示す。前の部位から流れてきた血液の一部は、核・筋肉・脂肪・皮膚を通過して静脈へ移動する。その他は次の部位の動脈へ流れる。特に末端部（手または脚）においては、一部の血液が毛細血管を通ること無く直接に動脈から表在静脈へ流れる（AVA 血流）。このため、四肢部位においては表在静脈が存在する。

動脈と静脈は核に存在するため、核との間で熱伝導が生じる。一方、表在静脈は皮膚層に存在するため、皮膚との間で熱伝導が生じる。この他、互いに隣接する動脈と静脈、核・筋肉・脂肪・皮膚との間でも熱交換が行われる。

各部位ともに皮膚層において外部環境との間で放射・対流・蒸発による熱交換が行われる。また、特に胸部位においては呼吸が行われるため、核と外部環境との間で直接に熱交換が行われる。

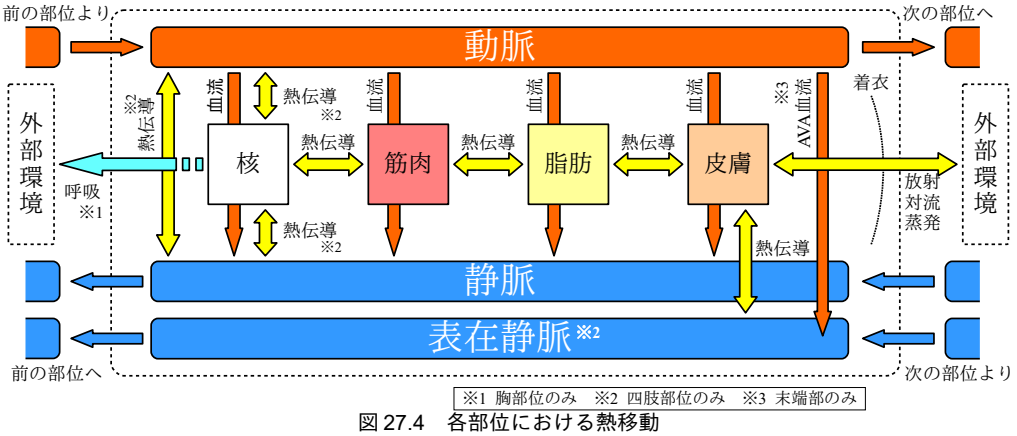


図 27.4　各部位における熱移動

5）血流の計算

全身の基礎血流 BF_b [(mL/s)]は式 27.66 に示す通り、体表面積 A_{du} [m^2]に心係数 CI（Cardiac Index）[(mL/s)/m^2]を乗じることで計算できる。心係数 CI は年齢とともに低下することが知られており、式 27.67 で計算する†1)。表 27.6 に基礎血流の配分比 R_{BF} [%]を示す†2)。式 27.66 によって計算した全身の基礎血流 BF_b に配分比 R_{BF} を乗じることで部位別組織別の基礎血流を計算する。

$$BF_b = CI \cdot A_{du} \tag{27.66}$$

$$CI = 115 - 2.4\,Age + 0.0167\,Age^2 + 3.56\,Age^3 \times 10^{-4} - 4.29\,Age^4 \times 10^{-6} \tag{27.67}$$

表 27.6　基礎血流の配分比 R_{BF} [%]

組織	頭 $i=0$	首 $i=1$	胸 $i=2$	背中 $i=3$	下腹部 $i=4$	肩 $i=5,8$	腕 $i=6,9$	手 $i=7,10$	大腿 $i=11,14$	下腿 $i=12,15$	脚 $i=13,16$
核	10.822	5.118	27.575	27.040	6.443	0.113	0.056	0.032	0.129	0.026	0.018
筋肉	0.210	0.099	2.714	2.713	4.349	0.454	0.237	0.028	0.303	0.024	0.004
脂肪	0.081	0.038	0.474	0.474	0.765	0.056	0.031	0.015	0.053	0.006	0.006
皮膚	1.974	0.112	0.678	0.509	0.783	0.314	0.175	0.384	0.502	0.224	0.322

血流は人間が活動して代謝量が増えるにつれて増加する。先に述べたとおり、本モデルでは外部仕事およびふるえによる熱産生は筋肉層においてのみ生じるとするため、代謝量の増加に伴う血流の増

†1　文献 27.24 に記載の数値から作成した回帰式であり、適用範囲は 15 < Age < 75 である。本モデルでは男女ともに同一の回帰式としたが、文献 27.23 によれば中高年齢層でやや女性が高い傾向を示すようである。

†2　各部位と皮膚層への配分比は JOS[27.8]、その他の組織への配分比は 65MN[27.19]による。

加も筋肉層においてのみ生じる。その増加量は単位仕事に対して必要な酸素量の物理的関係にもとづき、0.239 (mL/s)/W とする。即ち、筋肉層における血流は外部仕事およびふるえによる熱産生の関数として式 27.68 で計算できる。

$$BF_{i,ms} = BF_{b,i,ms} + 0.239\left(M_{shv,i} + W_{ex,i}\right) \tag{27.68}$$

皮膚層においては外部温熱環境に応じて体温調節を目的として血管収縮あるいは血管拡張が生じ、これによって血流が変化する（後述）。核および脂肪の血流は基礎血流で一定とする。

ある部位の動脈および静脈に流れる血液（$BF_{i,ar}$ と $BF_{i,ve}$）は、その部位以降の全ての部位の核・筋肉・脂肪・皮膚を通る血液の合算値である。ただし、腰に関しては脚の表在静脈の血液を静脈で受けるため、表在静脈分を加算する必要がある。

5）組織間の熱コンダクタンス

組織間の熱伝導を計算するために、各部位を球（頭）および円管（その他の部位）として取り扱う。図 27.5 に円管による部位のモデル化を示す。中心から順に核層、筋肉層、脂肪層、皮膚層で構成される多層円管として捉える。

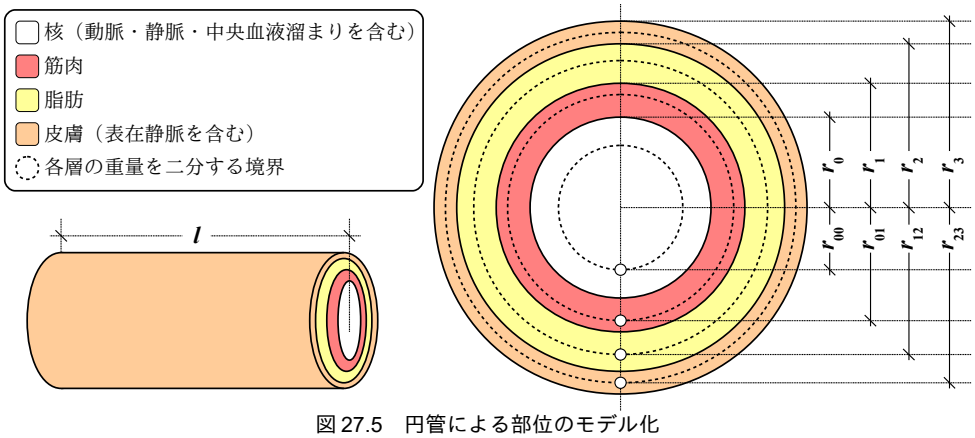


図 27.5　円管による部位のモデル化

核層は、骨・内臓等・動脈・静脈・中央血液を含み、皮膚は表在静脈を含むものとし、各層の重量は式 27.69 で計算する。

$$\begin{aligned} Wt_{i,cr} &= Wt_{i,0} + Wt_{i,1} + Wt_{i,5} + Wt_{i,6} + Wt_{i,8} \\ Wt_{i,ms} &= Wt_{i,2} \\ Wt_{i,ft} &= Wt_{i,3} \\ Wt_{i,sk} &= Wt_{i,4} + Wt_{i,7} \end{aligned} \tag{27.69}$$

円管の長さ l [m]が与えられれば、各層の重量が既知であるため、式 27.70 により層の外径 r_o [m]を計算することができる。$Wt_{i,sm}$ [kg]は計算対象の層に至るまでの全層の重量合算値である。ρ は人体の比重であり 1,000 kg/m^3 とする。

$$r_o = \sqrt{\frac{Wt_{i,sm}}{\pi \rho l}} \tag{27.70}$$

質点は図 27.5 に示したように各層の重量を二等分する仮想的な境界に設ける。質点間の熱コンダクタンス K [W/K]を計算するためには、式 27.71 に示す多層円管の熱コンダクタンスの公式を用いる。ただし、λ は熱伝導率 [W/(m·K)]、r_0 [m]は第 1 層の内径、r_1 [m]は第 1 層の外径（=第 2 層の内径）、r_2 [m]は第 2 層の外径である。核層と筋肉層の熱伝導率は 0.4184 W/(m·K)、脂肪層と皮膚層の

熱伝導率は 0.3347 W/(m·K)とする。

$$K=\frac{2\pi l}{\ln(r_1/r_0)/\lambda_0+\ln(r_2/r_1)/\lambda_1} \tag{27.71}$$

標準体躯における円管長さを表 27.7 に示す[†1]。身長 H [m]が標準体躯の身長である H_{st} = 1.72 m と異なる場合には $R_H=(H/H_{st})^{0.725}$ を乗じて補正する。

表 27.7　標準体躯における円管長さ [m]

首 i = 1	胸 i = 2	背中 i = 3	下腹部 i = 4	肩 i = 5, 8	腕 i = 6, 9	手 i = 7, 10	大腿 i = 11, 14	下腿 i = 12, 15	脚 i = 13, 16
0.075	0.182	0.170	0.257	0.343	0.217	0.480	0.542	0.267	0.625

頭は球としてモデル化する。この場合の各層の外径および熱コンダクタンスはそれぞれ式 27.72~27.73で計算する。

$$r_o=\left(\frac{3Wt_{i,sm}}{4\pi\rho}\right)^{1/3} \tag{27.72}$$

$$K=\frac{4\pi}{(1/r_0+1/r_1)/\lambda_0+(1/r_1+1/r_2)/\lambda_1} \tag{27.73}$$

表 27.8 に各部位の血管の熱コンダクタンスを示す[†2]。i=0~4 の部位に関しては 0 である。

表 27.8　各部位の血管の熱コンダクタンス

層	肩 i = 5, 8	腕 i = 6, 9	手 i = 7, 10	大腿 i = 11, 14	下腿 i = 12, 15	脚 i = 13, 16
動脈/静脈 × 核層	0.586	0.383	1.534	0.81	0.435	1.816
表在静脈 × 皮膚層	57.735	37.768	16.634	102.012	54.784	24.277
動脈 × 静脈	0.537	0.351	0.762	0.826	0.444	0.992

6）外部環境との熱交換

外部環境との熱交換に関しては、1）呼吸による熱損失、2）皮膚表面における顕熱交換および蒸発熱損失、3）皮膚と物体との接触による熱交換、をモデル化する。

呼吸による熱損失は式 27.31 と 27.32 で記した。ただし、乾球温度 t_a [°C]および水蒸気分圧 P_a [kPa]は頭部周囲の空気の値を用いることに注意する。

皮膚表面における顕熱・潜熱の交換は式 27.2 と式 27.3 で計算する。ただし放射伝達率 h_r [W/(m²·K)]、対流熱伝達率 h_c [W/(m²·K)]、相対気流速度 v [m/s]、着衣量 I_{clo} [clo]などは部位別の値を設定して計算する必要がある。放射熱伝達率は各部位の平均放射温度 T_{mr} [K]を用いて式 27.8 で計算できる。有効放射定数 εf_{eff}は立位で 0.73、座位で 0.70 である[27.5)]。標準条件（代謝量 1.0 met、相対気流速度 0.15 m/s）での対流熱伝達率はサーマルマネキンを用いた実測結果（表 27.9）を参考とする[27.26)][†3]。標準条件以外の条件下においては式 27.46 を応用して式 27.74 で補正する。後述するが、標準条件は乾球温度 28.8 °C、着衣量 0 clo であるため、$t_{cl,s}$は皮膚温度 $t_{sk,s}$と一致し、$t_{a,s}$は 28.8 °C である。また SET*の計算と同様に v は 0.15 m/s 以上の値とする。

$$h_c=Max\left(2.58\sqrt{v},\left(\frac{t_{cl}-t_a}{t_{cl,s}-t_{a,s}}\right)\right)\times h_{c,s} \tag{27.74}$$

†1　手や脚がやけに長細く、当初はこれが西欧人の標準体躯なのかと不思議に思ったが、どうやら体積と表面積の関係を抽象化して捉えた結果のようだ。とすれば、わざわざ頭だけ球にする必要も無いように思うのだが。

†2　厳密には体躯や血流量によって変化するが、本モデルでは一定値とした。詳細なモデル化が必要な場合には参考文献 27.25 を参照のこと。

†3　田辺らのモデルでは、対流熱伝達率の他、放射熱伝達率も部位別の値を設定する仕様となっている。放射環境の不均一性をモデルで表現する方法としては、放射熱伝達率を部位別に設定した後、一様の放射温度を与える方法と、放射温度自体を部位別に与える方法が考えられるが、本モデルでは後者の方法を取る。

表 27.9　部位別の対流熱伝達率 $h_{c,s}$

	頭 $i=0$	首 $i=1$	胸 $i=2$	背中 $i=3$	下腹部 $i=4$	肩 $i=5,8$	腕 $i=6,9$	手 $i=7,10$	大腿 $i=11,14$	下腿 $i=12,15$	脚 $i=13,16$
立位	4.48	4.48	2.97	2.91	2.85	3.61	3.55	3.67	2.80	2.04	2.04
座位	4.75	4.75	3.12	2.48	1.84	3.76	3.62	2.06	2.98	2.98	2.62

皮膚が直接に物体と接触している場合には、式 27.75 で皮膚と外界との間の熱流を計算する。$K_{mt,i}$ は接触物との間の熱コンダクタンス[W/(m²·K)]、cf は接触している皮膚の割合である。接触部に関しては対流・放射・蒸発による熱移動は無いとみなすため、第 2 項で(1-cf)を乗じる。飽和水蒸気圧 $P_{Ws,sk,i}$ [kPa]は皮膚温度の関数であるが、1 タイムステップ前の皮膚温度を用いて式 22.38 で計算し固定値とする。

$$\frac{Q_{sk,i}}{A_{du,i}} = cf_i K_{mt,i}(T_{sk,i} - T_{mt,i}) + (1-cf_i)\left\{h_i'(T_{sk,i} - T_{o,i}) + w h_e'(P_{Ws,sk,i} - P_{a,i})\right\} \tag{27.75}$$

7）熱平衡式

各部位において核、筋肉、脂肪、皮膚、動脈、静脈、表在静脈（四肢部位のみ）の 6 つまたは 7 つの質点を設け、熱平衡式をつくると式 27.76~27.81となる。ただし部位を表す添字 i は省略した。ρCp_b は血液の体積比熱であり 3.842 J/(mL K)とする。上付の*は血液の上流の部位（例えば腕の ar*は肩の動脈、腕の sv*は手の表在静脈）を表す。

・核（<·>は胸部位のみ）

$$C_{cr}\frac{dT_{cr}}{dt} = M_{cr} + \rho Cp_b BF_{cr}(T_{ar} - T_{cr}) + K_{cr-ve}(T_{ar} + T_{ve} - 2T_{cr}) + K_{cr-ms}(T_{ms} - T_{cr}) + \langle C_{res} + E_{res}\rangle \tag{27.76}$$

・筋肉

$$C_{ms}\frac{dT_{ms}}{dt} = M_{ms} + \rho Cp_b BF_{ms}(T_{ar} - T_{ms}) + K_{cr-ms}(T_{cr} - T_{ms}) + K_{ms-fat}(T_{fat} - T_{ms}) \tag{27.77}$$

・脂肪

$$C_{fat}\frac{dT_{fat}}{dt} = M_{fat} + \rho Cp_b BF_{fat}(T_{ar} - T_{fat}) + K_{ms-fat}(T_{ms} - T_{fat}) + K_{fat-sk}(T_{sk} - T_{fat}) \tag{27.78}$$

・皮膚

$$C_{sk}\frac{dT_{sk}}{dt} = M_{sk} + \rho Cp_b BF_{sk}(T_{ar} - T_{sk}) + K_{fat-sk}(T_{fat} - T_{sk}) + K_{sv-sk}(T_{sv} - T_{sk}) + cf\,K_{mt}(T_{mt} - T_{sk}) + (1-cf)\left\{h_i'(T_o - T_{sk}) + w h_e'(P_{Ws,sk,i} - P_{a,i})\right\} \tag{27.79}$$

・動脈

$$C_{ar}\frac{dT_{ar}}{dt} = \rho Cp_b BF_{ar}(T_{ar}^* - T_{ar}) + K_{cr-ve}(T_{cr} - T_{ar}) + K_{ve}(T_{ve} - T_{ar}) \tag{27.80}$$

・静脈（<<·>>は腰のみ）

$$C_{ve}\frac{dT_{ve}}{dt} = \rho_{ve} Cp_b\left\{BF_{cr}(T_{cr} - T_{ve}) + BF_{ms}(T_{ms} - T_{ve}) + BF_{fat}(T_{fat} - T_{ve}) + BF_{sk}(T_{sk} - T_{ve}) + \sum_{m=0} BF_{ve^*,m}(T_{ve^*,m} - T_{ve})\right\} + K_{cr-ve}(T_{cr} - T_{ve}) + K_{ve}(T_{ar} - T_{ve}) + \langle\langle \sum_{m=0} \rho_{ve} Cp_b BF_{sv,m}(T_{sv^*,m} - T_{ve})\rangle\rangle \tag{27.81}$$

・表在静脈（<<<<·>>>>は肩・腕・大腿・下腿、<<<<<·>>>>>は手・脚）

$$C_{sv}\frac{dT_{sv}}{dt} = K_{sv-sk}(T_{sk} - T_{sv}) + \rho Cp_b BF_{sv}(\langle\langle\langle T_{sv^*}\rangle\rangle\rangle + \langle\langle\langle\langle T_{ar}\rangle\rangle\rangle\rangle - T_{sv}) \tag{27.82}$$

中央血液溜まりの熱平衡は式 27.83 で表される。中央血液溜まりに接続する各部位の血流による熱

移動に関しての熱平衡式である。

$$C_{cb}\frac{dT_{cb}}{dt}=\rho_{ve}Cp_b\sum_{n=0}\left\{BF_{ve,n}\left(T_{ve,n}-T_{cb}\right)+BF_{sv,n}\left(T_{sv,n}-T_{cb}\right)\right\} \tag{27.83}$$

8）体温制御

体温制御は 2-Node モデルと同様に血管運動、発汗、ふるえを考慮する他、手および脚部位の AVA 血流をモデル化する。

体温制御の計算にあたっては、まず、制御の目標体温（セットポイント）を定める必要がある。このために、作用温度 28.8 °C、相対湿度 50 %、気流速度 0.1 m/s、着衣量 0 clo、代謝量 1.0 met の条件（PMV=0.0 となる）のもと、体温制御無しで計算を行い、平衡状態となる体温をセットポイントとする。以下、セットポイント温度を添字の *sp*（Set Point）で表現する。

・血管運動

皮膚の温度受容器からの統合信号を式 27.84 および式 27.85 で定義する。$R_{sk,SIG}$ [%]は信号を統合する際の全身の皮膚の重み付け係数であり、表 27.10 の値をとる。

$$Wrms=\sum_{i=0}^{16}\left(R_{skSIG,i}Max\left(T_{sk,i}-T_{sk,sp,i},0\right)\right) \tag{27.84}$$

$$Clds=\sum_{i=0}^{16}\left(R_{skSIG,i}Max\left(T_{sk,sp,i}-T_{sk,i},0\right)\right) \tag{27.85}$$

血管拡張に関する信号を式 27.86、血管収縮に関する信号を式 27.87 で定義し、これらを用いて血管運動を反映した各部位の血流を式 27.88 で計算する。ただし SIG_{head}は頭部の核における体温とセットポイントの差であり、式 27.89 で計算する。$R_{dil,i}$ [%]と $R_{str,i}$ [%]は血管運動の全身への配分比であり、表 27.10 の値をとる。R_{BF} [%]は標準体躯の基礎血流量（21.13 mL/s）に対する、計算対象としている人体の基礎血流の比率である。なお、血管運動は皮膚層においてのみ発生する。

$$SIG_{dil}=\left\{32.5SIG_{head}+2.08\left(Wrms-Clds\right)\right\}R_{BF} \tag{27.86}$$

$$SIG_{str}=-10.8SIG_{head}-10.8\left(Wrms-Clds\right) \tag{27.87}$$

$$BF_{sk,i}=\frac{BF_{sk,b,i}+R_{dil,i}SIG_{dil}}{1+R_{str,i}SIG_{str}}\times 2^{\left(T_{sk,i}-T_{sk,sp,i}\right)/6} \tag{27.88}$$

$$SIG_{head}=\left(T_{Head,cr}-T_{Head,cr,sp}\right) \tag{27.89}$$

・発汗

発汗に関する信号を式 27.90 で定義し、これを用いて各部位の発汗による熱損失を式 27.91 で計算する。R_{Adu} [%]は標準体躯の体表面積（1.87 m^2）に対する、計算対象としている人体体表面積の比率である。R_{sw} [%]は発汗の全身への配分比であり、表 27.10 の値をとる。式 27.33~27.36を用いて、各部位で最低でも 6 %の不感蒸泄が発生することを前提に蒸発熱損失を計算する。

$$SIG_{sw}=\left\{371.2SIG_{head}+33.64\left(Wrms-Clds\right)\right\}R_{Adu} \tag{27.90}$$

$$E_{sw,i}=R_{sw}SIG_{sw}\times 2^{\left(T_{sk,i}-T_{sk,sp,i}\right)/10} \tag{27.91}$$

・ふるえ

ふるえに関する信号を式 27.92 で定義し、これを用いて各部位のふるえによる熱産生を式 27.93 で計算する。R_{shv} [%]は熱産生の全身への配分比であり、表 27.10 の値をとる。既に式 27.61 で示したとおり、ふるえによる熱産生は各部位の筋肉層のみに配分される。

表 27.10　温度受容器の重み付け係数 $R_{sk,SIG}$ および体温調整反応量の配分比[%]

	頭 $i=0$	首 $i=1$	胸 $i=2$	背中 $i=3$	下腹部 $i=4$	肩 $i=5,8$	腕 $i=6,9$	手 $i=7,10$	大腿 $i=11,14$	下腿 $i=12,15$	脚 $i=13,16$
$R_{sk,SIG,i}$	5.47	1.46	14.92	13.21	21.22	2.27	1.17	9.23	5.01	2.51	1.67
$R_{dil,i}$	10.42	2.77	9.80	8.60	13.79	3.13	1.63	6.05	9.20	2.30	5.00
$R_{str,i}$	2.13	2.13	6.38	6.38	6.38	2.13	2.13	14.89	2.13	2.13	14.89
$R_{sw,i}$	6.40	1.70	14.60	12.90	20.60	5.10	2.60	1.55	7.30	3.60	1.75
$R_{shv,i}$	3.39	4.36	27.39	24.10	38.74	0.24	0.14	0.02	0.39	0.18	0.04

$$SIG_{shv}=-24.36\,Min\left(SIG_{head},0\right)\cdot Clds\cdot R_{Adu} \tag{27.92}$$

$$M_{shv,i}=SIG_{shv}R_{shv,i} \tag{27.93}$$

・AVA 血流

脚および手の AVA 血流の開度 O_{AVA} [-]は、全身の平均皮膚温度 t_{sk} [ºC]、全身の平均皮膚セットポイント $t_{sk,sp}$ [ºC]、体中心（胸、背中、下腹部）の平均核温度 $t_{cr,234}$ [ºC]、体中心の平均核セットポイント $t_{cr,234,sp}$ [ºC]を用いて式 27.94 で計算する[27.11]。ただし、α_{AVA} と β_{AVA} は係数であり、脚と手でそれぞれ表 27.11 の値をとる。式 27.95 に AVA 血流の計算式を示す。$BF_{AVA,max}$ は皮膚の最大血流量であり、皮膚の体積に 5,000 (mL/s)/m^3 を乗じた値とする[27.27] [27.28]。

$$O_{AVA}=0.265\left(t_{sk}-\left(t_{sk,sp}+\alpha_{AVA1}\right)\right)+0.953\left(t_{cr,123}-\left(t_{cr,123,sp}+\alpha_{AVA2}\right)\right)+0.9126 \tag{27.94}$$

$$BF_{AVA}=O_{AVA}BF_{AVA,max} \tag{27.95}$$

表 27.11　AVA 血流開度の計算係数

	α_{AVA1}	α_{AVA2}
手	-0.43	-0.1905
脚	-0.97	0.0095

27.3　計算法

27.3.1　2-Node モデルの計算

直接的に式 27.1 を解き、定常状態を求めることは困難であるため、まず、式 27.4 および式 27.5 を解くことで微小時間における核と皮膚の温度変化を求める。式 27.13~27.23で示した体温制御機構を働かせながら、60 min の計算を行うことで定常状態に到達させる。核と皮膚の温度変化は人体の比熱容量 Cp_h [J/(kg·K)]を用いて式 27.96 と 27.97 で計算する。ここで A_{du} は人体の表面積であり SET*計算のための標準体表面積としては 1.8 m^2 を採用する。

$$\frac{dT_{sk}}{dt}=\frac{Q_{sk}A_{du}}{Cp_hWt\,\alpha} \tag{27.96}$$

$$\frac{dT_{cr}}{dt}=\frac{Q_{cr}A_{du}}{Cp_hWt\left(1-\alpha\right)} \tag{27.97}$$

2-Node モデルの計算プログラムを 27.1 に示す。1~11 行および 39~48 行は定数宣言である。50~66 行で初期値を設定した後、73~177 行で 1 分ずつ体温を更新していくことで定常状態に到達させる。衣服の対流熱伝達率は不変のため、68~71 行でループの外で計算する。78~96 行は衣服の温度の計算処理である。式 27.8 で示したとおり、衣服の温度を求めるための放射熱伝達率自体が衣服の温度の関数となっているため、収束計算が必要となる。

108, 109 行は式 27.4 と式 27.5 で示される核と皮膚の熱収支であり定常状態では 0 となる。計算の初期では熱収支が合わないため、110~118 行で示すように式 27.96 と 27.97 を用いて温度変化を計算す

る。120 行以下では、得られた体温を前提に血管運動・発汗・ふるえなどを計算して体温制御を行う。以上の計算を繰り返し、60 分後の定常状態での体温等を出力する。

プログラム 27.1　2-Node モデル計算処理

Popolo.HumanBody.TwoNodeModel class

```
/// <summary>衣服の面積率[-]</summary>
private const double K_CLO = 0.25;

/// <summary>衣服の透湿係数[K/kPa]</summary>
private const double I_CLS = 0.45;

/// <summary>絶対温度と摂氏との変換定数</summary>
private const double CONVERT_C_TO_K = 273.15;

/// <summary>代謝量換算係数[(W/m2)/met]</summary>
private const double CONVERT_MET_TO_W = 58.2;

/// <summary>定常状態を計算する</summary>
/// <param name="drybulbTemperature">乾球温度[CDB]</param>
/// <param name="meanRadiantTemperature">平均放射温度[C]</param>
/// <param name="relativeHumidity">相対湿度[%]</param>
/// <param name="velocity">相対気流速度[m/s]</param>
/// <param name="clothing">着衣量[clo]</param>
/// <param name="basalMetabolism">基礎代謝量[W/m2]</param>
/// <param name="externalWork">外部仕事量[W/m2]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <param name="skinTemperature">出力:皮膚温度[C]</param>
/// <param name="coreTemperature">出力:コア温度[C]</param>
/// <param name="bodyTemperature">出力:平均体温[C]</param>
/// <param name="clothTemperature">出力:衣服温度[C]</param>
/// <param name="sensibleHFSkin">出力:皮膚からの顕熱損失[W/m2]</param>
/// <param name="latentHFSkin">出力:皮膚からの潜熱損失[W/m2]</param>
/// <param name="sensibleRespiration">出力:呼吸による顕熱損失[W/m2]</param>
/// <param name="latentRespiration">出力:呼吸による潜熱損失[W/m2]</param>
/// <param name="wettedness">出力:皮膚の平均濡れ率[-]</param>
public static void GetSteadyState
  (double drybulbTemperature, double meanRadiantTemperature, double relativeHumidity, double velocity,
  double clothing, double basalMetabolism, double externalWork, double atmosphericPressure,
  out double skinTemperature, out double coreTemperature, out double bodyTemperature,
  out double clothTemperature, out double sensibleHFSkin, out double latentHFSkin,
  out double sensibleRespiration, out double latentRespiration, out double wettedness)
{
  //定数宣言
  const double WEIGHT = 70d;  //標準体重[kg]
  const double BODY_SURFACE = 1.8;//標準体表面積[m2]
  const double C_SWEATING = 47.2; //発汗の係数[mg/(m2 s K)]
  const double C_VASODILATATION = 55.6;   //血管拡張の係数[L/(m2 s K)]
  const double C_VASOCONSTRICTION = 0.1;  //血管収縮の係数[1/K]
  const double SETPOINT_SKIN = 33.7;  //皮膚セットポイント温度[C]
  const double SETPOINT_CORE = 36.8;  //コアセットポイント温度[C]
  const double SETPOINT_BODY = 36.49; //体温セットポイント温度[C]
  const double NORMAL_BLOOD_FLOW = 1.75;  //標準状態の皮膚血流量[mL/(m2 s)]
  const double CRITICAL_WETTEDNESS = 0.85;//最大濡れ率[-]

  //水蒸気分圧[kPa]の計算
  double pa = relativeHumidity / 100 * Water.GetSaturationPressure(drybulbTemperature);
  //着衣抵抗[m2K/W]の計算
  double rcl = 0.155 * clothing;
  //着衣面積率[-]の計算
  double clothRate = 1d + K_CLO * clothing;

  //初期値を設定する
  sensibleHFSkin = sensibleRespiration = latentRespiration = wettedness = 0;
  skinTemperature = SETPOINT_SKIN;
  coreTemperature = SETPOINT_CORE;
  bodyTemperature = SETPOINT_BODY;
  double skinBloodFlow = NORMAL_BLOOD_FLOW;
  double mshiv = 0;
  double alpha = 0.1;
  latentHFSkin = 0.1 * basalMetabolism;
  double metabolism = basalMetabolism;   //ふるえ産熱は 0 とする

  //対流熱伝達率の計算 (代謝量か気流で決定)
  double chcv = 8.6 * Math.Pow(Math.Max(0.15, velocity), 0.53);
  double chcm = 5.66 * Math.Pow(Math.Max(0, basalMetabolism / CONVERT_MET_TO_W - 0.85), 0.39);
  double convectiveHTransCoef = Math.Max(chcv, chcm);

  //Δt=1min として 60min の繰り返し計算を行う
```

```
 74  const int DELTA_T = 1;
 75  clothTemperature = (skinTemperature + drybulbTemperature) / 2d;
 76  for (int tim = 0; tim < 60; tim += DELTA_T)
 77  {
 78    //衣服の表面温度を収束計算
 79    double operatingTemp, ra;
 80    while (true)
 81    {
 82      double ctOld = clothTemperature;
 83      //放射熱伝達率[W/(m2K)]の計算
 84      double hr = 4d * BLACK_CONSTANT * 0.72
 85        * Math.Pow((clothTemperature + meanRadiantTemperature) / 2d + CONVERT_C_TO_K, 3);
 86      //総合熱伝達率[W/(m2K)]の計算
 87      double hcr = hr + convectiveHTransCoef;
 88      //空気層顕熱抵抗[(m2K)/W]の計算
 89      ra = 1 / (clothRate * hcr);
 90      //作用温度[C]の計算
 91      operatingTemp = (hr * meanRadiantTemperature + convectiveHTransCoef * drybulbTemperature) / hcr;
 92      //衣服温度[C]の計算
 93      clothTemperature = (ra * skinTemperature + rcl * operatingTemp) / (ra + rcl);
 94      //衣服温度の更新量が0.01C以下で収束と判定
 95      if (Math.Abs(ctOld - clothTemperature) < 0.01) break;
 96    }
 97
 98    //皮膚からの顕熱損失量[W/(m2K)]の計算
 99    sensibleHFSkin = (skinTemperature - operatingTemp) / (ra + rcl);
100    //コアから皮膚への熱流[W/m2]の計算
101    double hfcs = (coreTemperature - skinTemperature) * (5.28 + 3.842 * skinBloodFlow);
102
103    //呼吸による熱損失量[W/m2]の計算
104    latentRespiration = metabolism * 0.017251 * (5.8662 - pa);
105    sensibleRespiration = metabolism * 0.0014 * (34 - drybulbTemperature);
106
107    //コアと皮膚への熱流[W/m2]を計算
108    double scr = metabolism - hfcs - latentRespiration - sensibleRespiration - externalWork;
109    double ssk = hfcs - sensibleHFSkin - latentHFSkin;
110    //コアと皮膚の熱容量[J/K]を計算
111    double tccr = 3492 * (1 - alpha) * WEIGHT;
112    double tcsk = 3492 * alpha * WEIGHT;
113    //コアと皮膚の温度変化量[K/s]を計算
114    double dtcr = (scr * BODY_SURFACE) / tccr;
115    double dtsk = (ssk * BODY_SURFACE) / tcsk;
116    //コアと皮膚の温度を更新
117    vcoreTemperature = coreTemperature + dtcr * (DELTA_T * 60);
118    vskinTemperature = skinTemperature + dtsk * (DELTA_T * 60);
119
120    //重み付けして体温を計算
121    bodyTemperature = alpha * skinTemperature + (1 - alpha) * coreTemperature;
122
123    //制御量の計算
124    double sDil = Math.Max(0, coreTemperature - SETPOINT_CORE);
125    double sStr = Math.Max(0, SETPOINT_SKIN - skinTemperature);
126    double sSw1 = Math.Max(0, bodyTemperature - SETPOINT_BODY);
127    double sSw2 = Math.Max(0, skinTemperature - SETPOINT_SKIN);
128    double sShv1 = Math.Max(0, SETPOINT_SKIN - skinTemperature);
129    double sShv2 = Math.Max(0, SETPOINT_CORE - coreTemperature);
130
131    //皮膚血流量[L/(m2 s)]の計算
132    skinBloodFlow = (NORMAL_BLOOD_FLOW + C_VASODILATATION * sDil) / (1d + C_VASOCONSTRICTION * sStr);
133    skinBloodFlow = Math.Max(Math.Min(skinBloodFlow, 25), 0.139);
134    //皮膚・コアの重量比を更新
135    alpha = 0.0417737 + 0.2069953 / (skinBloodFlow + 0.1626158);
136
137    //調節発汗による蒸発熱損失[W/m2]を計算
138    double msw = C_SWEATING * sSw1 * Math.Exp(sSw2 / 10.7);
139    double esw = 2.501 * msw;
140
141    //不感蒸泄による蒸発熱損失[W/m2]を計算
142    double lewis = 0.0555 * (skinTemperature + CONVERT_C_TO_K);
143    double latentHTransCoef = 1 / (rcl / (I_CLS * lewis) + 1d / (clothRate * convectiveHTransCoef * lewis));
144    double emax = latentHTransCoef * (Water.GetSaturationPressure(skinTemperature) - pa);
145    double wSw = esw / emax;
146    double eb = 0.06 * (emax - esw);
147    double wB = eb / emax;
148    wettedness = wSw + wB;
149    latentHFSkin = esw + eb;
150
151    //ぬれ率上限を上回る場合には蒸発熱損失等を修正
152    if (CRITICAL_WETTEDNESS < wettedness)
153    {
```

```
154       wettedness = CRITICAL_WETTEDNESS;
155       wSw = (CRITICAL_WETTEDNESS - 0.06) / 0.94;
156       esw = wSw * emax;
157       eb = 0.06 * (1 - wSw) * emax;
158       latentHFSkin = esw + eb;
159     }
160     //皮膚表面が露点以下の場合
161     if (emax < 0)
162     {
163       eb = 0;
164       esw = 0;
165       wB = CRITICAL_WETTEDNESS;
166       wSw = CRITICAL_WETTEDNESS;
167       latentHFSkin = emax;
168     }
169
170     //ふるえ熱産生量[W/m2]を計算
171     mshiv = 19.4 * sShv1 * sShv2;
172     //基礎代謝とふるえ熱産生を加算した値を代謝量とする
173     metabolism = basalMetabolism + mshiv;
174
175     //衣服の温度を更新
176     clothTemperature = (ra * skinTemperature + rcl * operatingTemp) / (ra + rcl);
177   }
178 }
```

SET*の計算プログラムを 27.2 に示す。2-Node モデルの計算結果である「衣服温度」「皮膚温度」「皮膚からの顕熱および潜熱損失」「皮膚の平均濡れ率」を入力として与える必要がある。23 行で標準着衣量を計算し、30~36 行で標準状態における顕熱伝達率 h_s'と潜熱伝達率 h_{es}'を計算する。38~46 行ではニュートン・ラフソン法を用いて式 27.40 を解いている。

プログラム 27.2　SET*の計算処理

Popolo.HumanBody.TwoNodeModel class

```
  1 /// <summary>SET*[C]を計算する</summary>
  2 /// <param name="meanRadiantTemperature">平均放射温度[C]</param>
  3 /// <param name="basalMetabolism">基礎代謝量[W/m2]</param>
  4 /// <param name="externalWork">外部仕事量[W/m2]</param>
  5 /// <param name="clothTemperature">衣服温度[C]</param>
  6 /// <param name="skinTemperature">皮膚温度[C]</param>
  7 /// <param name="sensibleHFSkin">皮膚からの顕熱損失[W/m2]</param>
  8 /// <param name="latentHFSkin">皮膚からの潜熱損失[W/m2]</param>
  9 /// <param name="wettedness">皮膚の平均濡れ率[-]</param>
 10 /// <returns>SET*[C]</returns>
 11 public static double GetSETStar
 12   (double meanRadiantTemperature, double basalMetabolism, double externalWork, double clothTemperature,
 13   double skinTemperature, double sensibleHFSkin, double latentHFSkin, double wettedness)
 14 {
 15   //放射熱伝達率[W/(m2K)]の計算
 16   double radiativeHTransCoef = 4d * BLACK_CONSTANT * 0.72
 17     * Math.Pow((clothTemperature + meanRadiantTemperature) / 2d + CONVERT_C_TO_K, 3);
 18   //対流熱伝達率[W/(m2 K)]の計算
 19   double convectiveHTransCoef = 5.66 * Math.Pow(Math.Max(0, basalMetabolism / CONVERT_MET_TO_W - 0.85), 0.39);
 20   convectiveHTransCoef = Math.Max(convectiveHTransCoef, 3d);
 21
 22   //標準着衣量[clo]の計算
 23   double sClothing = 1.3264 / ((basalMetabolism - externalWork) / CONVERT_MET_TO_W + 0.7383) - 0.0953;
 24
 25   //着衣抵抗[m2K/W]の計算
 26   double rcl = 0.155 * sClothing;
 27   //着衣面積率[-]の計算
 28   double clothRate = 1d + K_CLO * sClothing;
 29
 30   //潜熱伝達率[W/(m2 K)]の計算
 31   double lewis = 0.0555 * (skinTemperature + CONVERT_C_TO_K);
 32   double latentHTransCoef = 1 / (rcl / (I_CLS * lewis) + 1d / (clothRate * convectiveHTransCoef * lewis));
 33
 34   //顕熱伝達率[W/(m2K)]の計算
 35   double hcr = radiativeHTransCoef + convectiveHTransCoef;
 36   double sensibleHTransCoef = 1 / (1 / (clothRate * hcr) + rcl);
 37
 38   //反復計算でSET*を計算
 39   double hfSkin = sensibleHFSkin + latentHFSkin;
 40   double psk = Water.GetSaturationPressure(skinTemperature);
 41   Roots.ErrorFunction eFnc = delegate (double setStar)
 42   {
```

```
43     return hfSkin - sensibleHTransCoef * (skinTemperature - setStar)
44     - wettedness * latentHTransCoef * (psk - Water.GetSaturationPressure(setStar) * 0.5);
45   };
46   return Roots.Newton(eFnc, 26, 1e-3, 1e-3, 1e-3, 20);
47 }
```

SET*のみが必要な場合にも毎回 2-Node モデルを計算することは手間がかかるため、プログラム 27.1 とプログラム 27.2 を組み合わせ、外界条件から直接に SET*を計算するプログラムを 27.3 のように定義する。

プログラム 27.3　SET*の計算処理 2

Popolo.HumanBody.TwoNodeModel class

```
/// <summary>外界条件から SET*[C]を直接計算する</summary>
/// <param name="drybulbTemperature">乾球温度[CDB]</param>
/// <param name="meanRadiantTemperature">平均放射温度[C]</param>
/// <param name="velocity">相対気流速度[m/s]</param>
/// <param name="relativeHumidity">相対湿度[%]</param>
/// <param name="clothing">着衣量[clo]</param>
/// <param name="basalMetabolism">基礎代謝量[W/m2]</param>
/// <param name="externalWork">外部仕事量[W/m2]</param>
/// <param name="atmosphericPressure">大気圧[kPa]</param>
/// <returns>SET*[C]</returns>
public static double GetSETStarFromAmbientCondition
  (double drybulbTemperature, double meanRadiantTemperature, double relativeHumidity, double velocity,
  double clothing, double basalMetabolism, double externalWork, double atmosphericPressure)
{
  double st, ct, bt, clt, ss, ls, sr, lr, wd;
  GetSteadyState(drybulbTemperature, meanRadiantTemperature, relativeHumidity,
    velocity, clothing, basalMetabolism, externalWork, atmosphericPressure,
    out st, out ct, out bt, out clt, out ss, out ls, out sr, out lr, out wd);
  return GetSETStar(meanRadiantTemperature, basalMetabolism, externalWork, clt, st, ss, ls, wd);
}
```

【例題 27.1】

作用温度 22 °C、24 °C、26 °C、28 °C のそれぞれで人体の顕熱および潜熱負荷を計算せよ。また、各状態での SET*値も計算せよ。ただし活動内容は 1）劇場を想定した座位安静、2）事務所を想定した座位軽作業、3）店舗を想定した軽歩行、の 3 種とする。また、着衣量は 0.6 clo、相対気流速度は 0.3 m/s、相対湿度は 50 %とする。

【解】

計算処理をプログラム 27.4 に示す。代謝量は 1）座位安静: 45 W/m^2、2）座位軽作業: 65 W/m^2、3）軽歩行: 100 W/m^2 とした。計算結果は体表面積あたりの値であるため、16, 17 行で体表面積 1.8 m^2 を乗じる。結果をまとめると表 27.12 となる。

表 27.12　活動別の人体負荷および SET*値

	22 °C			24 °C			26 °C			28 °C		
	SH	LH	SET*	SH	LH	SET*	SH	LH	SET*	SH	LH	SET*
座位安静	91.6	22.7	19.3	79.2	21.9	21.6	66.9	21.3	23.9	57.8	21.4	26.1
座位軽作業	98.0	25.2	20.8	91.2	25.8	22.9	76.3	40.7	24.8	61.0	55.9	26.8
軽歩行	117.8	62.1	23.8	102.6	77.3	25.7	86.9	92.9	27.7	70.7	109.0	29.6

プログラム 27.4　活動別の人体負荷および SET*値の計算

```
private static void TwoNodeModelTest()
{
  double[] activity = new double[] { 45, 65, 100 };
  double[] opTemp = new double[] { 22, 24, 26, 28 };

  Console.WriteLine
    ("22CSH, 22CLH, 22CSET*, 24CSH, 24CLH, 24CSET*, 26CSH, 26CLH, 26CSET*, 28CSH, 28CLH, 28CSET*");
  double st, ct, bt, clt, ss, ls, sr, lr, wd;
  for (int i = 0; i < activity.Length; i++)
  {
    for (int j = 0; j < opTemp.Length; j++)
    {
      TwoNodeModel.GetSteadyState
        (opTemp[j], opTemp[j], 50, 0.3, 0.6, activity[i], 0, 101.325,
        out st, out ct, out bt, out clt, out ss, out ls, out sr, out lr, out wd);
      double sh = (ss + sr) * 1.8;    //顕熱負荷[W]
      double lh = (ls + lr) * 1.8;    //潜熱負荷[W]
      double setstar = TwoNodeModel.GetSETStar(opTemp[j], activity[i], 0, clt, st, ss, ls, wd);

```

```
20        Console.Write(sh.ToString("F1") + ", " + lh.ToString("F1") + ", " + setstar.ToString("F1") + ", ");
21      }
22      Console.WriteLine();
23    }
24 }
```

【例題 27.2】

事務所建物では夏季、中間期、冬季の設計用温湿度条件をそれぞれ 26 ℃ / 50 %、24 ℃ / 50 %、22 ℃ / 40 %、とすることが多い。これらの条件と SET*を等しくする温湿度条件の組み合わせを湿り空気線図状に記せ。代謝量は座位軽作業 65 W/m^2、相対気流速度は 0.2 m/s とし、着衣量は各季節でそれぞれ 0.6 clo、0.8 clo、1.1clo とする。

【解】

計算処理をプログラム 27.5 に示す。13~22 行で相対湿度を 30~100 %まで変化させる。15~19 行で誤差関数を定義し、19 行でニュートン・ラフソン法を用いて SET*が等しくなる温度を収束計算する。計算結果を第 4 章で作成した湿り空気線図の上にプロットすると図 27.6 が得られる。等比エンタルピー線に比較すると線が急傾斜になっている点に注意する必要がある。これは、相対湿度が高く、乾球温度が低い運転点を選択することにより、冷房時に湿り空気の比エンタルピーを高く維持しながら、同等の熱的快適性が得られる可能性を示している。この傾向は特に外気冷房の効果を最大化させようとする場合に有効となる[27.32)]。

プログラム 27.5　SET*を等しくする温湿度条件の組み合わせ計算

```
 1 private static void TwoNodeModelTest2()
 2 {
 3   //温度・湿度・着衣条件  0:夏季,1:冬季,2:中間期
 4   double[] dbt = new double[] { 26, 24, 22 };
 5   double[] hmd = new double[] { 50, 50, 40 };
 6   double[] clth = new double[] { 0.6, 0.8, 1.1 };
 7
 8   for (int i = 0; i < 3; i++)
 9   {
10     //標準条件の SET*
11     double stnd = TwoNodeModel.GetSETStarFromAmbientCondition
12       (dbt[i], dbt[i], hmd[i], 0.2, clth[i], 65, 0, 101.325);
13     for (int j = 30; j <= 100; j+=10)
14     {
15       Roots.ErrorFunction eFnc = delegate (double temp)
16       {
17         return stnd - TwoNodeModel.GetSETStarFromAmbientCondition(temp, temp, j, 0.2, clth[i], 65, 0,
18 101.325);
19       };
20       double set2 = Roots.Newton(eFnc, 24, 0.0001, 0.0001, 0.0001, 20);
21       Console.Write(", " + set2.ToString("F3"));
22     }
23     Console.WriteLine();
24   }
   }
```

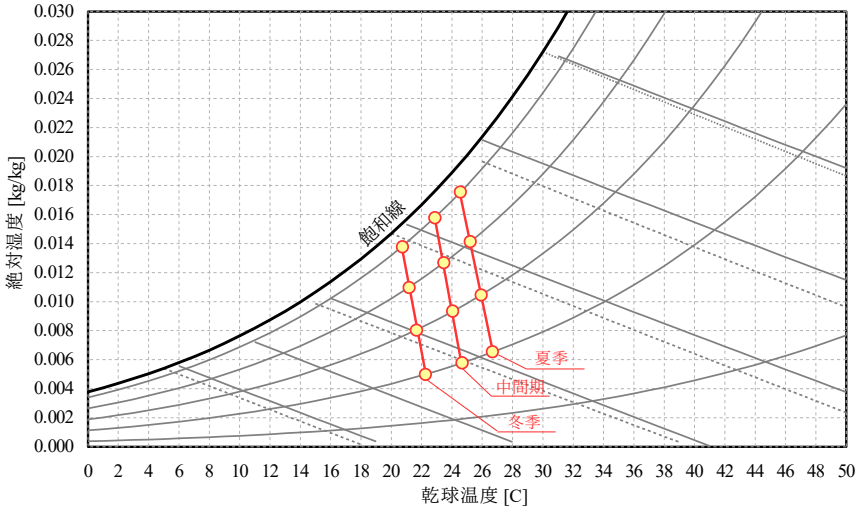


図 27.6　SET*が等しくなる温湿度条件

27.3.2　PMV, PPD の計算

プログラム 27.6 に人体への熱負荷計算処理を示す。32 行までで繰り返し計算に無関係の計算処理を行う。35 行目で着衣表面温度 T_{cl} の初期値を仮定し[†1]、式 27.55 に従い、45~56 行で反復計算を行う。式 27.53 を用いて 58~67 行で人体熱負荷の各要素を計算する。

プログラム 27.6　人体への熱負荷の計算処理

Popolo.HumanBody.ThermalComfort class

```
 1 /// <summary>人体への熱負荷[W/m2]を計算する</summary>
 2 /// <param name="drybulbTemperature">乾球温度[C]</param>
 3 /// <param name="meanRadiantTemperature">平均放射温度[C]</param>
 4 /// <param name="relativeHumidity">相対湿度[%]</param>
 5 /// <param name="relativeAirVelocity">相対気流速度[m/s]</param>
 6 /// <param name="clothing">着衣量[clo]</param>
 7 /// <param name="metabolicRate">代謝量[met]</param>
 8 /// <param name="externalWork">外部仕事量[met]</param>
 9 /// <returns>人体への熱負荷[W/m2]</returns>
10 public static double GetThermalLoad
11   (double drybulbTemperature, double meanRadiantTemperature, double relativeHumidity,
12   double relativeAirVelocity, double clothing, double metabolicRate, double externalWork)
13 {
14   double dbtA = CONVERT_C_TO_K + drybulbTemperature;
15   double mrtA = CONVERT_C_TO_K + meanRadiantTemperature;
16
17   //周囲の水蒸気分圧[kPa]の計算
18   double pa = relativeHumidity / 100d * Water.GetSaturationPressure(drybulbTemperature);
19
20   //代謝量[W/m2]の計算
21   double m = metabolicRate * CONVERT_MET_TO_W;
22   double mw = m - externalWork * CONVERT_MET_TO_W;
23
24   //着衣の熱抵抗[m2K/W]の計算
25   double rcl = 0.155 * clothing;
26   //着衣面積率[-]の計算
27   double fcl;
28   if (rcl < 0.078) fcl = 1.0 + 1.29 * rcl;
29   else fcl = 1.05 + 0.645 * rcl;
30
31   //強制対流による対流熱伝達率[W/(m2K)]
32   double hcf = 12.1 * Math.Sqrt(relativeAirVelocity);
33
34   //着衣表面温度の反復計算
35   double tcla = dbtA + (35.5 - drybulbTemperature) / (3.5 * rcl + 0.1);//初期値
36   double p1 = rcl * fcl;
37   double p2 = p1 * 3.96;
38   double p3 = p1 * 100;
39   double p4 = p1 * dbtA;
40   double p5 = 308.7 - 0.028 * mw + p2 * Math.Pow(mrtA / 100.0, 4);
41   double xn = tcla / 100.0;
42   double xf = xn;
43   double hc; //対流熱伝達率[W/(m2K)]
44   int iterNum = 0;
45   while (true)
46   {
47     if (150 < iterNum) throw new Exception("ThermalComofrt Class iteration error");
48     xf = (xf + xn) / 2d;
49     //対流熱伝達率[W/(m2K)]を更新
50     hc = Math.Max(hcf, 2.38 * Math.Pow(Math.Abs(100.0 * xn - dbtA), 0.25));
51     //状態値更新
52     xn = (p5 + p4 * hc - p2 * Math.Pow(xf, 4)) / (100d + p3 * hc);
53     //収束判定
54     if (Math.Abs(xn - xf) < 0.00015) break;
55     iterNum++;
56   }
57   //着衣表面温度[C]
58   double tcl = 100.0 * xn - CONVERT_C_TO_K;
59
60   //人体熱負荷の計算
61
62   double ediff = 3.05 * (5.733 - 0.00699 * mw - pa);  //皮膚表面からの潜熱損失[W/m2]
63   double esw = 0.42 * Math.Max(0, mw - CONVERT_MET_TO_W); //発汗による潜熱損失[W/m2]
64   double lres = 0.017 * m * (5.867 - pa); //呼吸による潜熱損失[W/m2]
65   double dres = 0.0014 * m * (34.0 - drybulbTemperature); //呼吸による顕熱損失[W/m2]
```

†1　この式の意味を考えてみたが、皮膚表面温度 T_{sk} を 35.5 ℃、着衣表面の対流熱伝達率 h_c を 3.5 W/(m²·K)と仮定し、$h_c\,(T_{cl} - T_a) = (T_{sk} - T_a) / R_{cl}$ の関係式を T_{cl} について解いたもののようだ。着衣量 0 の場合には R_{cl}=0 となるため、0 割りを防ぐ趣旨で 0.1 を加算したものと推測する。

```
66   double r = 3.96 * fcl * (Math.Pow(xn, 4) - Math.Pow(mrtA / 100.0, 4)); //着衣表面からの放射熱損失[W/m2]
67   double c = fcl * hc * (tcl - drybulbTemperature);   //着衣表面からの対流熱損失[W/m2]
68
69   //集計
70   return mw - (ediff + esw + lres + dres + r + c);
71 }
```

プログラム 27.7 に PMV および PPD の計算処理を示す。1~9 行は式 27.42 の実装である。直接に温熱 6 要素から計算もできるように、11~27 行でオーバーロードする。29~36 行は式 27.56 の実装である。

プログラム 27.7　PMV および PPD の計算処理

Popolo.HumanBody.ThermalComfort class

```
 1 /// <summary>PMV 値[-]を計算する</summary>
 2 /// <param name="metabolicRate">代-謝量[met]</param>
 3 /// <param name="thermalLoad">人体への熱負荷[W/m2]</param>
 4 /// <returns>PMV 値[-]</returns>
 5 public static double GetPMV(double metabolicRate, double thermalLoad)
 6 {
 7   double m = metabolicRate * CONVERT_MET_TO_W;
 8   return (0.303 * Math.Exp(-0.036 * m) + 0.028) * thermalLoad;
 9 }
10
11 /// <summary>PMV 値[-]を計算する</summary>
12 /// <param name="drybulbTemperature">乾球温度[CDB]</param>
13 /// <param name="meanRadiantTemperature">平均放射温度[C]</param>
14 /// <param name="relativeHumidity">相対湿度[%]</param>
15 /// <param name="relativeAirVelocity">相対気流速度[m/s]</param>
16 /// <param name="clothing">着衣量[clo]</param>
17 /// <param name="metabolicRate">代謝量[met]</param>
18 /// <param name="externalWork">外部仕事量[met]</param>
19 /// <returns>PMV 値[-]</returns>
20 public static double GetPMV
21   (double drybulbTemperature, double meanRadiantTemperature, double relativeHumidity,
22   double relativeAirVelocity, double clothing, double metabolicRate, double externalWork)
23 {
24   double thermalLoad = GetThermalLoad(drybulbTemperature, meanRadiantTemperature,
25     relativeHumidity, relativeAirVelocity, clothing, metabolicRate, externalWork);
26   return GetPMV(metabolicRate, thermalLoad);
27 }
28
29 /// <summary>PPD 値[%]を計算する</summary>
30 /// <param name="pmv">PMV 値</param>
31 /// <returns>PPD 値[%]</returns>
32 public static double GetPPD(double pmv)
33 {
34   double p2 = pmv * pmv;
35   return 100.0 - 95.0 * Math.Exp(-0.03353 * p2 * p2 - 0.2179 * p2);
36 }
```

PMV 制御は、制御対象を乾球温度や露点温度ではなく、人間の快適性に直結する PMV 値とすることで空調の消費エネルギー量を削減する手法である。このような制御を行う場合には、乾球温度を除く温熱 6 要素（相対湿度、相対気流速度、平均放射温度、着衣量、代謝量）と PMV 値から乾球温度を逆算する処理が必要となる。PMV 値に基づく乾球温度の計算処理をプログラム 27.8（1~35 行）に示す。14~24 行で乾球温度の上下限値を設定して PMV を求め、対応が不可能な PMV 値の場合には、上下限値を出力する。26~34 行でニュートン・ラプソン法を用いて指定の PMV 値に合致する乾球温度を求める。

プログラム 27.8　PMV 値に基づく乾球温度の逆算処理

Popolo.HumanBody.ThermalComfort class

```
 1 /// <summary>PMV 値[-]から乾球温度[C]を求める</summary>
 2 /// <param name="pmv">PMV 値[-]</param>
 3 /// <param name="meanRadiantTemperature">平均放射温度[C]</param>
 4 /// <param name="relativeAirVelocity">気流速度[m/s]</param>
 5 /// <param name="relativeHumidity">相対湿度[%]</param>
 6 /// <param name="clothing">着衣量[clo]</param>
 7 /// <param name="metabolicRate">代謝量[met]</param>
 8 /// <param name="externalWork">外部仕事量[met]</param>
 9 /// <returns>乾球温度[C]</returns>
```

```
10 public static double GetDrybulbTemperature
11   (double pmv, double meanRadiantTemperature, double relativeHumidity, double relativeAirVelocity,
12   double clothing, double metabolicRate, double externalWork)
13 {
14   //乾球温度上下限値
15   double MAXDB = 50;
16   double MINDB = -10;
17
18   //上下限値確認
19   double pmvh = GetPMV(MAXDB, meanRadiantTemperature, relativeHumidity,
20     relativeAirVelocity, clothing, metabolicRate, externalWork);
21   if (pmvh < pmv) return MAXDB;
22   double pmvl = GetPMV(MINDB, meanRadiantTemperature, relativeHumidity,
23     relativeAirVelocity, clothing, metabolicRate, externalWork);
24   if (pmv < pmvl) return MINDB;
25
26   //ニュートン・ラフソン法で収束計算
27   Roots.ErrorFunction eFnc = delegate (double dbTemp)
28   {
29     //誤差を評価
30     double pmv2 = GetPMV(dbTemp, meanRadiantTemperature, relativeHumidity,
31       relativeAirVelocity, clothing, metabolicRate, externalWork);
32     return pmv2 - pmv;
33   };
34   return Roots.Newton(eFnc, 26, 0.0001, 0.01, 0.01, 20);
35 }
```

【例題 27.3】

放射温度および相対湿度に基づいて、PMV が 0 となる乾球温度を求める線図を作成せよ。代謝量はタイピング作業（1.1 met）、相対気流速度は 0.1 m/s とする。夏季と冬季を想定し、冷房時は着衣量=0.65 clo、度暖房時は着衣量=0.80 clo とする。

【解】

計算処理をプログラム 27.9 に示す。相対湿度および放射温度を変化させ（11, 15, 28 行）、各条件において PMV=0 となる乾球温度を計算（20, 33 行）して CSV ファイルに書き出しを行う。計算結果をグラフ化すると図 27.7 が得られる。

プログラム 27.9　PMV=0 となる乾球温度、平均放射温度、相対湿度の計算処理

```
 1 private static void PMVTest()
 2 {
 3   //上下限値
 4   const int HMD_MAX = 70;
 5   const int HMD_MIN = 20;
 6
 7   using (StreamWriter sWriter =
 8   new StreamWriter("PMVTest.csv", false, Encoding.GetEncoding("Shift_JIS")))
 9   {
10     //タイトル行
11     for (int hmd = HMD_MIN; hmd <= HMD_MAX; hmd += 10) sWriter.Write("," + hmd.ToString("F0"));
12     sWriter.WriteLine();
13
14     sWriter.WriteLine("夏季");
15     for (int mrt = 20; mrt <= 40; mrt++)
16     {
17       sWriter.Write(mrt.ToString("F0"));
18       for (int hmd = HMD_MIN; hmd <= HMD_MAX; hmd += 10)
19       {
20         double pmv = ThermalComfort.GetDrybulbTemperature(0, mrt, hmd, 0.1, 0.65, 1.1, 0.0);
21         sWriter.Write("," + pmv.ToString("F2"));
22       }
23       sWriter.WriteLine();
24     }
25
26     sWriter.WriteLine();
27     sWriter.WriteLine("冬季");
28     for (int mrt = 10; mrt <= 30; mrt++)
29     {
30       sWriter.Write(mrt.ToString("F0"));
31       for (int hmd = HMD_MIN; hmd <= HMD_MAX; hmd += 10)
32       {
33         double pmv = ThermalComfort.GetDrybulbTemperature(0, mrt, hmd, 0.1, 0.80, 1.1, 0.0);
34         sWriter.Write("," + pmv.ToString("F2"));
35       }
36       sWriter.WriteLine();
37     }
38   }
39 }
```

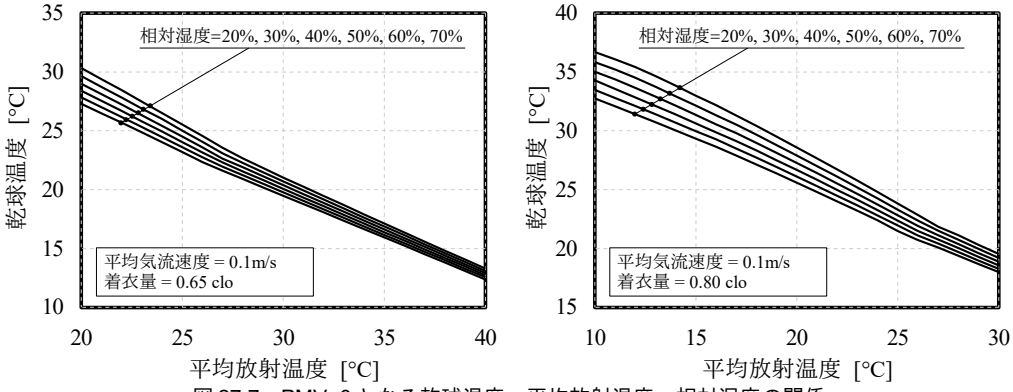


図 27.7　PMV=0 となる乾球温度、平均放射温度、相対湿度の関係

27.3.3　非定常モデルの計算

1）モデルの行列表現

27.2.3 節で示した熱平衡式を差分化して行列で表現すると式 27.98 が得られる。ただし、ベクトル TV_i、ZV_i と行列 BM_i は表 27.13~27.15である。また Tは一時点前の温度を表す。

$$ZV_i = BM_i\, TV_i \tag{27.98}$$

表 27.13 で下線をつけた項は別の部位の温度であるため、連成させて解く必要があり、人体全体を 1 つの行列で表現すると図 27.8 となる。ただし ZV_i'は ZV_iから下線部のある項を取り除いたベクトルである。各部位で核、筋肉、脂肪、皮膚、動脈、静脈の 6 の質点があり、さらに四肢部位においては表在静脈があるため、質点は全部で 6×5+7×12=114 に中央血液溜まりを加えた 115 点となる。従って図 27.8 は 115×115 の行列であり、この逆行列を解くことで体温を更新できる。なお、中央血液溜まりの熱平衡式 27.83 を差分化すると式 27.99 が得られる。

$$\frac{C_{cb}}{\Delta t}T'_{cb} = \left\{\frac{C_{cb}}{\Delta t} + \rho\, Cp_b \sum_{m=0}\left(BF_{ve,m} + BF_{sv,m}\right)\right\}T_{cb} - \sum_{m=0}\rho\, Cp_b\, BF_{ve,m}\, T_{ve^*,m} - \sum_{m=0}\rho\, Cp_b\, BF_{sv,m}\, T_{sv^*,m} \tag{27.99}$$

2）クラスの構成

非定常人体モデルを表す MultiNodeModel クラスを定義する。また、プログラムをわかりやすくするため、MultiNodeModel クラス内の入れ子クラスとして、部位を表す bodyPart クラスを定義する。bodyPart クラスは private とすることで、MultiNodeModel クラス以外からの利用ができないようにする。

3）bodyPart クラス

プログラム 27.10 に bodyPart クラスのクラス変数を示す。表 27.3 や表 27.4 の定数値を部位別に保持する変数である。既に記したように本モデルは 115 の状態変数（体温）を持つモデルであるが、23 行目の mOffset は各部位の体温が行列のどの要素を占めるかを示した変数である。行列の中での部位の並びは、中央血液溜まり、頭、首、胸、背中、腰、下腹部、左肩、左腕、左手、右肩、右腕、右手、左大腿部、左下腿部、左脚、右大腿部、右下腿部、右脚の順とし、組織の並びは、核、筋肉、脂肪、皮膚、動脈、静脈、表在静脈の順とする。具体的には、0: 中央血液溜まり温度、1: 頭部核温度、2: 頭部筋肉温度、3: 頭部皮膚温度...の順で並ぶ。static コンストラクタで行う mOffset の初期化処理をプログラム 27.11 に示す。紙面の都合上、他のクラス変数の初期化処理は省略する。

表 27.13　ベクトル TV_i^{-1}

T_{cr}	T_{ms}	T_{fat}	T_{sk}	T_{ar}	T_{ve}	T_{sv}

表 27.14　行列 BM_i

$\frac{C_{cr}}{\Delta t}+\rho Cp_b BF_{cr}$ $+2K_{cr-ve}+K_{cr-ms}$	$-K_{cr-ms}$			$-\rho Cp_b BF_{cr}$ $-K_{cr-ve}$	$-K_{cr-ve}$	
$-K_{cr-ms}$	$\frac{C_{ms}}{\Delta t}+\rho Cp_b BF_{ms}$ $+K_{cr-ms}+K_{ms-fat}$	$-K_{ms-fat}$		$-\rho Cp_b BF_{ms}$		
	$-K_{ms-fat}$	$\frac{C_{fat}}{\Delta t}+\rho Cp_{ve} BF_{fat}$ $+K_{ms-fat}+K_{fat-sk}$	$-K_{fat-sk}$	$-\rho Cp_{ve} BF_{fat}$		
		$-K_{fat-sk}$	$\frac{C_{sk}}{\Delta t}+\rho Cp_b BF_{sk}$ $+K_{fat-sk}+K_{sv-sk}$ $+cf\ K_{mt}+(1-cf)h_i^{'}$	$-\rho Cp_b BF_{sk}$		$-K_{sv-sk}$
$-K_{cr-ve}$				$\frac{C_{ar}}{\Delta t}+\rho Cp_b BF_{ar}$ $+K_{cr-ve}+K_{ve}$	$-K_{ve}$	
$-\rho Cp_b BF_{cr}$ $-K_{cr-ve}$	$-\rho Cp_b BF_{ms}$	$-\rho Cp_b BF_{fat}$	$-\rho Cp_b BF_{sk}$	$-K_{ve}$	$\frac{C_{ve}}{\Delta t}+\rho Cp_b$ $\{BF_{cr}+BF_{ms}$ $+BF_{fat}+BF_{sk}$ $+\sum_{m=0} BF_{ve^*}$ $+\langle\langle BF_{sv}\rangle\rangle\}$ $+K_{cr-ve}+K_{ve}$	
			$-K_{sv-sk}$	$\langle\langle\langle\langle-\rho Cp_b$ $BF_{sv}\rangle\rangle\rangle\rangle$		$\frac{C_{sv}}{\Delta t}+K_{sv-sk}$ $+\rho Cp_b BF_{sv}$

<<·>>は腰のみ、<<<<·>>>>は手と脚のみ

表 27.15　ベクトル ZV_i^{-1}

$\frac{C_{cr}}{\Delta t}T_{cr}^{'}+M_{cr}$ $-\langle C_{res}+E_{res}\rangle$	$\frac{C_{ms}}{\Delta t}T_{ms}^{'}+M_{ms}$	$\frac{C_{fat}}{\Delta t}T_{fat}^{'}+M_{fat}$	$\frac{C_{sk}}{\Delta t}T_{sk}^{'}+M_{sk}$ $+cf\ K_{mt}T_{mt}$ $+(1-cf)$ $\{(P_{a,i}-P_{Ws,sk,i})$ $wh_e^{'}+h_i^{'}T_o\}$	$\frac{C_{ar}}{\Delta t}T_{ar}^{'}$ $+\rho Cp_b BF_{ar}\underline{T_{ar^*}}$	$\frac{C_{ve}}{\Delta t}T_{ve}^{'}+\rho Cp_b\sum_{m=0}$ $\{BF_{ve^*,m}\underline{T_{ve^*,m}}+$ $\langle\langle BF_{sv,m}\underline{T_{sv^*,m}}\rangle\rangle\}$	$\frac{C_{sv}}{\Delta t}T_{sv}^{'}$ $+\langle\langle\langle\rho Cp_b$ $BF_{sv}\underline{T_{sv^*,m}}\rangle\rangle\rangle$

<·>は胸部位のみ、<<·>>は腰のみ、<<<·>>>は肩・腕・大腿・下腿のみ

$$
\begin{bmatrix} T_{cb}^{'} \\ ZV_1^{'} \\ ZV_2^{'} \\ ZV_3^{'} \\ ZV_4^{'} \\ ZV_5^{'} \\ ZV_6^{'} \\ ZV_7^{'} \\ ZV_8^{'} \\ ZV_9^{'} \\ ZV_{10}^{'} \\ ZV_{11}^{'} \\ ZV_{12}^{'} \\ ZV_{13}^{'} \\ ZV_{14}^{'} \\ ZV_{15}^{'} \\ ZV_{16}^{'} \\ ZV_{17}^{'} \end{bmatrix} =
\begin{bmatrix}
 & \boxed{\text{中央血液だまりへの血流に関する項}} & & & & & \\
BM_1 & BM_2 & & & & & \\
 & BM_3 & \boxed{\text{部位間血流による熱移動に関する項}} & & & & \\
 & & BM_4 & BM_5 & & & \\
 & & & BM_6 & \ddots & & \\
 & & & & BM_{12} & BM_{13} & BM_{14} \\
 & & & & & & BM_{15} \\
 & \boxed{\text{部位間血流による熱移動に関する項}} & & & & BM_{16} & \\
 & & & & & & BM_{17}
\end{bmatrix}
\begin{bmatrix} T_{cb} \\ TV_1 \\ TV_2 \\ TV_3 \\ TV_4 \\ TV_5 \\ TV_6 \\ TV_7 \\ TV_8 \\ TV_9 \\ TV_{10} \\ TV_{11} \\ TV_{12} \\ TV_{13} \\ TV_{14} \\ TV_{15} \\ TV_{16} \\ TV_{17} \end{bmatrix}
$$

図 27.8　人体熱平衡式の行列表現

プログラム **27.10**　**bodyPart** クラスのクラス変数

Popolo.HumanBody.MultiNodeModel.bodyPart class

```
/// <summary>重量比 1[-]</summary>
private static readonly Dictionary<Node, double> rWeight1;

/// <summary>重量比 2[-]</summary>
private static readonly Dictionary<Node, double[]> rWeight2;

/// <summary>表面積比[-]</summary>
private static readonly Dictionary<Node, double> rSurface;

/// <summary>円柱長さ[m]</summary>
private static readonly Dictionary<Node, double> cLength;

/// <summary>基礎代謝比[-]</summary>
private static readonly Dictionary<Node, double[]> bMetRate;

/// <summary>基礎血流比[-]</summary>
private static readonly Dictionary<Node, double[]> bBFRate;

/// <summary>血管の熱コンダクタンス[W/K]</summary>
private static readonly Dictionary<Node, double[]> hCdBLD;

/// <summary>行列のオフセット[-]</summary>
internal static readonly Dictionary<Node, int> mOffset;

/// <summary>立位の対流熱伝達率[W/(m2 K)]</summary>
private static readonly Dictionary<Node, double> cHTransferStand;

/// <summary>座位の対流熱伝達率[W/(m2 K)]</summary>
private static readonly Dictionary<Node, double> cHTransferSit;

/// <summary>温冷感信号の重み付け係数[-]</summary>
private static readonly Dictionary<Node, double> skinSignal;

/// <summary>発汗信号の重み付け係数[-]</summary>
private static readonly Dictionary<Node, double> sweatSignalR;

/// <summary>ふるえ信号の重み付け係数[-]</summary>
private static readonly Dictionary<Node, double> shivSignalR;

/// <summary>血管収縮信号の重み付け係数[-]</summary>
private static readonly Dictionary<Node, double> dilSignalR;

/// <summary>血管拡張信号の重み付け係数[-]</summary>
private static readonly Dictionary<Node, double> strSignalR;
```

プログラム **27.11**　**mOffset** の初期化処理

Popolo.HumanBody.MultiNodeModel.bodyPart class

```
mOffset = new Dictionary<Node, int>();
mOffset.Add(Node.Head, 1);
mOffset.Add(Node.Neck, 7);
mOffset.Add(Node.Chest, 13);
mOffset.Add(Node.Back, 19);
mOffset.Add(Node.Pelvis, 25);
mOffset.Add(Node.LeftShoulder, 31);
mOffset.Add(Node.LeftArm, 38);
mOffset.Add(Node.LeftHand, 45);
mOffset.Add(Node.RightShoulder, 52);
mOffset.Add(Node.RightArm, 59);
mOffset.Add(Node.RightHand, 66);
mOffset.Add(Node.LeftThigh, 73);
mOffset.Add(Node.LeftLeg, 80);
mOffset.Add(Node.LeftFoot, 87);
mOffset.Add(Node.RightThigh, 94);
mOffset.Add(Node.RightLeg, 101);
mOffset.Add(Node.RightFoot, 108);
```

プログラム 27.12 に列挙型の定義を示す。体の部位を示す Node と組織を示す Layer を定義する。

プログラム **27.12**　列挙型定義

Popolo.HumanBody.MultiNodeModel class

```
/// <summary>体の部位</summary>
[Flags]
public enum Node
{
  /// <summary>頭</summary>
```

```
 6   Head = 1,
 7   /// <summary>首</summary>
 8   Neck = 2,
 9   /// <summary>胸</summary>
10   Chest = 4,
11   /// <summary>背中</summary>
12   Back = 8,
13   /// <summary>下腹部</summary>
14   Pelvis = 16,
15   /// <summary>左肩</summary>
16   LeftShoulder = 32,
17   /// <summary>左腕</summary>
18   LeftArm = 64,
19   /// <summary>左手</summary>
20   LeftHand = 128,
21   /// <summary>右肩</summary>
22   RightShoulder = 256,
23   /// <summary>右腕</summary>
24   RightArm = 512,
25   /// <summary>右手</summary>
26   RightHand = 1024,
27   /// <summary>左大腿部</summary>
28   LeftThigh = 2048,
29   /// <summary>左下腿部</summary>
30   LeftLeg = 4096,
31   /// <summary>左脚</summary>
32   LeftFoot = 8192,
33   /// <summary>右大腿部</summary>
34   RightThigh = 16384,
35   /// <summary>右下腿部</summary>
36   RightLeg = 32768,
37   /// <summary>右脚</summary>
38   RightFoot = 65536,
39 }
40
41 /// <summary>組織</summary>
42 public enum Layer
43 {
44   /// <summary>核層</summary>
45   Core = 0,
46   /// <summary>筋肉層</summary>
47   Muscle = 1,
48   /// <summary>脂肪層</summary>
49   Fat = 2,
50   /// <summary>皮膚層</summary>
51   Skin = 4,
52   /// <summary>動脈</summary>
53   Artery = 8,
54   /// <summary>表在静脈</summary>
55   SuperficialVein = 16,
56   /// <summary>深部動脈</summary>
57   DeepVein = 32,
58   /// <summary>AVA</summary>
59   AVA = 64
60 }
```

プログラム 27.13 に定数宣言を示す。表在静脈に関する計算をする都合上、四肢部位および四肢末端部はプログラム内で条件分岐をかけることが多い特殊な部位であるため、37~44 行で両部位を示す定数を定義する。

プログラム 27.13　定数宣言

Popolo.HumanBody.MultiNodeModel class

```
 1 /// <summary>血液の体積比熱[J/(mLK)]</summary>
 2 public const double BLD_SPECIFICHEAT = 3.842;
 3
 4 /// <summary>代謝量換算係数[(W/m2)/met]</summary>
 5 private const double CONVERT_MET_TO_W = 58.2;
 6
 7 /// <summary>標準体躯の表面積[m2]</summary>
 8 private const double STANDARD_SURFACE_AREA = 1.87;
 9
10 /// <summary>標準体躯の血流量[mL/s]</summary>
11 private const double STANDARD_BLOOD_FLOW = 290.004 / 3.6;
12
13 /// <summary>標準体躯の重量[kg]</summary>
14 private const double STANDARD_WEIGHT = 74.43;
15
```

```
16 /// <summary>大気圧=101.325[kPa]</summary>
17 private const double ATMOSPHERIC_PRESSURE = 101.325;
18
19 /// <summary>骨の定圧比熱[J/(kgK)]</summary>
20 private const double SPECIFIC_HEAT_BORN = 2092;
21
22 /// <summary>脂肪の定圧比熱[J/(kgK)]</summary>
23 private const double SPECIFIC_HEAT_FAT = 2510;
24
25 /// <summary>その他の人体定圧比熱[J/(kgK)]</summary>
26 private const double SPECIFIC_HEAT_ELSE = 3766;
27
28 /// <summary>黒体の放射定数[W/(m2K4)]</summary>
29 private const double BLACK_CONSTANT = 5.67e-8;
30
31 /// <summary>絶対温度と摂氏との変換定数</summary>
32 private const double CONVERT_C_TO_K = 273.15;
33
34 /// <summary>衣服の透湿係数[K/kPa]</summary>
35 private const double I_CLS = 0.45;
36
37 /// <summary>四肢末端部位</summary>
38 private const Node TERMINAL_NODE =
39   Node.LeftHand | Node.RightHand | Node.LeftFoot | Node.RightFoot;
40
41 /// <summary>四肢部位</summary>
42 private const Node LIMBS =
43   TERMINAL_NODE | Node.LeftShoulder | Node.RightShoulder | Node.LeftArm |
44   Node.RightArm | Node.LeftThigh | Node.RightThigh | Node.LeftLeg | Node.RightLeg;
```

プログラム 27.14 にインスタンス変数とプロパティを示す。MultiNodeModel クラスから参照する変数に関してはアクセス修飾子を internal とする。

プログラム 27.14　インスタンス変数とプロパティ

Popolo.HumanBody.MultiNodeModel.bodyPart class

```
 1 /// <summary>人体</summary>
 2 private ImmutableMultiNodeModel body;
 3
 4 /// <summary>熱コンダクタンス[W/K]</summary>
 5 /// <remarks>
 6 /// 0:核-筋肉,1:筋肉-脂肪,2:脂肪-皮膚,3:核-血管,4:皮膚-血管,5:動脈-静脈,
 7 /// 6:皮膚-物体,7:皮膚-外界(顕熱),8:皮膚-外界(潜熱)
 8 /// </remarks>
 9 private double[] hConductance = new double[9];
10
11 /// <summary>最大 AVA 血流[mL/s]</summary>
12 private double maxAVA = 0;
13
14 /// <summary>皮膚層の基礎血流量[mL/s]</summary>
15 private double basalBloodFlow_Skin;
16
17 /// <summary>筋肉層の基礎血流量[mL/s]</summary>
18 private double basalBloodFlow_Muscle;
19
20 /// <summary>初期化中か否か</summary>
21 internal bool initializing { get; set; }
22
23 /// <summary>重量[kg]を取得する</summary>
24 internal double weight { get; private set; }
25
26 /// <summary>筋肉の重量[kg]を取得する</summary>
27 internal double muscleWeight { get; private set; }
28
29 /// <summary>表面積[m2]を取得する</summary>
30 internal double surfaceArea { get; private set; }
31
32 /// <summary>部位を取得する</summary>
33 internal Node node { get; private set; }
34
35 /// <summary>皮膚接触部の割合[-]を取得する</summary>
36 internal double contactPortionRate { get; private set; }
37
38 /// <summary>皮膚接触部の温度[C]を取得する</summary>
39 internal double materialTemperature { get; private set; }
40
41 /// <summary>対流熱伝達率[W/(m2 K)]を取得する</summary>
42 internal double convectiveHeatTransferCoefficient { get; private set; }
43
```

```
 44 /// <summary>放射熱伝達率[W/(m2 K)]を取得する</summary>
 45 internal double radiativeHeatTransferCoefficient { get; private set; }
 46
 47 /// <summary>Clo 値[clo]を取得する</summary>
 48 internal double clothingIndex { get; private set; }
 49
 50 /// <summary>相対気流速度[m/s]を取得する</summary>
 51 internal double velocity { get; private set; }
 52
 53 /// <summary>着衣表面温度[C]を取得する</summary>
 54 internal double clothTemperature { get; private set; }
 55
 56 /// <summary>平均放射温度[C]を取得する</summary>
 57 internal double meanRadiantTemperature { get; private set; }
 58
 59 /// <summary>乾球温度[C]を取得する</summary>
 60 internal double drybulbTemperature { get; private set; }
 61
 62 /// <summary>水蒸気分圧[kPa]を取得する</summary>
 63 internal double waterVaporPressure { get; private set; }
 64
 65 /// <summary>温度[C]を取得する</summary>
 66 internal Dictionary<Layer, double> temperatures { get; private set; }
 67
 68 /// <summary>熱容量[kJ/K]</summary>
 69 internal Dictionary<Layer, double> heatCapacity { get; private set; }
 70
 71 /// <summary>血流[mL/s]</summary>
 72 internal Dictionary<Layer, double> bloodFlow { get; private set; }
 73
 74 /// <summary>ふるえによる熱産生[W]を取得する</summary>
 75 internal double shiveringLoad { get; private set; }
 76
 77 /// <summary>外部仕事[W]を設定・取得する</summary>
 78 internal double externalWork { get; set; }
 79
 80 /// <summary>基礎代謝[W]を取得する</summary>
 81 internal Dictionary<Layer, double> basalMetabolicRate { get; private set; }
 82
 83 /// <summary>相当温度[C]を取得する</summary>
 84 internal double operatingTemperature { get; private set; }
 85
 86 /// <summary>中央血液溜まりの熱容量[J/K]を取得する</summary>
 87 internal double centralBloodHeatCapacity { get; private set; }
 88
 89 /// <summary>皮膚のセットポイント[C]を設定・取得する</summary>
 90 internal double setPoint_Skin { get; set; }
 91
 92 /// <summary>核のセットポイント[C]を設定・取得する</summary>
 93 internal double setPoint_Core { get; set; }
 94
 95 /// <summary>発汗による蒸発熱損失[W]を取得する</summary>
 96 internal double evaporativeHeatLoss_Sweat { get; private set; }
 97
 98 /// <summary>蒸発熱損失[W]を取得する</summary>
 99 internal double latentHeatLoss { get; private set; }
100
101 /// <summary>接続先の部位を取得する</summary>
102 internal List<bodyPart> downStreamParts { get; private set; }
103
104 /// <summary>接続元の部位を取得する</summary>
105 internal bodyPart upperStreamPart { get; private set; }
```

プログラム 27.15 にコンストラクタを示す。コンストラクタではプログラム 27.10 に示した各種の変数と MultiNodeModel クラスから与えられる人体全身の情報を用いて、部位ごとの情報を初期化する。31~38 行は体脂肪率による重量調整処理であり、式 27.57、式 27.58 を用いる。62~67 行で部位の体積を求め、頭か否かに応じて式 27.71 と式 27.73 を用いて熱コンダクタンスを計算する。

プログラム 27.15　コンストラクタ

Popolo.HumanBody.MultiNodeModel.bodyPart class

```
 1 /// <summary>インスタンスを初期化する</summary>
 2 /// <param name="body">人体</param>
 3 /// <param name="node">部位</param>
 4 /// <param name="bodyWeight">全身の体重[kg]</param>
 5 /// <param name="bodyHeight">身長[m]</param>
 6 /// <param name="bodySurfaceArea">全身の体表面積[m2]</param>
```

```
/// <param name="fatPercentage">体脂肪率[%]</param>
/// <param name="metabolicRate">部位の基礎代謝[W]</param>
/// <param name="basalBloodFlow">基礎血流量[mL/s]</param>
internal bodyPart
  (ImmutableMultiNodeModel body, Node node, double bodyWeight, double bodyHeight,
  double bodySurfaceArea, double fatPercentage, double metabolicRate, double basalBloodFlow)
{
  this.body = body;

  temperatures = new Dictionary<Layer, double>();
  heatCapacity = new Dictionary<Layer, double>();
  bloodFlow = new Dictionary<Layer, double>();
  basalMetabolicRate = new Dictionary<Layer, double>();
  downStreamParts = new List<bodyPart>();

  this.weight = bodyWeight * rWeight1[node];
  this.surfaceArea = bodySurfaceArea * rSurface[node];
  this.node = node;
  double[] rWt = (double[])rWeight2[node].Clone();
  double[] rMb = bMetRate[node];
  double[] rBFb = bBFRate[node];
  double len = cLength[node] * Math.Pow(bodyHeight / 1.72, 0.725);
  hCdBLD[node].CopyTo(hConductance, 3);

  //脂肪率による重量調整
  double rFat;
  if (fatPercentage < 0.15) rFat = rWt[3] * fatPercentage / 0.15;
  else rFat = (rWt[3] * (1 - fatPercentage) + fatPercentage - 0.15) / (1 - 0.15);
  double rr = (1 - rFat) / (1 - rWt[3]);
  for (int i = 0; i < rWt.Length; i++) rWt[i] *= rr * weight;
  rWt[3] = rFat * weight;
  muscleWeight = rWt[2];

  //基礎代謝[W]の計算
  basalMetabolicRate[Layer.Core] = metabolicRate * rMb[0];
  basalMetabolicRate[Layer.Muscle] = metabolicRate * rMb[1];
  basalMetabolicRate[Layer.Fat] = metabolicRate * rMb[2];
  basalMetabolicRate[Layer.Skin] = metabolicRate * rMb[3];

  //基礎血流[mL/s]の計算//核と脂肪は固定値
  bloodFlow[Layer.Core] = basalBloodFlow * rBFb[0];
  basalBloodFlow_Muscle = basalBloodFlow * rBFb[1];
  bloodFlow[Layer.Fat] = basalBloodFlow * rBFb[2];
  basalBloodFlow_Skin = basalBloodFlow * rBFb[3];

  //熱容量[J/K]の計算
  heatCapacity[Layer.Core] = rWt[0] * SPECIFIC_HEAT_BORN + rWt[1] * SPECIFIC_HEAT_ELSE;
  heatCapacity[Layer.Muscle] = rWt[2] * SPECIFIC_HEAT_ELSE;
  heatCapacity[Layer.Fat] = rWt[3] * SPECIFIC_HEAT_FAT;
  heatCapacity[Layer.Skin] = rWt[4] * SPECIFIC_HEAT_ELSE;
  heatCapacity[Layer.Artery] = rWt[5] * SPECIFIC_HEAT_ELSE;
  heatCapacity[Layer.DeepVein] = rWt[6] * SPECIFIC_HEAT_ELSE;
  heatCapacity[Layer.SuperficialVein] = rWt[7] * SPECIFIC_HEAT_ELSE;
  centralBloodHeatCapacity = rWt[8] * SPECIFIC_HEAT_ELSE;

  //各層の体積[m3]の計算
  double[] wt = new double[4];
  wt[0] = 0.001 * (rWt[0] + rWt[1] + rWt[5] + rWt[6] + rWt[8]); //核の体積[m3]
  wt[1] = rWt[2] / 1000d; //筋肉の体積[m3]
  wt[2] = rWt[3] / 1000d; //脂肪の体積[m3]
  wt[3] = 0.001 * (rWt[4] + rWt[7]);  //皮膚の体積[m3]

  //四肢末端部の場合には AVA 最大流量を計算
  if ((node & TERMINAL_NODE) != 0) maxAVA = wt[3] * 5000;

  //熱コンダクタンス[W/K]の計算
  double[] lmda = new double[] { 0.4184, 0.4184, 0.3347, 0.3347 };  //熱伝導率[W/mK]
  double[] rads = new double[7];
  rads[0] = wt[0] / 2;
  for (int i = 1; i < rads.Length; i++) rads[i] = rads[i - 1] + wt[i / 2] / 2;
  if (node == Node.Head)
  {
    //球とみなして計算
    for (int i = 0; i < rads.Length; i++) rads[i] = Math.Pow(rads[i] * 3d / (4d * Math.PI), 1d / 3d);
    for (int i = 0; i < 3; i++)
      hConductance[i] = 4d * Math.PI / ((1 / rads[2 * i] - 1 / rads[2 * i + 1]) / lmda[i]
        + (1 / rads[2 * i + 1] - 1 / rads[2 * i + 2]) / lmda[i + 1]);
  }
  else
  {
```

```
87     //多層円管とみなして計算
88     for (int i = 0; i < rads.Length; i++) rads[i] = Math.Sqrt(rads[i] / (Math.PI * len));
89     for (int i = 0; i < 3; i++)
90       hConductance[i] = 2d * Math.PI * len / (Math.Log(rads[2 * i + 1] / rads[2 * i]) / lmda[i]
91         + Math.Log(rads[2 * i + 2] / rads[2 * i + 1]) / lmda[i + 1]);
92   }
93 }
```

プログラム 27.16 に境界条件設定に関する処理を示す。1~10 行は皮膚表面での物体への接触状況を設定する処理であり、物体温度、物体への熱コンダクタンス、接触割合を設定する。12~18 行と 20~33 行は着衣量と外界条件の設定処理であり、これらの値は顕熱伝達率および潜熱伝達率に影響するため、35~85 行に定義した updateSkinHeatConductance メソッドで値を更新する。52~76 行で SET*の計算と同様の方法で着衣温度を収束計算する。ただし、各部位の対流熱伝達率と放射熱伝達率は式 27.8 と式 27.74 を用いて更新する。両者が求まれば式 27.6 と式 27.10 を用いて顕熱伝達率と潜熱伝達率を計算できる。

プログラム 27.16　境界条件設定処理

Popolo.HumanBody.MultiNodeModel.bodyPart class

```
 1 /// <summary>物体に接触させる</summary>
 2 /// <param name="temperature">物体の温度[C]</param>
 3 /// <param name="heatConductance">物体への熱コンダクタンス[W/K]</param>
 4 /// <param name="contactPortionRate">接触比率[-]</param>
 5 internal void contact(double temperature, double heatConductance, double contactPortionRate)
 6 {
 7   this.materialTemperature = temperature;
 8   this.hConductance[6] = heatConductance;
 9   this.contactPortionRate = contactPortionRate;
10 }
11
12 /// <summary>着衣量[clo]を設定する</summary>
13 /// <param name="clothingIndex">着衣量[clo]</param>
14 internal void setClothingIndex(double clothingIndex)
15 {
16   this.clothingIndex = clothingIndex;
17   updateSkinHeatConductance();
18 }
19
20 /// <summary>境界条件を設定する</summary>
21 /// <param name="velocity">気流速度[m/s]</param>
22 /// <param name="meanRadiantTemperature">平均放射温度[C]</param>
23 /// <param name="drybulbTemperature">乾球温度[C]</param>
24 /// <param name="waterVaporPressure">水蒸気分圧[kPa]</param>
25 internal void updateBoundary
26   (double velocity, double meanRadiantTemperature,double drybulbTemperature, double waterVaporPressure)
27 {
28   this.velocity = velocity;
29   this.meanRadiantTemperature = meanRadiantTemperature;
30   this.drybulbTemperature = drybulbTemperature;
31   this.waterVaporPressure = waterVaporPressure;
32   updateSkinHeatConductance();
33 }
34
35 /// <summary>皮膚表面の顕熱・潜熱伝達率を更新する</summary>
36 private void updateSkinHeatConductance()
37 {
38   //着衣面積率[-]の計算
39   double fcl = 1 + 0.25 * clothingIndex;
40   double rcl = 0.155 * clothingIndex;
41
42   double cht;
43   if (body.IsStanding) cht = cHTransferStand[node];
44   else cht = cHTransferSit[node];
45
46   //放射熱伝達率[W/(m2 K)]を更新する(衣服の表面温度を収束計算)
47   double vel = Math.Max(0.15, velocity);
48   double ra, eff;
49   clothTemperature = 30;
50   if (body.IsStanding) eff = 0.73;
51   else eff = 0.72;
52   while (true)
53   {
54     double ctOld = clothTemperature;
```

```
55     //放射熱伝達率[W/(m2K)]の計算
56     radiativeHeatTransferCoefficient = 4d * BLACK_CONSTANT * eff
57       * Math.Pow((clothTemperature + meanRadiantTemperature) / 2d + CONVERT_C_TO_K, 3);
58     //対流熱伝達率[W/(m2K)の計算]
59     if (initializing) convectiveHeatTransferCoefficient = cht;
60     else
61     {
62       convectiveHeatTransferCoefficient = cht * Math.Max(2.58 * Math.Sqrt(vel),
63         (clothTemperature - drybulbTemperature) / (setPoint_Skin - 28.8));
64     }
65     //総合熱伝達率[W/(m2K)]の計算
66     double hcr = radiativeHeatTransferCoefficient + convectiveHeatTransferCoefficient;
67     //空気層顕熱抵抗[(m2K)/W]の計算
68     ra = 1 / (fcl * hcr);
69     //作用温度[C]の計算
70     operatingTemperature = (radiativeHeatTransferCoefficient * meanRadiantTemperature
71       + convectiveHeatTransferCoefficient * drybulbTemperature) / hcr;
72     //衣服温度[C]の計算
73     clothTemperature = (ra * temperatures[Layer.Skin] + rcl * operatingTemperature) / (ra + rcl);
74     //衣服温度の更新量が 0.01C 以下で収束と判定
75     if (Math.Abs(ctOld - clothTemperature) < 0.01) break;
76   }
77
78   //顕熱伝達率[W/K]の計算
79   hConductance[7] = surfaceArea / (rcl + 1d / (fcl
80     * (radiativeHeatTransferCoefficient + convectiveHeatTransferCoefficient)));
81
82   //潜熱伝達率[W/K]の計算
83   double lewis = 0.0555 * (temperatures[Layer.Skin] + CONVERT_C_TO_K);
84   hConductance[8] = surfaceArea * lewis / (1 / (fcl * convectiveHeatTransferCoefficient) + rcl / I_CLS);
85 }
```

プログラム 27.17 に行列設定処理を示す。図 27.8 の行列に対して、各部位の要素を設定する処理である。7 行で部位に応じたオフセット（プログラム 27.11 参照）を取得する。11~61 行は部位内で完結する熱移動に関する要素であり、表 27.14 で示した行列 *BM* である。63~77 行は他の部位からの熱流に関する要素であり、表 27.15 で示したベクトル *ZV* の下線部である。79~95 行は表 27.15 のベクトル *ZV* から下線部を除いた内容である。

プログラム 27.17　行列設定処理

Popolo.HumanBody.MultiNodeModel.bodyPart class

```
 1 /// <summary>行列の要素を設定する</summary>
 2 /// <param name="bMatrix">B 行列</param>
 3 /// <param name="zVector">Z ベクトル</param>
 4 internal void makeMatrix(IMatrix bMatrix, IVector zVector)
 5 {
 6   double tStep = body.TimeStep;
 7   int os = mOffset[node]; //行列のオフセット
 8
 9   //BM 行列を生成
10   //核
11   bMatrix[os + 0, os + 0] = heatCapacity[Layer.Core] / tStep
12     + BLD_SPECIFICHEAT * bloodFlow[Layer.Core] + 2d * hConductance[3] + hConductance[0];
13   bMatrix[os + 0, os + 1] = -hConductance[0];
14   bMatrix[os + 0, os + 4] = -BLD_SPECIFICHEAT * bloodFlow[Layer.Core] - hConductance[3];
15   bMatrix[os + 0, os + 5] = -hConductance[3];
16   //筋肉
17   bMatrix[os + 1, os + 0] = -hConductance[0];
18   bMatrix[os + 1, os + 1] = heatCapacity[Layer.Muscle] / tStep
19     + BLD_SPECIFICHEAT * bloodFlow[Layer.Muscle] + hConductance[0] + hConductance[1];
20   bMatrix[os + 1, os + 2] = -hConductance[1];
21   bMatrix[os + 1, os + 4] = -BLD_SPECIFICHEAT * bloodFlow[Layer.Muscle];
22   //脂肪
23   bMatrix[os + 2, os + 1] = -hConductance[1];
24   bMatrix[os + 2, os + 2] = heatCapacity[Layer.Fat] / tStep
25     + BLD_SPECIFICHEAT * bloodFlow[Layer.Fat] + hConductance[1] + hConductance[2];
26   bMatrix[os + 2, os + 3] = -hConductance[2];
27   bMatrix[os + 2, os + 4] = -BLD_SPECIFICHEAT * bloodFlow[Layer.Fat];
28   //皮膚
29   bMatrix[os + 3, os + 2] = -hConductance[2];
30   bMatrix[os + 3, os + 3] = heatCapacity[Layer.Skin] / tStep
31     + BLD_SPECIFICHEAT * bloodFlow[Layer.Skin] + hConductance[2]
32     + contactPortionRate * hConductance[6] + (1 - contactPortionRate) * hConductance[7];
33   if ((node & LIMBS) != 0) bMatrix[os + 3, os + 3] += hConductance[4];
34   bMatrix[os + 3, os + 4] = -BLD_SPECIFICHEAT * bloodFlow[Layer.Skin];
```

```
35  //動脈
36  double upperFlow = bloodFlow[Layer.DeepVein] + bloodFlow[Layer.SuperficialVein];
37  bMatrix[os + 4, os + 0] = -hConductance[3];
38  bMatrix[os + 4, os + 4] = heatCapacity[Layer.Artery] / tStep +
39    BLD_SPECIFICHEAT * upperFlow + hConductance[3] + hConductance[5];
40  bMatrix[os + 4, os + 5] = -hConductance[5];
41  //静脈
42  bMatrix[os + 5, os + 0] = -BLD_SPECIFICHEAT * bloodFlow[Layer.Core] - hConductance[3];
43  bMatrix[os + 5, os + 1] = -BLD_SPECIFICHEAT * bloodFlow[Layer.Muscle];
44  bMatrix[os + 5, os + 2] = -BLD_SPECIFICHEAT * bloodFlow[Layer.Fat];
45  bMatrix[os + 5, os + 3] = -BLD_SPECIFICHEAT * bloodFlow[Layer.Skin];
46  bMatrix[os + 5, os + 4] = -hConductance[5];
47  bMatrix[os + 5, os + 5] = heatCapacity[Layer.DeepVein] / tStep
48    + BLD_SPECIFICHEAT * bloodFlow[Layer.DeepVein] + hConductance[3] + hConductance[5];
49  if (node == Node.Pelvis)
50    bMatrix[os + 5, os + 5] += BLD_SPECIFICHEAT * bloodFlow[Layer.SuperficialVein];
51  //四肢部位のみ（表在静脈）
52  if ((node & LIMBS) != 0)
53  {
54    bMatrix[os + 6, os + 3] = -hConductance[4];
55    if ((node & TERMINAL_NODE) != 0)
56      bMatrix[os + 6, os + 4] = -BLD_SPECIFICHEAT * bloodFlow[Layer.SuperficialVein];
57    bMatrix[os + 6, os + 6] = heatCapacity[Layer.SuperficialVein] / tStep +
58      BLD_SPECIFICHEAT * bloodFlow[Layer.SuperficialVein] + hConductance[4];
59
60    if ((node & LIMBS) != 0) bMatrix[os + 3, os + 6] = -hConductance[4];
61  }
62
63  //上流部位の動脈血流入
64  if (upperStreamPart != null)
65    bMatrix[os + 4, mOffset[upperStreamPart.node] + 4] = -BLD_SPECIFICHEAT * upperFlow;
66  else bMatrix[os + 4, 0] = -BLD_SPECIFICHEAT * upperFlow;
67
68  //下流部位の静脈血流入
69  foreach (bodyPart bp in downStreamParts)
70  {
71    //深部静脈
72    bMatrix[os + 5, mOffset[bp.node] + 5] = -BLD_SPECIFICHEAT * bp.bloodFlow[Layer.DeepVein];
73    //表在静脈
74    double downFlow = -BLD_SPECIFICHEAT * bp.bloodFlow[Layer.SuperficialVein];
75    if (node == Node.Pelvis) bMatrix[os + 5, mOffset[bp.node] + 6] += downFlow;
76    else bMatrix[os + 6, mOffset[bp.node] + 6] = downFlow;
77  }
78
79  //Zベクトルを生成
80  zVector[os + 0] = heatCapacity[Layer.Core] / tStep * temperatures[Layer.Core]
81    + basalMetabolicRate[Layer.Core];
82  zVector[os + 1] = heatCapacity[Layer.Muscle] / tStep * temperatures[Layer.Muscle]
83    + basalMetabolicRate[Layer.Muscle] + shiveringLoad + externalWork;
84  zVector[os + 2] = heatCapacity[Layer.Fat] / tStep * temperatures[Layer.Fat]
85    + basalMetabolicRate[Layer.Fat];
86  double wvSk = Water.GetSaturationPressure(temperatures[Layer.Skin]);
87  zVector[os + 3] = heatCapacity[Layer.Skin] / tStep * temperatures[Layer.Skin]
88    + basalMetabolicRate[Layer.Skin] + contactPortionRate * hConductance[6] * materialTemperature
89    + (1 - contactPortionRate) * (hConductance[7] * operatingTemperature) - latentHeatLoss;
90  zVector[os + 4] = heatCapacity[Layer.Artery] / tStep * temperatures[Layer.Artery];
91  zVector[os + 5] = heatCapacity[Layer.DeepVein] / tStep
92    * temperatures[Layer.DeepVein];
93  //四肢部位のみ（表在静脈）
94  if ((node & LIMBS) != 0) zVector[os + 6] =
95    heatCapacity[Layer.SuperficialVein] / tStep * temperatures[Layer.SuperficialVein];
96 }
```

プログラム 27.18 に bodyPart クラスのその他のメソッドを示す。1~45 行は制御の更新処理である。全身の状態から計算される制御値が引数として与えられるが、この計算法に関しては後述する。セットポイントの計算中は制御を停止させるが、この場合には 17~25 行が実行される。27~44 行は、発汗、ふるえ、血管収縮・拡張、AVA 血流の計算であり、それぞれ式 27.91、式 27.93、式 27.88、式 27.95 を用いて求める。

47~53 行は部位の接続処理である。血流の計算や他の部位からの熱移動の計算に用いるため、上流の部位と下流の部位を保存する。

55~80 行は血流の更新処理である。式 27.68 にもとづき、59, 60 行で筋肉層の血流を更新し、核、

筋肉、脂肪、皮膚の血流を合計する。この合計値と下流部位の血流とを足し合わせることで必要な血流を計算する。下流の血流は 66~71 行で計算するが、68 行では updateBloodFlow メソッドを再帰的に呼び出す。これにより末端までの血流が全て積算される。特に下腹部では左右大腿部の表在静脈が深部静脈に流れ込むため、75~79 行で血流の調整を行う。

プログラム 27.18　その他のメソッド

Popolo.HumanBody.MultiNodeModel.bodyPart class

```
 1 /// <summary>制御を更新する</summary>
 2 /// <param name="signal">制御信号</param>
 3 /// <param name="sweatSignal">発汗信号</param>
 4 /// <param name="shiveringSignal">ふるえ信号</param>
 5 /// <param name="vasodilatationSignal">血管拡張信号</param>
 6 /// <param name="vasoconstrictionSignal">血管収縮信号</param>
 7 /// <param name="avaRate">AVA 血流開度</param>
 8 internal void updateControl
 9   (double signal, double sweatSignal, double shiveringSignal,
10   double vasodilatationSignal, double vasoconstrictionSignal, double avaRate)
11 {
12   //最大蒸発熱損失[W]
13   double wvSk = Water.GetSaturationPressure(temperatures[Layer.Skin]);
14   double eMax = (1 - contactPortionRate) * hConductance[8] * (wvSk - waterVaporPressure);
15
16   //制御 OFF の場合
17   if (initializing)
18   {
19     evaporativeHeatLoss_Sweat = 0;
20     latentHeatLoss = eMax * 0.06;
21     shiveringLoad = 0;
22     bloodFlow[Layer.Skin] = basalBloodFlow_Skin;
23     bloodFlow[Layer.AVA] = 0;
24     return;
25   }
26
27   //係数計算
28   double dsp = temperatures[Layer.Skin] - setPoint_Skin;
29   double pow1 = sweatSignal * Math.Pow(2, dsp / 10d);
30   double pow2 = Math.Pow(2, dsp / 6d);
31
32   //蒸発熱損失[W]
33   evaporativeHeatLoss_Sweat = pow1 * sweatSignalR[node];
34   latentHeatLoss = eMax * Math.Min(0.85, 0.06 + 0.94 * evaporativeHeatLoss_Sweat / eMax);
35
36   //ふるえによる熱産生[W]
37   shiveringLoad = shiveringSignal * shivSignalR[node];
38
39   //皮膚血流量[mL/s]
40   bloodFlow[Layer.Skin] = (basalBloodFlow_Skin + dilSignalR[node]
41     * vasodilatationSignal) / (1 + strSignalR[node] * vasoconstrictionSignal) * pow2;
42
43   //AVA 血流量[mL/s]
44   bloodFlow[Layer.AVA] = maxAVA * Math.Max(0, Math.Min(1, avaRate));
45 }
46
47 /// <summary>体の部位を接続する</summary>
48 /// <param name="downStreamPart">接続先の体の部位</param>
49 internal void connect(bodyPart downStreamPart)
50 {
51   downStreamPart.upperStreamPart = this;
52   this.downStreamParts.Add(downStreamPart);
53 }
54
55 /// <summary>血流を更新する</summary>
56 internal void updateBloodFlow()
57 {
58   //血流の計算
59   bloodFlow[Layer.Muscle] = basalBloodFlow_Muscle + 0.239 * (externalWork + shiveringLoad);
60   double bfSum = bloodFlow[Layer.Core] +bloodFlow[Layer.Muscle] + bloodFlow[Layer.Fat] + bloodFlow[Layer.Skin];
61
62   //下流部位の血流を更新して静脈を計算
63   Layer dv = Layer.DeepVein;
64   Layer sv = Layer.SuperficialVein;
65   bloodFlow[dv] = bloodFlow[sv] = 0;
66   foreach (bodyPart bp in downStreamParts)
67   {
68     bp.updateBloodFlow();
69     bloodFlow[dv] += bp.bloodFlow[Layer.DeepVein];
```

```
 70     bloodFlow[sv] += bp.bloodFlow[Layer.SuperficialVein];
 71   }
 72   bloodFlow[Layer.Artery] = bloodFlow[dv] + bloodFlow[sv];
 73   bloodFlow[dv] += bfSum;
 74   bloodFlow[sv] += bloodFlow[Layer.AVA];
 75   if (node == Node.Pelvis)
 76   {
 77     bloodFlow[dv] += bloodFlow[sv];
 78     bloodFlow[sv] = 0;
 79   }
 80 }
 81
 82 /// <summary>制御信号を取得する</summary>
 83 /// <returns>制御信号</returns>
 84 internal double getSignal()
 85 { return (temperatures[Layer.Skin] - setPoint_Skin) * skinSignal[node]; }
 86
 87 /// <summary>皮膚からの顕熱損失[W]を計算する</summary>
 88 /// <returns>皮膚からの顕熱損失[W]</returns>
 89 internal double getSensibleHeatLoss()
 90 {
 91   return (temperatures[Layer.Skin] - operatingTemperature) * hConductance[7] * (1 - contactPortionRate)
 92     + (temperatures[Layer.Skin] - materialTemperature) * hConductance[6] * contactPortionRate;
 93 }
 94
 95 /// <summary>部位間の熱コンダクタンス[W/K]を取得する</summary>
 96 /// <param name="layer1">部位 1</param>
 97 /// <param name="layer2">部位 2</param>
 98 /// <returns>部位間の熱コンダクタンス[W/K]</returns>
 99 internal double getHeatConductance(Layer layer1, Layer layer2)
100 {
101   switch (layer1 | layer2)
102   {
103     case (Layer.Core | Layer.Muscle):
104       return hConductance[0];
105     case (Layer.Muscle | Layer.Fat):
106       return hConductance[1];
107     case (Layer.Fat | Layer.Skin):
108       return hConductance[2];
109     case (Layer.DeepVein | Layer.Core):
110     case (Layer.Artery | Layer.Core):
111       return hConductance[3];
112     case (Layer.SuperficialVein | Layer.Skin):
113       return hConductance[4];
114     case (Layer.Artery | Layer.DeepVein):
115       return hConductance[5];
116     default:
117       return 0;
118   }
119 }
120
121 /// <summary>体温[C]を更新する</summary>
122 /// <param name="tVector">体温ベクトル</param>
123 internal void updateTemperature(IVector tVector)
124 {
125   int os = mOffset[node];
126   temperatures[Layer.Core] = tVector[os];
127   temperatures[Layer.Muscle] = tVector[os + 1];
128   temperatures[Layer.Fat] = tVector[os + 2];
129   temperatures[Layer.Skin] = tVector[os + 3];
130   temperatures[Layer.Artery] = tVector[os + 4];
131   temperatures[Layer.DeepVein] = tVector[os + 5];
132   if ((node & LIMBS) != 0) temperatures[Layer.SuperficialVein] = tVector[os + 6];
133   updateSkinHeatConductance();
134 }
```

4）MultiNodeModel クラス

プログラム 27.19 にインスタンス変数とプロパティを示す。

プログラム 27.19　インスタンス変数・プロパティ

Popolo.HumanBody.MultiNodeModel class

```
 1 /// <summary>体の部位</summary>
 2 private Dictionary<Node, bodyPart> parts = new Dictionary<Node, bodyPart>();
 3
 4 /// <summary>筋肉の重量比[-]</summary>
 5 private Dictionary<Node, double> rMuscle = new Dictionary<Node, double>();
 6
 7 /// <summary>計算用行列</summary>
```

```
 8 private double[,] bMatrix = new double[115, 115];
 9
10 /// <summary>温度ベクトル</summary>
11 private double[] tVector = new double[115];
12
13 /// <summary>核の平均セットポイント[C]</summary>
14 private double averageCoreSetPoint;
15
16 /// <summary>皮膚の平均セットポイント[C]</summary>
17 private double averageSkinSetPoint;
18
19 /// <summary>置換ベクトル</summary>
20 private int[] permutaion = new int[115];
21
22 /// <summary>体重[kg]を取得する</summary>
23 public double Weight { get; private set; }
24
25 /// <summary>身長[m]を取得する</summary>
26 public double Height { get; private set; }
27
28 /// <summary>年齢[-]を取得する</summary>
29 public double Age { get; private set; }
30
31 /// <summary>男性か否か</summary>
32 public bool IsMale { get; private set; }
33
34 /// <summary>立位か否か</summary>
35 public bool IsStanding { get; private set; }
36
37 /// <summary>体脂肪率[-]を取得する</summary>
38 public double FatPercentage { get; private set; }
39
40 /// <summary>体表面積[m2]を取得する</summary>
41 public double SurfaceArea { get; private set; }
42
43 /// <summary>中央血液溜まりの温度[C]を取得する</summary>
44 public double CentralBloodTemperature { get; private set; }
45
46 /// <summary>呼吸による熱損失[W]を取得する</summary>
47 public double HeatLossByBreathing { get; private set; }
48
49 /// <summary>全身の基礎血流[mL/s]を取得する</summary>
50 public double BasalBloodFlow { get; private set; }
51
52 /// <summary>全身の血流[mL/s]を取得する</summary>
53 public double BloodFlow { get; private set; }
54
55 /// <summary>全身の基礎代謝[W]を取得する</summary>
56 public double BasalMetabolicRate { get; private set; }
57
58 /// <summary>全身の代謝量[W]を取得する</summary>
59 public double MetabolicRate { get; private set; }
60
61 /// <summary>計算時間間隔[sec]を取得する</summary>
62 public double TimeStep { get; private set; }
```

プログラム 27.20 にコンストラクタと初期化処理を示す。5~66 行がメインのコンストラクタであるが、1~3 行でオーバーロードしており、引数が無い場合には標準体躯で初期化する。式 27.12、式 27.67、式 27.62 を用いて 24 行、26 行、31 行で体表面積、心係数、基礎代謝をそれぞれ初期化する。37~44 行で bodyPart クラスを用いて 17 の部位のインスタンスを生成する。また、部位の筋肉重量を積算し、46 行で全身の筋肉重量比を計算する。この比率は外部仕事とふるえ熱産生を部位に配分する際に用いる。49~59 行は部位の接続処理である。62 行で体温を初期化した後、65 行でセットポイントを求める。

68~107 行はセットポイントの計算処理である。体温調節機能を OFF にし、PMV=0 となる条件下で定常状態まで（24 時間）計算する。定常状態での各部位の皮膚温（85~91 行）、体中心の温度（93~101 行）、全身の平均皮膚温度（103~106 行）をセットポイントとしてそれぞれ保存する。

プログラム **27.20**　コンストラクタ・初期化処理

Popolo.HumanBody.MultiNodeModel class

```
/// <summary>インスタンスを初期化する</summary>
/// <remarks>引数が無い場合には標準体躯とする</remarks>
public MultiNodeModel() : this(74.43, 1.72, 25, true, 0.15, true) { }

/// <summary>インスタンスを初期化する</summary>
/// <param name="weight">体重[kg]</param>
/// <param name="height">身長[m]</param>
/// <param name="age">年齢</param>
/// <param name="isMale">男性か否か</param>
/// <param name="fatPercentage">体脂肪率[%]</param>
/// <param name="isStanding">立位か否か</param>
public MultiNodeModel
  (double weight, double height, double age, bool isMale, double fatPercentage, bool isStanding)
{
  //情報保存
  Weight = weight;
  Height = height;
  Age = age;
  IsMale = isMale;
  IsStanding = isStanding;
  FatPercentage = fatPercentage;

  //体表面積[m2]を計算
  SurfaceArea = 0.202 * Math.Pow(Weight, 0.425) * Math.Pow(Height, 0.725);
  //心係数[(mL/s)/m2]を計算
  double ci = 115 + Age * (-2.4 + Age * (0.0167 + Age * (3.56e-4 - 4.29e-6 * Age)));
  //基礎血流[mL/s]を計算
  BasalBloodFlow = ci * SurfaceArea;

  //基礎代謝[W]を計算
  BasalMetabolicRate = (0.1004 + 1.97 * height + 0.0469 * weight - 0.005 * age);
  if (isMale) BasalMetabolicRate = (BasalMetabolicRate - 0.4925) / 0.0864;
  else BasalMetabolicRate = (BasalMetabolicRate - 0.4925 * 2) / 0.0864;

  //部位を作成
  double muscleSum = 0;
  Node[] nds = (Node[])Enum.GetValues(typeof(Node));
  foreach (Node nd in nds)
  {
    bodyPart bp = new bodyPart
      (this, nd, Weight, Height, SurfaceArea, fatPercentage, BasalMetabolicRate, BasalBloodFlow);
    parts.Add(nd, bp);
    muscleSum += bp.muscleWeight;
  }
  //筋肉の重量比を計算
  foreach (Node nd in parts.Keys) rMuscle[nd] = parts[nd].muscleWeight / muscleSum;

  //部位を接続
  parts[Node.Neck].connect(parts[Node.Head]);
  parts[Node.Pelvis].connect(parts[Node.LeftThigh]);
  parts[Node.Pelvis].connect(parts[Node.RightThigh]);
  parts[Node.LeftThigh].connect(parts[Node.LeftLeg]);
  parts[Node.RightThigh].connect(parts[Node.RightLeg]);
  parts[Node.LeftLeg].connect(parts[Node.LeftFoot]);
  parts[Node.RightLeg].connect(parts[Node.RightFoot]);
  parts[Node.LeftShoulder].connect(parts[Node.LeftArm]);
  parts[Node.RightShoulder].connect(parts[Node.RightArm]);
  parts[Node.LeftArm].connect(parts[Node.LeftHand]);
  parts[Node.RightArm].connect(parts[Node.RightHand]);

  //体温を初期化
  InitializeTemperature(36);

  //セットポイント初期化
  initializeSetPoint();
}

/// <summary>セットポイントを初期化する</summary>
private void initializeSetPoint()
{
  //制御を OFF
  foreach (Node nd in parts.Keys)
  {
    parts[nd].initializing = true;
    parts[nd].setClothingIndex(0);
  }

  //PMV=0 となる境界条件を設定
```

```
 79   UpdateBoundary(0, 28.8, 28.8, 50);
 80   SetMetabolicRate(1);
 81
 82   //定常状態まで計算 (24 時間)
 83   for (int i = 0; i < 24; i++) Update(3600);
 84
 85   //セットポイント設定
 86   foreach (Node nd in parts.Keys)
 87   {
 88     parts[nd].initializing = false;
 89     parts[nd].setPoint_Skin = parts[nd].temperatures[Layer.Skin];
 90     parts[nd].setPoint_Core = parts[nd].temperatures[Layer.Core];
 91   }
 92
 93   //平均セットポイントを作成
 94   //体中心の核
 95   Layer core = Layer.Core;
 96   double capSum = parts[Node.Chest].heatCapacity[core]
 97     + parts[Node.Pelvis].heatCapacity[core] + parts[Node.Back].heatCapacity[core];
 98   averageCoreSetPoint =
 99     (parts[Node.Chest].setPoint_Core * parts[Node.Chest].heatCapacity[core]
100     + parts[Node.Pelvis].setPoint_Core * parts[Node.Pelvis].heatCapacity[core]
101     + parts[Node.Back].setPoint_Core * parts[Node.Back].heatCapacity[core]) / capSum;
102
103   //全身の皮膚
104   averageSkinSetPoint = 0;
105   foreach (Node bp in parts.Keys) averageSkinSetPoint += parts[bp].setPoint_Skin * parts[bp].surfaceArea;
106   averageSkinSetPoint /= SurfaceArea;
107 }
108
109 /// <summary>体温を初期化する</summary>
110 /// <param name="temperature">初期化する温度[C]</param>
111 public void InitializeTemperature(double temperature)
112 {
113   //行列に設定
114   for (int i = 0; i < tVector.Length; i++) tVector[i] = temperature;
115   //各部位への設定処理
116   CentralBloodTemperature = temperature;
117   foreach (Node nd in parts.Keys) parts[nd].updateTemperature(tVector);
118 }
```

プログラム 27.21 に状態更新処理を示す。引数はタイムステップである。後退差分のため計算が発散する可能性は小さいが、制御系は前タイムステップの状態値にもとづいて計算されるため、体温がある程度安定するまではテイムステップを 60 sec 程度にすると良い。9 行で制御を更新した後、11~43 行で表 27.14 の BM 行列を作成し、46 行で連立方程式を解いて 49, 50 行で結果を保存する。15~23 行では各部位のインスタンスメソッド（プログラム 27.17 参照）を用いて BM 行列を更新する。31~42 行は式 27.99 で示した中央血液溜まりに関する項である。

53~123 行は制御の更新処理である。56~69 行でセットポイントと体温の差を取得し、71~77 行で式 27.86、式 27.87、式 27.90、式 27.92 を用いて制御信号を計算する。79~94 行は式 27.94 にもとづく AVA 血流の計算である。96~104 行で各部位に制御信号を送り、制御を更新（プログラム 27.18 参照）する。106~114 行は血流更新処理であり、中央血液溜まりに直接に接続された部位の updateBloodFlow メソッドを呼び出す。116~122 行で全身の代謝を積算する。

プログラム 27.21　状態更新処理

Popolo.HumanBody.MultiNodeModel class

```
 1 /// <summary>状態を更新する</summary>
 2 /// <param name="timeStep">計算間隔[sec]</param>
 3 public void Update(double timeStep)
 4 {
 5   this.TimeStep = timeStep;
 6   IVector zVector = new Vector(115);
 7
 8   //制御を更新
 9   updateControl();
10
11   //計算用疎行列を用意
```

```
12   SparseMatrix bMatrix = new SparseMatrix(115, 115);
13
14   double mr = 0;
15   foreach (Node nd in parts.Keys)
16   {
17     //行列に各部位の項を設定
18     parts[nd].makeMatrix(bMatrix, zVector);
19     //代謝量を積算
20     mr += parts[nd].shiveringLoad + parts[nd].externalWork
21       + parts[nd].basalMetabolicRate[Layer.Core] + parts[nd].basalMetabolicRate[Layer.Muscle]
22       + parts[nd].basalMetabolicRate[Layer.Fat] + parts[nd].basalMetabolicRate[Layer.Skin];
23   }
24
25   //胸部の呼吸の項を追加
26   bodyPart head = parts[Node.Head];
27   HeatLossByBreathing = mr * (0.0014 * (34 - head.drybulbTemperature)
28     + 0.017251 * (5.8662 - head.waterVaporPressure));
29   zVector[13] -= HeatLossByBreathing;
30
31   //中央血液溜まりの項を追加
32   bMatrix[0, 0] = parts[Node.Chest].centralBloodHeatCapacity / timeStep;
33   bodyPart[] bps = new bodyPart[] { parts[Node.Neck], parts[Node.LeftShoulder],
34     parts[Node.RightShoulder], parts[Node.Pelvis], parts[Node.Chest], parts[Node.Back]};
35   for (int i = 0; i < bps.Length; i++)
36   {
37     double dvf = BLD_SPECIFICHEAT * bps[i].bloodFlow[Layer.DeepVein];
38     double svf = BLD_SPECIFICHEAT * bps[i].bloodFlow[Layer.SuperficialVein];
39     bMatrix[0, 0] += (dvf + svf);
40     bMatrix[0, bodyPart.mOffset[bps[i].node] + 5] = -dvf;
41     bMatrix[0, bodyPart.mOffset[bps[i].node] + 6] = -svf;
42   }
43   zVector[0] = parts[Node.Chest].centralBloodHeatCapacity / timeStep * CentralBloodTemperature;
44
45   //連立代数方程式を解く
46   bMatrix.SolveLinearEquation(zVector, ref tVector);
47
48   //設定
49   foreach (Node nd in parts.Keys) parts[nd].updateTemperature(tVector);
50   CentralBloodTemperature = tVector[0];
51 }
52
53 /// <summary>制御信号を計算する</summary>
54 private void updateControl()
55 {
56   //全身の温冷感信号の計算
57   double cldSignal, wrmSignal;
58   cldSignal = wrmSignal = 0;
59   foreach (Node bp in parts.Keys)
60   {
61     double sig = parts[bp].getSignal();
62     if (0 < sig) wrmSignal += sig;
63     else cldSignal += sig;
64   }
65   double signal = wrmSignal + cldSignal;
66
67   //頭部の核の温冷感信号
68   Layer core = Layer.Core;
69   double sHead = parts[Node.Head].temperatures[core] - parts[Node.Head].setPoint_Core;
70
71   //発汗・ふるえ・血管収縮・血管拡張の信号を計算
72   double sfRate = SurfaceArea / STANDARD_SURFACE_AREA;
73   double sweatSignal = Math.Max(0, (371.2 * sHead + 33.64 * signal)) * sfRate;
74   double shiveringSignal = (-24.36 * Math.Max(0, -sHead) * cldSignal) * sfRate;
75   double vasoconstrictionSignal = Math.Max(0, -10.8 * sHead - 10.8 * signal);
76   double vasodilatationSignal = Math.Max(0, 32.5 * sHead + 2.08 * signal);
77   vasodilatationSignal *= BasalBloodFlow / STANDARD_BLOOD_FLOW;
78
79   //AVA血流開度信号を計算
80   //体中心（腰・胸・背中）の平均温度を計算
81   double capSum = parts[Node.Chest].heatCapacity[core]
82   + parts[Node.Pelvis].heatCapacity[core] + parts[Node.Back].heatCapacity[core];
83   double aveCore = 0;
84   aveCore += parts[Node.Chest].temperatures[core] * parts[Node.Chest].heatCapacity[core];
85   aveCore += parts[Node.Pelvis].temperatures[core] * parts[Node.Pelvis].heatCapacity[core];
86   aveCore += parts[Node.Back].temperatures[core] * parts[Node.Back].heatCapacity[core];
87   aveCore /= capSum;
88   //全身の平均皮膚温を計算
89   double atSkin = GetAverageSkinTemperature();
90   //AVA開度の計算
91   double ovaHand = 0.265 * (atSkin - (averageSkinSetPoint - 0.43))
```

```
 92   + 0.953 * (aveCore - (averageCoreSetPoint - 0.1905)) + 0.9126;
 93   double ovaFoot = 0.265 * (atSkin - (averageSkinSetPoint - 0.97))
 94   + 0.953 * (aveCore - (averageCoreSetPoint + 0.0095)) + 0.9126;
 95
 96   //皮膚血管運動・発汗・ふるえ熱生産・AVA 血流を計算
 97   foreach (Node bp in parts.Keys)
 98   {
 99     if ((bp == Node.LeftHand) || (bp == Node.RightHand))
100       parts[bp].updateControl
101         (signal, sweatSignal, shiveringSignal, vasodilatationSignal, vasoconstrictionSignal, ovaHand);
102     else parts[bp].updateControl
103       (signal, sweatSignal, shiveringSignal, vasodilatationSignal, vasoconstrictionSignal, ovaFoot);
104   }
105
106   //血流更新処理//中央血液溜まり直結部以外はメソッド内で再帰的に呼び出し
107   BloodFlow = 0;
108   Node[] nds =
109     new Node[] { Node.Neck, Node.Chest, Node.Back, Node.Pelvis, Node.LeftShoulder, Node.RightShoulder };
110   for (int i = 0; i < nds.Length; i++)
111   {
112     parts[nds[i]].updateBloodFlow();
113     BloodFlow += parts[nds[i]].bloodFlow[Layer.DeepVein] + parts[nds[i]].bloodFlow[Layer.SuperficialVein];
114   }
115
116   //全身代謝量[W]を更新
117   MetabolicRate = BasalMetabolicRate;
118   foreach (Node nd in parts.Keys)
119   {
120     bodyPart part = parts[nd];
121     MetabolicRate += part.externalWork + part.shiveringLoad;
122   }
123 }
```

プログラム 27.22 に境界条件設定処理を示す。1~7 行は代謝量の設定処理である。基礎代謝を減じた値を外部仕事とし、筋肉の重量比に従って各部位に設定する。9~50 行は外界条件、着衣量、物体接触の設定であり、bodyPart インスタンスのメソッドを呼び出して処理を行う。

プログラム 27.22　境界条件設定処理

Popolo.HumanBody.MultiNodeModel class

```
 1 /// <summary>代謝量[met]を設定する</summary>
 2 /// <param name="met">代謝量[met]</param>
 3 public void SetMetabolicRate(double met)
 4 {
 5   met = Math.Max(0, CONVERT_MET_TO_W * met * SurfaceArea - BasalMetabolicRate);
 6   foreach (Node bp in parts.Keys) parts[bp].externalWork = rMuscle[bp] * met;
 7 }
 8
 9 /// <summary>境界条件を更新する</summary>
10 /// <param name="velocity">気流速度[m/s]</param>
11 /// <param name="meanRadiantTemperature">平均放射温度[C]</param>
12 /// <param name="drybulbTemperature">乾球温度[C]</param>
13 /// <param name="relativeHumidity">相対湿度[%]</param>
14 public void UpdateBoundary
15   (double velocity, double meanRadiantTemperature, double drybulbTemperature, double relativeHumidity)
16 {
17   double hr = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
18     (drybulbTemperature, relativeHumidity, ATMOSPHERIC_PRESSURE);
19   double wvp = MoistAir.GetWaterVaporPartialPressureFromHumidityRatio(hr, ATMOSPHERIC_PRESSURE);
20   foreach (Node bp in parts.Keys)
21     parts[bp].updateBoundary(velocity, meanRadiantTemperature, drybulbTemperature, wvp);
22 }
23
24 /// <summary>境界条件を更新する</summary>
25 /// <param name="node">更新する部位</param>
26 /// <param name="velocity">気流速度[m/s]</param>
27 /// <param name="meanRadiantTemperature">平均放射温度[C]</param>
28 /// <param name="drybulbTemperature">乾球温度[C]</param>
29 /// <param name="relativeHumidity">相対湿度[%]</param>
30 public void UpdateBoundary
31 (Node node, double velocity, double meanRadiantTemperature, double drybulbTemperature, double relativeHumidity)
32 {
33   double hr = MoistAir.GetHumidityRatioFromDryBulbTemperatureAndRelativeHumidity
34     (drybulbTemperature, relativeHumidity, ATMOSPHERIC_PRESSURE);
35   double wvp = MoistAir.GetWaterVaporPartialPressureFromHumidityRatio(hr, ATMOSPHERIC_PRESSURE);
36   parts[node].updateBoundary(velocity, meanRadiantTemperature, drybulbTemperature, wvp);
37 }
38
```

```
39 /// <summary>着衣量[clo]を設定する</summary>
40 /// <param name="node">部位</param>
41 /// <param name="clo">着衣量[clo]</param>
42 public void SetClothingIndex(Node node, double clo) { parts[node].setClothingIndex(clo); }
43
44 /// <summary>物体に接触させる</summary>
45 /// <param name="node">部位</param>
46 /// <param name="temperature">物体の温度[C]</param>
47 /// <param name="heatConductance">物体への熱コンダクタンス[W/K]</param>
48 /// <param name="contactPortionRate">接触比率[-]</param>
49 public void Contact(Node node, double temperature, double heatConductance, double contactPortionRate)
50 { parts[node].contact(temperature, heatConductance, contactPortionRate); }
```

プログラム 27.23 に情報取得処理を示す。

プログラム 27.23　情報取得処理

Popolo.HumanBody.MultiNodeModel class

```
 1 /// <summary>気流速度[m/s]を取得する</summary>
 2 /// <param name="node">体の部位</param>
 3 /// <returns>気流速度[m/s]</returns>
 4 public double GetVelocity(Node node) { return parts[node].velocity; }
 5
 6 /// <summary>平均放射温度[C]を取得する</summary>
 7 /// <param name="node">体の部位</param>
 8 /// <returns>平均放射温度[C]</returns>
 9 public double GetMeanRadiantTemperature(Node node) { return parts[node].meanRadiantTemperature; }
10
11 /// <summary>乾球温度[C]を取得する</summary>
12 /// <param name="node">体の部位</param>
13 /// <returns>乾球温度[C]</returns>
14 public double GetDrybulbTemperature(Node node) { return parts[node].drybulbTemperature; }
15
16 /// <summary>相当温度[C]を取得する</summary>
17 /// <param name="node">体の部位</param>
18 /// <returns>相当温度[C]</returns>
19 public double GetOperatingTemperature(Node node) { return parts[node].operatingTemperature; }
20
21 /// <summary>相対湿度[%]を取得する</summary>
22 /// <param name="node">体の部位</param>
23 /// <returns>相対湿度[%]</returns>
24 public double GetRelativeHumidity(Node node)
25 {
26   double hr = MoistAir.GetHumidityRatioFromWaterVaporPartialPressure
27     (parts[node].waterVaporPressure, ATMOSPHERIC_PRESSURE);
28   return MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
29     (parts[node].drybulbTemperature, hr, ATMOSPHERIC_PRESSURE);
30 }
31
32 /// <summary>着衣量[clo]を取得する</summary>
33 /// <param name="node">体の部位</param>
34 /// <returns>着衣量[clo]</returns>
35 public double GetClothingIndex(Node node) { return parts[node].clothingIndex; }
36
37 /// <summary>組織間の熱コンダクタンス[W/K]を取得する</summary>
38 /// <param name="node">部位</param>
39 /// <param name="layer1">組織 1</param>
40 /// <param name="layer2">組織 2</param>
41 /// <returns>組織間の熱コンダクタンス[W/K]</returns>
42 public double GetHeatConductance(Node node, Layer layer1, Layer layer2)
43 { return parts[node].getHeatConductance(layer1, layer2); }
44
45 /// <summary>代謝量[W]を取得する</summary>
46 /// <returns>代謝量[W]</returns>
47 public double GetMetabolicRate(Node node, Layer layer)
48 {
49   double bm = parts[node].basalMetabolicRate[layer];
50   if (layer != Layer.Muscle) return bm;
51   else return bm + parts[node].externalWork + parts[node].shiveringLoad;
52 }
53
54 /// <summary>体温[C]を取得する</summary>
55 /// <param name="node">体の部位</param>
56 /// <param name="layer">層</param>
57 /// <returns>体温[C]</returns>
58 public double GetTemperature(Node node, Layer layer) { return parts[node].temperatures[layer]; }
59
60 /// <summary>血流[mL/s]を取得する</summary>
61 /// <param name="node">体の部位</param>
62 /// <param name="layer">層</param>
63 /// <returns>血流[mL/s]</returns>
```

```
64 public double GetBloodFlow(Node node, Layer layer) { return parts[node].bloodFlow[layer]; }
65
66 /// <summary>平均皮膚温[C]を取得する</summary>
67 /// <returns>平均皮膚温[C]</returns>
68 public double GetAverageSkinTemperature()
69 {
70   double ctSum = 0;
71   foreach (Node bp in parts.Keys)
72   {
73     bodyPart bPart = parts[bp];
74     ctSum += bPart.temperatures[Layer.Skin] * bPart.surfaceArea;
75   }
76   return ctSum / SurfaceArea;
77 }
78
79 /// <summary>皮膚表面からの顕熱損失[W]を取得する</summary>
80 /// <param name="node">部位</param>
81 /// <returns>皮膚表面からの顕熱損失[W]</returns>
82 public double GetSensibleHeatLoss(Node node) { return parts[node].getSensibleHeatLoss(); }
83
84 /// <summary>皮膚表面からの潜熱損失[W]を取得する</summary>
85 /// <param name="node">部位</param>
86 /// <returns>皮膚表面からの潜熱損失[W]</returns>
87 public double GetLatentHeatLoss(Node node) { return parts[node].latentHeatLoss; }
```

【例題 27.4】

人体の各部位各組織の温度の推移を計算するプログラムを作成せよ。

【解】

計算処理をプログラム 27.24 に示す。全身の相当温度を 28.8 ℃ とした場合と左大腿部のみを 35 ℃ とした場合の最終的な熱平衡状態をそれぞれ図 27.9 と図 27.10 に示す[†1]。

プログラム 27.24　人体体温計算処理

```
 1 private static void MultiNodeModelTest(double operatingTemperature)
 2 {
 3   //人体モデル作成
 4   MultiNodeModel mn = new MultiNodeModel();
 5   mn.InitializeTemperature(35);
 6   mn.UpdateBoundary(0, operatingTemperature, operatingTemperature, 50);
 7
 8   //四肢部位
 9   const MultiNodeModel.Node LIMBS =
10     MultiNodeModel.Node.LeftHand | MultiNodeModel.Node.RightHand |
11     MultiNodeModel.Node.LeftFoot | MultiNodeModel.Node.RightFoot |
12     MultiNodeModel.Node.LeftShoulder | MultiNodeModel.Node.RightShoulder |
13     MultiNodeModel.Node.LeftArm | MultiNodeModel.Node.RightArm |
14     MultiNodeModel.Node.LeftThigh | MultiNodeModel.Node.RightThigh |
15     MultiNodeModel.Node.LeftLeg | MultiNodeModel.Node.RightLeg;
16
17   using (StreamWriter sWriter = new StreamWriter("MultiNodeTest.csv", false,Encoding.GetEncoding("Shift_JIS")))
18   {
19     //タイトル行
20     sWriter.Write("中央血液溜まり,");
21     foreach (MultiNodeModel.Node nd in Enum.GetValues(typeof(MultiNodeModel.Node)))
22       sWriter.Write(nd.ToString() + "核,筋肉,脂肪,皮膚,動脈,静脈,表在静脈,");
23     sWriter.WriteLine();
24
25     //時系列データ書き出し
26     for (int i = 0; i < 600; i++)
27     {
28       sWriter.Write(mn.CentralBloodTemperature + ",");
29       foreach (MultiNodeModel.Node nd in Enum.GetValues(typeof(MultiNodeModel.Node)))
30       {
31         sWriter.Write(mn.GetTemperature(nd, MultiNodeModel.Layer.Core) + ",");
32         sWriter.Write(mn.GetTemperature(nd, MultiNodeModel.Layer.Muscle) + ",");
33         sWriter.Write(mn.GetTemperature(nd, MultiNodeModel.Layer.Fat) + ",");
34         sWriter.Write(mn.GetTemperature(nd, MultiNodeModel.Layer.Skin) + ",");
35         sWriter.Write(mn.GetTemperature(nd, MultiNodeModel.Layer.Artery) + ",");
36         sWriter.Write(mn.GetTemperature(nd, MultiNodeModel.Layer.DeepVein) + ",");
37         if ((nd & LIMBS) == 0) sWriter.Write("0,");
38         else sWriter.Write(mn.GetTemperature(nd, MultiNodeModel.Layer.SuperficialVein) + ",");
39       }
40       sWriter.WriteLine("");
41       mn.Update(60);  //状態更新
42     }
43   }
44 }
```

†1　非定常人体モデルはかなり複雑なプログラムであるため、相当なデバッグ作業が予想される。その際に重要な手がかりになるのは図 27.9 下に示したような熱収支図である。熱収支の異常が把握できればバグの所在にも見当がつく。

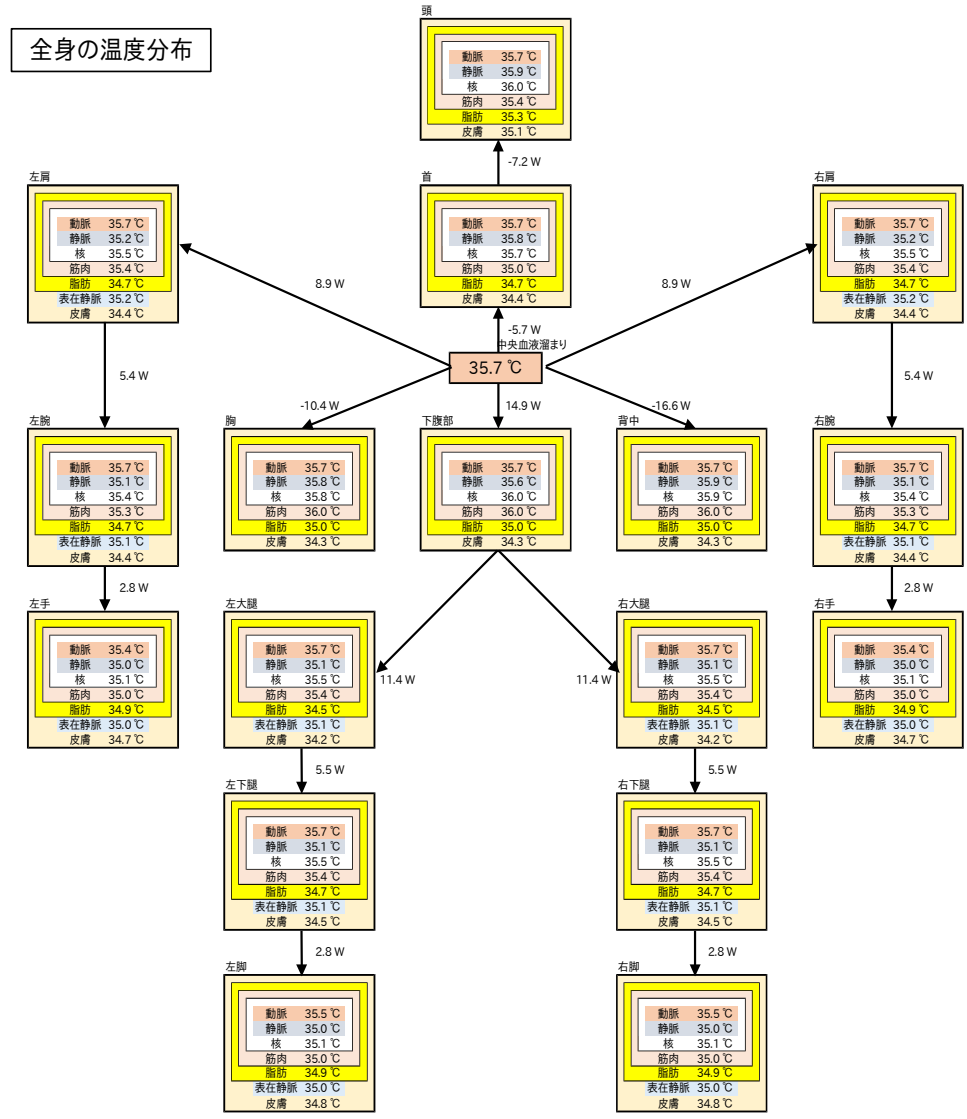


図 27.9　全身と代表部位の温度分布・熱流（28.8 °C / 50 %条件）

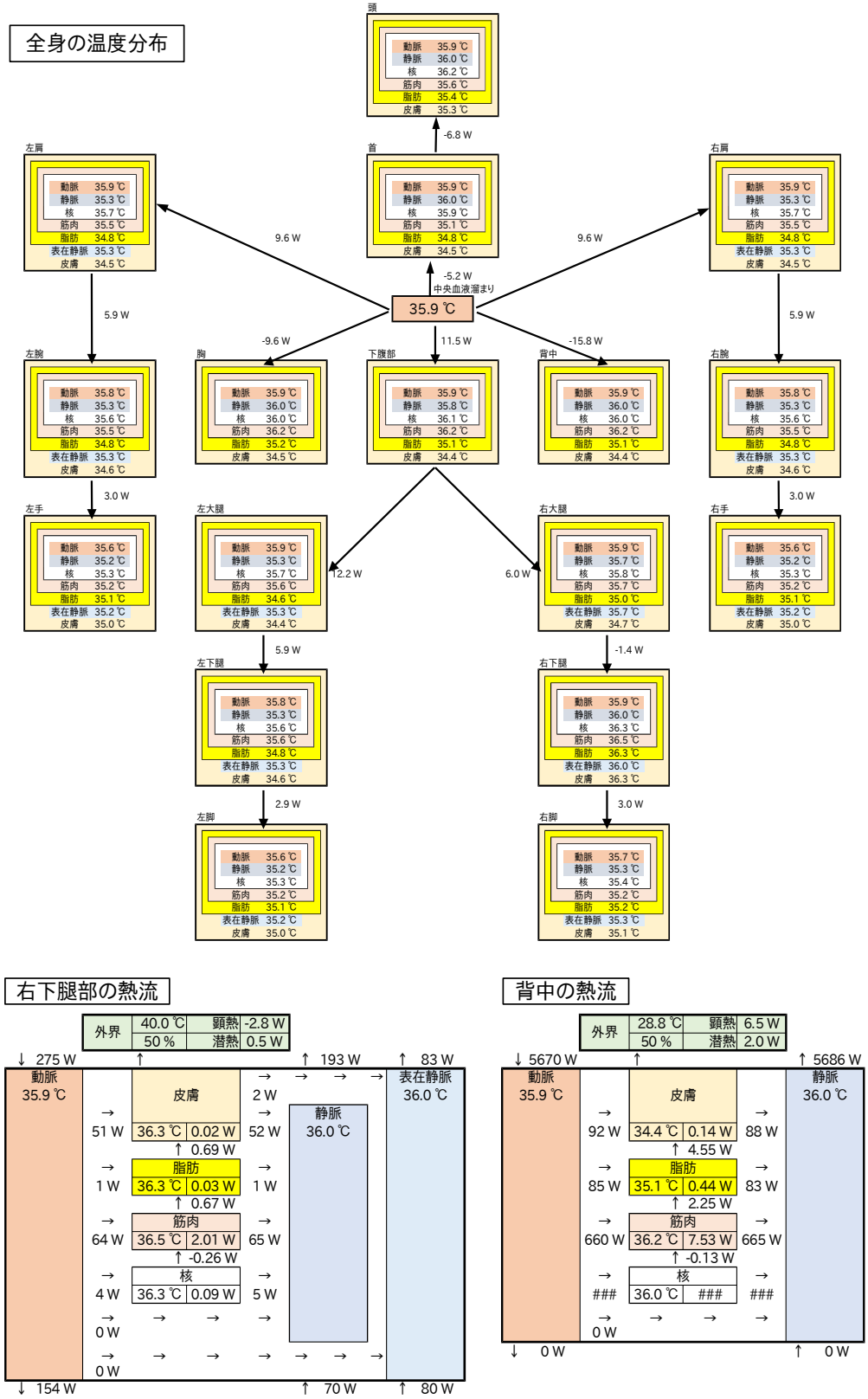


図 27.10　全身と代表部位の温度分布・熱流（右下腿部のみ 40 °C 条件）

【第 27 章 記号表】

A_{du} ：体表面積 **[m²]**
Age ：年齢 **[歳]**
O_{AVA} ：**AVA** 血管の開度 **[-]**
BF ：血流 **[mL/(m²·s)]**
C ：係数
C ：熱容量 **[kJ/K]**
C_{cl}, C_{clr} ：着衣からの対流熱損失 **[W], [W/m²]**
C_{dW} ：水分コンダクタンス **[kg/(s·kPa·m²)]**
cf ：物体に接触する皮膚面積割合 **[-]**
CI ：心係数 **[(mL/s)/m²]**
Cp ：比熱 **[kJ/(kg·K)]**
C_{res}, C_{resr} ：呼吸による顕熱損失 **[W], [W/m²]**
E_b ：不感蒸泄による熱損失量 **[W/m²]**
E_{max} ：皮膚表面の最大潜熱損失 **[W], [W/m²]**
E_{res}, E_{resr} ：呼吸による潜熱損失 **[W], [W/m²]**
E_{sw}, E_{swr} ：発汗による潜熱損失 **[W], [W/m²]**
E_{sk}, E_{skr} ：皮膚からの潜熱損失 **[W], [W/m²]**
f_{cl} ：人体表面に対する衣服表面積比率 **[-]**
f_{eff} ：有効放射面積率 **[-]**
H ：身長 **[m]**
h' ：皮膚~外界への顕熱伝達率 **[W/(m²·K)]**
h_c ：対流熱伝達率 **[W/(m²·K)]**
h_e' ：皮膚~外界への潜熱伝達率 **[W/(m²·kPa)]**
h_r ：放射熱伝達率 **[W/(m²·K)]**
I_{clo} ：着衣量 **[clo]**
I_{cls} ：標準化着衣量 **[clo]**
i_{cl} ：衣服の透湿係数 **(0.45 kPa/K)**
K ：熱コンダクタンス **[W/K]**
L ：人体への負荷 **[W/m²]**
l ：円柱長さ **[m]**
L_e ：ルイス数 **[K/kPa]**
M, M_r ：代謝 **[W], [W/m²]**
M_b ：基礎代謝 **[W/m²]**
Met ：活動量 **[met] (1 met = 58.15 W/m²)**
M_{shv} ：ふるえによる熱産出 **[W/m²]**
m_{swr} ：発汗量 **[mg/(m²·s)]**
P_a ：水蒸気分圧 **[kPa]**
PMV ：予測平均温冷感申告値 **[-]**
PPD ：予測不満足者率 **[-]**
P_{Ws} ：飽和水蒸気圧 **[kPa]**
$P_{Ws,SET}$ ：温度 **SET** における飽和水蒸気圧 **[kPa]**
SET ：新標準有効温度 **[°C]**
Q_{cs} ：コアから皮膚への熱流 **[W/m²]**
R ：比率 **[%]**
r ：径 **[m]**
R_{cl}, R_{clr} ：着衣からの放射熱損失 **[W], [W/m²]**
SIG_{dil} ：血管拡張の制御信号
SIG_{rstr} ：血管収縮の制御信号
SIG_{sw} ：発汗の制御信号
SIG_{shv} ：ふるえの制御信号
T ：絶対温度 **[K]**
T_{mr} ：外界の平均放射温度 **[K]**
T_o ：外界の作用温度 **[K]**
v ：相対気流速度 **[m/s]**
V_{res} ：呼吸量 **[kg/s]**
w ：皮膚の濡れ率 **[-]**
w_b ：不感蒸泄による皮膚の濡れ率 **[-]**
w_{sw} ：発汗による皮膚の濡れ率 **[-]**
W_{ex}, W_{exr} ：外部仕事 **[W], [W/m²]**
Wt ：体重 **[kg]**
W ：絶対湿度 **[kg/kg]**
WK ：外部仕事 **[met]**
α ：コアと皮膚の重量比 **[-]**
γ_{35} ：**35 °C** の水の蒸発潜熱 **2,418 kJ/kg**
εf_{eff} ：放射率 **[-]**
λ ：熱伝導率 **[W/(m·K)]**
ρ ：比重量 **[kg/m³]**
ρCp_b ：血液の体積比熱 **(3.842 J/(mL·K))**
σ ：黒体の放射定数 **5.67×10^{-8} W/(m²·K⁴)**

sub scripts ：

sk ：皮膚
cl ：衣服
cr ：核
a ：外界
sv ：表在静脈
st ：標準体躯
AVA ：**AVA**
sp ：セットポイント
b ：熱的中立状態
r ：体表面積あたり
res ：呼気
ms ：筋肉
ft ：脂肪
ar ：動脈
ve ：静脈
Head ：頭部

【第 27 章 参考文献】

27.1） Ralph F. Goldman: Thermal Manikins, Their Origins and Role, 6th International Thermal Manikin and Modeling Meeting, pp.3-18, 2006

27.2） A.P. Gagge, Ph.D., A.P. Fobelets, L.G. Berglund, P.E., Ph.D. : A Standard Predictive Index of Human Response to the Thermal Environment, ASHRAE Transactions, vol.92, 1986, pp.709-730

27.3） Stolwijk, J. A. J. and J. D. Hardy, Temperature regulation in man – a theoretical study, Pflugers Arch. 291, pp.129-162, 1966

27.4） A. P. Gagge, Y. Nishi and R. R. Gonzalez: Standard Effective Temperature - a single temperature index of temperature sensation and thermal discomfort, Proceedings of the CIB Commision W45 symposium held at the building research station, pp 229-250, 1972

27.5） Fanger, P.O. :Thermal comfort, McGraw-Hill, New York, 1972

27.6） ASHRAE Standard, 55-2013, Thermal Environmental Conditions for Human Occupancy

27.7） ISO 7730, Ergonomics of the thermal environment - Analytical determination and interpretation of thermal comfort using calculation of the PMV and PPD indices and local thermal comfort criteria

27.8） S. Tanabe, Y. Kobayashi, Development of JOS-2 human thermoregulation model with detailed vascular system, Building and Environment, Vol. 66, pp.1-10, 2013

27.9）　田辺新一, 小林弘造, 小川一晃: 温熱環境評価のための人体数値計算モデル COM の開発, 日本建築学会環境系論文集, No. 599, pp.31-38, Jan., 2006
27.10）J.A.J. Stolwijk: A Mathematical Model of Physiological Temperature Regulation in Man, NASA Contractor report 1855, 1971
27.11）竹森利和, 中島健, 庄司裕子: 人体モデルの開発（熱的快適性評価のための基本モデル開発）, 日本機械学会論文集（B 編）, Vol. 61, No. 584, pp.1513-1520, April 1995
27.12）Carol Elaine Smith: A Transient Three-Dimensional Model of the Human Thermal System. PhD Thesis, Kansas State University, 1991
27.13）石野久彌, 郡公子, 佐藤豊: 人体 Two-Node Model の簡易化と応用に関する研究 : 人体発熱負荷推定への応用, 日本建築学会計画系論文報告集, Vol.451, pp.67-74, 1993
27.14）T. Oohori, L. G. Berglund, A. P. Gagge: Comparison of Current Two-Parameter indices of Vapor Permeation of Clothing – As Factors Governing Thermal Equilibrium and Human Comfort, ASHRAE Transactions, Vol.90, Part 2, pp.85-101, 1985
27.15）DuBois D, DuBois EF: A formula to estimate the approximate surface area if height and weight be known. Archives of Internal Medicine. 17, 863-71, 1916
27.16）Asmussen, E., and Nielsen, M.: Studies on the regulation of respiration in heavy work. Acta Physiol. Scand. 12, pp.171-188, 1946
27.17）Liddel, F. D. K.: Estimation of energy expenditure from expired air. J. Appl. Physiol. 18, I, pp.25-29, 1963
27.18）McCutchan J. W., and Taylor, C. L.: Respiratory heat exchange with varying temperatures and humidity of inspired air. J. Appl. Physiol. 4, pp.121-135, 1951
27.19）田辺新一, 中野淳太, 小林弘造: 温熱環境評価のための 65 分割体温調節モデルに関する研究, 日本建築学会計画系論文報告集, No. 541, pp.9-16, 2001
27.20）Ganpule AA, Tanaka S, Ishikawa Takata K, Tabata I. Interindividual variability in metabolic rates in Japanese subjects, European Journal of Clinical Nutrition, Vol.61, Issue.11, pp.1256–1261, 2007
27.21）Harris, Benedict: A biometric study of basal metabolism in man, 1919, Carnegie Institution of Washington
27.22）Grollman, A.: Physiological variations in the cardiac output of man. VI. The value of the cardiac output of the normal individual in the basal, resting condition. Am. J. Physiol., Vol.90, pp.210-217, 1929
27.23）新谷冨士雄, 渡辺真司, 稲木一元, 田中政: 心係数の男女差, 心臓, Vol.14, No.12, pp.1466-1472, 1982
27.24）渡部哲也, 元田憲, 加藤紀久: 正常人の心拍出量と循環時間, 心臓, Vol.5, No.8, pp.1053-1061, 1973
27.25）John W. Mitchell and Glen E. Myers: An analytical model of the counter-current heat exchange phenomena, Biophysical Journal, Vol.8, pp.897-911, 1968
27.26）市原真希, 斉藤正文, 西村美加, 田辺新一: サーマルマネキンを用いた立位・座位人体各部位の放射・対流熱伝達率の測定, 日本建築学会計画系論文集, Vol.501, pp.45-51, 1997
27.27）Roddie, I.C., Shepherd, J.T and Whelam, R.F., J.Physiol. 136, pp.489-498, 1957
27.28）Shepherd, J.T: Physiology of circulation in human limbs in health and disease (Philadelphia) W.B.Saunders Co., 1963
27.29）D. Mitchell. Convective Heat Transfer in Man and other Animals, Heat Loss from Animals and Man, Butterworth Publishing Inc., London, 1974
27.30）佐藤大樹, 大黒雅之, 吉田伸治: 人体各部の投影面積率を用いた室内空間における人体の放射受熱量評価に関する研究 （第 1 報） 人体形状モデルによる投影面積率算出と至近距離壁面との形態係数評価 , 空気調和・衛生工学会論文集, No. 202, pp.1~9, 2014.1
27.31）佐藤大樹, 大黒雅之, 吉田伸治: 人体各部の投影面積率を用いた室内空間における人体の放射受熱量評価に関する研究 （第 2 報） 部位別放射受熱量の算出と不均一放射環境の評価への適用 , 空気調和・衛生工学会論文集, No. 214, pp.9~17, 2015.1
27.32）中原信生: ビル・建築設備の省エネルギー 設計・管理技術の基礎から応用まで , No.47 条件緩和による風量低減と外気冷房効果の増大, 財団法人省エネルギーセンター, 第 1 版, 2001
27.33）William Heberden: An Account of the Heat of July, 1825: together with Some Remarks Upon Sensible Cold, Philosophical Transactions of the Royal Society of London, Vol.116, No.1/3, 1826

第28章　空調設備システムの熱的快適性評価
(Evaluation of Air Conditioning Systems)

28.1　概要

第 22 章~第 25 章では熱負荷計算を解説した。熱負荷計算モデルと第 26 章で解説した空気調和機のモデルを組み合わせれば、ゾーンの温湿度調整に必要な冷温水量や、過負荷になった場合のゾーンの温湿度変動を予測することができる。また、第 27 章で解説した人体モデルに、ゾーンの温湿度や壁表面温度を入力すれば、空間を熱的快適性の面から評価することもできる。

本章では、オフィスビルを例に取り、熱負荷計算モデルの作成と空調機モデルとの連成の方法を示す。また、エネルギー消費と熱的快適性、両面からの性能予測と評価を行う。

28.2　二次側システムモデルの作成

28.2.1　建物熱負荷計算

1）計算対象

計算対象とする建物の平面図および断面図を図 28.1 に示す。図 28.1 は「平成 25 年 1 月 エネルギーの使用の合理化に関する法律」にもとづく一次エネルギー消費量算定用 Web プログラム（WEBPRO）で、モデル建物として例示された建物である [28.1) 28.2)]。空気調和・衛生工学会の「建物のエネルギーシミュレーション評価手法ガイドライン」においても採用されており、今後、日本においては、本建物を用いたケーススタディなどの蓄積が進むと予測できる。建物全体としては 7 階建のオフィスビルであり、1 階にはエントランスホールなどもあるが、本書では基準階のみを計算対象とする。立地は東京都（北緯 35.7°、東経 139.8°）とする。

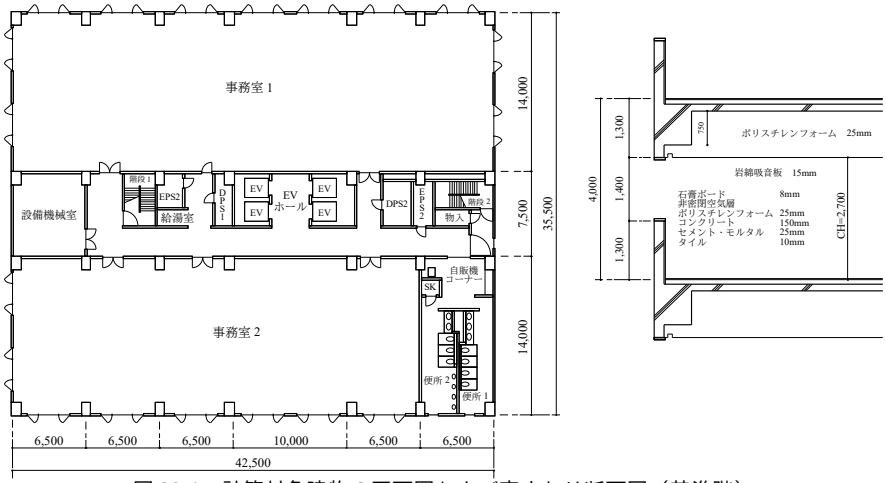


図 28.1　計算対象建物の平面図および窓まわり断面図（基準階）

図 28.2 に空調のゾーニングを示す。事務室 1 と事務室 2 に関しては、外壁から 5m の範囲をペリメータゾーンとし、ペリメータ系統の空調機を計画する。共用部は EV ホールおよび廊下を FCU に

よって空調する。自販機コーナーおよび便所は温調しないが、専有部からの余剰空気を廊下経由でパスして排気する[†1]。その他の諸室は非空調室とし、隣室温度差係数を 0.5 として計算する。

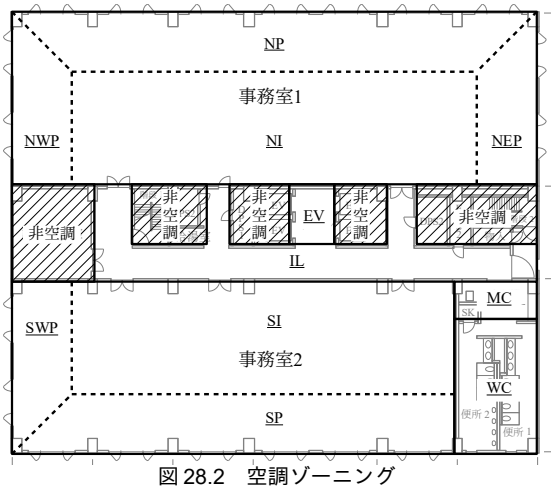


図 28.2　空調ゾーニング

表 28.1 に空調対象室の諸元を示す。隙間風は室容積あたりで 0.1 回/h とする。事務室に関しては家具の熱容量として空気の熱容量の 10 倍を見込む。活動量は事務作業相当（1.2 met）とする。

表 28.1　空調対象室諸元

RM No.	ZN No.	記号	室名	面積 [m²]	天井高 [m]	内部発熱			空調機系統	外気量 [CMH/m²]
						人員密度 [人/m²]	照明 [W/m²]	機器 [W/m²]		
0	0	NI	事務室 1	292.5	2.7	0.1	12.0	12.0	AHU-1I	5.0
	1	NWP		57.5	2.7	0.1	12.0	12.0	AHU-1P	5.0
	2	NP		187.5	2.7	0.1	12.0	12.0	AHU-1P	5.0
	3	NEP		57.5	2.7	0.1	12.0	12.0	AHU-1P	5.0
1	0	SI	事務室 2	279.0	2.7	0.1	12.0	12.0	AHU-2I	5.0
	1	SWP		57.5	2.7	0.1	12.0	12.0	AHU-2P	5.0
	2	SP		167.5	2.7	0.1	12.0	12.0	AHU-2P	5.0
2	0	EV	EV ホール	12.0	2.6	0.0	15.0	0.0	FCU-1	PASS
	1	IL	廊下	144.0	2.4	0.0	15.0	0.0	FCU-2~4	PASS
	2	MC	自販機コーナー	21.8	2.4	0.0	15.0	0.0	FCU-5	PASS
	3	WC	便所	69.2	2.4	0.0	15.0	0.0	-	PASS

図 28.3 に外気のフローを示す。

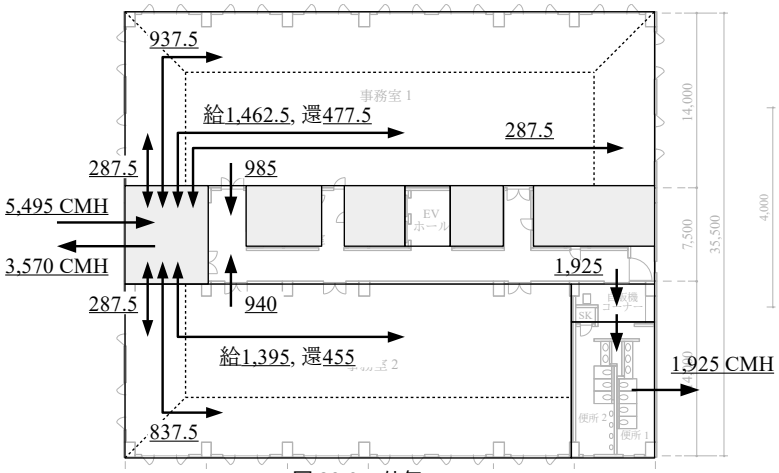


図 28.3　外気フロー

†1　本例ではゾーン間換気の計算練習のために単独空調は設けないこととしたが、外壁に面する便所で冷暖房ができないためにクレームとなることは多い。特に北側便所の場合には冬季の冷え込み対策が求められる。

外気は西の空調機械室より導入して外気処理空調機で調整した後、事務室 1 および事務室 2 の各ゾーンに給気する。ペリメータゾーンの給還気量は等しくするが、インテリアゾーンでは還気を絞り、一部を廊下へパスする。廊下へパスした余剰空気は自販機コーナーを経て便所から屋外に直接に排気する。事務室の還気は空調機械室から排気する。空間的に隣接するゾーン間には隣接長さあたりで、150 CMH/m のゾーン間換気を設定する。ゾーン隣接長さを表 28.2 に示す。

表 28.2　ゾーン隣接長さ [m]

	NI	NWP	NP	NEP	SI	SWP	SP	EV	IL	MC	WC
NI	-	9.0	32.5	9.0	-	-	-	-	-	-	-
NWP	9.0	-	7.0	-	-	-	-	-	-	-	-
NP	32.5	7.0	-	7.0	-	-	-	-	-	-	-
NEP	9.0	-	7.0	-	-	-	-	-	-	-	-
SI	-	-	-	-	-	9.0	31.0	-	-	-	-
SWP	-	-	-	-	9.0	-	7.0	-	-	-	-
SP	-	-	-	-	31.0	7.0	-	-	-	-	-
EV	-	-	-	-	-	-	-	-	3.0	-	-
IL	-	-	-	-	-	-	-	3.0	-	2.0	-
MC	-	-	-	-	-	-	-	-	2.0	-	2.0
WC	-	-	-	-	-	-	-	-	-	2.0	-

図 28.4 に室使用スケジュールを示す。土日の照明および人体発熱は事務所・廊下ともに常に 0 %である。また、空調運転時間は事務所が 7:00~21:00 まで、廊下が 8:00~21:00 までとする。温湿度設定値は夏季（6~9 月）が 26 ºC / 50 %、冬季（12~3 月）が 22 ºC / 40 %、中間期（4, 5, 10, 11 月）が 24 ºC / 50 %とする。暦は 2006 年を前提とし、祝日は 1/1~1/3, 1/9, 2/11, 3/21, 4/29, 5/3~5/5, 7/17, 9/18, 9/23, 10/9, 11/3, 11/23, 12/23, 12/29 とする。

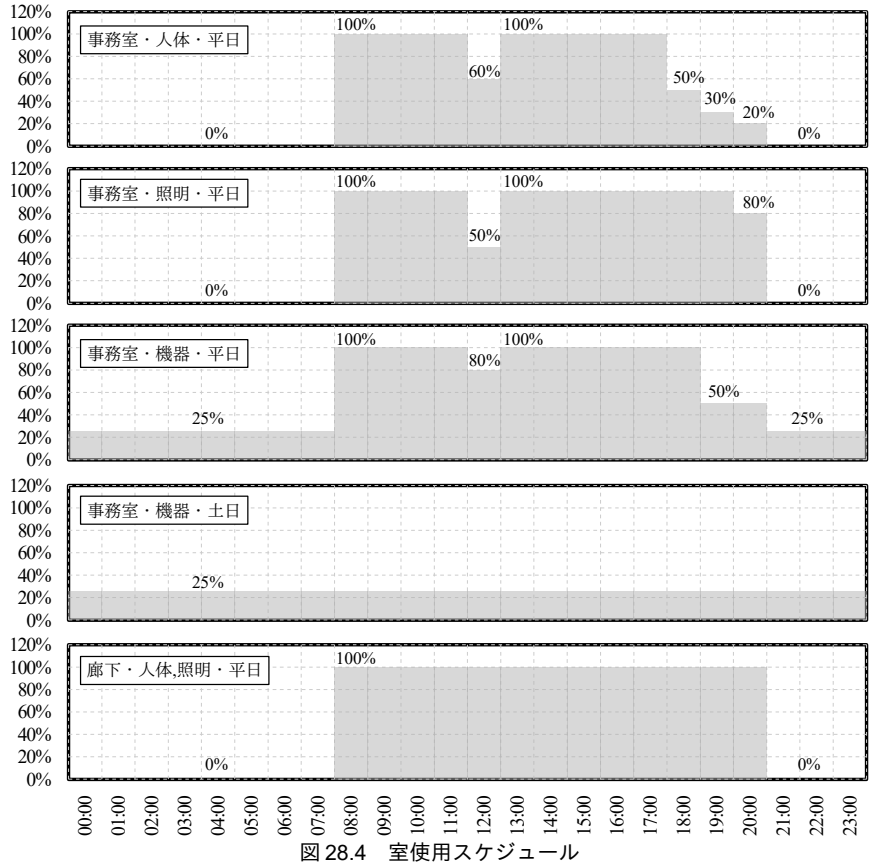


図 28.4　室使用スケジュール

表 28.3 に壁の構成・物性値・厚みを示す。対流熱伝達率は室内側が 4 W/(m²·K)、室外側が 18 W/(m²·K)とする。室内側短波長放射率と外壁日射吸収率は 0.9、長波長放射率は 0.8 とする。

表 28.3　壁構成・物性・厚み

記号	部位	材料	熱伝導率 [W/(m·K)]	容積比熱 [kJ/(m³·K)]	厚み [mm]
EX	外壁（一般部）	石膏ボード（内）	0.220	830	8
		非密閉中空層	-	-	-
		押出法スチレンフォーム 保温板 1 種	0.040	33	25
		コンクリート	1.6	2000	150
		セメント・モルタル	1.5	1600	25
		タイル（外）	1.3	2000	10
EXBM	外壁（梁部）	押出法スチレンフォーム 保温板 1 種（内）	0.040	33	25
		コンクリート	1.6	2000	750
		セメント・モルタル	1.5	1600	25
		タイル（外）	1.3	2000	10
FL	天井	ロックウール化粧吸音板	0.064	290	12
		石膏ボード	0.220	830	10
		非密閉中空層	-	-	-
		コンクリート	1.6	2000	150
		非密閉中空層	-	-	-
		ビニル系床材	0.190	2000	3
IN	内壁	石膏ボード	0.220	830	12
		非密閉中空層	-	-	-
		石膏ボード	0.220	830	12

外壁および内壁一覧を表 28.4 と表 28.5 に示す。基準階の計算であるため、各空調ゾーンで天井と床を用意し、自分自身との熱流を計算する必要があることに注意する（第 25 章 25.2.2 節 参照）。

表 28.4　外壁一覧

No.	ゾーン	方位	壁構成	面積 [m²]	No.	ゾーン	方位	壁構成	面積 [m²]
0	NWP	W	EX	32.8	7	SWP	W	EXBM	12.6
1	NWP	W	EXBM	12.6	8	SP	S	EX	79.7
2	NP	N	EX	94.5	9	SP	S	EXBM	32.4
3	NP	N	EXBM	38.3	10	MC	E	EX	16.8
4	NEP	E	EX	32.8	11	WC	E	EX	42.6
5	NEP	E	EXBM	12.6	12	WC	S	EX	26.0
6	SWP	W	EX	32.8	-	-	-	-	-

表 28.5　内壁一覧

No.	ゾーン 1	ゾーン 2	壁構成	面積 [m²]	No.	ゾーン 1	ゾーン 2	壁構成	面積 [m²]
13	NI	NI	FL	292.5	31	SWP	SWP	FL	57.5
14	NI	NI	FL	292.5	32	SWP	SWP	FL	57.5
15	NI	IL	IN	16.2	33	SWP	非空調室	IN	13.5
16	NI	EV	IN	8.1	34	SP	SP	FL	167.5
17	NI	非空調室	IN	64.8	35	SP	SP	FL	167.5
18	NWP	NWP	FL	57.5	36	SP	WC	IN	13.5
19	NWP	NWP	FL	57.5	37	EV	非空調室	IN	31.2
20	NWP	非空調室	IN	13.5	38	EV	EV	FL	12.0
21	NP	NP	FL	187.5	39	EV	EV	FL	12.0
22	NP	NP	FL	187.5	40	IL	MC	IN	10.8
23	NEP	NEP	FL	57.5	41	IL	非空調室	IN	158.4
24	NEP	NEP	FL	57.5	42	IL	IL	FL	144.0
25	NEP	非空調室	IN	13.5	43	IL	IL	FL	144.0
26	SI	IL	IN	79.7	44	MC	WC	IN	16.9
27	SI	MC	IN	8.0	45	MC	MC	FL	21.8
28	SI	非空調室	IN	28.4	46	MC	MC	FL	21.8
29	SI	SI	FL	279.0	47	WC	WC	FL	69.2
30	SI	SI	FL	279.0	48	WC	WC	FL	69.2

窓一覧を表 28.6 に示す。いずれも透明フロート単板ガラス（8mm）とし、透過率は 81.5 %、反射率 7.2 %とする。ブラインドは中間色、反射率は両面 66 %、奥行き 25mm、ピッチ 22.5mm とする。

表 28.6　窓一覧

No.	ゾーン	方位	面積 [m²]	No.	ゾーン	方位	面積 [m²]
0	NWP	W	10.6	3	SWP	W	10.6
1	NP	N	37.2	4	SP	S	31.9
2	NEP	E	10.6	-	-	-	-

2）計算プログラムの作成

次章を含め、本建物モデルの作成に関連する処理はすべて「SHASE_SimulationTest」クラスにまとめる。後々に簡単に感度解析ができるように、モデルの設定を調整するためのフラグを用意する。感度解析用のフラグをプログラム 28.1 に示す。

プログラム 28.1　感度解析用フラグ

SHASE_SimulationTest class

```
/// <summary>高断熱仕様か否か</summary>
public static bool IS_HIGH_INSULATION = false;

/// <summary>CO2 制御実施の真偽</summary>
public static bool USE_CO2CNTRL = false;

/// <summary>外気冷房の真偽</summary>
public static bool USE_OA_COOLING = false;

/// <summary>全熱交換器導入の真偽</summary>
public static bool USE_REGENERATOR = false;

/// <summary>VWV 制御導入の真偽</summary>
public static bool USE_VWV_SYSTEM = false;

/// <summary>高効率熱源導入の真偽</summary>
public static bool IS_HIGHEFF_HSOURCE = false;

/// <summary>クールビズ実施の真偽</summary>
public static bool DO_COOLBIZ = false;
```

プログラム 28.2 に建物モデルの作成処理を示す。11~39 行で表 28.3 にもとづいて壁の構成を作成する。感度解析用フラグに応じて断熱仕様を切り替える点に注意する。41~88 行で表 28.1 にもとづいてゾーンを作成する。57~68 行は内部発熱要素の設定である。本問題用に開発した内部発熱クラスであり、詳細は後述（プログラム 28.3）する。70~88 行で家具の熱容量と隙間風の量を設定する。90~111 行で表 28.4 と表 28.5 にもとづいて壁を作成する。113~124 行でガラス物性の配列を作成し、125~145 行で窓を作成する。以上で作成した各種のインスタンスを引数に与え、147~151 行で多数室モデルのインスタンスを生成する。室は「事務室 1」「事務室 2」「廊下」「EV ホール」「自販機コーナー」「便所」の 6 つとし、153~164 行でゾーンを割り当てる。166~188 行で表 28.4 と表 28.5 にもとづき内壁と外壁を各ゾーンに割り当てる。隣室が非空調室の場合には 186 行に示すように隣室温度差係数を用いる点に注意する。190~195 行で窓を各ゾーンに割り当て、197~203 行で窓の短波長を優先的に配分する壁表面を設定する。本モデルでは 70 %がそれぞれのペリメータゾーンの床に配分されるとする。206 行で建物モデルのインスタンスを生成する。209~221 行は表 28.2 のゾーン隣接長さにもとづくゾーン間換気設定処理である。

プログラム 28.2　建物モデルの作成処理

SHASE_SimulationTest class

```
/// <summary>建物モデルを作成する</summary>
/// <returns>建物モデル</returns>
public static BuildingThermalModel MakeBuildingThermalModel()
{
  //傾斜面の作成（四方位）///////////////
  Incline incN = new Incline(Incline.Orientation.N, 0.5 * Math.PI);
  Incline incE = new Incline(Incline.Orientation.E, 0.5 * Math.PI);
  Incline incW = new Incline(Incline.Orientation.W, 0.5 * Math.PI);
```

```
 9   Incline incS = new Incline(Incline.Orientation.S, 0.5 * Math.PI);
10
11   //壁構成を作成////////////////////////
12   WallLayer[] exWL = new WallLayer[6];  //外壁一般部分
13   exWL[0] = new WallLayer("タイル", 1.3, 2000, 0.010);
14   exWL[1] = new WallLayer("セメント・モルタル", 1.5, 1600, 0.025);
15   exWL[2] = new WallLayer("コンクリート", 1.6, 2000, 0.150);
16   if(IS_HIGH_INSULATION) exWL[3] = new WallLayer("押出ポリスチレンフォーム３種", 0.028, 33, 0.025);
17   else exWL[3] = new WallLayer("押出ポリスチレンフォーム１種", 0.040, 33, 0.025);
18   exWL[4] = new AirGapLayer("非密閉中空層", false, 0.05);
19   exWL[5] = new WallLayer("石膏ボード", 0.22, 830, 0.008);
20
21   WallLayer[] exbmWL = new WallLayer[4];  //外壁梁部分
22   exbmWL[0] = new WallLayer("タイル", 1.3, 2000, 0.010);
23   exbmWL[1] = new WallLayer("セメント・モルタル", 1.5, 1600, 0.025);
24   exbmWL[2] = new WallLayer("コンクリート", 1.6, 2000, 0.750);
25   if (IS_HIGH_INSULATION) exbmWL[3] = new WallLayer("押出ポリスチレンフォーム３種", 0.028, 33, 0.025);
26   else exbmWL[3] = new WallLayer("押出ポリスチレンフォーム１種", 0.040, 33, 0.025);
27
28   WallLayer[] flWL = new WallLayer[6];  //床・天井
29   flWL[0] = new WallLayer("ビニル系床材", 0.190, 2000, 0.003);
30   flWL[1] = new AirGapLayer("非密閉中空層", false, 0.05);
31   flWL[2] = new WallLayer("コンクリート", 1.6, 2000, 0.150);
32   flWL[3] = new AirGapLayer("非密閉中空層", false, 0.05);
33   flWL[4] = new WallLayer("石膏ボード", 0.220, 830, 0.009);
34   flWL[5] = new WallLayer("ロックウール化粧吸音板", 0.064, 290, 0.015);
35
36   WallLayer[] inWL = new WallLayer[3];  //内壁
37   inWL[0] = new WallLayer("石膏ボード", 0.220, 830, 0.012);
38   inWL[1] = new AirGapLayer("非密閉中空層", false, 0.05);
39   inWL[2] = new WallLayer("石膏ボード", 0.220, 830, 0.012);
40
41   //ゾーンを作成////////////////////////
42   Zone[] zn0 = new Zone[4];
43   Zone[] zn1 = new Zone[3];
44   Zone[] zn2 = new Zone[4];
45   zn0[0] = new Zone("NI", 292.5 * 2.7 * 1.2);
46   zn0[1] = new Zone("NWP", 57.5 * 2.7 * 1.2);
47   zn0[2] = new Zone("NP", 187.5 * 2.7 * 1.2);
48   zn0[3] = new Zone("NEP", 57.5 * 2.7 * 1.2);
49   zn1[0] = new Zone("SI", 279.0 * 2.7 * 1.2);
50   zn1[1] = new Zone("SWP", 57.5 * 2.7 * 1.2);
51   zn1[2] = new Zone("SP", 167.5 * 2.7 * 1.2);
52   zn2[0] = new Zone("EV", 12.0 * 2.6 * 1.2);
53   zn2[1] = new Zone("IL", 144.0 * 2.4 * 1.2);
54   zn2[2] = new Zone("MC", 21.8 * 2.4 * 1.2);
55   zn2[3] = new Zone("WC", 69.2 * 2.4 * 1.2);
56
57   //内部発熱を設定
58   zn0[0].AddHeatGain(new MyHeatGain(0.1 * 292.5, 12 * 292.5, 12 * 292.5, true));
59   zn0[1].AddHeatGain(new MyHeatGain(0.1 * 57.5, 12 * 57.5, 12 * 57.5, true));
60   zn0[2].AddHeatGain(new MyHeatGain(0.1 * 187.5, 12 * 187.5, 12 * 187.5, true));
61   zn0[3].AddHeatGain(new MyHeatGain(0.1 * 57.5, 12 * 57.5, 12 * 57.5, true));
62   zn1[0].AddHeatGain(new MyHeatGain(0.1 * 279.0, 12 * 279.0, 12 * 279.0, true));
63   zn1[1].AddHeatGain(new MyHeatGain(0.1 * 57.5, 12 * 57.5, 12 * 57.5, true));
64   zn1[2].AddHeatGain(new MyHeatGain(0.1 * 167.5, 12 * 167.5, 12 * 167.5, true));
65   zn2[0].AddHeatGain(new MyHeatGain(0, 15 * 12.0, 0, false));
66   zn2[1].AddHeatGain(new MyHeatGain(0, 15 * 144.0, 0, false));
67   zn2[2].AddHeatGain(new MyHeatGain(0, 15 * 21.8, 0, false));
68   zn2[3].AddHeatGain(new MyHeatGain(0, 15 * 69.2, 0, false));
69
70   ////隙間風と熱容量設定
71   for (int i = 0; i < zn0.Length; i++)
72   {
73     zn0[i].HeatCapacity = zn0[i].AirMass * 1006 * 10;
74     zn0[i].VentilationRate = zn0[i].AirMass / 3600d * 0.1;
75     zn0[i].InitializeAirState(22, 0.0105);
76   }
77   for (int i = 0; i < zn1.Length; i++)
78   {
79     zn1[i].HeatCapacity = zn1[i].AirMass * 1006 * 10;
80     zn1[i].VentilationRate = zn1[i].AirMass / 3600d * 0.1;
81     zn1[i].InitializeAirState(22, 0.0105);
82   }
83   for (int i = 0; i < zn0.Length; i++)
84   {
85     zn2[i].HeatCapacity = 0;
86     zn2[i].VentilationRate = zn2[i].AirMass / 3600d * 0.1;
87     zn2[i].InitializeAirState(22, 0.0105);
88   }
```

```
 89
 90  //壁体の作成//////////////////////////
 91  Wall[] walls = new Wall[49];
 92  walls[0] = new Wall(32.8, exWL);
 93  walls[1] = new Wall(12.6, exbmWL);
 94  walls[2] = new Wall(94.5, exWL);
 95  walls[3] = new Wall(38.3, exbmWL);
 96  walls[4] = new Wall(32.8, exWL);
 97  walls[5] = new Wall(12.6, exbmWL);
 98  //・・・
 99  //以下略
100
101  //壁の初期化
102  for (int i = 0; i < walls.Length; i++)
103  {
104    walls[i].ShortWaveAbsorptanceF = walls[i].ShortWaveAbsorptanceB = 0.8;
105    walls[i].LongWaveEmissivityF = walls[i].LongWaveEmissivityB = 0.9;
106    walls[i].RadiativeCoefficientF = walls[i].RadiativeCoefficientB = 5;
107    if (i <= 12) walls[i].ConvectiveCoefficientF = 18;
108    else walls[i].ConvectiveCoefficientF = 4;
109    walls[i].ConvectiveCoefficientB = 4;
110    walls[i].Initialize(20);
111  }
112
113  //窓を作成
114  double[] TAU_WIN, RHO_WIN;
115  if (IS_HIGH_INSULATION)
116  {
117    TAU_WIN = new double[] { 0.627, 0.815 }; //ガラスの透過率リスト[-]
118    RHO_WIN = new double[] { 0.211, 0.072 }; //ガラスの反射率リスト[-]
119  }
120  else
121  {
122    TAU_WIN = new double[] { 0.815 }; //ガラスの透過率リスト[-]
123    RHO_WIN = new double[] { 0.072 }; //ガラスの反射率リスト[-]
124  }
125  Window[][] win = new Window[3][];
126  win[0] = new Window[3];
127  win[1] = new Window[2];
128  win[2] = new Window[0];
129  win[0][0] = new Window(10.6, TAU_WIN, RHO_WIN, incW);
130  win[0][1] = new Window(37.2, TAU_WIN, RHO_WIN, incN);
131  win[0][2] = new Window(10.6, TAU_WIN, RHO_WIN, incE);
132  win[1][0] = new Window(10.6, TAU_WIN, RHO_WIN, incW);
133  win[1][1] = new Window(31.9, TAU_WIN, RHO_WIN, incS);
134  for (int i = 0; i < win.Length; i++)
135  {
136    for (int j = 0; j < win[i].Length; j++)
137    {
138      VenetianBlind blind = new VenetianBlind(25, 22.5, 0, 0, 0.66, 0.66);
139      blind.SlatAngle = 0;
140      win[i][j].SetShadingDevice(1, blind);
141      win[i][j].ConvectiveCoefficientF = 18;
142      win[i][j].ConvectiveCoefficientB = 4;
143      win[i][j].LongWaveEmissivityF = win[i][j].LongWaveEmissivityB = 0.9;
144    }
145  }
146
147  //多数室の作成
148  MultiRooms[] mRm = new MultiRooms[3];
149  mRm[0] = new MultiRooms(1, zn0, walls, win[0]);
150  mRm[1] = new MultiRooms(1, zn1, walls, win[1]);
151  mRm[2] = new MultiRooms(4, zn2, walls, win[2]);
152
153  //ゾーンを室に登録
154  mRm[0].AddZone(0, 0);
155  mRm[0].AddZone(0, 1);
156  mRm[0].AddZone(0, 2);
157  mRm[0].AddZone(0, 3);
158  mRm[1].AddZone(0, 0);
159  mRm[1].AddZone(0, 1);
160  mRm[1].AddZone(0, 2);
161  mRm[2].AddZone(0, 0);
162  mRm[2].AddZone(1, 1);
163  mRm[2].AddZone(2, 2);
164  mRm[2].AddZone(3, 3);
165
166  //外壁を登録
167  mRm[0].AddWall(1, 0, false); mRm[0].SetOutsideWall(0, true, incW);
168  mRm[0].AddWall(1, 1, false); mRm[0].SetOutsideWall(1, true, incW);
```

```
169   mRm[0].AddWall(2, 2, false); mRm[0].SetOutsideWall(2, true, incN);
170   mRm[0].AddWall(2, 3, false); mRm[0].SetOutsideWall(3, true, incN);
171   mRm[0].AddWall(3, 4, false); mRm[0].SetOutsideWall(4, true, incE);
172   mRm[0].AddWall(3, 5, false); mRm[0].SetOutsideWall(5, true, incE);
173   mRm[1].AddWall(1, 6, false); mRm[1].SetOutsideWall(6, true, incW);
174   mRm[1].AddWall(1, 7, false); mRm[1].SetOutsideWall(7, true, incW);
175   mRm[1].AddWall(2, 8, false); mRm[1].SetOutsideWall(8, true, incS);
176   mRm[1].AddWall(2, 9, false); mRm[1].SetOutsideWall(9, true, incS);
177   mRm[2].AddWall(2, 10, false); mRm[2].SetOutsideWall(10, true, incE);
178   mRm[2].AddWall(3, 11, false); mRm[2].SetOutsideWall(11, true, incE);
179   mRm[2].AddWall(3, 12, false); mRm[2].SetOutsideWall(12, true, incS);
180
181   //内壁を登録
182   mRm[0].AddWall(0, 0, 13);
183   mRm[0].AddWall(0, 0, 14);
184   mRm[0].AddWall(0, 15, true); mRm[2].AddWall(1, 15, false);
185   mRm[0].AddWall(0, 16, true); mRm[2].AddWall(0, 16, false);
186   mRm[0].AddWall(0, 17, false); mRm[0].UseAdjacentSpaceFactor(17, true, 0.5);
187   //・・・
188   //以下略
189
190   //窓を登録
191   mRm[0].AddWindow(1, 0);
192   mRm[0].AddWindow(2, 1);
193   mRm[0].AddWindow(3, 2);
194   mRm[1].AddWindow(1, 0);
195   mRm[1].AddWindow(2, 1);
196
197   //ペリメータ床に短波長優先配分
198   const double SW_RATE_TO_FLOOR = 0.7;
199   mRm[0].SetSWDistributionRateToFloor(0, 18, true, SW_RATE_TO_FLOOR);
200   mRm[0].SetSWDistributionRateToFloor(1, 21, true, SW_RATE_TO_FLOOR);
201   mRm[0].SetSWDistributionRateToFloor(2, 23, true, SW_RATE_TO_FLOOR);
202   mRm[1].SetSWDistributionRateToFloor(0, 31, true, SW_RATE_TO_FLOOR);
203   mRm[1].SetSWDistributionRateToFloor(1, 34, true, SW_RATE_TO_FLOOR);
204
205   //建物モデルの作成
206   BuildingThermalModel bModel = new BuildingThermalModel(mRm);
207   bModel.TimeStep = 3600;
208
209   //ゾーン間換気の設定
210   const double cvRate = 150d * 1.2 / 3600d;
211   bModel.SetCrossVentilation(0, 0, 0, 1, 9.0 * cvRate);
212   bModel.SetCrossVentilation(0, 0, 0, 2, 32.5 * cvRate);
213   bModel.SetCrossVentilation(0, 0, 0, 3, 9.0 * cvRate);
214   bModel.SetCrossVentilation(0, 1, 0, 2, 7.0 * cvRate);
215   bModel.SetCrossVentilation(0, 2, 0, 3, 7.0 * cvRate);
216   bModel.SetCrossVentilation(1, 0, 1, 1, 9.0 * cvRate);
217   bModel.SetCrossVentilation(1, 0, 1, 2, 31.0 * cvRate);
218   bModel.SetCrossVentilation(1, 1, 1, 2, 7.0 * cvRate);
219   bModel.SetCrossVentilation(2, 0, 2, 1, 3.0 * cvRate);
220   bModel.SetCrossVentilation(2, 1, 2, 2, 2.0 * cvRate);
221   bModel.SetCrossVentilation(2, 2, 2, 3, 2.0 * cvRate);
222
223   return bModel;
224 }
```

プログラム 28.3 に内部発熱要素クラスの定義を示す。IHeatGain クラスを実装する。29~40 行のコンストラクタでは最大発熱と事務室か否かの別を保存する。69~130 行で図 28.4 に示したスケジュールに従って負荷率を計算し、これに最大発熱を乗じて各時刻の内部発熱を計算する。

プログラム 28.3　内部発熱要素クラス

SHASE_SimulationTest.MyHeatGain class

```
 1 /// <summary>内部発熱クラス</summary>
 2 public class MyHeatGain : IHeatGain
 3 {
 4
 5   /// <summary>事務室か否か</summary>
 6   private bool isOffice;
 7
 8   /// <summary>最大人数[人]を取得する</summary>
 9   public double MaxOccupancy { get; private set; }
10
11   /// <summary>最大照明発熱[W]を取得する</summary>
12   public double MaxLightingLoad { get; private set; }
13
14   /// <summary>最大機器発熱[W]を取得する</summary>
```

```
15  public double MaxPlugLoad { get; private set; }
16
17  /// <summary>人数[人]を取得する</summary>
18  public double Occupancy { get; private set; }
19
20  /// <summary>照明発熱[W]を取得する</summary>
21  public double LightingLoad { get; private set; }
22
23  /// <summary>機器発熱[W]を取得する</summary>
24  public double PlugLoad { get; private set; }
25
26  /// <summary>現在時刻</summary>
27  private DateTime currentDT;
28
29  /// <summary>初期化する</summary>
30  /// <param name="maxOccupancy">最大人数[人]</param>
31  /// <param name="maxLightingLoad">最大照明発熱[W]</param>
32  /// <param name="maxPlugLoad">最大機器発熱[W]</param>
33  /// <param name="isOffice">事務室か否か</param>
34  public MyHeatGain(double maxOccupancy, double maxLightingLoad, double maxPlugLoad, bool isOffice)
35  {
36    MaxOccupancy = maxOccupancy;
37    MaxLightingLoad = maxLightingLoad;
38    MaxPlugLoad = maxPlugLoad;
39    this.isOffice = isOffice;
40  }
41
42  /// <summary>顕熱取得[W]の内、対流成分を取得する</summary>
43  /// <param name="zone">発熱要素が属するゾーン</param>
44  /// <returns>顕熱取得の対流成分[kW]</returns>
45  public double GetConvectiveHeatGain(ImmutableZone zone)
46  {
47    updateLoad(zone.MultiRoom.CurrentDateTime);
48    return (LightingLoad + PlugLoad + Occupancy * 80) * 0.4;
49  }
50
51  /// <summary>顕熱取得[W]の内、放射成分を取得する</summary>
52  /// <param name="zone">発熱要素が属するゾーン</param>
53  /// <returns>顕熱取得の放射成分[kW]</returns>
54  public double GetRadiativeHeatGain(ImmutableZone zone)
55  {
56    updateLoad(zone.MultiRoom.CurrentDateTime);
57    return (LightingLoad + PlugLoad + Occupancy * 80) * 0.6;
58  }
59
60  /// <summary>発生水分[kg/s]を取得する</summary>
61  /// <param name="zone">発熱要素が属するゾーン</param>
62  /// <returns>発生水分[kg/s]</returns>
63  public double GetWaterGain(ImmutableZone zone)
64  {
65    updateLoad(zone.MultiRoom.CurrentDateTime);
66    return Occupancy * 40 / 2500000d;
67  }
68
69  /// <summary>負荷を更新する</summary>
70  /// <param name="dt">現在の日時</param>
71  private void updateLoad(DateTime dt)
72  {
73    if (currentDT == dt) return;
74    else currentDT = dt;
75
76    double rLt, rOcc, rPlg;
77    if (isOffice) //事務室の場合
78    {
79      if (IsHoliday(dt) || (dt.Hour < 8 || 21 <= dt.Hour))
80      {
81        rOcc = 0;
82        rLt = 0;
83        rPlg = 0.25;
84      }
85      else if (20 <= dt.Hour && dt.Hour < 21)
86      {
87        rOcc = 0.2;
88        rLt = 0.8;
89        rPlg = 0.5;
90      }
91      else if (19 <= dt.Hour && dt.Hour < 20)
92      {
93        rOcc = 0.3;
94        rLt = 1.0;
```

```
 95         rPlg = 0.5;
 96       }
 97       else if (18 <= dt.Hour && dt.Hour < 19)
 98       {
 99         rOcc = 0.5;
100         rLt = 1.0;
101         rPlg = 1.0;
102       }
103       else if (12 <= dt.Hour && dt.Hour < 13)
104       {
105         rOcc = 0.6;
106         rLt = 0.5;
107         rPlg = 0.8;
108       }
109       else
110       {
111         rOcc = 1.0;
112         rLt = 1.0;
113         rPlg = 1.0;
114       }
115     }
116     else //廊下などの場合
117     {
118       rOcc = 0;
119       rLt = 0;
120       rPlg = 0.0;
121       if (!IsHoliday(dt) && 8 <= dt.Hour && dt.Hour < 21)
122       {
123         rOcc = 1.0;
124         rLt = 1.0;
125       }
126     }
127     LightingLoad = MaxLightingLoad * rLt;
128     PlugLoad = MaxPlugLoad * rPlg;
129     Occupancy = MaxOccupancy * rOcc;
130   }
131 }
```

プログラム 28.3 の 80 行では土日祝祭日の場合の条件分岐を設けている。この処理をプログラム 28.4 に示す。

プログラム 28.4　土日祝日判定処理

SHASE_SimulationTest class

```
 1 /// <summary>土日祝日か否かを判定する</summary>
 2 /// <param name="dt">現在の日時</param>
 3 /// <returns>土日祝日の場合に true</returns>
 4 public static bool IsHoliday(DateTime dt)
 5 {
 6   if (dt.DayOfWeek == DayOfWeek.Saturday) return true;
 7   if (dt.DayOfWeek == DayOfWeek.Sunday) return true;
 8   if (dt.Month == 1 && dt.Day == 2) return true;
 9   if (dt.Month == 1 && dt.Day == 3) return true;
10   if (dt.Month == 1 && dt.Day == 9) return true;
11   if (dt.Month == 2 && dt.Day == 11) return true;
12   if (dt.Month == 3 && dt.Day == 21) return true;
13   if (dt.Month == 4 && dt.Day == 29) return true;
14   if (dt.Month == 5 && dt.Day == 3) return true;
15   if (dt.Month == 5 && dt.Day == 4) return true;
16   if (dt.Month == 5 && dt.Day == 5) return true;
17   if (dt.Month == 7 && dt.Day == 17) return true;
18   if (dt.Month == 9 && dt.Day == 18) return true;
19   if (dt.Month == 9 && dt.Day == 23) return true;
20   if (dt.Month == 10 && dt.Day == 9) return true;
21   if (dt.Month == 11 && dt.Day == 3) return true;
22   if (dt.Month == 11 && dt.Day == 23) return true;
23   if (dt.Month == 12 && dt.Day == 23) return true;
24   if (dt.Month == 12 && dt.Day == 29) return true;
25   return false;
26 }
```

プログラム 28.5 に空調制御の設定処理を示す。3~20 行で季節に応じて設定温湿度を判定し、22~41 行で空調の発停を切り替える。

プログラム 28.5　空調制御の設定処理

SHASE_SimulationTest class

```
 1 public static void SetHVACControl(BuildingThermalModel bModel)
```

```
 2 {
 3   //温湿度判定
 4   double dbt, hrt;
 5   DateTime dt = bModel.CurrentDateTime;
 6   if (6 <= dt.Month && dt.Month < 10)
 7   {
 8     dbt = 26;
 9     hrt = 0.00930;
10   }
11   else if (dt.Month == 12 || dt.Month < 4)
12   {
13     dbt = 22;
14     hrt = 0.00656;
15   }
16   else
17   {
18     dbt = 24;
19     hrt = 0.01050;
20   }
21
22   for (int i = 0; i < bModel.MultiRoom.Length; i++)
23   {
24     for (int j = 0; j < bModel.MultiRoom[i].ZoneNumber; j++)
25     {
26       bool ffloat;
27       if (IsHoliday(dt)) ffloat = true;
28       else if (i == 2) ffloat = (dt.Hour < 7 || 21 <= dt.Hour) || (j == 4);
29       else ffloat = (dt.Hour < 8 || 21 <= dt.Hour);
30       if (ffloat)
31       {
32         bModel.ControlHeatSupply(i, j, 0);
33         bModel.ControlWaterSupply(i, j, 0);
34       }
35       else
36       {
37         bModel.ControlDrybulbTemperature(i, j, dbt);
38         bModel.ControlHumidityRatio(i, j, hrt);
39       }
40     }
41   }
42 }
```

プログラム 28.6 に年間熱負荷計算処理を示す。

4 行の配列に気象データを与えて計算を行う。他のプログラムと計算結果を比較するため、以降の計算では東京都の標準年気象データを用いたが、第 7 章の気象データ作成プログラムなどを用いて 79~102 行のメソッドで作成しても良い。ただし、第 7 章のプログラムは毎正時の日射を出力するため、標準年気象データとは太陽位置が 30 分ずれることに注意する必要がある（32 行や 49 行など）。

熱容量が大きい建物では壁体内の初期温度が計算結果に少なくない影響を与える。この影響を取り除くため、検討対象よりも前の一定期間についても計算を行うことがあり、この計算処理を助走計算と呼ぶ。26~41 行が助走計算処理である。1 月~12 月が検討対象であるが、12 月 1 日~31 日の計算を行うことで 1 月 1 日時点の壁体内温度を初期化する。助走計算として 1 月 1 日時点の 24 時間分のデータを用いて周期定常状態になるまで反復計算を行うという方法もある。

43~75 行で 8760 時間分の熱負荷計算処理を行う。48~50 行で外気条件、53 行で空調の状態を更新し（プログラム 28.5）、54~56 行で熱平衡を解く。

プログラム 28.6　年間熱負荷計算

```
 1 private static void SHASE_HeatLoadTest()
 2 {
 3   //気象データ読み込み
 4   double[] dbt, hrt, dnr, dhr, ncr;
 5   //4行に記載の配列に気象データを読み込む処理
 6
 7   //建物モデル作成
 8   BuildingThermalModel building = SHASE_SimulationTest.MakeBuildingThermalModel();
 9   Sun sun = new Sun(Sun.City.Tokyo);
10
```

```
11   using (StreamWriter sWriter = new StreamWriter("output.csv", false, Encoding.GetEncoding("Shift_JIS")))
12   {
13     //タイトル行書き出し
14     sWriter.Write(",,外気乾球温度,外気絶対湿度,法線面直達日射,水平面全天日射,夜間放射");
15     for (int j = 0; j < building.MultiRoom.Length; j++)
16     {
17       ImmutableMultiRooms mRM = building.MultiRoom[j];
18       for (int k = 0; k < mRM.ZoneNumber; k++)
19       {
20         string nm = mRM.Zones[k].Name;
21         sWriter.Write(", " + nm + ":室温," + nm + ":絶対湿度," + nm + ":顕熱負荷," + nm + ":潜熱負荷");
22       }
23     }
24     sWriter.WriteLine();
25
26     //12 月を助走計算期間とする
27     DateTime dt = new DateTime(2006, 12, 1, 0, 0, 0);
28     for (int i = 0; i < 31 * 24; i++)
29     {
30       //気象条件更新
31       sun.SetGlobalHorizontalRadiation(dhr[8016 + i], dnr[8016 + i]);
32       sun.Update(dt.AddMinutes(30)); //過去一時間のデータのため 30 分シフト
33       building.UpdateOutdoorCondition(dt, sun, dbt[8016 + i], hrt[8016 + i], ncr[8016 + i]);
34
35       //熱平衡を更新
36       SHASE_SimulationTest.SetHVACControl(building);
37       building.ForecastHeatTransfer();
38       building.ForecastWaterTransfer();
39       building.FixState();
40       dt = dt.AddHours(1);
41     }
42
43     //8760 時間の計算実行
44     dt = new DateTime(2006, 1, 1, 0, 0, 0);
45     for (int i = 0; i < 8760; i++)
46     {
47       //気象条件更新
48       sun.SetGlobalHorizontalRadiation(dhr[i], dnr[i]);
49       sun.Update(dt.AddMinutes(30)); //過去一時間のデータのため 30 分シフト
50       building.UpdateOutdoorCondition(dt, sun, dbt[i], hrt[i], ncr[i]);
51
52       //熱平衡を更新
53       SHASE_SimulationTest.SetHVACControl(building);
54       building.ForecastHeatTransfer();
55       building.ForecastWaterTransfer();
56       building.FixState();
57
58       //書き出し
59       sWriter.Write(dt.ToShortDateString() + "," + dt.ToShortTimeString());
60       sWriter.Write("," + dbt[i] + "," + hrt[i] + "," + dnr[i] + "," + dhr[i] + "," + ncr[i]);
61       for (int j = 0; j < building.MultiRoom.Length; j++)
62       {
63         ImmutableMultiRooms mRM = building.MultiRoom[j];
64         for (int k = 0; k < mRM.ZoneNumber; k++)
65         {
66           ImmutableZone zn = mRM.Zones[k];
67           sWriter.Write
68             (", " + zn.Temperature + ", " + zn.HumidityRatio + ", " + zn.HeatSupply + ", " + zn.WaterSupply);
69         }
70       }
71       sWriter.WriteLine();
72
73       if (dt.Hour == 0) Console.WriteLine(dt.ToShortDateString());
74       dt = dt.AddHours(1);
75     }
76   }
77 }
78
79 public static void MakeWeatherData
80   (int seed, out double[] dbt, out double[] hrt, out double[] dnr, out double[] dhr, out double[] ncr)
81 {
82   double[] rad;
83   bool[] fcf;
84   dnr = new double[8760];
85   dhr = new double[8760];
86   ncr = new double[8760];
87   RandomWeather.MakeWeather(seed, RandomWeather.Location.Tokyo, 1, out dbt, out hrt, out rad, out fcf);
88   for (int i = 0; i < hrt.Length; i++) hrt[i] *= 0.001; //g/kg→kg/kg に換算
89
90   DateTime dt = new DateTime(2014, 1, 1, 0, 0, 0);
```

```
 91   Sun sun = new Sun(Sun.City.Tokyo);
 92   for (int i = 0; i < 8760; i++)
 93   {
 94     sun.Update(dt);
 95     sun.SeparateGlobalHorizontalRadiation(rad[i], Sun.SeparationMethod.Udagawa);
 96     dnr[i] = sun.DirectNormalRadiation;
 97     dhr[i] = sun.DiffuseHorizontalRadiation;
 98     double vp = MoistAir.GetWaterVaporPartialPressureFromHumidityRatio(hrt[i], 101.325);
 99     ncr[i] = Sky.GetNocturnalRadiation(dbt[i], fcf[i] ? 0 : 5, vp);
100     dt = dt.AddHours(1);
101   }
102 }
```

図 28.5 に空調ゾーン別の年間積算負荷と最大負荷を示す。図 28.6 に代表日における事務室の負荷時系列を示す。NewHASP と BEST による計算結果も参考に表示した。概ね、整合した結果が得られていることがわかる。

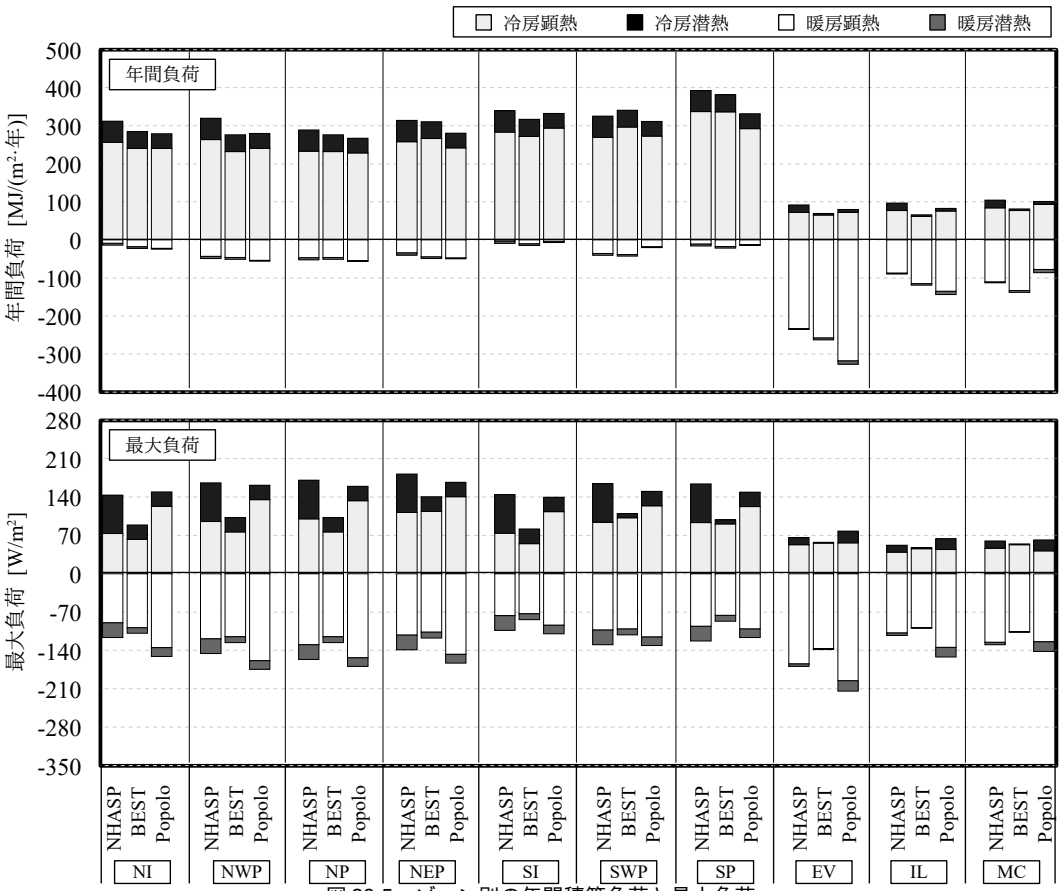


図 28.5　ゾーン別の年間積算負荷と最大負荷

表 28.7　ゾーン別の年間積算負荷と最大負荷

	冷房積算 [MJ/(m²·年)]			暖房積算 [MJ/(m²·年)]			冷房ピーク [W/m²]			暖房ピーク [W/m²]		
	NHASP	BEST	Popolo	NHASP	BEST	Popolo	NHASP	BEST	Popolo	NHASP	BEST	Popolo
NI	312	285	275	14	22	21	143	89	149	116	109	148
NWP	320	276	275	49	51	48	166	103	161	146	126	171
NP	289	276	263	52	51	49	170	103	159	157	126	166
NEP	314	311	276	40	49	42	182	141	166	139	117	159
SI	339	317	329	9	14	6	144	82	140	103	84	107
SWP	325	341	308	41	43	15	164	110	150	130	112	128
SP	393	381	329	16	22	10	164	98	148	123	87	114
EV	92	70	79	235	262	311	66	57	78	169	138	215
IL	96	66	82	89	119	138	52	47	65	113	99	152
MC	104	81	99	112	138	84	60	54	62	130	107	142

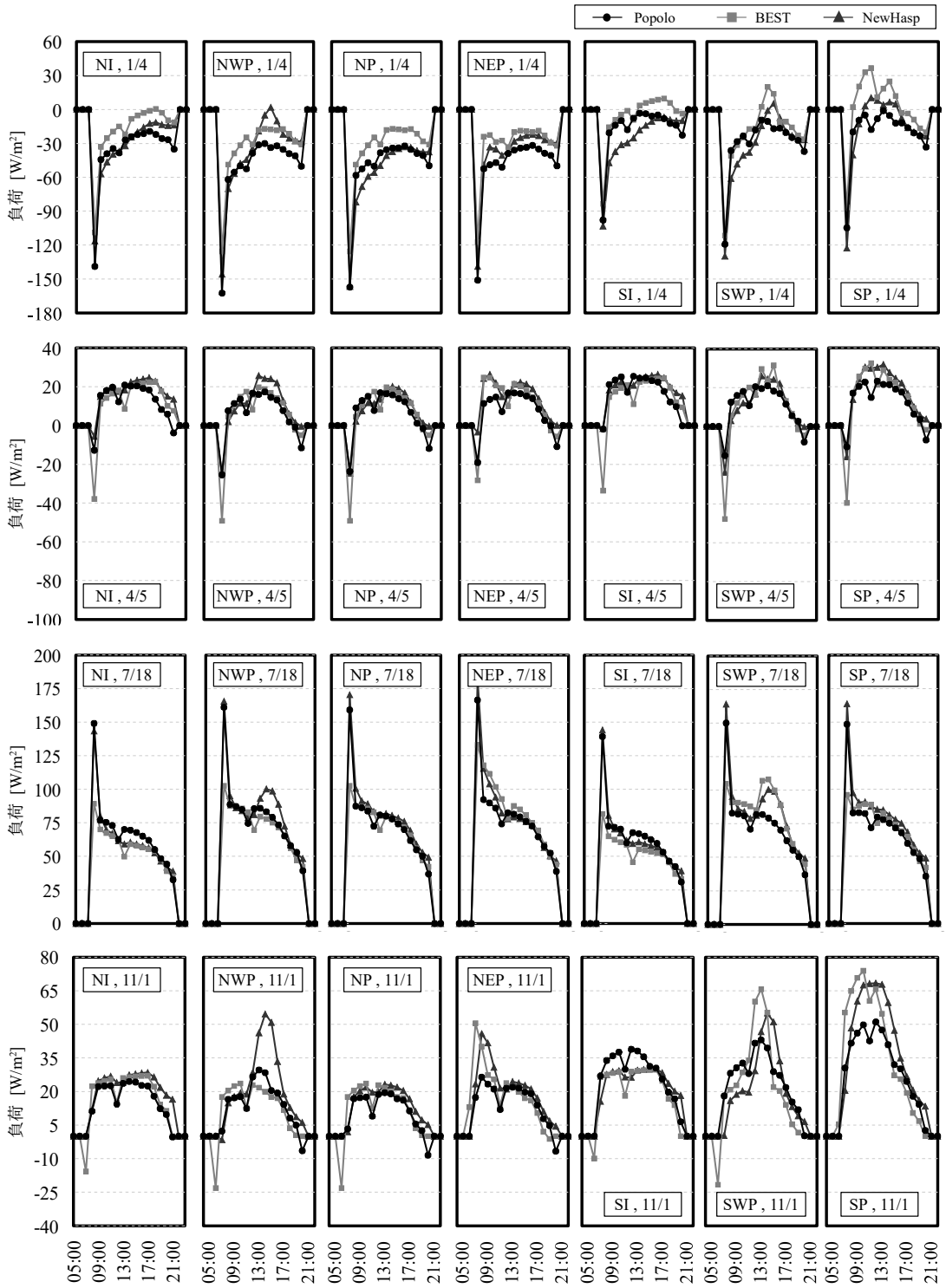


図 28.6　代表日時系列データ（事務室 1）

28.2.2　二次側空調システムの計算

現実の建物では 28.2.1 節で計算したように温度と湿度の両方を完全に制御することはできない。例えば再熱コイルが無ければ夏場の湿度は成り行きとなるし、休日明けなどで蓄熱負荷が大きい時には過負荷となり設定温湿度を満足できない時間帯も生じる。そこで、第 26 章で作成した空調機システムモデルを用いて、図 28.1 の建物を題材に建物熱負荷と空調機の連成計算を行う。空調機の仕様を表 28.8 に示す。ただし、AHU-1P に関しては表 26.1 と同じ仕様とする。

表 28.8　空調機の仕様

AHU-1I					
給気ファン	風量	6,212 CMH	還気ファン	風量	5,700 CMH
	静圧	400 Pa		静圧	200 Pa
	消費電力	3.11 kW		消費電力	1.30 kW
加湿器	水滴下気化式 15.4 kg/h				
冷水コイル	空気条件	入口：27.46 ºC / 12.06 g/kg 出口：14.30 ºC / 9.32 g/kg	温水コイル	空気条件	入口：17.46 ºC / 5.54 g/kg 出口：36.10 ºC / 5.54 g/kg
	冷水条件	7 ºC → 12.6 ºC		温水条件	50 ºC → 43.6 ºC
	冷水流量	111 L/min		温水流量	93 L/min
	冷却能力	43.5 kW		加熱能力	41.5 kW
	列数×段数	6 列×22 段		列数×段数	4 列×22 段
	正面面積	820 mm×910 mmH		正面面積	820 mm×910 mmH
	フロー	ハーフフロー		フロー	ハーフフロー

AHU-2P					
給気ファン	風量	5,106 CMH	還気ファン	風量	4,712 CMH
	静圧	400 Pa		静圧	200 Pa
	消費電力	2.53 kW		消費電力	1.04 kW
加湿器	水滴下気化式 11.2 kg/h				
冷水コイル	空気条件	入口：27.46 ºC / 12.06 g/kg 出口：14.30 ºC / 9.32 g/kg	温水コイル	空気条件	入口：17.46 ºC / 5.54 g/kg 出口：36.10 ºC / 5.54 g/kg
	冷水条件	7 ºC → 12.6 ºC		温水条件	50 ºC → 43.6 ºC
	冷水流量	88 L/min		温水流量	93 L/min
	冷却能力	34.5 kW		加熱能力	32.3 kW
	列数×段数	6 列×22 段		列数×段数	4 列×22 段
	正面面積	820 mm×910 mmH		正面面積	820 mm×910 mmH
	フロー	ハーフフロー		フロー	ハーフフロー

AHU-2I					
給気ファン	風量	5,730 CMH	還気ファン	風量	5,242 CMH
	静圧	400 Pa		静圧	200 Pa
	消費電力	3.01 kW		消費電力	1.29 kW
加湿器	水滴下気化式 14.6 kg/h				
冷水コイル	空気条件	入口：27.46 ºC / 12.06 g/kg 出口：14.30 ºC / 9.32 g/kg	温水コイル	空気条件	入口：17.46 ºC / 5.54 g/kg 出口：36.10 ºC / 5.54 g/kg
	冷水条件	7 ºC → 12.6 ºC		温水条件	50 ºC → 43.6 ºC
	冷水流量	104 L/min		温水流量	86 L/min
	冷却能力	40.5 kW		加熱能力	38.4 kW
	列数×段数	6 列×20 段		列数×段数	4 列×20 段
	正面面積	820 mm×910 mmH		正面面積	820 mm×910 mmH
	フロー	ハーフフロー		フロー	ハーフフロー

※フィンピッチ 2.9 mm、厚み 0.2 mm、熱伝導率 237 W/(m·K)、チューブ内外径 14.6 mm、15.8 mm

プログラム 28.7 に空調機システムクラスのインスタンス作成処理を示す。CAV と VAV で比較を行うこととし、第 3 引数で切り替える。4 行の建物熱負荷計算モデル作成処理はプログラム 28.2 と同じである。5 行で 49~102 行のメソッドを用いて空調機を作成する。処理内容は例題 26.2 とプログラム 26.10 と同様であるが、プログラム 28.1 で示した感度解析用フラグを用いて全熱交換器と外気冷房の採否を設定できるようにする（USE_REGENERATOR、USE_OA_COOLING）。全熱交換器を採用す

る場合には 68 行に示すようにファンの静圧も高くする必要がある点に注意する。8~31 行で CAV または VAV を作成し、33~46 行で空調機システムクラスに設定する。

プログラム **28.7**　空調機システムクラスのインスタンス作成処理

SHASE_SimulationTest class

```
 1 public static void MakeAHUSystem(out BuildingThermalModel bModel, out AHUSystem ahuSystem, bool isCAVSystem)
 2 {
 3   //二次側システムモデル作成
 4   bModel = MakeBuildingThermalModel();
 5   AirHandlingUnit[] ahu = makeAHUs();
 6   ahuSystem = new AHUSystem(bModel, ahu);
 7
 8   //CAV,VAV を作成
 9   const double MIN_RATE = 0.2;
10   //NI 系統
11   AHUSystem.VolumeController[] vcNI = new AHUSystem.VolumeController[1];
12   vcNI[0] = new AHUSystem.VolumeController
13     (0, 0, 6212d / 3600 * 1.2, 5700d / 3600 * 1.2, 6212d / 3600 * 1.2 * MIN_RATE);
14   //NP 系統
15   AHUSystem.VolumeController[] vcNP = new AHUSystem.VolumeController[3];
16   vcNP[0] = new AHUSystem.VolumeController
17     (0, 1, 1543d / 3600 * 1.2, 1433d / 3600 * 1.2, 1543d / 3600 * 1.2 * MIN_RATE);
18   vcNP[1] = new AHUSystem.VolumeController
19     (0, 2, 4151d / 3600 * 1.2, 3857d / 3600 * 1.2, 4151d / 3600 * 1.2 * MIN_RATE);
20   vcNP[2] = new AHUSystem.VolumeController
21     (0, 3, 1782d / 3600 * 1.2, 1656d / 3600 * 1.2, 1782d / 3600 * 1.2 * MIN_RATE);
22   //SI 系統
23   AHUSystem.VolumeController[] vcSI = new AHUSystem.VolumeController[1];
24   vcSI[0] = new AHUSystem.VolumeController
25     (1, 0, 5730d / 3600 * 1.2, 5242d / 3600 * 1.2, 5730d / 3600 * 1.2 * MIN_RATE);
26   //SP 系統
27   AHUSystem.VolumeController[] vcSP = new AHUSystem.VolumeController[2];
28   vcSP[0] = new AHUSystem.VolumeController
29     (1, 1, 1510d / 3600 * 1.2, 1393d / 3600 * 1.2, 1510d / 3600 * 1.2 * MIN_RATE);
30   vcSP[1] = new AHUSystem.VolumeController
31     (1, 2, 3596d / 3600 * 1.2, 3319d / 3600 * 1.2, 3596d / 3600 * 1.2 * MIN_RATE);
32
33   if (isCAVSystem)
34   {
35     ahuSystem.SetCAV(0, vcNI);
36     ahuSystem.SetCAV(1, vcNP);
37     ahuSystem.SetCAV(2, vcSI);
38     ahuSystem.SetCAV(3, vcSP);
39   }
40   else
41   {
42     ahuSystem.SetVAV(0, vcNI);
43     ahuSystem.SetVAV(1, vcNP);
44     ahuSystem.SetVAV(2, vcSI);
45     ahuSystem.SetVAV(3, vcSP);
46   }
47 }
48
49 private static AirHandlingUnit[] makeAHUs()
50 {
61   AirHandlingUnit[] ahus = new AirHandlingUnit[4];
62   CrossFinHeatExchanger cCoil, hCoil;
63   CentrifugalFan saFan, raFan;
64   RotaryRegenerator regenerator;
65   double msa, mra, moa;
66
67   double dPHEX = 0;
68   if (USE_REGENERATOR) dPHEX = 0.2;
69
70   //AHU-1I を作成
71   msa = 6212d / 3600 * 1.2;
72   mra = 5700d / 3600 * 1.2;
73   moa = 1463d / 3600 * 1.2;
74   cCoil = new CrossFinHeatExchanger(0.82, 0.910, 6, 24, msa, 27.46, 0.01206, 95,
75     111d / 60, 111d / 60, 7, CrossFinHeatExchanger.WaterFlowType.HalfFlow, 43.5, true);
76   hCoil = new CrossFinHeatExchanger(0.82, 0.910, 4, 24, msa, 17.46, 0.00554, 95,
77     93d / 60, 93d / 60, 50, CrossFinHeatExchanger.WaterFlowType.HalfFlow, 41.5, true);
78   saFan = new CentrifugalFan(0.85 + dPHEX, msa / 1.2, 0.85 + dPHEX, msa / 1.2, 3, true);
79   raFan = new CentrifugalFan(0.35 + dPHEX, mra / 1.2, 0.35 + dPHEX, mra / 1.2, 3, true);
80   saFan.MinimumRotationRatio = 0.4;
81   raFan.MinimumRotationRatio = 0.4;
82   regenerator = new RotaryRegenerator
83     (0.34, moa * 3600 / 1.2, (moa + mra - msa) * 3600 / 1.2, true, 34.4, 0.0194, 26, 0.0105);
```

```
 84   ahus[0] = new AirHandlingUnit
 85     (cCoil, hCoil, AirHandlingUnit.HumidifierType.DropPervaporation, saFan, raFan, regenerator);
 86   ahus[0].SetOutdoorAirFlowRange(moa, msa);
 87   ahus[0].SetAirFlowRate(mra, msa);
 88
 89   //AHU-1P, AHU-2I, AHU-2P の作成処理は AHU-1I と同様のため省略
 90
 91   //共通の設定
 92   for (int i = 0; i < ahus.Length; i++)
 93   {
 94     if(USE_OA_COOLING) ahus[i].OutdoorAirCooling = AirHandlingUnit.OutdoorAirCoolingControl.Enthalpy;
 95     else ahus[i].OutdoorAirCooling = AirHandlingUnit.OutdoorAirCoolingControl.None;
 96     ahus[i].BypassRegenerator = !USE_REGENERATOR;
 97     ahus[i].MinimizeAirFlow = true;
 98     ahus[i].UpperTemperatureLimit = 37d;
 99     ahus[i].LowerTemperatureLimit = 15d;
100   }
101   return ahus;
102 }
```

プログラム 28.8 に空調制御の設定処理を示す。プログラム 28.5 とほぼ同様であるが、事務室に関しては 28~57 行に示すように、VAV または CAV コントローラを介して制御する。また、34 行に示すように、始業時の 7:00 に関しては外気量を 0 にする外気カット制御を行う。10, 11 行と 17, 18 行はクールビズとウォームビズの場合の温度設定処理であり、これに関しては後述する

プログラム 28.8　空調制御の設定処理（空調機-熱負荷 連成）

SHASE_SimulationTest class

```
 1 public static void ControlAHUSystem(BuildingThermalModel bModel, AHUSystem ahuSystem)
 2 {
 3   //温湿度判定
 4   double dbt = 0;
 5   double hrt = 0;
 6   DateTime dt = bModel.CurrentDateTime;
 7   bool midSeason = false;
 8   if (6 <= dt.Month && dt.Month < 10)
 9   {
10     if (DO_COOLBIZ) dbt = 28;
11     else dbt = 26;
12     hrt = 0.00930;
13     for (int i = 0; i < 4; i++) ahuSystem.Controllers[i].Mode = AHUSystem.OperatingMode.Cooling;
14   }
15   else if (dt.Month == 12 || dt.Month < 4)
16   {
17     if (DO_COOLBIZ) dbt = 20;
18     else dbt = 22;
19     hrt = 0.00656;
20     for (int i = 0; i < 4; i++)
21     {
22       ahuSystem.Controllers[i].Mode = AHUSystem.OperatingMode.Heating;
23       ahuSystem.Controllers[i].MinimumHumidity = hrt;
24     }
25   }
26   else midSeason = true;
27
28   //事務室//CAV,VAV コントローラを制御
29   if ((IsHoliday(dt)) || (dt.Hour < 7 || 21 <= dt.Hour) || midSeason)
30     for (int i = 0; i < 4; i++) ahuSystem.Controllers[i].Mode = AHUSystem.OperatingMode.ShutOff;
31   else
32   {
33     //外気カットウォーミングアップ設定
34     if (dt.Hour == 7) for (int i = 0; i < 4; i++) ahuSystem.SetOutdoorAirFlow(i, 0, 0);
35     else
36     {
37       ahuSystem.SetOutdoorAirFlow(0, 1463d / 3600 * 1.2, 1463d / 3600 * 1.2);
38       ahuSystem.SetOutdoorAirFlow(1, 1513d / 3600 * 1.2, 1513d / 3600 * 1.2);
39       ahuSystem.SetOutdoorAirFlow(2, 1395d / 3600 * 1.2, 1395d / 3600 * 1.2);
40       ahuSystem.SetOutdoorAirFlow(3, 1125d / 3600 * 1.2, 1125d / 3600 * 1.2);
41     }
42
43     //ゾーン温度を設定(VAV 用)
44     ahuSystem.ControlZoneTemperature(0, 0, dbt);
45     ahuSystem.ControlZoneTemperature(1, 0, dbt);
46     ahuSystem.ControlZoneTemperature(1, 1, dbt);
47     ahuSystem.ControlZoneTemperature(1, 2, dbt);
48     ahuSystem.ControlZoneTemperature(2, 0, dbt);
49     ahuSystem.ControlZoneTemperature(3, 0, dbt);
```

```
50    ahuSystem.ControlZoneTemperature(3, 1, dbt);
51    //(CAV 用)
52    for (int i = 0; i < ahuSystem.AHUs.Length; i++)
53    {
54      ahuSystem.Controllers[i].IsRATemperatureControl = true;
55      ahuSystem.Controllers[i].SetpointTemperature = dbt;
56    }
57  }
58
59  //廊下・便所は直接に熱負荷計算
60  for (int i = 0; i < bModel.MultiRoom[2].ZoneNumber; i++)
61  {
62    bool ffloat = (IsHoliday(dt)) || (dt.Hour < 7 || 21 <= dt.Hour) || (i == 3) || midSeason;
63    if (ffloat)
64    {
65      bModel.ControlHeatSupply(2, i, 0);
66      bModel.ControlWaterSupply(2, i, 0);
67    }
68    else
69    {
70      bModel.ControlDrybulbTemperature(2, i, dbt);
71      bModel.ControlHumidityRatio(2, i, hrt);
72    }
73  }
74
75  //風の流れを設定
76  if (!IsHoliday(dt) && (dt.Hour < 8 || 21 <= dt.Hour))
77  {
78    bModel.SetAirFlow(0, 0, 2, 1, 985d / 3600 * 1.2);
79    bModel.SetAirFlow(1, 0, 2, 1, 940d / 3600 * 1.2);
80    bModel.SetAirFlow(2, 1, 2, 2, 1925d / 3600 * 1.2);
81    bModel.SetAirFlow(2, 2, 2, 3, 1925d / 3600 * 1.2);
82  }
83  else
84  {
85    const double cvRate = 150d * 1.2 / 3600d;
86    bModel.SetAirFlow(2, 1, 2, 2, 0);
87    bModel.SetAirFlow(2, 2, 2, 3, 0);
88    bModel.SetCrossVentilation(2, 1, 2, 2, 2.0 * cvRate);
89    bModel.SetCrossVentilation(2, 2, 2, 3, 2.0 * cvRate);
90  }
91
92  if (USE_CO2CNTRL) ApplyCO2Control(ahuSystem, bModel.CurrentDateTime);
93 }
```

プログラム 28.9 に年間熱負荷計算処理を示す。引数の真偽値で CAV と VAV の切り替えを可能とする。全体の流れはプログラム 28.6 と同様であるが、63 行に示すようにゾーンの状態は空調機の運転状態から計算する方式とする。従って、過負荷や過剰処理が表現でき、温度と湿度は必ずしも一定に維持されない。そこで、温熱環境の良否を確認するため、AHU の運転状態に加え、ゾーンの平均放射温度および PMV を計算して出力する（66~114 行）。

プログラム 28.9　年間熱負荷計算（空調機-熱負荷 連成）

```
 1 private static void SHASE_AHUTest(bool isCAVSystem)
 2 {
 3   //気象データ読み込み
 4   double[] dbt, hrt, dnr, dhr, ncr;
 5   SHASE_SimulationTest.LoadWeatherData("3639999.has", out dbt, out hrt, out dnr, out dhr, out ncr);
 6
 7   //建物モデル作成
 8   BuildingThermalModel bModel;
 9   AHUSystem ahuSystem;
10   SHASE_SimulationTest.MakeAHUSystem(out bModel, out ahuSystem, isCAVSystem);
11   Sun sun = new Sun(Sun.City.Tokyo);
12
13   using (StreamWriter sWriter = new StreamWriter("output.csv", false, Encoding.GetEncoding("Shift_JIS")))
14   {
15     //タイトル行書き出し
16     sWriter.Write(",,外気乾球温度,外気絶対湿度,法線面直達日射,水平面全天日射,夜間放射");
17     for (int j = 0; j < bModel.MultiRoom.Length; j++)
18     {
19       for (int k = 0; k < bModel.MultiRoom[j].ZoneNumber; k++)
20       {
21         ImmutableZone zn = bModel.MultiRoom[j].Zones[k];
22         sWriter.Write(
23           ", " + zn.Name + ":室温" + ", " + zn.Name + ":絶対湿度" +
```

```
24          ", " + zn.Name + ":平均放射温度" + ", " + zn.Name + ":PMV");
25        }
26      }
27      ImmutableAirHandlingUnit[] ahus = ahuSystem.AHUs;
28      for (int j = 0; j < ahus.Length; j++) sWriter.Write(",AHU" + j + ":冷熱処理");
29      for (int j = 0; j < ahus.Length; j++) sWriter.Write(",AHU" + j + ":温熱処理");
30      for (int j = 0; j < ahus.Length; j++) sWriter.Write(",AHU" + j + ":加湿量");
31      for (int j = 0; j < ahus.Length; j++) sWriter.Write(",AHU" + j + ":給気ファン動力");
32      for (int j = 0; j < ahus.Length; j++) sWriter.Write(",AHU" + j + ":還気ファン動力");
33      sWriter.WriteLine(",冷水流量,冷水還温度,冷熱負荷,温水流量,温水還温度,温熱負荷");
34
35      //12月を助走計算期間とする
36      DateTime dt = new DateTime(2006, 12, 1, 0, 0, 0);
37      for (int i = 0; i < 31 * 24; i++)
38      {
39        //気象条件更新
40        sun.SetGlobalHorizontalRadiation(dhr[8016 + i], dnr[8016 + i]);
41        sun.Update(dt.AddMinutes(30)); //過去一時間のデータのため30分シフト
42        bModel.UpdateOutdoorCondition(dt, sun, dbt[8016 + i], hrt[8016 + i], ncr[8016 + i]);
43
44        //熱平衡を更新
45        SHASE_SimulationTest.ControlAHUSystem(bModel, ahuSystem);
46        ahuSystem.ForecastReturnWaterTemperature(7, 50);
47        bModel.FixState();
48        dt = dt.AddHours(1);
49      }
50
51      //8760時間の計算実行
52      dt = new DateTime(2006, 1, 1, 0, 0, 0);
53      for (int i = 0; i < 8760; i++)
54      {
55        //気象条件更新
56        sun.SetGlobalHorizontalRadiation(dhr[i], dnr[i]);
57        sun.Update(dt.AddMinutes(30)); //過去一時間のデータのため30分シフト
58        bModel.UpdateOutdoorCondition(dt, sun, dbt[i], hrt[i], ncr[i]);
59        ahuSystem.OutdoorAir = new MoistAir(dbt[i], hrt[i]);
60
61        //空調制御//熱平衡を確定
62        SHASE_SimulationTest.ControlAHUSystem(bModel, ahuSystem);
63        ahuSystem.ForecastReturnWaterTemperature(7, 50);
64        bModel.FixState();
65
66        //書き出し
67        sWriter.Write(dt.ToShortDateString() + "," + dt.ToShortTimeString());
68        sWriter.Write("," + dbt[i] + "," + hrt[i] + "," + dnr[i] + "," + dhr[i] + "," + ncr[i]);
69        //着量を設定
70        double clo = 0.7; //中間期
71        if (SHASE_SimulationTest.DO_COOLBIZ)
72        {
73          if (6 <= dt.Month && dt.Month <= 9) clo = 0.5;  //夏季着衣量
74          else if (dt.Month == 12 || dt.Month <= 3) clo = 0.92;  //冬季着衣量
75        }
76        for (int j = 0; j < bModel.MultiRoom.Length; j++)
77        {
78          for (int k = 0; k < bModel.MultiRoom[j].ZoneNumber; k++)
79          {
80            //PMV計算
81            ImmutableZone zn = bModel.MultiRoom[j].Zones[k];
82            double tRad = zn.GetMeanSurfaceTemperature();
83            double rHmd = MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
84              (zn.Temperature, zn.HumidityRatio, 101.325);
85            double pmv = ThermalComfort.GetPMV(zn.Temperature, tRad, rHmd, 0.2, clo, 1.2, 0); //事務作業相当
86            if (6 <= dt.Month && dt.Month <= 9 && SHASE_SimulationTest.DO_COOLBIZ)
87              pmv = ThermalComfort.GetPMV(zn.Temperature, tRad, rHmd, 0.55, clo, 1.2, 0); //事務作業相当//CLBZ
88            sWriter.Write(", " + zn.Temperature + ", " + zn.HumidityRatio + ", " + tRad + ", " + pmv);
89          }
90        }
91
92        //冷水温水流量・還温度・負荷を計算
94        for (int j = 0; j < ahus.Length; j++) sWriter.Write("," + ahus[j].CoolingCoil.HeatTransfer);
95        for (int j = 0; j < ahus.Length; j++) sWriter.Write("," + ahus[j].HeatingCoil.HeatTransfer);
96        for (int j = 0; j < ahus.Length; j++) sWriter.Write("," + ahus[j].WaterConsumption);
97        for (int j = 0; j < ahus.Length; j++) sWriter.Write("," + ahus[j].SupplyAirFan.GetElectricConsumption());
98        for (int j = 0; j < ahus.Length; j++) sWriter.Write("," + ahus[j].ReturnAirFan.GetElectricConsumption());
99        double tco, tho, mcc, mhc, qc, qh;
100       tco = tho = mcc = mhc = qc = qh = 0;
101       for (int j = 0; j < ahus.Length; j++)
102       {
103         mcc += ahus[j].CoolingCoil.WaterFlowRate;
104         mhc += ahus[j].HeatingCoil.WaterFlowRate;
```

```
105          tco += ahus[j].CoolingCoil.WaterFlowRate * ahus[j].CoolingCoil.OutletWaterTemperature;
106          tho += ahus[j].HeatingCoil.WaterFlowRate * ahus[j].HeatingCoil.OutletWaterTemperature;
107          qc += ahus[j].CoolingCoil.HeatTransfer;
108          qh += ahus[j].HeatingCoil.HeatTransfer;
109        }
110        if (mcc == 0) tco = 7;
111        else tco /= mcc;
112        if (mhc == 0) tho = 50;
113        else tho /= mhc;
114        sWriter.WriteLine("," + mcc + "," + tco + "," + qc + "," + mhc + "," + tho + "," + qh);
115
116        if (dt.Hour == 0) Console.WriteLine(dt.ToShortDateString());
117        dt = dt.AddHours(1);
118      }
119    }
120 }
```

プログラム 28.9 を計算した結果を集計すると、空調機の年間負荷と搬送動力は図 28.7 となる。参考に ACSS, LCEM, BEST などの結果も記載した。VAV 制御による大きな搬送動力の削減が確認できる。

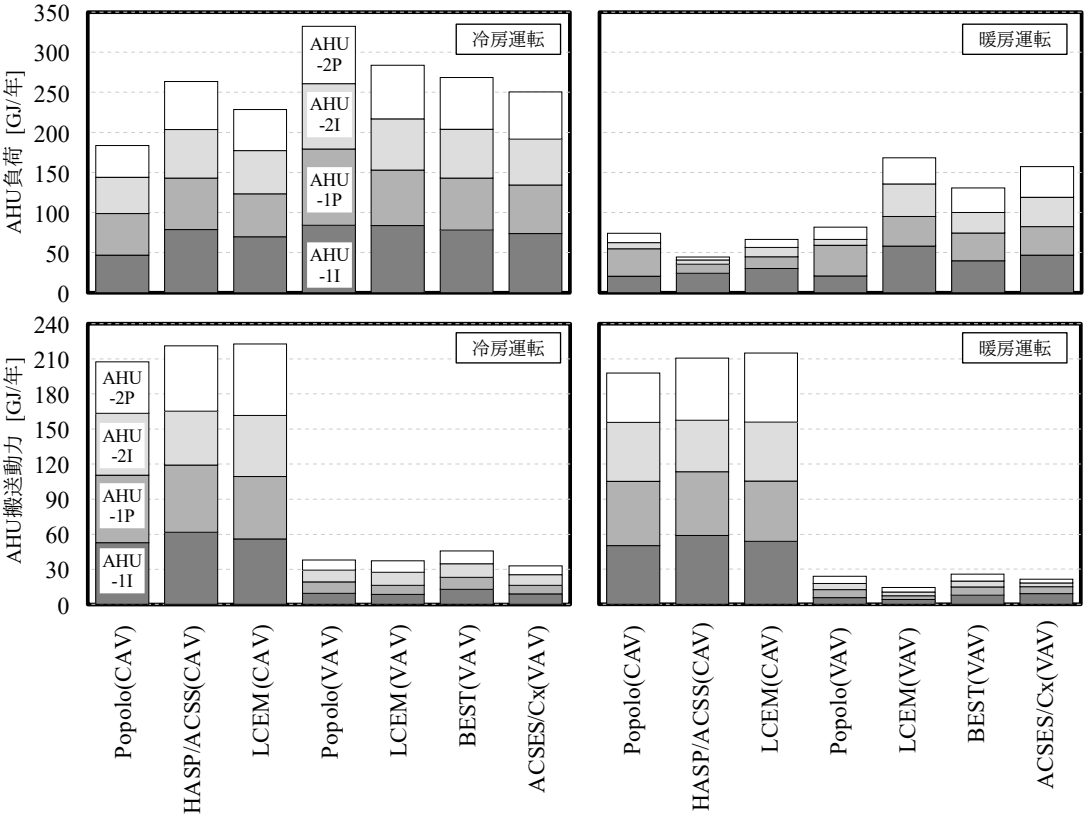


図 28.7　空調機の年間負荷と搬送動力

28.3　二次側空調システムの評価

28.3.1　PMV による温熱環境評価

プログラム 28.9 の結果（VAV 制御）にもとづき、冷房時と暖房時の各ゾーンの PMV を集計すると、図 28.8 に示す頻度分布が得られる。冷房時はやや正側に、暖房時はやや負側に寄っており、ISO の PMV 推奨範囲である±0.5 をやや外れている。ペリメータ部は外乱の影響を受けやすいため、冷房時、暖房時ともに、インテリア部の方が PMV 値が安定していることもわかる。

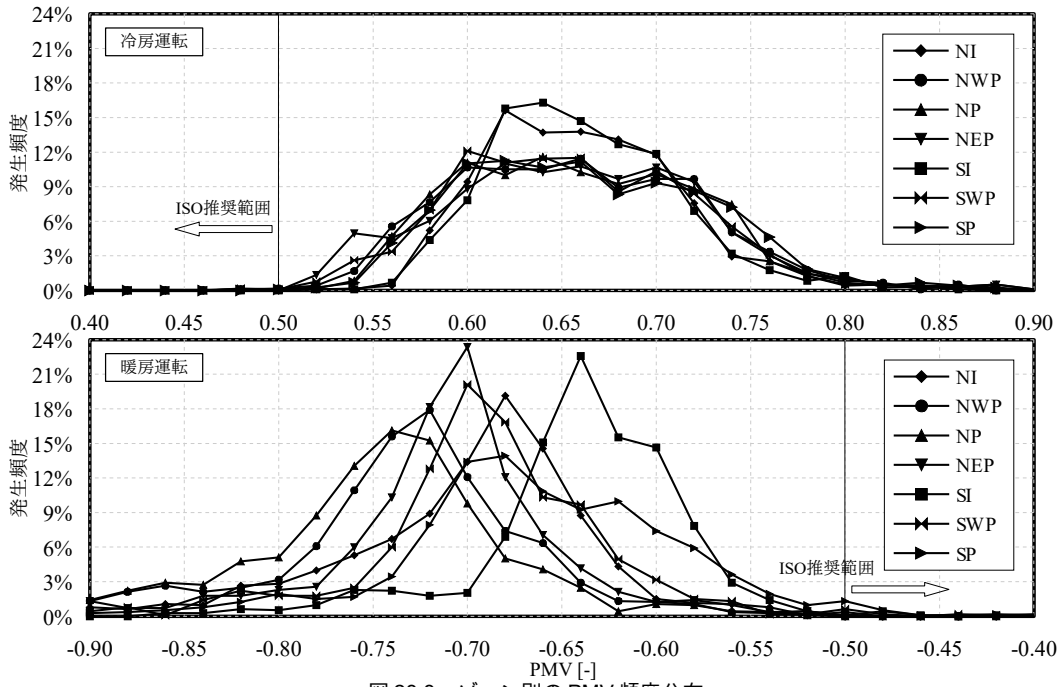


図 28.8　ゾーン別の PMV 頻度分布

28.3.2　空調設定温度の緩和

同等の温熱環境を確保したまま空調負荷を減少させる建物運用法としてクールビズまたはウォームビズという概念がある。夏季と冬季の空調設定温度をそれぞれ 28 ºC と 20 ºC に変更することで空調負荷を減少させ、同時に服装を薄着または厚着にすることで PMV を同等にするという狙いである。シミュレーション上でこのような運用の効果を確認する。

プログラム 28.1 の定数である DO_COOLBIZ を true にすることで、プログラム 28.8 の空調設定温度（10 行と 16 行）をそれぞれ 28 ºC と 20 ºC として年間熱負荷計算を実行する。ただし、プログラム 28.9 の 73, 74, 86, 87 行は削除して着衣量と気流速は据え置く。この結果、図 28.9 に示す PMV の分布（全ゾーン積算）が得られる。ゾーンの乾球温度が変化した影響を受けて冷房期の PMV は増大し、暖房期の PMV は低下し、それぞれ温熱環境が悪化する。注意すべきは暖房期に比べて冷房期の PMV 悪化の幅が大きい点である。暖房期の平均 PMV は-0.68 から-1.17 へと、約 0.49 悪化することに対し、冷房期の平均 PMV は 0.68 から 1.21 へと約 0.53 悪化する。これは設定温度の変更が相対湿度にも影響を与えたからである。図 28.10 に冷房期の相対湿度の頻度分布を示す。顕熱負荷が低下したことにより、空調機が処理する顕熱比が上昇し、絶対湿度が高い側にシフトすることがわかる。即ち、2 ºC の乾球温度の上昇に加えて相対湿度の上昇効果があるため、冷房期の設定温度緩和は熱的快適性に大きく影響を与える。

PMV 値の検討においては、平均放射温度=室温と簡略化して計算を行うことも多いが、現実の壁面は冬季であれば室よりも低く、夏季であれば室よりも高い。図 28.11 に NWP ゾーンにおける代表日の室温と平均放射温度の推移を示す。夏季においては室温は 26 ºC に保たれるが平均放射温度はこれよりも 1 ºC 程度高い値で推移することがわかる。冬季も同様であるが、日射の影響が夏と冬とで逆に作用するため、夏の PMV は朝と夕が比較的良好で、冬の PMV は昼の PMV が良い値となる。年間

を通じて空調時間帯の平均放射温度は、冷房時が 27.2 ºC、暖房時が 21.3 ºC となった。

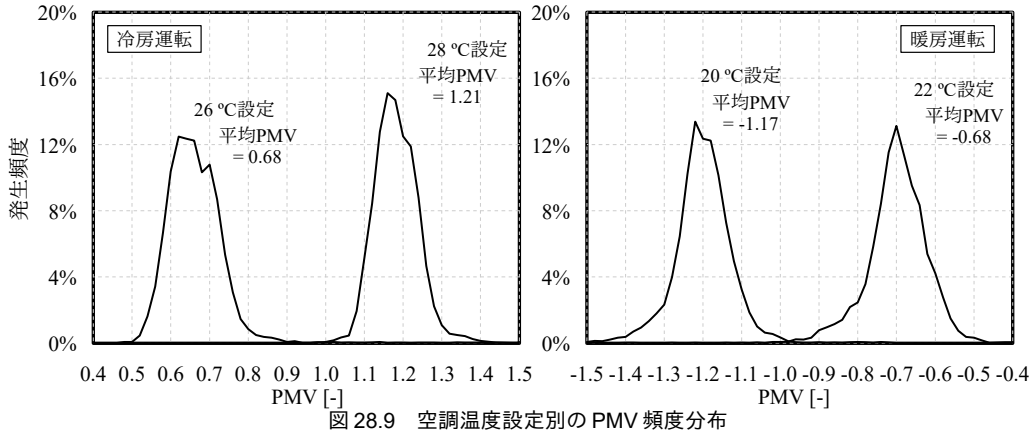


図 28.9　空調温度設定別の PMV 頻度分布

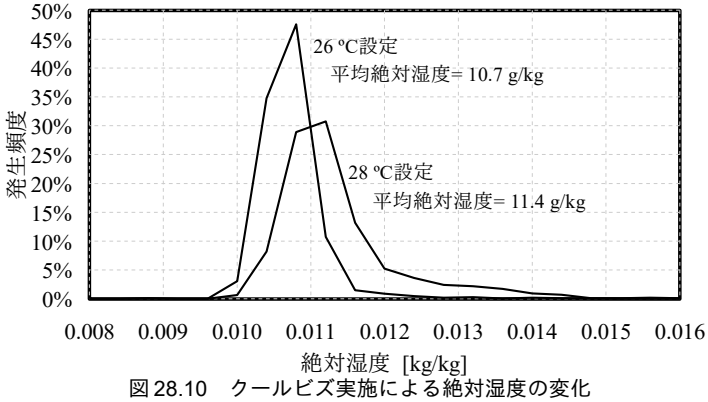


図 28.10　クールビズ実施による絶対湿度の変化

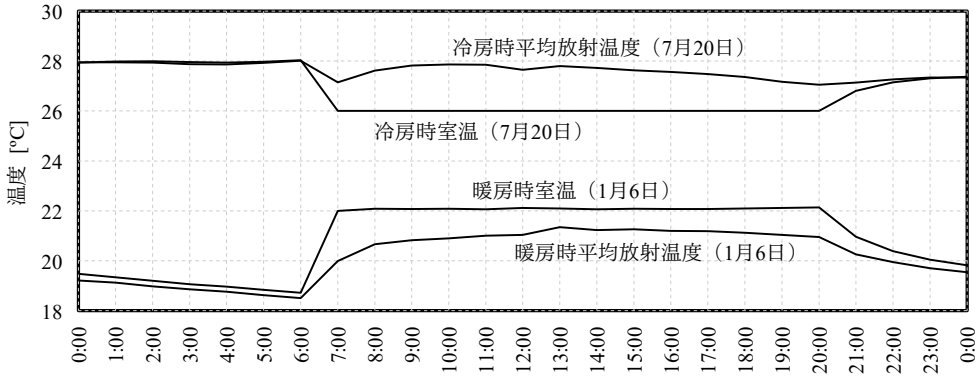


図 28.11　NWP ゾーンにおける代表日の室温と平均放射温度の推移

着衣量の変更によって温熱環境の改善を試みる。プログラム 28.10 にクールビズ・ウォームビズ時の着量を推定する処理（1~12 行）を示す。引数 1~4 で与えた温湿度および着衣条件での PMV 値を計算し、これと同等の PMV 値となる着衣量を二分法によって求める。ただし、比較対象の温湿度条件は引数 5~7 で与える。15 行と 18 行は、本例でのクールビズおよびウォームビズの計算例である。室温と平均放射温度との温度差は通常空調時と温度設定緩和時とで同じとした。

プログラム 28.10　クールビズ・ウォームビズ時の着衣量推定処理

```
 1 private static double estimateCLOValue
 2   (double tdb1, double trd1, double hrt1, double clo1, double tdb2, double trd2, double hrt2)
 3 {
 4   double rel1 = MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio(tdb1, hrt1, 101.325);
```

```
 5   double rel2 = MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio(tdb2, hrt2, 101.325);
 6   double pmv1 = ThermalComfort.GetPMV(tdb1, trd1, rel1, 0.2, clo1, 1.1, 0);
 7
 8   Roots.ErrorFunction eFnc = delegate (double clo2)
 9   { return pmv1 - ThermalComfort.GetPMV(tdb2, trd2, rel2, 0.2, clo2, 1.1, 0); };
10
11   return Roots.Bisection(eFnc, 0.01, 2.0, 1e-4, 1e-4, 50);
12 }
13
14 //クールビズの計算例
15 estimateCLOValue(26, 27.2, 0.0107, 0.7, 28, 29.2, 0.012);
16
17 //ウォームビズの計算例
18 estimateCLOValue(22, 21.3, 0.00656, 0.7, 20, 19.3, 0.00656);
```

計算を実行すると冷房時の着衣量は 0.30 clo、暖房時の着衣量は 0.92 clo となる。一般的なオフィスでの業務姿を前提とすると暖房時の 0.92 clo は可能だが、0.30 clo はかなり業種が制限される。参考に国土交通省の「クールビズ/ウォームビズ空調システム導入ガイドライン[28.3)]」に提示されたクールビズ・ウォームビズの着衣例を表 28.9 に示す[†1)]。そこで、夏季の着衣量は 0.5 とし、手元 USB ファンのような採涼アイテムを用いて気流感で調整を試みる。プログラム 28.11 にクールビズ時の気流速を推定する処理（1~12 行）を示す。プログラム 28.10 と同様の処理であり、二分法を用いて基準条件と PMV 値が一致する気流速を計算する。15 行を実行すると気流速=0.55 m/s が求まる[†2)]。

表 28.9　クールビズ・ウォームビズ時の着衣例

	基準		Cool BIZ		Warm BIZ	
	着衣	clo	着衣	clo	着衣	clo
男性	ブリーフ	0.04	ブリーフ	0.04	ブリーフ	0.04
	T シャツ	0.08	T シャツ	0.08	T シャツ	0.08
	靴下（ふくらはぎの長さ）	0.03	靴下（ふくらはぎの長さ）	0.03	靴下（ふくらはぎの長さ）	0.03
	ネクタイ	0.01	半袖シャツ	0.19	ネクタイ	0.01
	長袖シャツ	0.25	ズボン（薄手）	0.15	長袖シャツ	0.25
	ズボン（薄手）	0.15	靴	0.02	長袖セーター（薄手）	0.25
	靴	0.02	-	-	ズボン（薄手）	0.15
	-	-	-	-	靴	0.02
	合計	0.62	合計	0.51	合計	0.87
女性	ショーツ	0.03	ショーツ	0.03	ショーツ	0.03
	ブラ	0.01	ブラ	0.01	ブラ	0.01
	袖なしシャツ	0.08	袖なしシャツ	0.08	袖なしシャツ	0.08
	ストッキング	0.02	ストッキング	0.02	ストッキング	0.02
	長袖シャツ	0.25	半袖シャツ	0.19	長袖シャツ	0.25
	スカート（厚手）	0.23	スカート（薄手）	0.23	長袖セーター（薄手）	0.25
	サンダル	0.02	サンダル	0.02	スカート（厚手）	0.23
	-	-	-	-	サンダル	0.02
	合計	0.64	合計	0.58	合計	0.89

プログラム 28.11　クールビズ時の気流速推定処理

```
 1 private static double estimateVelocity(double tdb1, double trd1, double hrt1, double clo1, double vel1,
 2   double tdb2, double trd2, double hrt2, double clo2)
 3 {
 4   double rel1 = MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio(tdb1, hrt1, 101.325);
 5   double rel2 = MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio(tdb2, hrt2, 101.325);
 6   double pmv1 = ThermalComfort.GetPMV(tdb1, trd1, rel1, vel1, clo1, 1.1, 0);
 7
 8   Roots.ErrorFunction eFnc = delegate (double vel2)
 9   { return pmv1 - ThermalComfort.GetPMV(tdb2, trd2, rel2, vel2, clo2, 1.1, 0); };
10
11   return Roots.Bisection(eFnc, 0.01, 2.0, 1e-4, 1e-4, 50);
12 }
13
14 //クールビズの計算例
15 estimateVelocity(26, 27.2, 0.0107, 0.7, 0.2, 28, 29.2, 0.012, 0.5);
```

†1　clo 値は筆者が設定。

†2　第 27 章で解説したように、PMV は体の部位を考慮しないモデルであるため、ここで言う 0.55 m/s は全身の表面がくまなく 0.55m/s に晒されることを意味している。実際には扇風機周辺にある上半身の一部のみが冷却の対象となるであろうから、物理的にどの程度の風速が求められるかを具体的に把握したい場合には第 27 章の多節点人体モデルの検討が必要である。計算するより実験してしまった方が話が早そうではあるが。

プログラム 28.9 の 73 と 74 行を有効にしてクールビズとウォームビズ実施時に着衣量を 0.50 clo と 0.92 clo に変更する。夏季の気流速を 0.55m/s として PMV の分布を再計算すると図 28.12 が得られる。夏季、冬季ともにほぼ同等の PMV 値が得られることがわかる。ただし、大熊らは被験者実験行い、たとえ体全体としての着衣量が高くても現実には薄着とせざるをえない部位が残るため、冬期 20 ℃の環境においては局所的に寒さを感じることがあると報告している[28.5)]。

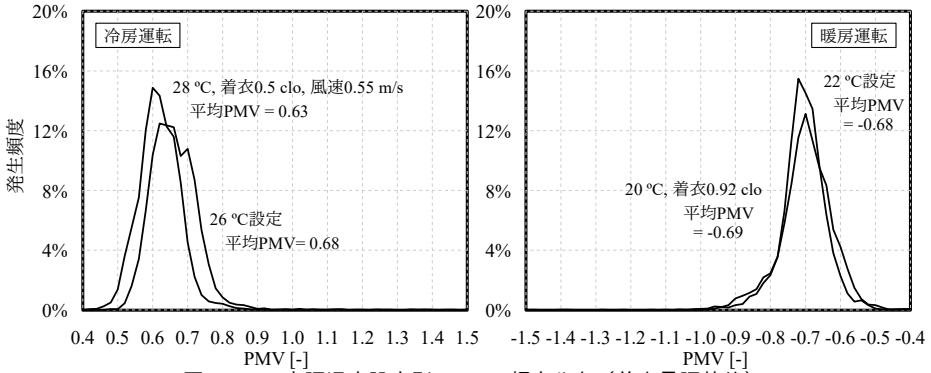


図 28.12　空調温度設定別の PMV 頻度分布（着衣量調整後）

図 28.13 に空調設定温度の緩和による空調負荷の変化を示す。年間では冷房負荷が 332→268 GJ/年（19%減）、暖房負荷が 82→53 GJ/年（35%減）となった。気流速の調整を行わず、乾球温度のみで PMV 値を同等に保った場合のエネルギー消費量に関しては崔らが報告を行っている[28.4)]。冬期 1.0 clo、夏期 0.6 clo に着衣を緩和した場合には、年間 0.8 clo で運用した場合に比較して約 10 %程度空調負荷が減少するとしている。温度設定値緩和の効果としては内外温度差減少による貫流熱への影響よりも外気負荷削減の方が大きいであろうから、換気量の設定によって効果は大きく変動するだろう。

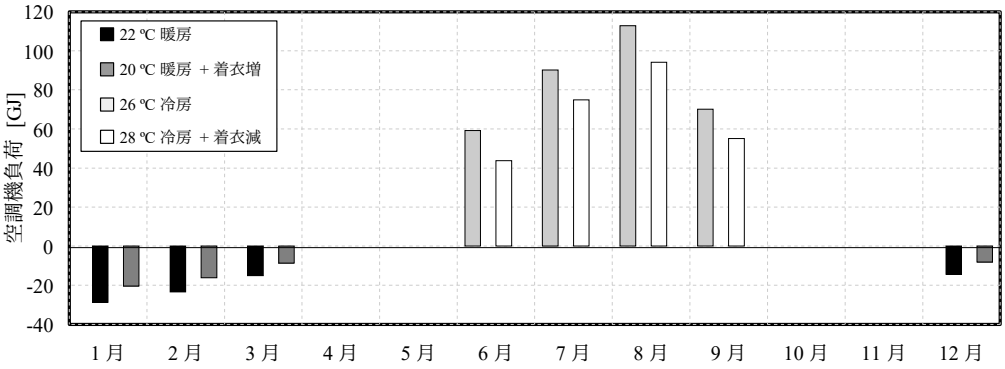


図 28.13　空調設定温度の緩和による空調負荷の変化

【第 28 章 参考文献】

28.1）　平成 25 年 省エネルギー基準に準拠した算定・判断の方法及び解説, I 非住宅建築物（第二版）, 財団法人建築環境・省エネルギー機構, 2013.05

28.2）　平成 25 年 エネルギーの使用の合理化に関する法律（昭和 54 年法律第 49 号）, エネルギーの使用の合理化に関する建築主等及び特定建築物の所有者の判断の基準（平成 25 年経済産業省・国土交通省告示第 1 号）

28.3）　国土交通省大臣官房官庁営繕部 設備・環境課, 官庁施設におけるクールビズ/ウォームビズ空調システム導入ガイドライン, H21.7

28.4）　崔軍, 渡辺俊行: 居住者の温冷感を考慮した空調負荷計算法に関する研究, -ウォームビズとクールビズによる負荷削減効果の定量的評価について- , 日本建築学会環境系論文集, 第 79 巻, 第 695 号, pp.73-81, 2014.01

28.5）　大熊涼子, 石野久彌, 中山哲士: オフィス着衣条件における冬期 20ºC 暖房の温熱環境, 日本建築学会学術講演梗概集, pp.1035-1036, 2006.09

第29章　建築熱環境システムの総合評価
(Comprehensive Evaluation of Building Thermal System)

29.1　概要

一次側の熱源設備システムモデルを用いてエネルギーを評価する方法に関しては第 II 編において解説した。また、建物熱負荷計算と空調機からなる二次側空調システムモデルを用いて室内温熱環境を評価する方法に関しては前章までに解説した。本章ではこれらの両システムを連成させて、エネルギーと温熱環境の両面から建物を総合的に評価する方法について解説を行う。

29.2　建物熱環境システムモデルの作成

29.2.1　連成計算の考え方

一次側システムと二次側システムは冷水および温水を媒介に熱をやり取りする。従って、両者を連成させるためには、冷水と温水の温度と流量の組み合わせが両モデルで成立している必要がある。流量を固定した場合、冷水および温水の温度と熱量の関係は図 29.1 の通りとなる。

冷水往温度が低ければ空調機において空気と冷水との間に生じる温度差が大きくなるため、二次側空調設備は、より多くの熱を処理することができる。従って図 29.1 左に示すように、冷水温度の低下に伴って二次側処理熱量は大きくなり、左上がりの線が描ける[†1)]。一方で冷水往温度が低くなると、冷凍機の能力は低下（圧縮式冷凍機における圧縮比の増大、吸収式冷凍機における溶液循環比の増大）するため、一次側熱源設備が製造可能な熱量は低下する。従って図 29.1 左に示すように、冷水温度の低下に伴って一次側製造熱量は小さくなり、左下がりの線が描ける。この関係を受けて 2 つの線が交わる点で往温度は実現する。温水の場合にはこの逆の傾向となり、図 29.1 右に示す関係となる。ただし、現実には一次側システムの熱源は冷水および温水の出口温度設定値を持っているため、この値を超えてまで能力を低下させるような運転状態にはならない[†2)]。すなわち、一次側熱源システムが冷水を設定温度まで下げられる場合、あるいは温水を設定温度まで上げられる場合には、その値が連成計算の解であり、この場合には収束計算は不要である。収束計算が発生する状態とは、一次側システムが過負荷となり、冷水温度が上昇または温水温度が低下してしまう状態である。

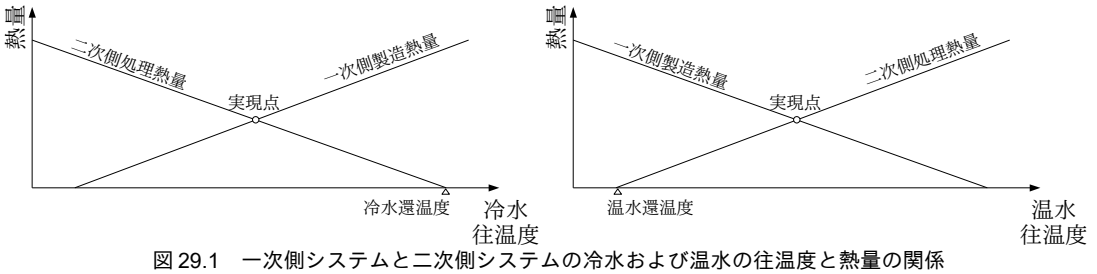


図 29.1　一次側システムと二次側システムの冷水および温水の往温度と熱量の関係

†1　図は説明のために傾向のみを示しており、現実には線形ではない。一次側システムの温度・熱量の関係も同様である。
†2　極めて負荷の小さい場合には製造した冷水（温水）が使用しきれずに冷水温度が低下（上昇）することはある。しかしこの場合にはもはやシステムは安定せず、熱源が異常停止するため、シミュレーションでは表現しない。

29.2.2　二次側空調システムインターフェースの作成

本書では第 26 章において、外気処理機能付の空気調和機による二次側空調システムのモデルについて解説したが、現実には二次側空調システムの構成は様々である。そこで、後々のプログラムの拡張を容易にするために、二次側空調システム一般を表わすインターフェースを定義する。プログラム 29.1 に二次側空調システムインターフェースを示す。37 行の ForecastReturnWaterTemperature メソッドが最も重要であり、本メソッドを用いて往温度から還温度と所要流量を予測し、プロパティを用いて計算結果を取得する。一次側システムモデルと連成するように繰り返し計算を行い、収束した時点で 40 行の FixState メソッドで状態を確定する。

プログラム 29.1　二次側空調システムインターフェース

Popolo.HVAC.SubSystem.IAirConditioningSystemModel interface

```
 1 /// <summary>二次側空調システムモデル</summary>
 2 public interface IAirConditioningSystemModel
 3 {
 4   /// <summary>現在の日時を設定・取得する</summary>
 5   DateTime CurrentDateTime { get; set; }
 6
 7   /// <summary>タイムステップを設定・取得する</summary>
 8   double TimeStep { get; set; }
 9
10   /// <summary>外気条件を設定・取得する</summary>
11   ImmutableMoistAir OutdoorAir { get; set; }
12
13   /// <summary>計算対象の建物熱負荷計算モデルを取得する</summary>
14   ImmutableBuildingThermalModel BuildingThermalModel { get; }
15
16   /// <summary>冷水往温度[C]を取得する</summary>
17   double ChilledWaterSupplyTemperature { get; }
18
19   /// <summary>冷水還温度[C]を取得する</summary>
20   double ChilledWaterReturnTemperature { get; }
21
22   /// <summary>冷水流量[kg/s]を取得する</summary>
23   double ChilledWaterFlowRate { get; }
24
25   /// <summary>温水往温度[C]を取得する</summary>
26   double HotWaterSupplyTemperature { get; }
27
28   /// <summary>温水還温度[C]を取得する</summary>
29   double HotWaterReturnTemperature { get; }
30
31   /// <summary>温水流量[kg/s]を取得する</summary>
32   double HotWaterFlowRate { get; }
33
34   /// <summary>冷温水還温度を予想する</summary>
35   /// <param name="chilledWaterSupplyTemperature">冷水往温度[C]</param>
36   /// <param name="hotWaterSupplyTemperature">温水往温度[C]</param>
37   void ForecastReturnWaterTemperature(double chilledWaterSupplyTemperature,double hotWaterSupplyTemperature);
38
39   /// <summary>状態を確定する</summary>
40   void FixState();
41 }
```

29.2.3　建物熱環境システムモデルの作成

一次側熱源システムである HeatSourceSystemModel クラス（第 21 章）、前節で作成した二次側空調システムである IAirConditioningSystemModel インターフェース、建物熱負荷を計算する BuidlingThermalModel クラス（第 25 章）の 3 つを連成させる建物熱環境システムモデル（HVACSystemModel class）を作成する。インスタンス変数、プロパティ、コンストラクタの定義をプログラム 29.2 に示す。

事務所ビルなどでは基準階があり、同じ構成の二次側空調システムが繰り返されることが多い。このような場合には 1 つの二次側空調システムモデルを解き、基準階の数だけ倍数をかければ計算負荷が大きく削減できる。2 行目の acFactor はこの倍数を保持する配列である。

76~88 行がコンストラクタであり、上記の 3 つのオブジェクトを引数にとる。1 つの建物に対して複数の二次側空調システムで対応する場合もあるため、IAirConditioningSystemModel は配列とする。それぞれの倍数は 1 で初期化する。

33 行と 52 行は温水と冷水の上下限温度である。図 29.1 に示したように過負荷の場合には温水温度が低下、あるいは冷水温度が上昇するが、際限なく温度変化が生じるとはせず、上下限値の範囲内で計算を行うことにする。

プログラム 29.2　インスタンス変数、プロパティ、コンストラクタの定義

Popolo.HVAC.HVACSystemModel class

```
 1 /// <summary>空調機系統の倍数</summary>
 2 private int[] acFactor;
 3
 4 /// <summary>現在の日時を取得する</summary>
 5 public DateTime CurrentDateTime { get; private set; }
 6
 7 /// <summary>計算時間間隔[sec]</summary>
 8 private double timeStep = 3600;
 9
10 /// <summary>計算時間間隔[sec]を設定・取得する</summary>
11 public double TimeStep
12 {
13   get { return timeStep; }
14   set
15   {
16     timeStep = value;
17     bModel.TimeStep = value;
18     foreach (IAirConditioningSystemModel acs in acModels) acs.TimeStep = value;
19     hsModel.TimeStep = value;
20   }
21 }
22
23 /// <summary>建物熱負荷計算モデル</summary>
24 private BuildingThermalModel bModel;
25
26 /// <summary>二次側空調システムモデルリスト</summary>
27 private IAirConditioningSystemModel[] acModels;
28
29 /// <summary>熱源システムモデル</summary>
30 private HeatSourceSystemModel hsModel;
31
32 /// <summary>温水下限温度[C]を設定・取得する</summary>
33 public double HotWaterLowerLimitTemperature { get; set; } = 35;
34
35 /// <summary>温水往温度設定値[C]を設定・取得する</summary>
36 public double HotWaterSupplyTemperatureSetpoint
37 {
38   get { return hsModel.HotWaterSupplyTemperatureSetpoint; }
39   set { hsModel.HotWaterSupplyTemperatureSetpoint = value; }
40 }
41
42 /// <summary>温水往温度[C]を取得する</summary>
43 public double HotWaterSupplyTemperature { get; private set; } = 45;
44
45 /// <summary>温水還温度[C]を取得する</summary>
46 public double HotWaterReturnTemperature { get; private set; } = 40;
47
48 /// <summary>温水流量[kg/s]を取得する</summary>
49 public double HotWaterFlowRate { get; private set; }
50
51 /// <summary>冷水上限温度[C]を設定・取得する</summary>
52 public double ChilledWaterUpperLimitTemperature { get; set; } = 15;
53
54 /// <summary>冷水往温度設定値[C]を設定・取得する</summary>
55 public double ChilledWaterSupplyTemperatureSetpoint
56 {
57   get { return hsModel.ChilledWaterSupplyTemperatureSetpoint; }
58   set { hsModel.ChilledWaterSupplyTemperatureSetpoint = value; }
59 }
60
61 /// <summary>冷水往温度[C]を取得する</summary>
62 public double ChilledWaterSupplyTemperature { get; private set; } = 7;
63
64 /// <summary>冷水還温度[C]を取得する</summary>
```

```
65 public double ChilledWaterReturnTemperature { get; private set; } = 12;
66
67 /// <summary>冷水流量[kg/s]を取得する</summary>
68 public double ChilledWaterFlowRate { get; private set; }
69
70 /// <summary>未処理加熱負荷[kW]を取得する</summary>
71 public double RemainingHeatingLoad { get; private set; }
72
73 /// <summary>未処理冷却負荷[kW]を取得する</summary>
74 public double RemainingCoolingLoad { get; private set; }
75
76 /// <summary>インスタンスを初期化する</summary>
77 /// <param name="bModel">建物熱負荷計算モデル</param>
78 /// <param name="acModels">二次側空調システムモデルリスト</param>
79 /// <param name="hsModel">熱源システムモデル</param>
80 public HVACSystemModel
81   (BuildingThermalModel bModel, IAirConditioningSystemModel[] acModels, HeatSourceSystemModel hsModel)
82 {
83   this.bModel = bModel;
84   this.acModels = acModels;
85   this.hsModel = hsModel;
86   acFactor = new int[acModels.Length];
87   for (int i = 0; i < acFactor.Length; i++) acFactor[i] = 1;
88 }
```

プログラム 29.3 に状態設定処理を示す。1~20 行は外気条件の設定処理である。日時、太陽情報、外気温湿度、夜間放射を建物熱負荷計算モデル、二次側空調モデル、一次側熱源モデルに設定する。22~26 行は空調システムの倍数設定処理である。

プログラム 29.3　状態設定処理

Popolo.HVAC.HVACSystemModel class

```
 1 /// <summary>外気条件を更新する</summary>
 2 /// <param name="dTime">現在の日時</param>
 3 /// <param name="sun">太陽</param>
 4 /// <param name="temperature">外気乾球温度[C]</param>
 5 /// <param name="humidityRatio">外気絶対湿度[kg/kg]</param>
 6 /// <param name="nocRadiation">夜間放射[W/m2]</param>
 7 public void UpdateOutdoorCondition
 8   (DateTime dTime, ImmutableSun sun, double temperature, double humidityRatio, double nocRadiation)
 9 {
10   MoistAir outdoorAir = new MoistAir(temperature, humidityRatio);
11   CurrentDateTime = dTime;
12   bModel.UpdateOutdoorCondition(dTime, sun, temperature, humidityRatio, nocRadiation);
13   foreach (IAirConditioningSystemModel acs in acModels)
14   {
15     acs.CurrentDateTime = dTime;
16     acs.OutdoorAir = outdoorAir;
17   }
18   hsModel.CurrentDateTime = dTime;
19   hsModel.OutdoorAir = outdoorAir;
20 }
21
22 /// <summary>空調システムの倍数を設定する</summary>
23 /// <param name="airConditioningSystemIndex">空調システム系統番号</param>
24 /// <param name="factor">倍数[-]</param>
25 public void SetACFactor(int airConditioningSystemIndex, int factor)
26 { acFactor[airConditioningSystemIndex] = factor; }
```

プログラム 29.4 に状態更新処理を示す。

1~20 行は冷水および温水の往温度にもとづき、還温度と所要流量を集計する処理である。7~15 行で各二次側空調システムの将来状態予測を行い、還温度と流量の積和を計算する。16~19 行で流量で除することで平均温度を求める。

22~95 行が状態更新処理である。まず 29 行で、往温度設定値が実現できると仮定した場合の還温度と流量を計算する。この値を用いて 32 行で一次側システムの計算を行う。当初仮定した通りに往温度が設定値となれば計算は終了であるが、過負荷の場合に対応するために 35~61 行（冷水）と 63~89 行（温水）で過負荷判定を行う。

36 行で冷却能力過不足を確認し、過負荷でなければ往温度=往温度設定値とする（61 行）。過負荷

の場合には温度上昇が生じる。この場合には、まず上限値で計算を行い（39 行、40 行）、上限値でも過負荷が続く場合には不足する能力を未処理負荷として計上する（44~46 行）。上限温度で処理可能である場合には、往温度設定値から上限値までの間に実現する温度があるということであるから、二分法を用いて収束計算で往温度を求める（50~58 行）。加熱運転に関しては往温度と往温度設定値の大小関係が冷却運転と逆になるだけで、計算処理は同様である。

冷水および温水の往温度を求めた後、91~94 行で状態を確定する。

プログラム 29.4　状態更新処理

Popolo.HVAC.HVACSystemModel class

```
 1 /// <summary>冷温水の還温度[C]を計算する</summary>
 2 /// <param name="tcwSP">冷水往温度</param>
 3 /// <param name="thwSP">温水往温度</param>
 4 private void calcReturnWaterState(double tcwSP, double thwSP)
 5 {
 6   ChilledWaterFlowRate = HotWaterFlowRate = ChilledWaterReturnTemperature = HotWaterReturnTemperature = 0;
 7   for (int i = 0; i < acModels.Length; i++)
 8   {
 9     IAirConditioningSystemModel acm = acModels[i];
10     acm.ForecastReturnWaterTemperature(tcwSP, thwSP);
11     ChilledWaterReturnTemperature += acm.ChilledWaterReturnTemperature * acm.ChilledWaterFlowRate*acFactor[i];
12     HotWaterReturnTemperature += acm.HotWaterReturnTemperature * acm.HotWaterFlowRate * acFactor[i];
13     ChilledWaterFlowRate += acm.ChilledWaterFlowRate * acFactor[i];
14     HotWaterFlowRate += acm.HotWaterFlowRate * acFactor[i];
15   }
16   if (ChilledWaterFlowRate == 0) ChilledWaterReturnTemperature = tcwSP;
17   else ChilledWaterReturnTemperature /= ChilledWaterFlowRate;
18   if (HotWaterFlowRate == 0) HotWaterReturnTemperature = thwSP;
19   else HotWaterReturnTemperature /= HotWaterFlowRate;
20 }
21
22 /// <summary>状態を更新する</summary>
23 public void Update()
24 {
25   //未処理負荷初期化
26   RemainingCoolingLoad = RemainingHeatingLoad = 0;
27
28   //二次側からの還温度を計算
29   calcReturnWaterState(ChilledWaterSupplyTemperatureSetpoint, HotWaterSupplyTemperatureSetpoint);
30
31   //一次側過負荷判定
32   hsModel.ForecastSupplyWaterTemperature
33     (ChilledWaterFlowRate, ChilledWaterReturnTemperature, HotWaterFlowRate, HotWaterReturnTemperature);
34
35   //冷却能力不足の場合
36   if (hsModel.IsOverLoad_C)
37   {
38     //往温度上限値で計算
39     calcReturnWaterState(ChilledWaterUpperLimitTemperature, HotWaterSupplyTemperatureSetpoint);
40     hsModel.ForecastSupplyWaterTemperature
41       (ChilledWaterFlowRate, ChilledWaterReturnTemperature, HotWaterFlowRate, HotWaterReturnTemperature);
42
43     //未処理負荷発生
44     if (ChilledWaterUpperLimitTemperature < hsModel.ChilledWaterSupplyTemperature)
45       RemainingCoolingLoad = ChilledWaterFlowRate * WATER_SPECIFIC_HEAT *
46         (hsModel.ChilledWaterSupplyTemperature - ChilledWaterUpperLimitTemperature);
47     //往温度上昇
48     else
49     {
50       Roots.ErrorFunction eFnc = delegate (double tcSply)
51       {
52         calcReturnWaterState(tcSply, HotWaterSupplyTemperatureSetpoint);
53         hsModel.ForecastSupplyWaterTemperature
54         (ChilledWaterFlowRate, ChilledWaterReturnTemperature, HotWaterFlowRate, HotWaterReturnTemperature);
55         return tcSply - hsModel.ChilledWaterSupplyTemperature;
56       };
57       ChilledWaterSupplyTemperature = Roots.Bisection
58         (eFnc, ChilledWaterSupplyTemperatureSetpoint, ChilledWaterUpperLimitTemperature, 0.01, 0.01, 10);
59     }
60   }
61   else ChilledWaterSupplyTemperature = ChilledWaterSupplyTemperatureSetpoint;
62
63   //加熱能力不足の場合
64   if (hsModel.IsOverLoad_H)
```

```
65  {
66    //往温度下限値で計算
67    calcReturnWaterState(ChilledWaterSupplyTemperature, HotWaterLowerLimitTemperature);
68    hsModel.ForecastSupplyWaterTemperature
69      (ChilledWaterFlowRate, ChilledWaterReturnTemperature, HotWaterFlowRate, HotWaterReturnTemperature);
70
71    //未処理負荷発生
72    if (hsModel.HotWaterSupplyTemperature < HotWaterLowerLimitTemperature)
73      RemainingHeatingLoad = HotWaterFlowRate * WATER_SPECIFIC_HEAT *
74        (HotWaterLowerLimitTemperature - hsModel.HotWaterSupplyTemperature);
75    //往温度低下
76    else
77    {
78      Roots.ErrorFunction eFnc = delegate (double tcSply)
79      {
80        calcReturnWaterState(ChilledWaterSupplyTemperature, tcSply);
81        hsModel.ForecastSupplyWaterTemperature
82        (ChilledWaterFlowRate, ChilledWaterReturnTemperature, HotWaterFlowRate, HotWaterReturnTemperature);
83        return tcSply - hsModel.HotWaterSupplyTemperature;
84      };
85      HotWaterSupplyTemperature = Roots.Bisection
86        (eFnc, HotWaterLowerLimitTemperature, HotWaterSupplyTemperatureSetpoint, 0.01, 0.01, 10);
87    }
88  }
89  else HotWaterSupplyTemperature = HotWaterSupplyTemperatureSetpoint;
90
91  //状態確定
92  hsModel.FixState();
93  foreach (IAirConditioningSystemModel ac in acModels) ac.FixState();
94  bModel.FixState();
95 }
```

29.3　建築熱環境システムの総合評価

29.3.1　基準システムの作成

第 28 章で作成した事務所ビルの二次側空調モデルと第 21 章の例題 21.2 で作成した一次側熱源システムモデルを組み合わせ、基準となる建物熱環境システムモデルを作成する。プログラム 29.5 にモデルの作成処理を示す。3~13 行は二次側および一次側システムモデルの作成処理であり、これらを引数に与えて 16 行で建築熱環境システムモデルを作成する。基準階型のオフィスビルであり、エントランスホールなどを除く 2~7 階の 6 層を中央熱源システムで空調する。従って、20 行に示すように二次側空調システムの倍率を 6 倍に設定する。

25~141 行が各時刻の熱環境計算処理である。既に第 21 章と第 28 章で解説した二次側と一次側の個別の年間計算処理とほぼ同様であり、12 月のデータで助走計算を行った後、1 年間の各時刻の計算を行い、CSV 形式で出力する。

プログラム 29.5　基準となる建物熱環境システムモデルの作成処理

```
 1 private static void SHASE_HVACSystemTest(string outputFile)
 2 {
 3   //建物モデル作成
 4   BuildingThermalModel bModel;
 5   AHUSystem ahuSystem;
 6   SHASE_SimulationTest.MakeAHUSystem(out bModel, out ahuSystem, false);
 7   Sun sun = new Sun(Sun.City.Tokyo);
 8
 9   //熱源モデル作成
10   HeatSourceSystemModel hss;
11   AirHeatSourceModularChillersSystem ahSystem;
12   DirectFiredAbsorptionChillerSystem arSystem;
13   SHASE_SimulationTest.MakeHeatSourceSystem(out hss, out ahSystem, out arSystem);
14
15   //HVAC モデル作成
16   HVACSystemModel hvacSystem = new HVACSystemModel
17     (bModel, new IAirConditioningSystemModel[] { ahuSystem }, hss);
18   hvacSystem.ChilledWaterSupplyTemperatureSetpoint = 7;
19   hvacSystem.HotWaterSupplyTemperatureSetpoint = 50;
20   hvacSystem.SetACFactor(0, 6); //2F-7F
```

```
21
22  //気象データ読み込み
23  double[] dbt, hrt, dnr, dhr, ncr;
24  //上記配列に気象データを読み込む処理
25  using (StreamWriter sWriter = new StreamWriter(outputFile, false, Encoding.GetEncoding("Shift_JIS")))
26  {
27    //タイトル行書き出し
28    //外界条件
29    sWriter.Write(",,外気乾球温度,外気絶対湿度,法線面直達日射,水平面全天日射,夜間放射");
30    //一次側
31    sWriter.Write(", 冷水往温度, 冷水還温度, 温水往温度, 温水還温度");
32    sWriter.Write(", AHP 電力, AHP 冷水ポンプ電力, AHP 温水ポンプ電力, AHP 冷却, AHP 加熱, AHP 台数");
33    sWriter.Write(", AR ガス, AR 電力, AR 冷水ポンプ電力, AR 温水ポンプ電力");
34    sWriter.Write(", AR 冷却水ポンプ電力, AR 冷却塔電力, AR 冷却, AR 加熱");
35    sWriter.Write(", 冷水二次ポンプ電力, 温水二次ポンプ電力");
36    sWriter.Write(", 二次側冷水流量, 二次側温水流量, 冷水バイパス流量, 温水バイパス流量");
37    //二次側
38    for (int j = 0; j < bModel.MultiRoom.Length; j++)
39    {
40      for (int k = 0; k < bModel.MultiRoom[j].ZoneNumber; k++)
41      {
42        string nm = bModel.MultiRoom[j].Zones[k].Name;
43        sWriter.Write(", " + nm + ":室温, " + nm + ":絶対湿度, " + nm + ":平均放射温度," + nm + ":PMV");
44      }
45    }
46    ImmutableAirHandlingUnit[] ahus = ahuSystem.AHUs;
47    for (int j = 0; j < ahus.Length; j++)
48    {
49      string sb = ",AHU" + j;
50      sWriter.Write(sb + ":冷却" + sb + ":加熱" + sb + ":加湿" + sb + ":SA ファン" + sb + ":RA ファン");
51    }
52    sWriter.WriteLine();
53
54    //12 月を助走計算期間とする
55    DateTime dt = new DateTime(2006, 12, 1, 0, 0, 0);
56    for (int i = 0; i < 31 * 24; i++)
57    {
58      //気象条件更新
59      sun.SetGlobalHorizontalRadiation(dhr[8016 + i], dnr[8016 + i]);
60      sun.Update(dt.AddMinutes(30)); //過去一時間のデータのため 30 分シフト
61      hvacSystem.UpdateOutdoorCondition(dt, sun, dbt[8016 + i], hrt[8016 + i], ncr[8016 + i]);
62
63      //熱平衡を更新
64      SHASE_SimulationTest.ControlAHUSystem(bModel, ahuSystem);
65      SHASE_SimulationTest.ControlHeatSourceSystem(hss);
66      hvacSystem.Update();
67      dt = dt.AddHours(1);
68    }
69
70    //8760 時間の計算実行
71    dt = new DateTime(2006, 1, 1, 0, 0, 0);
72    for (int i = 0; i < 8760; i++)
73    {
74      //気象条件更新
75      sun.SetGlobalHorizontalRadiation(dhr[i], dnr[i]);
76      sun.Update(dt.AddMinutes(30)); //過去一時間のデータのため 30 分シフト
77      hvacSystem.UpdateOutdoorCondition(dt, sun, dbt[i], hrt[i], ncr[i]);
78
79      //状態更新
80      SHASE_SimulationTest.ControlAHUSystem(bModel, ahuSystem);
81      SHASE_SimulationTest.ControlHeatSourceSystem(hss);
82      hvacSystem.Update();
83
84      //書き出し
85      //外界条件
86      sWriter.Write(dt.ToShortDateString() + "," + dt.ToShortTimeString());
87      sWriter.Write("," + dbt[i] + "," + hrt[i] + "," + dnr[i] + "," + dhr[i] + "," + ncr[i]);
88      //一次側
89      sWriter.Write(
90        "," + hvacSystem.ChilledWaterSupplyTemperature +
91        "," + hvacSystem.ChilledWaterReturnTemperature +
92        "," + hvacSystem.HotWaterSupplyTemperature +
93        "," + hvacSystem.HotWaterReturnTemperature);
94      ImmutableAirHeatSourceModularChillers ahp = ahSystem.AirHeatSourceModularChillers;
95      sWriter.Write(
96        "," + (ahp.ElectricConsumption * ahp.OperatingNumber) +
97        "," + (ahSystem.ChilledWaterPump.GetElectricConsumption() +
98        "," + ahSystem.HotWaterPump.GetElectricConsumption()) +
99        "," + ahp.CoolingLoad + "," + ahp.HeatingLoad + "," + ahSystem.OperatingChillerNumber);
101     ImmutableDirectFiredAbsorptionChiller ar = arSystem.DirectFiredAbsorptionChiller;
```

```
102       sWriter.Write(
103         "," + ar.FuelConsumption + "," + ar.ElectricConsumption +
104         "," + (arSystem.ChilledWaterPump.GetElectricConsumption() +
105         "," + arSystem.HotWaterPump.GetElectricConsumption()) +
106         "," + arSystem.CoolingWaterPump.GetElectricConsumption() +
107         "," + arSystem.CoolingTower.ElectricConsumption +
108         "," + ar.CoolingLoad + "," + ar.HeatingLoad);
109       ImmutablePumpSystem ps2c = hss.ChilledWaterPumpSystem;
110       ImmutablePumpSystem ps2h = hss.HotWaterPumpSystem;
111       sWriter.Write(
112         "," + ps2c.GetElectricConsumption() + "," + ps2h.GetElectricConsumption() +
113         "," + (ps2c.TotalFlowRate * 1000) + "," + (ps2h.TotalFlowRate * 1000) +
114         "," + hss.ChilledWaterBypassFlowRate + "," + hss.HotWaterBypassFlowRate);
115       //二次側
116       double clo = 0.7; //中間期着衣量
117       if (6 <= dt.Month && dt.Month <= 9) clo = 0.5;  //夏季着衣量
118       else if (dt.Month == 12 || dt.Month <= 3) clo = 0.92;  //冬季着衣量
119       for (int j = 0; j < bModel.MultiRoom.Length; j++)
120       {
121         for (int k = 0; k < bModel.MultiRoom[j].ZoneNumber; k++)
122         {
123           ImmutableZone zn = bModel.MultiRoom[j].Zones[k];
124           double tRad = zn.GetMeanSurfaceTemperature();
125           double rHmd = MoistAir.GetRelativeHumidityFromDryBulbTemperatureAndHumidityRatio
126             (zn.Temperature, zn.HumidityRatio, 101.325);
127           double pmv = ThermalComfort.GetPMV(zn.Temperature, tRad, rHmd, 0.2, clo, 1.2, 0);
128           sWriter.Write(", " + zn.Temperature + ", " + zn.HumidityRatio + ", " + tRad + ", " + pmv);
129         }
130       }
131       for (int j = 0; j < ahus.Length; j++)
132         sWriter.Write(
133           "," + ahus[j].CoolingCoil.HeatTransfer + "," + (-ahus[j].HeatingCoil.HeatTransfer) +
134           "," + ahus[j].WaterConsumption + "," + ahus[j].SupplyAirFan.GetElectricConsumption() +
135           "," + ahus[j].ReturnAirFan.GetElectricConsumption());
136       sWriter.WriteLine();
137
138       if (dt.Hour == 0) Console.WriteLine(dt.ToShortDateString());
139       dt = dt.AddHours(1);
140     }
141   }
142 }
```

計算結果のエネルギー消費量を集計すると図 29.2 が得られる。空気熱源ヒートポンプが 50 %超、吸収式冷温水機が約 15 %であり、熱源のエネルギー消費の大きさが確認できる。ポンプや冷却塔などの熱源補機類が約 15 %、空気搬送が 15 %である。ただし、この内訳は計算対象である空調室のみのものであり、現実のオフィスビルでは非空調室の換気ファン（多くの場合は INV 制御などは行わず固定値で電力を消費する）が存在するため、搬送系の消費エネルギーの割合はもう少し高くなる。

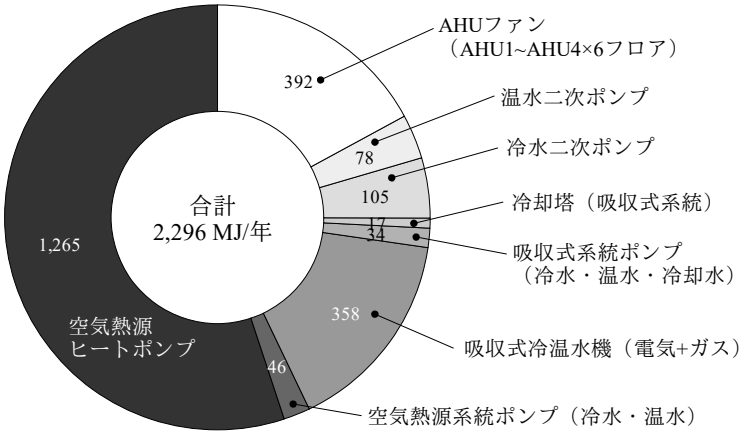


図 29.2　基準ケースの年間一次エネルギー消費量の内訳

熱的快適性の面から評価を行うため、放射熱環境が比較的厳しい（南面のために直射日光があたる）と推測できる SP ゾーンの PMV 頻度分布を作成した結果を図 29.3 に示す。ただし、居住者がいる 8:00~20:59 までのデータのみを対象とした。第 28 章で計算した結果とほぼ同様であり、ISO の推

奨範囲である±0.5 をやや逸脱している。

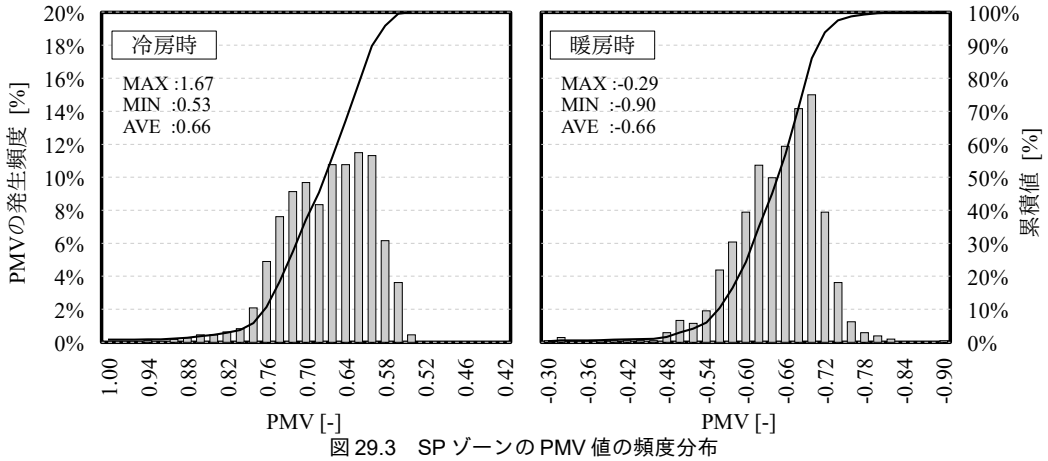


図 29.3　SP ゾーンの PMV 値の頻度分布

室内環境および設備システムの挙動を時系列で把握するため、6 月 1 日の時刻別の SP ゾーンの乾球温度と冷温水往還温度をグラフ化した結果を図 29.4 に示す。本計算では中間期である 4 月 1 日から 5 月 31 日までは非空調としているため、6 月 1 日時点では室内がかなり高温になる。従って空調開始時に大きな蓄熱負荷が生じ、7~8 時台は熱源システムが過負荷状態となり、往温度設定値である 7 ℃を満足できない。当然、ゾーンの温度も設定値である 26 ℃まで下がらず、PMV 値も高くなり、年間の最大値である 1.67 はこの日の 8 時に発生する。9 時以降は熱源システムは 7 ℃の冷水供給が可能となるが、二次側システムの空調機が過負荷状態となるために 17 時までは設定値を満足できない[†1]。

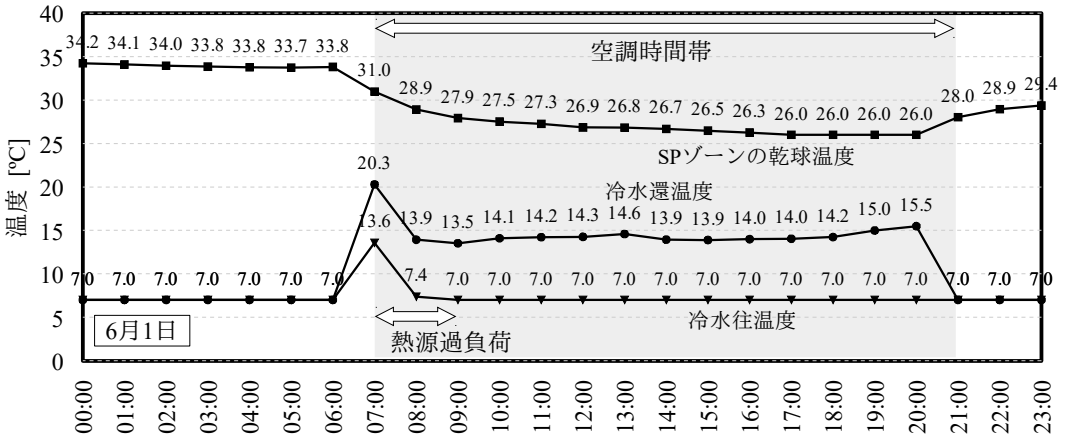


図 29.4　SP ゾーンの乾球温度と冷水往還温度の時刻別データ（6 月 1 日）

29.3.2　建物・設備仕様と運用の変更と効果の把握

前節で作成した基準システムに対して、建物・設備仕様と運用の変更を行うことで省エネルギー化と温熱環境の改善を試みる。

1）建物・設備仕様と運用の変更

変更内容は下記のとおりである。

・外皮断熱性能の強化

断熱材を押出ポリスチレンフォーム 1 種ではなく 3 種とし、熱伝導率を 0.040 W/(m·K)から 0.028

†1　現実には中間期であっても暑くて仕事ができなければ空調を開始して室温を下げるだろうから、このような非常に大きな蓄熱負荷が生じる状況は考えられない。過負荷状況のシミュレーション上の挙動を理解するために都合が良いために取り上げた。

W/(m·K)に変更する。窓ガラスを透明フロートシングルガラスから LowE ペアガラスに変更する。

・CO2 制御の導入

CO2 制御により AHU の外気導入量を人員密度に比例させて増減させる。外気導入量の調整処理をプログラム 29.6 に示す。人員密度は第 28 章の図 28.4 で示したとおりであり、このスケジュールに従って外気導入量の低減率を 4~9 行で特定し、各 AHU の外気導入量下限値を 11~19 行で設定する。外気導入量自体ではなく下限値を設定する理由は、外気条件によっては外気冷房が成立して積極的に導入量を増やしたい場合があるからである。

プログラム 29.6　CO2 制御による AHU 外気導入量の調整

SHASE_SimulationTest class

```
 1 public static void ApplyCO2Control(AHUSystem ahuSystem, DateTime dTime)
 2 {
 3   double oaRatio;
 4   if (IsHoliday(dTime) || (dTime.Hour < 8 || 21 <= dTime.Hour)) oaRatio = 0;
 5   else if (20 <= dTime.Hour && dTime.Hour < 21) oaRatio = 0.2;
 6   else if (19 <= dTime.Hour && dTime.Hour < 20) oaRatio = 0.3;
 7   else if (18 <= dTime.Hour && dTime.Hour < 19) oaRatio = 0.5;
 8   else if (12 <= dTime.Hour && dTime.Hour < 13) oaRatio = 0.6;
 9   else oaRatio = 1.0;
10
11   AirHandlingUnit ahu;
12   ahu = (AirHandlingUnit)ahuSystem.AHUs[0];
13   ahu.SetOutdoorAirFlowRange(1463d / 3600 * 1.2 * oaRatio, ahu.MaxOAFlowRate);
14   ahu = (AirHandlingUnit)ahuSystem.AHUs[1];
15   ahu.SetOutdoorAirFlowRange(1513d / 3600 * 1.2 * oaRatio, ahu.MaxOAFlowRate);
16   ahu = (AirHandlingUnit)ahuSystem.AHUs[2];
17   ahu.SetOutdoorAirFlowRange(1395d / 3600 * 1.2 * oaRatio, ahu.MaxOAFlowRate);
18   ahu = (AirHandlingUnit)ahuSystem.AHUs[3];
19   ahu.SetOutdoorAirFlowRange(1125d / 3600 * 1.2 * oaRatio, ahu.MaxOAFlowRate);
20 }
```

・外気冷房の導入

外気条件が良い場合に外気による冷却を行う。外気導入の是非は比エンタルピー基準によって行う。第 26 章で解説したように AirHandlingUnit クラスの OutdoorAirCooling で設定を行う。

・全熱交換器の導入

AHU に全熱交換器を導入し、外気負荷を抑制する。既に第 26 章のプログラム 26.10 では AirHandlingUnit クラスのインスタンスに全熱交換器を設定しているため、BypassRegenerator プロパティを true にすれば良い。全熱交換器を導入すると給気ファンと還気ファンの必要静圧が上がるため、ファン動力は増加する点に注意する。（プログラム 28.7　78, 79 行）

・冷温水の変流量制御の導入

冷水一次ポンプ、温水一次ポンプ、冷水二次ポンプ、温水二次ポンプを定速運転から吐出圧一定制御に変更する。

・熱源の高効率化

空気熱源ヒートポンプを高効率機種とし、冷却運転時の COP を 3.0 から 4.6 に、加熱運転時の COP を 3.0 から 3.5 に変更する。部分負荷時の性能を向上させるため、効率重視の台数制御とし、MaximizeEfficiency プロパティを true に変更する。また、直焚吸収冷温水機をインバータ機に変更する。

・クールビズの実施

詳細は第 28 章で解説した通りである。夏季の設定温度を 28.0 ºC、冬季を 20.0 ºC とし、熱的快適性を同等に保つために着衣量と採涼アイテムを導入する。

2）消費エネルギー効果の評価

前記の変更はすべてプログラム 28.1 に示した SHASE_SimulationTest クラスの public static プロパティで設定可能である（第 21 章と第 28 章のプログラムを参照）。プログラム 29.7 に段階的に変更を施した場合の計算処理を示す。3~9 行ですべての変更フラグを false に設定し、10 行で基準ケース（Case A）の計算を行う。12~31 行で 1 つずつ変更し、Case B~H として計算結果を出力する[†1]。

プログラム 29.7　建物・設備仕様と運用の変更と計算結果の出力処理

```
 1 private static void SHASE_HVACSystemTest()
 2 {
 3   SHASE_SimulationTest.IS_HIGH_INSULATION = false;  //高断熱仕様フラグ
 4   SHASE_SimulationTest.USE_CO2CNTRL = false;        //CO2 制御フラグ
 5   SHASE_SimulationTest.USE_OA_COOLING = false;      //外気冷房フラグ
 6   SHASE_SimulationTest.USE_REGENERATOR = false;     //全熱交換器フラグ
 7   SHASE_SimulationTest.USE_VWV_SYSTEM = false;      //VWV 制御フラグ
 8   SHASE_SimulationTest.IS_HIGHEFF_HSOURCE = false;  //高効率熱源フラグ
 9   SHASE_SimulationTest.DO_COOLBIZ = false;          //クールビズ実施フラグ
10   SHASE_HVACSystemTest("outputA.csv");
11
12   SHASE_SimulationTest.IS_HIGH_INSULATION = true;
13   SHASE_HVACSystemTest("outputB.csv");
14
15   SHASE_SimulationTest.USE_CO2CNTRL = true;
16   SHASE_HVACSystemTest("outputC.csv");
17
18   SHASE_SimulationTest.USE_OA_COOLING = true;
19   SHASE_HVACSystemTest("outputD.csv");
20
21   SHASE_SimulationTest.USE_REGENERATOR = true;
22   SHASE_HVACSystemTest("outputE.csv");
23
24   SHASE_SimulationTest.USE_VWV_SYSTEM = true;
25   SHASE_HVACSystemTest("outputF.csv");
26
27   SHASE_SimulationTest.IS_HIGHEFF_HSOURCE = true;
28   SHASE_HVACSystemTest("outputG.csv");
29
30   SHASE_SimulationTest.DO_COOLBIZ = true;
31   SHASE_HVACSystemTest("outputH.csv");
32 }
```

エネルギー消費量を集計した結果を図 29.5 に示す。

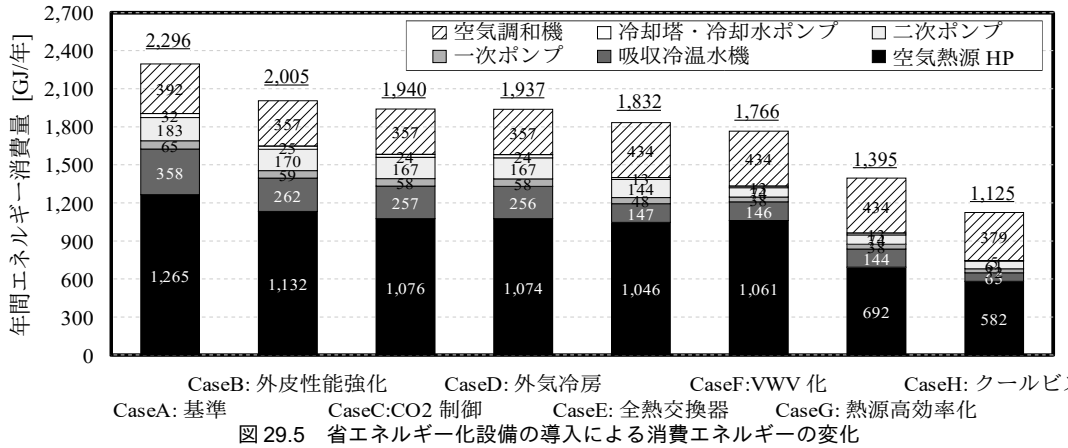


図 29.5　省エネルギー化設備の導入による消費エネルギーの変化

基準ケースは図 29.2 と同じである。外皮性能の強化や CO_2 制御はそもそもの負荷発生を抑制するため、熱源、熱源補機、空気搬送のすべてに効果的であることがわかる。外気冷房の導入に伴うエネルギー消費量の削減幅は小さい。本計算では中間期に完全に空調を停止しており、外気冷房が真に有効である期間がそもそも空調対象となっていないことが原因である。中間期空調を行う場合、あるい

†1　通常、省エネルギー化の手順としては、いきなり設備仕様を改善するのではなく、まず、断熱性能などの建築躯体としての性能を向上させて負荷の発生を根本から絶ち、その後、不可避の負荷に対して高効率な設備システムで対応するという手順を踏む。このような方法をヒエラルキーアプローチと呼ぶ。

は冬季にも冷房需要があるような場合には、本計算とは異なる結果となるであろう。全熱交換器を導入すると熱源および補機のエネルギー消費量は減少するが、空調機のファン静圧が上昇するために搬送エネルギーは増加する。変流量制御を導入するとポンプの搬送エネルギーは減少する。しかし温度差が拡大する結果、僅かであるが熱源のエネルギー消費量は増大する。熱源の高効率化を実施すると空気熱源ヒートポンプと吸収式冷温水機の消費エネルギーが減少する。以上のハード的な変更（Case A~Case G）をすべて実施すると、基準ケースと比較して約 60 %程度の消費エネルギーとなる。さらにソフト的な工夫として Case H のクールビズを実施すれば、基準ケースに比較してエネルギー消費量は 47 %となる。

3）温熱環境への影響評価

前記の変更に伴い、温熱環境がどのように変化するかを検討する。基準となる Case A、断熱性能を強化した Case B、クールビズを導入した Case H のそれぞれの PMV 発生頻度分布を図 29.6 に示す。居住者がいる 8:00~20:59 までのデータのみを計上した。Case C~G に関しては、計算上は室内温熱環境にほぼ影響を与えないため、Case B と同様と考えれば良い。

断熱を強化すると外気条件の変動に対して安定的となるため、頻度分布の尖度が大きくなることがわかる。また、断熱性能の向上に伴い、室内表面の温度が室温に近づくため、特に暖房時には放射温熱環境の改善が生じ、PMV がプラス側にシフトする。一方で、冷房時の PMV は平均値としては断熱強化の前後で変わらない。これは暖房時とは異なり、朝夕などの時間帯においては室温よりも外気温度が低く、室内表面温度が外気温度に近い方が有利な時間帯も多いためであろう。

クールビズを導入すると頻度分布の尖度はさらに大きくなる。これは温度設定値の緩和により室内外の温度差が小さくなり、放射温度と乾球温度が近づいたためである。着衣の調整と採涼アイテムによる相対気流速の拡大により、Case B とほぼ同等の PMV 値を実現できており、基準の Case A と比較すると平均値としてはやや改善されている。従って、一連の省エネルギー化設備の導入と運用の変更により、エネルギー消費は基準ケースに比較して約半分に抑えつつ、熱的快適性は若干の改善が可能であったということになる[†1)]。

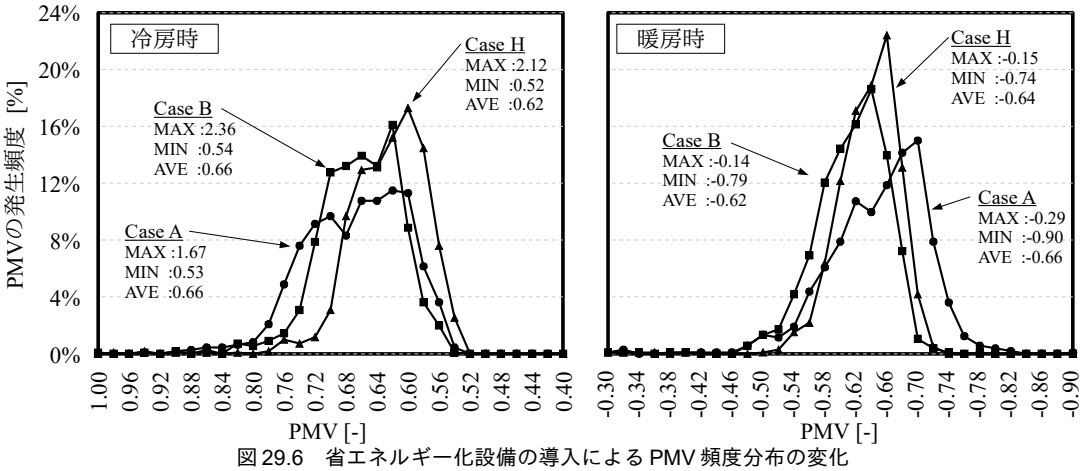


図 29.6　省エネルギー化設備の導入による PMV 頻度分布の変化

†1　エネルギー消費と温熱環境を組み合わせた統合的な評価としては、熱環境と人間の作業効率との関係を推定し、「知的生産性」と称して経済価値に結びつける研究もある。しかし、人間の「知的な」生産とは、外部入力によって支配される従属変数なのであろうか。筆者は「知的」と銘打つような活動への欲望は独立変数であり、これを従属変数と捉えて最終的にはたかがエネルギー消費に還元できてしまうようなモデルをたてることは、人間の知的好奇心・探求心をひどく矮小化する行為のように思う。回帰直線を著しく逸脱する外れ値こそが畏れるべき人間の「知性」では無いだろうか。

第30章　建築熱環境計算の未来
(The Future of Thermal Environmental Computing)

30.1　建築熱環境計算の過去と現在

本章の題は「建築熱環境計算の未来」であるが、大部分の記述は過去の歴史に費やした。未来は現在と過去から独立に存在するものではなく、現在を生きる我々の歴史観と社会性を基礎に生み出されるものであるためである。歴史学者の斉藤は「未来とは無限の可能性の世界」であるとしながらも、未来を引き寄せるものは人間であり「人間の経験から推して、実現の可能性の極めて乏しい事態を予想し、意欲するものはいない。予見は、実現の可能性が高いと考えられるものに自ずから限定されている[30.46]」としている。そして同時に、人間は現在の境遇を前提にその維持や変更を望む。この意味では、未来を形成する基礎は人間の過去の歴史認識と現在の社会的境遇である。本章では、熱環境計算の未来に影響を与える、コンピュータの歴史、建築環境工学の歴史[†1]、建築熱環境計算の歴史を概説し、熱環境計算の未来を占う。

30.1.1　コンピュータ概略史

建築熱環境計算の前提となるコンピュータ自体の歴史を概説する[30.1) 30.2)]。

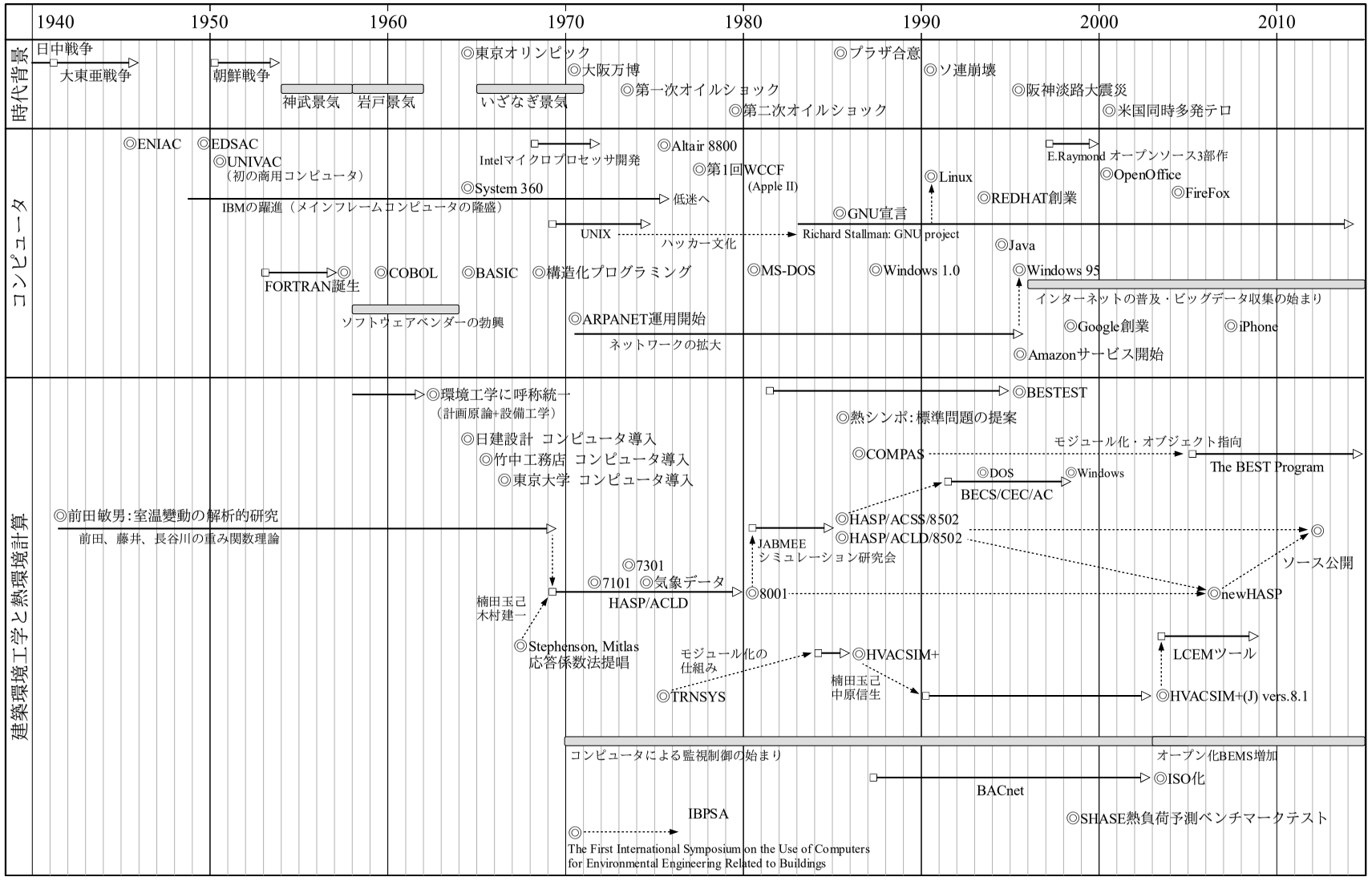
1）コンピュータの商用化

人間の演算処理を外部化するという意味では、古来から使われている計算尺のような計算用具もコンピュータに近い存在である。しかし、産業革命を受けて人間以外からの動力を演算処理に活用するという観点からの演算処理の自動化を志向した初めての機械は、Charles Babbage による Analitical Engine（1833 年）である。今日のコンピュータの直接の祖先としては、J.W. Mauchly と J.P. Eckert による初の電子式計算機である ENIAC（1946 年）が挙げられる。Eckert らはさらに 1951 年に初のプログラム内蔵型計算機である EDVAC を世に出し、ここにソフトウェアという概念が登場した。

1950 年には初の商用の計算機である UNIVAC が販売され、それまでは研究段階にあったコンピュータの商用化が大きく進展する。ただし、この頃の計算機は今日我々が保有しているようなパーソナルコンピュータではなく、メインフレームと呼ばれた巨大な計算機である。米国ではいくつかの商用メインフレーム販売企業があらわれたが、その中でも IBM は大きな躍進をみせ、1970 年代の半ばまで圧倒的なシェアを握った。1940 年台にはコンピュータの主用途は研究のための数値計算であったが、商用化の拡大に伴い、一般企業が事務機として導入する事例が増大した。

今日においても科学技術計算分野で大きなシェアを持つ FORTRAN 言語はこの頃に誕生し、IBM 社の John Backus が 1954 年に開発を始め、1957 年 4 月に完成させた。業務用計算でシェアを握る COBOL の開発は 1959 年、ライトなコンピュータ利用者層の支持を得た BASIC の開発は 1964 年である。1950 年代頃はソフトとハードは今日ほど分離していなかったが、 FORTRAN 言語が誕生した 1950 年代の終わり頃からソフトウェアの開発のみを担当するソフトウェア会社が市場に出現した。

†1　本分野の歴史に関する体系的な先行研究としては石原正雄による研究[30.89) 30.90) 30.91) 30.92)]と井上宇市による研究[30.44)]が挙げられる。


1940
1950
1960
1970
1980
1990
2000
2010
時代背景
日中戦争
大東亜戦争
朝鮮戦争
神武景気
岩戸景気
いざなぎ景気
◎東京オリンピック
◎大阪万博
◎第一次オイルショック
◎第二次オイルショック
◎プラザ合意
◎ソ連崩壊
◎阪神淡路大震災
◎米国同時多発テロ
コンピュータ
◎ENIAC
◎EDSAC
◎UNIVAC
（初の商用コンピュータ）
◎System 360
IBMの躍進（メインフレームコンピュータの隆盛）
低迷へ
Intelマイクロプロセッサ開発
◎Altair 8800
◎第1回WCCF
(Apple II)
UNIX
ハッカー文化
Richard Stallman: GNU project
◎GNU宣言
◎Linux
◎REDHAT創業
E.Raymond オープンソース3部作
◎OpenOffice
◎FireFox
◎Java
◎Windows 95
◎Windows 1.0
◎MS-DOS
◎FORTRAN誕生
◎COBOL
◎BASIC
◎構造化プログラミング
ソフトウェアベンダーの勃興
◎ARPANET運用開始
ネットワークの拡大
インターネットの普及・ビッグデータ収集の始まり
◎Google創業
◎iPhone
◎Amazonサービス開始
建築環境工学と熱環境計算
◎環境工学に呼称統一
（計画原論+設備工学）
◎日建設計 コンピュータ導入
◎竹中工務店 コンピュータ導入
◎東京大学 コンピュータ導入
◎BESTEST
◎熱シンポ：標準問題の提案
◎COMPAS
モジュール化・オブジェクト指向
The BEST Program
◎DOS
◎Windows
BECS/CEC/AC
◎前田敏男：室温變動の解析的研究
前田、藤井、長谷川の重み関数理論
◎HASP/ACSS/8502
◎HASP/ACLD/8502
JABMEE
シミュレーション研究会
◎
ソース公開
◎newHASP
◎7301
◎7101
◎気象データ
HASP/ACLD
◎8001
楠田玉己
木村建一
◎Stephenson, Mitlas
応答係数法提唱
モジュール化の
仕組み
◎HVACSIM+
楠田玉己
中原信生
◎TRNSYS
LCEMツール
◎HVACSIM+(J) vers.8.1
コンピュータによる監視制御の始まり
オープン化BEMS増加
◎ISO化
BACnet
◎SHASE熱負荷予測ベンチマークテスト
IBPSA
◎
The First International Symposium on the Use of Computers
for Environmental Engineering Related to Buildings

2）パーソナルコンピュータの普及

1960 年台の終わりから 70 年台の始めにかけて、真空管やトランジスタを用いた従来の中央演算装置に代わるものとして小型のマイクロプロセッサが開発された。Intel による 4bit CPU である Intel 4004 はその代表格である。大量生産によりマイクロプロセッサの価格は従来の中央演算装置よりも遥かに低く、4004 は販売当初においても約$1,000 であった。中央演算装置の低価格化を背景に、1975 年には初の個人向けコンピュータである Altair 8800（4004 の後継である 8 bit CPU の Intel 8080 を搭載）が販売され、その価格は僅か$400 であった。しかし、パーソナルコンピュータ（以下 PC: Personal Computer）の市場が大きく開く契機は 1976 年の第 1 回 WCCF（West Coast Computer Fair）である。WCCF ではアップル社が開発した Apple II によって CPU、キーボード、モニタ、記憶用ドライブという今日の PC の基本形が提示された。

この頃の PC のインターフェースは文字入力により操作を行う CUI（Character User Interface）であったが、1984 年にはアップル社からマウスを用いた直感的な操作が可能な GUI（Graphical User Interface）を持つマッキントッシュが販売され、1987 年にはマイクロソフト社から GUI を持つ OS である Windows 1.0 が販売された。その後、計算能力の向上に伴い GUI の操作性は向上を続け、1988 年の Windows 2.1 を経て 1993 年に発売された Windows 3.1 において、GUI を持った PC は一般の人々に広く普及することとなった。また、インターネットという技術も PC の普及を牽引した。インターネットの前身である ARPANET（Advanced Research Projects Agency Network）は既に 1971 年に運用を開始しており、当初は 4 つのノードを結ぶネットワークであった。その後、ネットワークは拡張を続けるが、特に一般の人々がインターネットに接続する起爆剤となったものは 1995 年に発売された Windows 95 であり、同 OS にはインターネット関連機能が標準搭載されていた。

3）オープンソース・ソフトウェアの誕生

1990 年台のもう 1 つの大きな動きはオープンソース・ソフトウェアの発展である。既に述べたとおり、当初のコンピュータ産業においてはソフトウェアとハードウェアは不可分の存在であり、パッケージ化された総体が販売対象となっていたが、1960 年代以降はソフトウェアが独立して売買の対象となった。これに伴いソースコードに対する著作権主張が一般化したが、このような状況に対して批判の声を上げた者が、当時 MIT でプログラマをしていた Richard Stallman である。Richard Stallman は、プログラマの相互扶助と博愛の精神にもとづいてソースコードは自由（フリー）であるべきだと主張し、権利的に自由な OS の開発を 1983 年に開始した[30.13)]。「権利的に自由である」とはソースコードの複製、改変、再頒布を許諾するということであり、このようなソフトウェアをフリーソフトウェアと呼ぶ[30.12)]。また、この運動から生まれた一連のソフトウェア群を GNU ソフトウェアと呼ぶ。この運動の最も大きな成果の 1 つは Linux の誕生である。Linux は 1991 年当時、ヘルシンキ大学の学生であった Linus Torvalds が自作の小さな OS を公開したことが開発のきっかけであり、これに様々なプログラマが GNU ソフトウェアなどを用いて機能追加を行い、巨大で堅牢なフリーの OS へと成長させていった。1990 年台は特にサーバー用途での Linux のシェア拡大が注目されていたが、今日ではブラウザ、マルチメディア処理、オフィススイートソフトなども備えており、一般業務での活

用が可能な段階となっている。Eric Raymond は、トップダウンの開発とは異なる、このような多数のプログラマによる分散的開発の長所を論考しており、このような体制で開発されたソフトウェアをオープンソース・ソフトウェアと呼ぶ[30.17]。オープンソース・ソフトウェアの活動は当初に Richard Stallman が重視した倫理性（ソフトウェアは自由（フリー）であるべき）からは離れ、企業の営利追求とも矛盾しない 1 つの開発方法論として進展しているようである。

4）データ集積と機械学習の高度化

今日では人間社会の様々な場面にコンピュータが散りばめられ、また、インターネットを中心とした通信技術の発達に支えられて、これらのコンピュータは相互に情報を連携して協働している。この結果、人間活動に関わる大量のデジタルデータが日々生産され、蓄積される状況が生まれた。

近年の 1 つの大きな動きは、これらのデータの処理をコンピュータに外部化する技術の開発と発達である。もちろん歴史的にはコンピュータの登場自体、人間が行っていた演算作業の外部化を目的としたものであるが、近年の外部化はこれとは質的に異なる。人間は何事かの事象に価値を見出し、この価値ある事象を実現するための手順を考え、その手順を実行に移す。従来の外部化はこの手順の実行に向けられたものであり、実行すべき手順は人間が考えるものであった。しかし近年の外部化はこの手順の策定自体を外部化しようとするものである。人間に残された作業は価値の判断のみとなる。

特にこの外部化技術が目覚ましく発達したコンピュータ将棋の分野に即して具体的に解説する[30.8)30.9)]。コンピュータ将棋は 1980 年頃から商用ソフトウェアの販売が開始され、その後も連綿と改良が続けられてきたが、2005 年に登場した bonanza によって大きなブレイクスルーがもたらされた。コンピュータ将棋は指し手の選択を行うために、局面の良し悪しを判断する評価関数を持つ。従来、この評価関数のパラメータはプログラム開発者である人間が設定するものであった。しかし bonanza ではプロ棋士が実際に指した 6 万の棋譜にもとづき、評価関数のパラメータをコンピュータ側に自動推定させるという手法を取った。即ち、人間側に求められた作業はその局面が優れているか否かについて価値判断を行う（この場合にはプロ棋士の棋譜に一致するか否かという情報の提供）だけであり、その価値をもたらすための手順の策定（評価関数のパラメータはどうあるべきかという判断）はコンピュータ側に任せられたということである。この手法により bonanza は 2006 年の第 16 回世界コンピュータ将棋選手権大会で初出場にして初優勝を成し遂げた。

このような手法を実行するためには大量の価値判断の記録が不可欠であり、将棋の場合にはプロ棋士の棋譜がそれを担った。一方で、先に述べたとおり、将棋以外の分野（建築分野を含む）においてもこのようなデータ蓄積は日々進行している。今日でもインターネットを介した商取引記録や IC カードに記録された行動情報などは記録されているが、記録の対象となる人間行動の範囲は今後さらに拡大し、これらがすべて価値判断の集積としてコンピュータの学習対象となる。また、近年、ディープラーニングと呼ばれる技術に先導されて第 3 次の人工知能ブームが生じているが、これは端的に言えば大量のデータにもとづいて極めて非線形性の強いモデルを自動生成する機械学習の技術である。デジタル化と通信・記録・機械学習の技術発達により、コンピュータへの外部化の範囲が拡大し続けているという状況が現在、生じている。

30.1.2 建築環境工学概略史

1）計画原論

今日、建築環境工学と呼ばれている学問分野の歴史は比較的浅く†1)、体系的な書籍としては1934年（昭9）に渡辺要があらわした「計画原論（高等建築学叢書 第13巻）」に遡る[30.4)]。本書の契機となったのは内田祥三による東京大学図書館の計画であり、渡辺要が内田祥三のもとで設計・監督にあたった際に現実に直面した計画上の諸問題の解決に端を発している[30.3) 30.121)]。渡辺の計画原論の目次は下記の通りである。

第1章　室内気候（その1）	第7章　昼光照明
第2章　室内気候（その2）	第8章　音響
第3章　自然換気	第9章　開口及建具
第4章　伝熱	第10章　家具
第5章　日照の基礎的事項	第11章　間取
第6章　日射	第12章　廊下、広間及階段

第9章~第12章は共著の長倉謙介による章である。当初、渡辺らは広範な計画基本原則を包含する学を構想していたが、次第に、第1章~第8章に示されるような物理現象の科学的説明と設計法が計画原論と呼ばれるようになった。環境工学分野の主要な項目がおさえられており、今日の教科書としても十分に通用する章立てである。ただし、伝熱理論に関しては壁体の非定常伝熱にとどまり、室温変動と熱負荷には触れられていない点には注意が必要である†2)。東京大学では本書の刊行後の1940年（昭15）に「建築学第2講座 建築計画第1」が設けられ、平山嵩が計画原論の講義を開始する†3)。

世代論としてくくるならば、計画原論の第一世代は渡辺要、東京大学の平山嵩、早稲田大学の佐藤武夫、木村幸一郎、日本大学の船越義房、横浜国立大学の佐藤鑑、京都大学の前田敏男などであろう†4)。さらに石原正雄、小木曽定彰、斉藤平蔵†5)、小林陽太郎、後藤滋などが第二世代としてこれに続き、熱、光、音などの研究を発展させる†6)。特に室温変動理論に関しては1941年（昭16）の前田敏男による「室温變動の解析的研究」を皮切りに†7)、建築研究所の藤井正一や東北大学の長谷川房雄などが研究を盛り上げ、重み関数理論を発展させる†8)。また、熱環境の計算に関しては、木村幸一郎の弟子である木村建一が後に米国より応答係数法を伝え[30.28)]、斉藤平蔵のもとからHASP/ACLDの生みの親である松尾陽が輩出されることになる。

---

†1　後藤滋, 紀谷文樹らは日本における建築環境学の先駆者として森林太郎（森鴎外）を挙げている[30.30) 30.88)]。

†2　渡辺は昭和9年に建築学会で「建築計画に関する二三の問題」という題で講演を行っているが、光と音の問題を取り上げており熱的な話題には触れていない[30.22)]。なお、この講演は渡辺の機能主義的な考え方が強く表れた興味深い講演であり、例えば「材料も又、建物の機能に対して最も忠実でなかればならぬ、或はローマンチシズムというようなことに陥らぬようによく注意しなければならぬ」という発言も記録されている。アテネ憲章の翌年の講演ということも影響しているかもしれないが。

†3　渡辺は「計画原論」の刊行前の昭和6年に日本大学で建築計画の講座を担当しており、実質的にはこの時点で既に計画原論に相当する講義が開始されていた[30.121)]。また、ほぼ同時期に早稲田大学では佐藤武夫による建築音響学（1922~）と木村幸一郎による照明学（1935~）の講義が行われている。

†4　聴竹居で知られる藤井厚二[30.36)]は、ここに挙げた第1世代の人々に比較して10年ほど年長である。理論が体系化される前に実作を伴いながら建築環境学を探求した者であり、世代としてくくることのできない特別な存在と言える。

†5　その厳しい研究に対する姿勢から、鬼平犯科帳にならって鬼の平蔵と呼ばれていたらしい。

†6　主に建築設備工学に関わる人物ついては後述のため、ここでは挙げない。

†7　厳密には前田は同研究報告の中で本分野における2つの先行研究を挙げている[30.25) 30.26)]。

†8　長谷川は「室温変動理論における三つの立場」という題目で3者の室温変動理論の違いを比較整理している[30.23)]。

2）建築設備工学

計画原論と対を成し、今日の建築環境工学を形成するもう 1 つの学の系譜が、建築設備工学である。暖房や照明の技術は古来よりあったが、ここで言う建築設備とは主に明治期に海外から導入が開始された西洋式の技術である。

今日では建築設備といえば主に機械設備（空調と衛生）と電気設備であるが、これらの技術の学術的な中心である電気学会、照明学会、空気調和・衛生工学会は明治維新後間もない 1888 年（電）と大正年間である 1916 年（照）と 1917 年（空衛）にそれぞれ創立されている。熱環境計算に特に関連の深い「空気調和・衛生工学会」は創立当初は「煖房冷蔵協會」という名称であった。1873 年（明 6）には工部大学校で日本初の蒸気暖房が行われており、協会創立時において暖房はすでに 40 年を超える歴史を持っていたが、冷房に関しては 1907 年に富士瓦斯紡績保土ヶ谷工場で初の井水冷却が行われたという段階にあった。しかもこれは工場空調であるから、製品の生産性向上を目的としたものであり、人間の快適性を目的にした保健空調に関しては協会創立後の 1924 年（大 13）の邦楽座の空調設備を待たなければならない [30.40]。従って、煖房冷蔵協會の「冷蔵」とは冷房ではなく、主に食品冷蔵や冷凍を意図したものであり、現に発起人の中には日東製氷や帝國冷蔵などの冷凍産業業者が含まれていた†1)。

アカデミーの側から建築設備工学を先導した初期の人物としては東京大学の中村達太郎が挙げられる。中村は大正年間に建築衛生に関する 2 冊の書を刊行し [30.32) 30.33)]、先の煖房冷蔵協會の発起総会においても発起人の一人として参加している†2)。一方で、設備系講義の整備は早稲田大学が早く、建築学科が開設された 3 年後の 1913 年（大 2）には「衛生設備」の科目が設けられた。さらに、同大学を卒業した大澤一郎と桜井省吾は米国イリノイ大学に設備工学の研究のために留学し、今日の建築設備工学関連の講座の流れを創作する [30.29) 30.126)]。その後、大澤一郎は芝浦工業大学、日本大学、関東学院大学でも教鞭をとり、関東学院大学では日本初の建築設備工学科を開設する。大澤一郎が関東学院大学に移った後、早稲田大学では主に非常勤講師による設備教育が行われたが、その中の 1 人である安東勝男が大成建設で井上宇市に出会い、1953 年に設備系専任講師として迎える [30.43) 30.118)]。井上は大学着任前後に「建築設備ポケットブック（1952 年）」と「空気調和ハンドブック（1956 年）」を出版している。当時の参考書の多くは一般的記述にとどまるものが多かったが、井上による両資料は現実の設計を行うために必要な各種の図表や例題が盛り込まれており、今日においても空調設備設計者必携の資料として版を重ねている。また、井上研究室は後に HASP/ACSS の主要な開発者の 1 人である猪岡達夫を輩出する。

先に述べたように蒸気暖房は既に明治期に我が国に導入されていたため、暖房に関する設備工学は、必要に迫られながら現場においても大きな発達を見せていた。当時はまだ設備設計という職能は確立しておらず†3)、ドイツのケルチング社の蒸気暖房システムを輸入していた商社である高田商会が

†1　その後、これらの会員は日本冷凍協會（今日の日本冷凍空調学会）を立ち上げ、煖房冷蔵協會を去っている。当時の記録には「日本冷凍協會の成立に対しては暫く其の成行きに任せ」と穏当な表現となっているが [30.35)]、後年に書かれた斉藤省三の回顧録からはかなりの恨み節が感じられる [30.34)]。

†2　中村は本会が暖房と冷蔵だけではなく建築衛生全般を取り扱うのであれば、会の名称を「建築衛生協會」とすべきではないかと強く主張した [30.31)]。発起総会では多数決により「暖房冷蔵協會」が採用されたが、その後、1927 年（昭 2）に中村が主張した名称によく似た「衛生工業協会」に改称されたことは興味深い。

†3　設備設計という職能に関しては北浦重之の発起により行われた意見交換会の記録が煖房冷蔵協会誌に残っている [30.41)]。またほぼ同時期に丹羽重光による「煖房の設計料に就て」という論文が掲載されている [30.42)]。

直接に設計と施工を行っていた。高田商会は一時期は国内の暖房工事をほぼ独占しており、同社からは関藤国助や柳町政之助など、昭和期の空調設備を担う多くの技術者が巣立った。特に柳町は後に日本初の冷凍機による冷房、床暖房システム、放射空調システム、太陽熱利用暖房システムなども手がけており、在野の立場から設備工学の地平を切り開いた存在と言える†1)。その先見性は多くの実務者の心をつかみ、例えばある特集記事[30.52)]では、中島康孝、中原信生、千葉孝男がそれぞれに敬愛の念を込めて当時を回顧している。また、冷凍機を用いた冷房にも早くから注目し、1929年（昭4）にターボ冷凍機を視察するために100日間ほどの米国旅行を行い[30.38) 30.39) 30.117)]、ニュージャージーでCariierを訪ねて昼食をともにしている†2)。柳町が1919年（大8）に発行した「煖房と換気（前後編）[30.53) 30.54)]」は空調設備に関する初の体系的な学術書であるが、この時期に既に冷房についての記載がある†3)。

3）建築環境工学へ

計画原論の学問としての深度化に伴い、研究者によって本分野の呼称が分かれるようになった。例えば、建築環境学[30.100)]、建築設計理論[30.101)]、設備原論[30.102)]、室内環境計画[30.103)]、建築保健工学[30.104)]などである。例えば、佐藤鑑は環境を自然的環境と創造的環境に分類し、後者をさらに人為的理学的環境と社会的文化的環境に分けた上で、両者を建築環境学の対象としている。以下は佐藤による建築環境学の定義である。

「*建築環境学とは人類の創造による人為的物理化学的環境及び社会的文化的環境と人類との相関関係を究明し、これによりよりよき新たなる環境を創造する方法を求め、以て人間の活動圏を拡大すると共にこれを安全化し能率化せんとするにある*[30.100)]」

このような背景のもと、昭和33年頃から日本建築学会の研究協議会において名称に関する議論が開始され[30.122) 30.123) 30.124) 30.125)]、建築設備工学の概念と合流して昭和37年に建築環境工学という名称が与えられた。

30.1.3　建築熱環境計算概略史

1）期間熱負荷計算のはじまり

楠田によれば、米国ASHRAE Journalに掲載された、空調技術にコンピュータを適用した最初の論文はSoumeraiと楠田の研究である[30.18)]。FORTRAN言語が誕生して2年後の1959年のことであったが、本研究ではFORTRANは使われておらず、コンピュータはBendix社のG-15が採用された。楠田によれば、この頃には核開発または航空力学に関わる研究者が高速なコンピュータを優先的に使用しており、建築物理に関する研究に使えることは稀であった[30.10)]。日本では1966年に東京大学にHITAC5020が導入され、私大の研究者の利用も許されていた[30.47)]。また民間の設計会社としては日建設計が1964年[30.87)]にコンピュータを初めて導入している。施工会社としては竹中工務店による1965年[30.105)]の導入が初のようである。

建築環境分野へのコンピュータの導入が活発化した原動力の1つは、年間時系列負荷計算への期待

†1　わずか数年間の極々短期であるが、柳町は井上宇市着任前に早稲田大学で非常勤講師も務めている。

†2　非常に活動的であり、米国の各所で冷房設備が設けられた映画館やオフィスをめぐり、また、有効温度（ET: Effective Temperature）で有名なヤグローにもボストンで面会している。

†3　渡辺要は東京大学図書館の暖房設計を懐古し、当時の主な和書としては中村達太郎の「換気煖房の計算必携」と柳町政之助の「煖房と換気」のみであったと語っている。ただし「換気煖房の計算必携」は東京大学図書館の設計施工期間の1924~1928年よりも後の1931年に刊行されたと記録されており、順序が前後する。

であった。特定の時間に発生するピーク負荷であれば手計算であっても予測可能であるが、年間 8760 時間の室温と負荷を求めるためにはコンピュータによる自動計算が必須となる[†1]。米国では 1960 年代に建築設計へのコンピュータ利用に興味を持つ者達の非営利団体である APEC（Automated Procedures for Engineering Consultants）が組織され、空調の時系列負荷に関する最初のプログラムが開発された。1967 年には Stephenson と Mitalas により応答係数法の理論が提唱され[30.21]、この理論を取り込みながら 1974 年には楠田が現在の Energy Plus と BLAST の前身である NBSLD（National Bureau of Standards Load Determination）を開発した[30.19) 30.20]。早稲田大学の木村建一は、たまたま応答係数法の理論が提唱された 1967 年から Stephenson のもとへ留学しており、1968 年の ASHRAE 大会において楠田と出会う[30.48]。楠田から ASHRAE の年間空調熱負荷員会の成果[30.51]を入手した木村は空気調和・衛生工学会にこれを送った。日本では東京大学の松尾陽を中心に委員会[†2]が組織されて 1971 年に HASP/ACLD/7101 が開発される[30.55]。HASP/ACLD は米国 ASHRAE 版のソフトウェアに比較して 2 つの特色があった[30.110]。1 つは数表を用いずに任意の構成の多層壁から応答係数を算出する点にあり、これは前田、長谷川、藤井らによる重み関数理論の研究の蓄積があったことが大きい。もう 1 つは、直角三角波入力の応答係数を導入することで、日本で主流の非連続空調に対応した点である。 HASP/ACLD/7101 の実際のコーディングは木村研究室出身の石野久彌が担当しており、石野は 2016 年現在開発が進められている The BEST Program の中心的存在でもある。

HASP/ACLD/7101 の開発は空気調和・衛生工学会の委員会中間報告として機関誌である空気調和・衛生工学の 1972 年 3 月号に掲載されたが、注目すべきは同号において「手計算による最大負荷計算法」についての報告が行われていることである。同報告をまとめた松尾の言葉を借りれば「電算機によって計算の内容がブラックボックス化されることにより、技術者の常識の欠如、空調技術の退歩を招くおそれ」があったためである。また、本開発に対する言葉ではないが、渡辺要は別のところで「コンピュータは精密計算機械として高度な技術計算を期待することも重要であり、筋ではあるが、他方でよりいっそうの略算法を開発すべきであろう」と述べている[30.5]。熱環境計算を得意とする者は、とかく手段の新しさや精度の高さにこだわりがちであるが、目的に応じた手段と精度という観点からは略算の意義は非常に大きいということは再認識する必要がある。

建築環境分野におけるコンピュータ利用の活性化の流れを受け、1970 年には The First International Symposium on the Use of Computers for Environmental Engineering Related to Buildings が開催され、400 名を超える研究者が参加した[30.49]。本シンポジウムの主催者は楠田であり、第 2 回がフランスのパリ、第 3 回がカナダのバンフ、第 4 回が日本の東京で開催され、現在の国際建物シミュレーション協会（IBPSA: International Building Performance Simulation Association）の創立に繋がった[30.50]。IBPSA 主催の国際会議である Building Simulation は今日に至るまで隔年で開催が継続されている。

2）空調システムシミュレーションへの展開

熱負荷計算プログラムである HASP/ACLD/7101 は数度のバージョンアップとバグ修正を重

†1　日建設計の牧英二の 1970 年の記事によれば、従来の手計算によるピーク負荷計算の自動化は既に設計の現場に導入されており、将来への期待としてランダムな条件に対応できる時系列計算が語られている[30.62]。

†2　委員は松尾陽、横山浩一、木村建一、武田仁、石野久彌、滝澤博と、現在では伝説的な面々であるが、当時は皆若手であり、新しい技術への期待と興奮の中で HASP/ACLD は生み出された[30.48) 30.110]。

ね、1980 年に HASP/ACLD/8001 がソースコード付の書籍で刊行された[30.57]。また、日本の標準年気象データも 1974 年に整備されており[30.61]、この時点で研究者ではない実務者が期間熱負荷計算を行うための一応の準備は整ったことになる。ただし、当時はまだパーソナルコンピュータという概念が生まれて数年の時期であり、技術者たちは各社が持つメインフレームで HASP/ACLD を実行していたようである。パーソナルコンピュータ（当時はマイクロコンピュータと呼称）に向けた MICRO-HASP/1982 が開発されたのは 1982 年である。HASP/ACLD/8001 の完成を受け、建築熱環境計算は新しい目標に向かって進み始める。建築設備システムとの連成計算による期間的なエネルギー消費の予測である。このため、1980 年に日本建築設備士協会に空気調和設備シミュレーション研究会が設けられ、6 年間の歳月をかけて設備システムシミュレーションプログラムである HASP/ACSS/8502 が開発された[30.59) 30.60]。HASP/ACSS の開発は HASP/ACLD と同様に東京大学の松尾陽が中心となって行われたが、設備システム部分の計算に関しては日建設計の猪岡達夫が大きな役割を担った。猪岡は後に HASP/ACSS を基礎に、首都大学東京の永田明寛、鹿児島大学の二宮秀與、大阪市立大学の西岡真稔、東京理科大学の長井達夫らとともに、省エネルギー法の CEC/AC 計算を行うためのプログラムである BECS/CEC/AC を開発する[30.120]。また、この頃には漸く PC が十分に普及し、当時の状況について松尾は 1984 年の空気調和・衛生工学誌において「電算機というものを、そのための特別の素養を持った専門家の手から解き放ち、『普通の人』が使うものにしてしまった」と記して、建築技術者の PC 活用に期待を表している。

HASP/ACLD と HASP/ACSS は日本で最もよく知られた熱負荷・設備システムシミュレーションプログラムであるが、この時期に開発が進められていたプログラムはこれだけではない。計算機資源の入手が容易になったことを受け、民間企業や大学研究機関の各所でプログラム開発が行われた。その結果、これらのプログラムの機能や計算結果の相互比較について関心がもたれるようになった。前記の HASP/ACSS の開発委員会では国内の 9 つのプログラムを収集し、これに米国の DOE を加え、同一のオフィスビルを対象に冷暖房負荷とエネルギー消費量を計算して比較を行っている[30.58]。また、日本建築学会ではプログラムの相互比較を行うための標準的な計算対象建物の検討が行われた。1985 年に滝沢博による事務所用標準問題と宇田川光弘による住宅用標準問題が示された[30.63) 30.64]。住宅用標準問題に関しては長谷川房雄による TASP-1[30.76]、土屋喬雄による SOLPAC[30.77]、宇田川による MALTEP[30.75]、松浦茂による SAPL-1[30.82]、坂本雄三による BRIMAP[30.78]、武田仁による LESCOM[30.79]、浦野良美による PSSP/MV1[30.80]、小玉祐一朗と武政孝治による PASSWORK[30.83]、通算産業省による ESPAR/M[30.84]、荒谷登による Succs 85[30.81]、計 10 の国内のプログラムの計算結果が相互に比較された[†1]。標準問題の検討は国内においては 1985 年の提案後、1 年ほど継続した後に終了する[30.74]。一方、米国の NREL（National Renewable Energy Laboratory）では 1981 年より同様の検討が続けられ[30.65]、1995 年に熱負荷計算部分が BESTEST（Building Energy Simulation Test）として公開された[30.66]。日本の標準問題に比較すると、BESTEST で計算する建物は形態や材料が遥かに抽象化されており、熱環境計算に影響を与える要素ごとの比較という観点では優れている[†2]。また、現在は設備システム

†1　この後、MALTEP は EESLISM へ、PASSWORK は SolarDesigner へ、BRIMAP は SMASH へ、それぞれ発展する。

†2　林徹夫によれば、短期間であるが日本においても標準問題提案後に同様の検討が行われた。しかし成果物が報告できるまでは検討が進まずに途中で終了してしまい、BESTEST の成果をみて後悔を感じたようだ[30.74]。

シミュレーション部分の開発が進められている[30.67]。

3）設備システムの最適化と制御系シミュレーション

1970 年頃から設備の監視制御を目的に建物にコンピュータが導入され始める。例えば 1971 年に竣工した日本 IBM 本社ビルの監視制御システムは、中間階と地下階に設けた設備管理センターと防災センターにそれぞれマイクロコンピュータを設置して建物附属設備と防災設備の管理を行う、当時の最先端のシステムであった[30.71]。情報化の進展に伴って監視対象が広くなり分単位での状態記録が行われるようになると、これを解析するためのシミュレーションプログラムが必要となる。しかし、HASP/ACSS のようにエネルギー予測を目的に 1 時間単位で計算を行うプログラムは直接には適用できない。そこで制御の適正化やシステムの故障検知を目的に分秒単位でシステムを模擬できる制御系の動的シミュレーションプログラムが求められるようになる。このような背景から 1984 年には米国 NBS において分秒単位の空調システムシミュレーションを行うプログラムである HVACSIM+の開発が始まり、1986 年に完成した[30.69) 30.70]。HVACSIM+は public domain のソフトウェアであったため、名古屋大学の中原信生は 1987 年に NBS の楠田からソースコードを入手し、1990 年頃からプログラムの改良を始める。1995 年以降は空気調和・衛生工学会に委員会が組織され、1998 年に様々な機能を追加した HVACSIM+(J)が公開され、以降 2003 年まで改良が続けられた[30.72]。プログラムの改良は主に中原研究室の朱穎心（現 清華大学）、丹羽英治（現 日建設計総合研究所）、渡辺剛（現 NTT ファシリティーズ）によって行われた。丹羽と渡辺は後に国土交通省官庁営繕部から公開される「LCEM（Life Cycle Energy Management）のためのシステムシミュレーションツール（以下、LCEM ツール）」の開発で中心的な役割を担うことになる。

4）モジュール化とプログラムの再利用

HASP/ACLD/7101 はメインプログラムと 5 つサブルーチンで構成される約 2,000 行のプログラムであった。一方、HASP/ACSS/8502 のサブルーチンは 140 を超え、プログラム行数は約 20,000 行に増加した。そこで石野らは HASP/ACSS/8502 の完成後、1980 年代の中頃からプログラムの細分化と再利用性の確保という問題に取り組み[30.95) 30.96) 30.97) 30.98]、COMPAS（Component Program Library for Air-conditioning Systems）を開発する。サブルーチンの入出力仕様を明確にし、これらを組み合わせることで汎用の目的にプログラムを再利用しようとするものである。

一方、情報分野では 1960 年代後半に構造化プログラミングが提唱され[30.85]、プログラムの大規模化に伴うエラーの頻出や開発期間の長期化への対応が検討された。構造化プログラミングでは、段階的抽象化によりプログラムの機能を分解することで、特定の階層の実装の詳細を他の階層から分離する。この考え方を直接的に取り入れた建築環境分野のプログラムとして、米国ウィスコンシン大学で開発された TRNSYS（TRaNsient SYStem Simulation Tool）が挙げられる。TRNSYS は Sanford Klein によるウィスコンシン大学の博士論文[30.86]の成果であり、1975 年に一般公開（version 6.0）されて以降、今日に至るまで改良が続けられている（2012 年 version 17.1）。その最も大きな特徴は、TYPE と呼ばれるモジュールを自由に接続してシステムを構成するという仕組みにある。TYPE の入出力仕様は明確に抽象化されているため、これを用いて非線形連立代数方程式（NLEs: Non Linear Equations）と連立常微分方程式（ODEs: Ordinary Differential Equations）を解くソルバの機能は、個別

の TYPE の具体的な実装とは切り離される。従って、ユーザーはそれぞれの検討したい問題に即して TYPE を自由に定義することができ、これを既存の TYPE と連成させて解くことができる。モジュール化はプログラムコードの再利用性を高め、この結果、ユーザーによる多くの TYPE が公開されている。このような考え方は後発のモジュール型プログラムである HVACSIM+にも受け継がれている。また、プラットフォームに求められるものは結局は NLEs と ODEs であると考え、近年では MATLAB と Simulink のような汎用の動的シムテム解析ソフトウェアを用いて計算を行うという方法も広がりつつある[30.99)]。

2003 年から 2009 年にかけて国土交通省官庁営繕部で開発された LCEM ツールもオブジェクト化セルズ法という仕組みでプログラムの再利用性の向上を図ったソフトウェアである。LCEM ツールは表計算ソフトウェア上で動作するが、数式が設定された複数のセル群を 1 つのモジュールと見立て、これらを連結することでシステムモデルを作成する。ただし、LCEM ツールにおける再利用性の確保は、ソフトウェアの維持管理や更新を主目的としたものではなく、ライフサイクルマネージメントを行うためのモデルの連続性という観点にもとづくものである。即ち、企画設計から運用改修に至るまでのライフサイクルを通じて一貫した指標で連続的に性能評価を行うため、建設活動の段階に応じてマクロにもミクロにも検討が可能なプラットフォームを作り、その上でモデルを継続的に使用（再利用）する[†1)]ということである。

情報分野では 1994 年に Java 言語が公開され、機能の抽象化がプログラム言語レベルで表現できるようになった。Java 言語を用いてプログラムの再利用性向上を狙った例としては、2005 年から国土交通省住宅局の主導のもとに開発が進められている The BEST（Building Energy Simulation Tool）Program（以下、BEST）が挙げられる[30.94) †2)]。

30.2 建築熱環境計算の未来

30.2.1 オープンソースの意味

日本における建築熱環境計算ソフトウェアが抱える問題として、その維持・保守・改良のための継続的な活動が不十分であるという点は度々指摘されており、この打開策としてソフトウェアのオープン化が主張されることは多い。しかしこの検討の前提として「オープン化」の意味を明確にする必要がある。これまでもソフトウェアのオープン化に関する議論は数多く行われてきたが「オープン化」の定義が各自で異なり、意味をなしていない場合があるからである[30.95) 30.97) 30.107)]。

先に述べた情報分野におけるオープンソース・ソフトウェアの大きな特徴は、ソースコードが公開されていることと、その再頒布が自由であるということにある。オープンソース・ソフトウェアを用いて営利活動を営むことは自由であり、ソフトウェアの対価を求めることさえ認められている。ただし、再頒布が自由であるため、ソフトウェアを入手した第三者がさらに安い価格で再頒布することも可能であり、実質的にはソフトウェア自体に対して対価を要求することは難しい。従って、通常は当該ソフトウェアに関わるサービス（修正、改良、コンサルティングなど）を提供して対価を得るという形式となる。

†1 「継続的に使用する」とは同じ機器特性を使い続けるという意味ではない。設計変更などがあった場合にはモデルを更新し、その前後の性能を定量的に把握して性能の見込みに対する連続性を保証するということである。

†2 なにかと比較になることが多い LCEM と BEST であるが、開発者の布陣という意味では BEST は HASP/ACSS、LCEM は HVACSIM+(J)に繋がり、前者は計画原論側、後者は実務性向が強く建築設備工学側の人間が多いように思う。

上記の意味でオープン化している熱環境計算ソフトウェアとしては、英国 Strathclyde 大学の ESP-r が挙げられる。ESP-r は 1970 年代から開発が継続されており、ライセンスとして GPL（General Public License）が適用されている。また、HASP/ACLD/8502 と HASP/ACSS/8502 は 1985 年から有償で販売され、これを購入することでソースコードを入手することもできたが、派生的なソフトウェアの開発や再頒布は明示的には認められていなかった。しかし 2012 年から一般に無償で公開されることになり、使用許諾書を読む限りは、現在は情報分野におけるオープンソース・ソフトウェアと極めて近いライセンスである。

GPL はソースコードを入手した第三者にもソースコードの公開と再頒布を義務付けている。つまり、原情報の作成者が著作権を留保し、これを積極的に利用することで、ソースコードをもとに作成された二次的な創作物を含めてオープンであり続けることを強制するものである†1)。しかし、このような制約があると、オリジナルのソースコードに付加した部分に企業秘密に関わる情報が含まれる場合などに利用が難しくなる。そこで、派生的ソフトウェアに対してはソースコードの公開までは要求しないというライセンスがあり、EnergyPlus が採用している BSD（Berkeley Software Distribution）ライセンスはその代表例である。ただし、EnergyPlus は過去にユーザーの種別に応じて複数のライセンスを揃えていた時期もあり、試行錯誤の結果、現在の BSD ライセンスに落ち着いたという歴史は知っておく必要がある[30.109)]。

そもそも著作権自体を完全に放棄することでオープン化するという手法もある。public domain である HVACSIM+はこの意味でのオープン化が達成されている。また、日本においていくつかの改良が加えられた HVACSIM+(J)も同様に public domain である。しかし、これは開発者である中原信生の思想が支えた結果であり†2)、public domain をもとに生み出された派生ソフトウェアが常に public domain となる保証は無いことには注意しなければならない。

上記は多人数が関わる大規模なソフトウェア開発の例であるが、多くの大学研究機関や企業の担当部署では 1 人~数名で開発が行われていることが実情であろう。EESLISM の開発者である宇田川によれば、EESLISM には明確なライセンスは設けず、問い合わせがある度にソフトウェアを配布していたとのことである。実態としては多くの研究機関でこのような運用がなされていると推測できる。

30.2.2　サービス業としての可能性

熱環境計算技術者はどのように生き残れば良いのかという問題がある。製品としてソフトウェアを販売するという道は従来からあるが、2 つの理由により、この業種の将来は明るくないと筆者は考える。第一に、ソフトウェアは継続的なメンテナンス（機能拡充とバグ修正）を必要とするため、1 度の売り切りにより将来に向けて果てしない製品保証をすることは困難である†3)。第二に、計算結果の

†1　多くの場合は情報を隠蔽するために用いられる著作権（コピーライト）を用いて、逆にその保護対象が公開され続けることを保証するという意味で、この仕組をコピーレフトと呼ぶ。公開の対象はオリジナルのソースコードだけではなく、第三者が追加した部分にも適用される。フリーソフトウェアの拡大再生産を狙った Richard Stallman の戦略であり、その効果により本条項をウィルス条項と呼ぶ者もある[30.16)]。

†2　参考文献 30.106 の議事には new HASP の公開について中原が強く迫る様子が記録されている。猪岡によれば、そもそも HASP の生みの親である松尾陽も BECS 開発当時からその公開を望んでいたようだが、開発者の一声だけですぐにどうにかなるものではないようで、難しい問題である。なお、商用化に関して松尾は以下の言葉を残している。「コードが細部まで行き届いたトータルなものになるにつれて、商品価値を生じ、知的所有権などの対象にもなってくる。HASP/ACLD 程度のものでも一部そうした動きがあり、ちょっとしたトラブルにもなったことがある。筆者は商用化を一概に排斥するつもりはない。むしろ本当に行き届いたよいものができるならば、しかるべき企業に頼んで商品に仕上げてもらい、適正な価格で購入すれば好都合だと思っている」[30.110)]

†3　例えば BECS/CEC/AC for windows は 1998 年に販売されたが、筆者が 2008 年に猪岡にヒアリングをした頃にもユーザーからの問い合わせが断続的にあった。

信頼性を保証するためには、究極的にはソースコードが開示されていなければ不可能であり、これは製品としてソフトウェアを販売するという仕組みと矛盾する。

熱環境計算技術者が進むべき道は製造業者ではなくサービス業者であると筆者は考える。即ち、ソフトウェアの開発により直接的に対価を受けるのではなく、ソフトウェアに関連する行為（利用に関する解説の提供、計算結果の解釈の提供、利用の代行など）について対価を求めるのである。このようなサービス業が成立するのであれば、オープンソース・ソフトウェアは業界への参入者が共通して用いることのできる社会的なインフラと位置づけることができる。幸いなことに、熱環境計算技術者がサービス業者として市場での競争力を高めたいのであれば、2 つの点でオープンソース・ソフトウェアの開発に貢献する価値がある。1 つ目は名声の獲得である。情報産業においてサービスの価格はサービス提供者の格（名声）によって決まる。これを梅棹忠夫は「お布施の原理†1)」と呼び、Eric Raymond は「贈与経済†2)」と呼んだ。2 つ目は実質的な能力の獲得である。ソフトウェア自体では他者との差別化を図ることはできないため、サービス業者はソフトウェアに関するノウハウを習得する必要がある。このための一つの有力な方法はオープンソース・ソフトウェア開発への参加であり、例えば、526 名のオープンソース・ソフトウェア開発者に対する Boston consulting group による調査では、オープンソース・ソフトウェア開発への参加のメリットとしてアンケート回答者の 87 %が「コミュニティでの評判の向上」、92 %が「知識の増大」を挙げている[30.111)]。

2）サービス対象となる問題の拡張

熱環境計算の研究は、歴史的には前田、藤井、長谷川らの壁体の非定常熱伝導、単室の非定常熱負荷計算、設備システムを含めたエネルギー計算へと問題の範囲を拡張してきた。今後も 1 つの流れとしては解くべき問題の範囲の拡張があると予想でき、第 20 章であつかった経済性と環境性[30.112)]や第 27 章であつかった熱的快適性はその代表例である。ただし、おそらく従来とは異なり「連成」はあまり問題にならず、むしろ、どのような情報を境界条件として受け渡し、計算結果を解釈するための方法をどのように担保するかということが問題となるだろう。例えば 1985 年に開発された HASP/ACSS は熱負荷と設備システムを完全に連成させて、設備の過負荷状態を室温変動にフィードバックすることができたが、後に 1998 年に HASP/ACSS を下敷きに開発された BECS/CEC/AC は未処理負荷の概念を導入して厳密な連成は取りやめている。猪岡によれば、過負荷状態による室温と設定値の乖離について理解できない設計者がいたため、より解釈が容易な未処理負荷の概念が生まれたとのことである。建物のエネルギー消費を例えば不動産価値や人間の熱的快適性などの問題につなげる際に、連成問題として捉えることにどれだけの価値があるのかは一考が必要である。この問題に限らず、おそらく理学的な厳密さに関しては技術者の誠意によって十分過ぎるほどに供給が維持されるが、問題を解釈するための観点の設定という社会科学的な検討に関しては今後も供給不足が続くであろう。

30.2.3　制度化という陥穽

法律や認証制度への対応は、サービス提供の 1 つの形態となりうる。例えば、米国の環境認証制度である LEED（Leadership in Energy & Environmental Design）では、認証取得のために建築設備システムのエネルギー計算を義務付けている。また、日本の「エネルギーの使用の合理化等に関する法律」

†1　お布施の額を決定するものは、お経の情報量をビットで計測した値ではなく、坊さんと檀家の格という二つの人間の社会的位置であるというもの[30.11)]。

†2　計算機能力や記憶領域がもはや不足することのない環境においては、希少性を前提とした交換経済は成立せず、仲間内の評判が成功の唯一の尺度となるため、贈与活動が活発化するという指摘[30.17)]。

では、建物のエネルギー消費量を計算して申請する必要があり、過去には「BECS/CEC/AC」が、現在では「BEST」と「エネルギー消費性能計算プログラム（WEBPRO）」がこれを担っている。このような制度に対応するための熱環境計算の代行は報酬を受けるべきサービスである。一方で、注意すべき点は、制度は公平性を重視するがために形式を偏重してしまう可能性があるということである。賛否のあり得る問題に対しても公平な結果を提供するためには、入力条件や計算過程の自由度は狭められる傾向となる。このような傾向は、性能の低い建物をマニュアル化可能な水準まで持ち上げるという底上げとしては有効だが、一方で、先導的な建築は適正に評価されない可能性がある。実質的な価値の創出に挑戦して結果責任を負うよりも、計算過程の法適合性という形式的操作に注意を払うことが技術者にとっての保身になる†1)。しかし、このような形式的操作こそ、コンピュータに代替させるべき創造性の欠落した作業であり、これが熱環境計算技術者の主たる業務になるとすれば、その未来は暗い。社会的な背景により建築の環境性能への関心が高まり、その性能についての制度が強化されることは、一面においては熱環境計算に関わる技術者の発言権が大きくなるという効果もある。しかし同時に、制度の上に安住・寄生する技術者を生み[30.108]、長期的には熱環境計算技術者の質を低下せしめ職を失わせる危険性すらあることを認識する必要がある。

30.2.4　帰納的方法への対応

本書では理論式にもとづく演繹的なモデルの作成法を中心に解説を行ってきたが、情報分野における機械学習の発達の流れを受け、今後は帰納的なモデル化への対応が必要になると予想できる。建築熱環境分野で機械学習が大きな成果を挙げた例として、空気調和・衛生工学会による負荷予測ベンチマークテストが挙げられる[30.113]。本テストは、蓄熱運転制御の適正化に必要な負荷予測技術の相互比較と情報交換を目的に実施されたテストである。東京都の現実の建物の負荷データを用いて前日の天気情報などから翌日の熱負荷を予測するという問題であった。19 の参加者が負荷予測性能を競い、首位のニューラルネットワークを用いた手法は、ピーク負荷に対して 3 %を下回る誤差率となった。同様のテストは米国 ASHRAE においても開催されており[30.116]、日本と同様、機械学習による予測が優秀な成績をおさめた。

大量の BEMS データの蓄積もまた、帰納的な方法を後押しする。筆者は設備システムの運転データをニューラルネットワークに学習させていくつかの設備機器のモデルを作成したが[30.114) 30.115]、少なくとも内挿性能に関しては簡単な物理モデルを超える性能を示すと判断している。近年では、設備システム側よりもむしろ居住域にいる人間側の情報の蓄積が大きく進んでおり、これらの両方の情報を含めた機械学習が始まるだろう。人間の快適性は物理式に支配された設備機器よりも遥かに個別性と不確実性が大きく、実験室実験ではなく現場でパラメータを推定する機械学習が有効であると予想できる。しかし現実にそこで熱環境計算技術者の担うべき役割は簡単ではない。前記の日本と米国の熱負荷予測ベンチマークテストの優勝者は、いずれも建築環境系の研究者ではなく情報系の研究者であったことは認識すべきである。完全なデータドリブンでのモデル化であれば建築環境系の技術者が積み上げた理論は不要であり、情報系研究者のデータ処理の問題となる。建築環境系の技術者が情報系に鞍替えせず、自身の分野と歴史的な連続性を保って機械学習の時代を生きるためには、理論式にもとづく演繹的モデルと観測データにもとづく帰納的モデルの狭間の領域にその道を見出す必要がある。

†1　筆者は実務を行っていた際に、一部の構造設計者が制度を笠に着て構造設計者としての検討を放棄する場面を何度か見た。建築構造分野は建築環境分野に比較して法的な縛りが大きいことも 1 つの理由であろう。

機械学習との住み分けを考える際には内挿と外挿の違いを意識する必要がある。機械学習は既に経験した範囲や経験しうることが明示されている範囲、即ち内挿範囲に関しては非常に有効である。一方で、論理の積み上げにより、従来は存在しなかった範囲に問題を拡張（外挿）することはできない。自動車の制御可能な機構が明示されれば、その制御値を揺らしながら自動走行のためのパラメータを獲得することは可能であろうが、自動車という概念をもとに海を渡る船や空を飛ぶ飛行機という概念を着想することはできない。少なくとも現在のところは、機械学習は内挿の問題について特に有効であり、新たな概念を自発的に生み出すことはない。外挿範囲（問題の拡張）に関しては当面は人間による理論と論理の積み上げが有効である。

30.2.5　熱環境計算技能者の道

熱環境システムの最適化のためには、より積極的に熱環境計算技術を活用すべきであるという主張がなされることは多い。確かに、計算上は各種の制御値や運転方法を仮想的に試行錯誤することができるため、実システムに比較すれば遥かに効率的かつ踏み込んだ検討が可能になる。しかし、これはすべてのシステムの運用にあたって熱環境計算技術者が直接に介在すべきということを意味するものではない。熱環境システムに関わる様々な可能性を試行錯誤できるような人材は希少性が高く、消費エネルギーの多寡によりその費用を回収することは困難である。現実には、すべての熱環境システムに対して個性を反映したほぼ満点の最適化を追求することは非効率である。例えば全体を 95 %と 5 %に分割した上で 95 %に対してコストのかからない 70 点程度の準最適化を狙った方が、社会的に必要な労力との関係という観点では有利であろう。さらに、この 95 %はおそらく先に述べた内挿の問題であり、人間の発想が無くとも機械学習により非常に安価に対応できる可能性が高い。一方で 5 %の創造性を要求される問題は、比率が少ないということにより軽視されるべきではない。この 5 %は先駆的なシステムである可能性が高く、ここで行われる試行錯誤は残余の 95 %のシステムの将来の指針となりうるからである。この意味では、希少性の高い熱環境計算技術者を育てて創造性の高い問題に従事させるということは大きな外部経済性を持っていると言える。先駆的な問題への挑戦によって生まれた技能を技術化し、残余の 95 %のシステムに還元することができれば社会的な余剰が拡大するためである。ただし、このような外部効果を内部化するための安易な仕掛けをつくることには、筆者は反対をしたい。合目的的な冷たい機構などは無くとも、我々は自ずから技に学び戯れることが可能であると信じるからである。

【第 30 章 参考文献】

30.1） M.キャンベル, W. アスプレイ: コンピュータ 200 年史, 山本菊男 訳, 海文堂出版, 1999
30.2） 情報処理学会 歴史特別委員会 編: 日本のコンピュータ史, オーム社, 2010
30.3） 村松貞治郎: 日本建築家山脈 -復刻版- , 東大山脈-三人目の巨頭 内田祥三-, pp.55, 2005
30.4） 渡辺要, 長倉謙介: 計画原論, 1934
30.5） 渡辺要先生追悼出版刊公開: 渡辺要, 鹿島研究所出版会, 1972
30.6） 佐藤鑑, 私の回想と環境工学の生い立ち, pp.11, 1975
30.7） 新建築学体系 10 巻 環境物理学の歴史: 彰国社, 1984
30.8） 松原仁: 情報フロンティアシリーズ 将棋とコンピュータ, 共立出版, 1994
30.9） 松原仁: コンピュータ将棋の進歩 プロ棋士に並ぶ, 共立出版, 2012
30.10） Tamaki Kusuda: Early history and future prospects of building system simulation, Proceedings of Building Simulation 1999, Kyoto Japan.
30.11） 梅棹忠夫: 情報の文明学 情報産業論（お布施の原理）, 中公文庫, p.60, 1999
30.12） The Free Software Definition <http://www.gnu.org/philosophy/free-sw.html> (accessed 2008-07-19)
30.13） Richard M. Stallman, 長尾高広 訳 : フリーソフトウェアと自由な社会, 株式会社アスキー, pp.71, 2003.05
30.14） Open source group Japan : オープンソースの定義, 八田真行 訳, <http://www.opensource.jp/osd/osd-japanese.html>(accessed 2008.09.22)
30.15） ローレンス・レッシグ : CODE–インターネットの合法・違法・プライバシー-, 翔泳社, 2001
30.16） Steven Weber : 山形裕生, 守岡桜 訳, オープンソースの成功 政治学者が分析するコミュニティの可能性, 毎日コミュニケーションズ, 2007
30.17） Eric S. Raymond : The Cathedral and the Bazaar, <http://www.catb.org/~esr/writings/cathedral-bazaar> (accessed 2008-09-20)
30.18） Soumerai, H., Kusuda, T., & Ditfach, J.: Digital computer applied to Compressor design analysis, ASHRAE Journal, 1(7), pp.43-49, 1959
30.19） Tamami Kusuda, NBSLD, the Computer Program for Heating And Cooling Loads in Buildings, Builidng Series 69, National Bureau of Standards, Washington, DC, 1976
30.20） T. Kusuda, NBSLD, Computer Program for Heating and Cooling Loads in Buildings, NBSIR pp.74-574, National Bureau of Standards, Washington, DC, 1974
30.21） Stephenson D. G. and Mitalas G. P. : Cooling Load Calculations by Thermal Response Factor Method, ASHRAE Transactions, Vol. 73, Part 1, 1967
30.22） 渡辺要: 建築計画に関する二三の問題, 建築雑誌, No.49, p.588-594, 1935.5
30.23） 長谷川房雄: 室温変動理論における三つの立場, 日本建築学会論文報告集, 第 69 号, pp.25-28, 1961
30.24） 前田敏男: 室温変動の解析的研究, 日本建築学会論文集, Vol 21, pp.77-84, 1941
30.25） 野手悌士: 両側の温度の振動する壁体熱伝導に就いて, 衛生工業協會誌, 第 4 巻, pp613
30.26） 赤井清康: 周期的変化の場合の中空球の熱伝導の一問題, 気象集誌, 第 18 巻, p.338
30.27） 建築環境工学文献題目集 1887-1963, 平山記念出版会編, 1973.10
30.28） 木村建一: アメリカの新しい空調負荷計算法について, 日本建築学会秋季学術講演会前刷集, pp.79-81, 1969
30.29） 空気調和・衛生工学, 特集 わが国の設備教育の歴史-大沢・桜井両先生を中心にして, pp.1033-1084, Vol.52, No.11, 1978
30.30） 後藤滋: 環境工学序説, 人間と生活環境, Vol.4, 1996
30.31） 煖房冷蔵協會発起人総会議事速記録, 煖房冷蔵協會誌, Vol.1, 大正 6 年 12 月
30.32） 中村達太郎: 建築衛生家屋排水の話, 松相社, 1920
30.33） 中村達太郎: 建築衛生除塵装置と汚水処分, 松相社, 1921
30.34） 斉藤省三: 協会名称ならびに定款検討委員会の動きと空気調和衛生工学会 , 附 煖房冷蔵協会と社団法人衛生工業協会についての思い出, 空気調和・衛生工学雑誌, vol.x, pp.xx, 1962
30.35） 煖房冷蔵協會誌 第 21 号: 新旧役員協議, 1927
30.36） 藤井厚二: 日本の住宅 -普及版-, 岩波書店, 1928
30.37） 日本建築学会:近代日本建築学発達史, 11 編 建築教育, 1972
30.38） 柳町政之助: 米国旅行談 其の一, 衛生工業協會誌, 第 3 巻, 第 11 号, 1929
30.39） 柳町政之助: 米国旅行談 其の二, 衛生工業協會誌, 第 3 巻, 第 12 号, 1929
30.40） 邦楽座の建築設備, 煖房冷蔵協會誌, 第 20 号, 大正 14 年 1 月
30.41） 意見交換会（北浦重之:建築家並に建築物の機械設備に関する技術家諸君に対する希望）, 煖房冷蔵協會誌, 第 20 号, 大正 14 年 1 月
30.42） 丹羽重光: 煖房設計料に就て, 衛生工業協會誌, 第 2 巻, 第 3 号, 1928
30.43） 井上宇市: 「建築設備」と私, 丸善株式会社, 1989
30.44） 井上宇市: 冷凍空調史, 日本冷凍空調設備工業連合会, 1993

30.45） マーガレット・インゲルス: 空気調和の父 ウィリス・ハヴィランド・キャリア（復刻版）, 東洋キャリア工業株式会社, 1990
30.46） 斉藤孝: 歴史と歴史学, 東京大学出版会, p.79, 1975
30.47） 木村建一: かんきょう随想 第4回 コンピュータ利用の曙, 建材試験情報7, pp.26-28, 2005
30.48） 木村建一: かんきょう随想 第5回 楠田博士との出会い, 建材試験情報9, pp.40-42, 2005
30.49） Kusuda, T.: Proceedings of the First International Symposium on the Use of Computers for Environmental Engineering Related to Buildings, National Bureau of Standards, 1970
30.50） Kusuda, T.: How to promote international collaboration?, Pan pacific symposium on building and urban environmental conditioning in asia, Nagoya, Japan, 1995.3
30.51） Lokmanhekim, M.: Proposed Procedures for Determining Heating and Cooling Loads for Energy Calculations, ASHRAE, 1968
30.52） 蓄熱技術の歴史・現在・未来, 建築設備と配管工事, Vol.50, No.10, 2012.8
30.53） 柳町政之助: 煖房と換気 前編, 大日本工業学会, 1920
30.54） 柳町政之助: 煖房と換気 後編, 大日本工業学会, 1922
30.55） 木村建一, 石野久彌: 電算機による動的空調負荷計算法, 空気調和・衛生工学, 46巻 3号, pp.3-38, 1972
30.56） 松尾陽: 手計算による動的空調負荷計算法, 空気調和・衛生工学, 46巻 3号, pp.39-76, 1972
30.57） 松尾陽, 横山浩一, 石野久彌, 川元昭吾: 空調設備の動的熱負荷計算入門, 日本建築設備士協会, 1980
30.58） 日本建築設備士協会 空気調和設備シミュレーション研究会: 国内の空調装置シミュレーションプログラムの比較, 空気調和・衛生工学, 第58巻, 第7号, pp.65-78 (pp.683-696), 1984
30.59） 松尾陽 ,猪岡達夫 ,横山浩一: 空調システムのエネルギーシミュレーションプログラム ：第1報 プログラムの概要と計算結果の検討, 空気調和・衛生工学会 学術講演会論文集, pp.425-428, 1985
30.60） 空調システム標準シミュレーションプログラム HASP/ACSS/8502 プログラミングガイド, 日本建築設備士協会, 1986
30.61） 標準気象データに関する研究, 空気調和・衛生工学, 第48巻, 第7号, pp.85-107, 1974
30.62） 牧英二: 設備計画とコンピューター, 日本建築学会 建築雑誌, 第85巻, 昭和45年2月号, pp.127-131, 1970
30.63） 滝沢博: 標準問題の提案（オフィス用標準問題）, 日本建築学会環境工学委員会 熱分科会第15回熱シンポジウム, pp.35-42, 1985
30.64） 宇田川光弘: 標準問題の提案（住宅用標準問題）, 日本建築学会環境工学委員会 熱分科会第15回熱シンポジウム, pp.23-33, 1985
30.65） R. Judkoff, D. Wortman, and B. O'Doherty: A Methodology for Validating Building Energy Analysis Simulations, SERI/TR-254-1508. Golden, CO: Solar Energy Research Institute, 1983
30.66） R. Judkoff, J. Neymark: Internatinal Energy Agency Building Energy Simulation Test (BESTEST) and Diagnostic Method, National Renewable Energy Laboratory, 1995
30.67） J. Neymark, R. Judkoff, G. Knabe, M. Durig: NREL/CP-550-29828, HVAC BESTEST, A Procedure for Testing the Ability of Whole Building Energy Simulation Programs to Model Space Conditioning Equipment, National Renewable Energy Laboratory, 2001
30.68） 中原信生: 環境・エネルギー性能の最適化のためのBEMSビル管理システム, 社団法人空気調和・衛生工学会, 丸善株式会社, pp.315, 2001
30.69） Daniel R. Clark: HVACSIM+ Building Systems and Equipment Simulation Program-Reference Manual, National Bureau of Standards NBSIR 84-2996
30.70） Daniel R. Clark and William B. May, Jr: HVACSIM+ Building Systems and Equipment Simulation Program-Users Guide, National Bureau of Standards NBSIR 84-2996
30.71） 新建築2010年9月別冊 日本IBM本社ビル 1971-2009 建築とファシリティマネジメントのライフタイム記録, 2010
30.72） HVACSIM+(J) 利用者マニュアル, 2004
30.73） 江口和雄: 建物の伝熱算法, 伝熱算法の発展と現状, 熱シンポジウム 予稿集 1回, pp.1~9
30.74） 林徹夫: 規格・標準化の検討課題（その2）ベンチマークテスト, 日本建築学会 環境工学委員会, 熱環境小委員会第30回熱シンポジウム, pp79-88, 2000
30.75） 宇田川光弘, 木村建一 ：多数室室温変動の実用的計算手法と断熱雨戸の熱的効果の検討例, 日本建築学会論文報告集, 第265号, 1978.03
30.76） 長谷川房雄, 石川善美, 松本博 ：室内相互ふく射を考慮した多数室室温変動, 日本建築学会論文報告集, 第323号, 1983.1
30.77） 土屋喬雄 ：パッシブソーラーシステムの熱性能に関する研究, 日本太陽エネルギー学会第8回研究発表会講演論文集, 1983
30.78） 坂本雄三 ：省エネルギー住宅システムの開発（建設省総プロ報告書）, （財）住宅・建設省エネ機構, 1983
30.79） 住機能向上調査委員会 ：LESCOM改訂作業報告書, （社）日本住宅設備システム協会, 1983.06

30.80）渡辺俊行 ：壁体表面放射収支の定量化とその室温変動解析への応用 , 日本建築学会建築環境工学論文集第 4 号, 1982.11
30.81）荒谷登, 絵内正道 ：逐次積分法による室温及び負荷変動の解析 , 北海道大学工学部研究報告 , No.51, 1968.12
30.82）松浦茂: パーソナルコンピューティング時代における APL を用いた連続系シミュレーションパッケージの基本設計概念とアルゴリズム, 北海道大学工学部研究報告, p.113, 1982
30.83）伊藤直明, 土屋喬雄, 高間三郎, 高倉直, 小玉祐一朗, 梅干野晃, 須永修通, 武政孝治: パッシブソーラーシステムの計画プロセスに関する研究 その 1 計画プロセスの基本的考え方, 日本建築学会 学術講演梗概集. 計画系, pp.509-510, 1982
30.84）M. Ishizuka, K.Kobayashi, T.Nakamura: Validation of Simulation Program for Passive Solar Houses, Proceedings of the Building Energy Simulation Conference 1985, pp.360-367, 1985
30.85）E.W.ダイクストラ: 構造化プログラミング, 野下浩平 訳, サイエンス社, 1975
30.86）Klein, Sanford: A design procedure for solar heating systems, Thesis (Ph. D.), University of Wisconsin Madison, 1976
30.87）橋本喬行: 北浜五丁目十三番地から 日建設計の 100 年, 創元社, p.151, 1999
30.88）紀谷文樹 他: 建築環境設備学 新訂版, 第 I 編 4 章 建築環境設備学の歴史, 彰国社, 2003
30.89）石原正雄: 環境調整の技術史 1 暖房の技術 1~4, 日本建築学会近畿支部 研究報告集, 1976~1982
30.90）石原正雄: 環境調整の技術史 2 換気の技術 1~4, 日本建築学会近畿支部 研究報告集, 1977~1983
30.91）石原正雄: 環境調整の技術史 3 空気調整の技術 1~5, 空気調和・衛生工学会学術講演会論文集, 1977~1981
30.92）石原正雄: 建物における熱・空気環境調整の技術史序説 -ヨーロッパ・アメリカにおける 19 世紀までの-, pp.201-211, 1984
30.93）時田繁, 松縄堅, 丹羽英治, 他: ライフサイクルエネルギーマネージメントのための空調システムシミュレーション開発 (第 1~34 報), 空気調和・衛生工学会大会学術講演論文集, 2005~2010
30.94）村上周三, 松尾陽, 坂本雄三, 石野久彌, 大塚雅之, 赤坂裕, 滝澤総, 野原文男: 外皮・躯体と設備・機器の総合エネルギーシミュレーションツール「BEST」の開発 (その 1) BEST 開発の背景と趣旨, 空気調和・衛生工学会大会学術講演論文集, pp.1969~1972, 2007
30.95）石野久彌, 郡公子: 空調ソフトウェアのコンポーネント・ライブラリーの開発, 空気調和・衛生工学会学術講演会論文集, pp.517-520, 1986
30.96）石野久彌: 空気調和設備委員会熱負荷算法小委員会成果報告, 空調ソフトウェア・COMPAS に関する研究, 空気調和・衛生工学, pp.69~78, 1988
30.97）空気調和衛生工学会 空気調和設備委員会 熱負荷算法小委員会（S57.04~H01.03） 報告書
30.98）H.Ishino, M.Shukuya and K.Kohri : Development of a Component Program Library for Building Enegy Simulation, A Japanese Experience, Proceeding of Buuilding Simulation 89, pp.363-pp.369, 1989
30.99）吉田治典, 王福林, 小野永吉, 新宮浩丈, 堀川晋, 田中宏昌, 高村秀明: シミュレーションを用いた複合熱源システムの最適組合せ運転法に関する研究, 空気調和・衛生工学会 学術講演会論文集, pp.2129~2132, 2007
30.100）佐藤鑑: 建築環境学, 紀元社, 1948
30.101）平山嵩: 建築設計理論, 科学政策会, 1948
30.102）大澤一郎, 小林陽太郎: 設備工学講座 第 1 設備原論, 共立出版, 1958
30.103）建築学大系 第 22, 室内環境計画, 1957 年
30.104）伊藤正文: 建築保健工学, 産業図書, 1948
30.105）竹中工務店 web サイト, 竹中のあゆみ, <http://www.takenaka.co.jp/corp/archive/years> (accessed 2016-08-20)
30.106）空気調和・衛生工学会 空気調和設備委員会空調システム動的シミュレーション評価小委員会 H8.04~H11.03 報告書
30.107）空気調和・衛生工学会 熱負荷・システムシミュレーション法小委員会, 熱負荷・空調ソフトウェアの現状と将来シンポジウム資料, 2006
30.108）ミルトン・フリードマン: 資本主義と自由, 村井章子 訳, 日経 BP 社, 2008
30.109）富樫英介, 田辺新一: 建築熱感強シミュレーションソフトをオープン化する方法について , 日本建築学会環境工学委員会 熱環境運営委員会 第 39 回 熱シンポジウム 『熱環境シミュレーションの拡がり』, pp.45~52, 2009
30.110）松尾陽: 伝熱シミュレーション・過去→未来, 日本建築学会 環境工学委員会 熱環境運営委員会 第 39 回熱シンポジウム 『熱環境シミュレーションの拡がり』, pp.5~8, 2009
30.111）Bates, J., DiBona, C., Lakhani, K.R., & Wolf, B.:. The Boston Consulting Group Hacker Survey. Open Source Development Network (OSDN), (release 0.73, 2003)
30.112）富樫英介: 設備システムの省エネルギー化が不動産価値に与える影響の定量的評価方法に関する研究 : 第 1 報-直接還元法に基づく省エネルギー化投資の評価方法とケーススタディ , 空気調和・衛生工学会論

文集 Vol.196, pp.29-37, 2013
30.113） 川島実: 特集 蓄熱空調システムの運転制御の適正化 負荷予測ベンチマークテストの概要報告, 空気調和・衛生工学 第 73 巻 第 9 号, pp.37-42, 1999
30.114） 富樫英介, 田辺新一, 渡邊進介, 高橋慎一, 渡辺剛: 建築系環境・情報マネージメントシステムに関する研究 (第 5 報) BEMS データとリカレント型ニューラルネットワークによるコミッショニング用動的シミュレーションモデルの開発, 空気調和・衛生工学会 学術講演会論文集, pp.641-644, 2005
30.115） 富樫英介, 田辺新一, 横山計三, 高橋慎一, 渡邊進介, 渡辺剛, 田中太一: 建築系環境・情報マネージメントシステムに関する研究 (第 7 報) ソフトコンピューティングを用いた設備機器性能モデルの作成, 空気調和・衛生工学会 学術講演会論文集, pp.1681-1684, 2006
30.116） Jeff S. Haberl, Sabaratnan Thamilseran: Great Energy Predictor Shootout II: Measuring retrofit savings - overview and discussion of results, ASHRAE Transactions 102, pp.419-435, Jan., 1996
30.117） 柳町政之助: 我国に於けるターボ冷凍機の誕生を懐古して, 日本冷凍空調協会, 冷凍, 第 30 巻, 第 332 号, pp.1-5, 1955
30.118） 木村建一: 井上先生の光る足跡（特集 井上先生の思い出）, 空気調和・衛生工学会 機関誌, 第 85 巻, 第 1 号, 2011
30.119） 牧英二, 猪岡達夫: シミュレーション・プログラム, 空気調和・衛生工学会 機関誌 特集 空調計画への電算機応用, 第 47 巻, 第 7 号, pp.602, 1973
30.120） 猪岡達夫: 省エネルギー法に基づく CEC/AC のシミュレーションプログラム BECS, 日本建築学会環境工学委員会 熱環境小委員会 第 29 回 熱シンポジウム, pp.75-84
30.121） 渡辺要, 木村宏, 村松貞次郎: 「計画原論」前後, 建築雑誌, 77 巻, 915 号, 日本建築学会, pp.489-495, 1962
30.122） 佐藤鑑: 環境調整（計画原論）の 2,3 の問題, 建築雑誌, 74 巻, 874 号, 日本建築学会, pp.1-5, 1959
30.123） 佐藤鑑: 計画原論の役割について, 建築雑誌, 69 巻, 813 号, 日本建築学会, pp.1-5, 1954
30.124） 前田敏男: 建築計画原論の発展のために, 建築雑誌, 73 巻, 857 号, 日本建築学会, pp.39-42, 1958
30.125） 前田敏男: 計画原論の進むべき道, 建築雑誌, 74 巻, 867 号, 日本建築学会, pp.23-24, 1959
30.126） アーバナの火 工学博士櫻井省吾生誕 100 年記念誌, 櫻井研究所, 1999

クラス一覧

名前空間	class, interface 名称	機能	章
Popolo.Electric	PhotovoltaicPanel	太陽光発電パネル	18
Popolo. HumanBody	MultiNodeModel	多質点人体モデル	27
	ThermalComfort	熱的快適性に関する静的メソッドを提供する	27
	TwoNodeModel	2-Node モデルに関する静的メソッドを提供する	27
Popolo.HVAC	HVACSystemModel	建物・熱源・空調設備連成モデル	29
Popolo.HVAC. AirConditioner	AirHandlingUnit	空気調和機	28
Popolo.HVAC. Circuit	CentrifugalFan	遠心ファン	16
	CentrifugalPump	遠心ポンプ	16
	CircuitNetwork	回路網	17
	CircuitNode	回路網の節点	17
	Conduit	管内流れに関する静的メソッドを提供する	17
	FluidMachinery	流体機械に関する抽象クラス	16
	ICircuitBranch	回路網の流路	17
	PumpSystem	ポンプシステム	16
	Regulator	流量制御バルブ・ダンパ	17
	SimpleCircuitBranch	圧力損失が体積流量の 2 乗に比例する単純な流路	17
	WaterPipe	水配管	1, 17
Popolo.HVAC. HeatExchanger	AirToAirFlatPlateHeatExchanger	空気対空気の静止型熱交換器クラス	11
	CoolingTower	冷却塔	12
	CrossFinCondensor	プレートフィン付管 凝縮器	10
	CrossFinEvaporator	プレートフィン付管 蒸発器	10
	CrossFinHeatExchanger	プレートフィン付管 熱交換器	9
	HeatExchange	熱交換に関する静的メソッドを提供する	8
	PlateHeatExchanger	プレート熱交換器	8
	RotaryRegenerator	回転型熱交換器	11
Popolo.HVAC. HeatSource	AbsorptionRefrigerationCycle	吸収冷凍サイクルに関する静的メソッドを提供する	15
	AirHeatSourceModularChillers	空気熱源モジュールヒートポンプ	14
	Boiler	温水・蒸気ボイラに関する静的メソッドを提供する	13
	DetailedCentrifugalInverterChiller	インバータターボ冷凍機	14
	DirectFiredAbsorptionChiller	直だき二重効用吸収冷温水発生機	15
	HotWaterAbsorptionChiller	温水吸収冷凍機	15
	HotWaterBoiler	温水ボイラ	13
	ICentrifugalChiller	ターボ冷凍機	14
	SimpleCentrifugalChiller	特性式による簡易ターボ冷凍機	14
	SteamBoiler	蒸気ボイラ	13
Popolo.HVAC. SolarEnergy	FlatPlateSolarCollector	平板式集熱器	18
	SimpleSolarCollector	特性式による簡易集熱器	18
Popolo.HVAC. SubSystem	AHUSystem	AHU による二次側空調システム	28
	AirHeatSourceModularChillersSystem	空気熱源モジュールヒートポンプによる 熱源サブシステム	21
	CentrifugalChillerSystem	ターボ冷凍機による熱源サブシステム	21
	DirectFiredAbsorptionChillerSystem	直焚吸収冷温水機による熱源サブシステム	21
	HeatSourceSystemModel	熱源一次システム	21

Popolo.HVAC.SubSystem	HotWaterBoilerSystem	温水ボイラによる熱源サブシステム	21
	IAirConditioningSystemModel	二次側空調システム	28
	IHeatSourceSubSystem	熱源設備サブシステム	21
	MultipleStratifiedWaterTankSystem	温度成層型蓄熱槽による熱源サブシステム	21
Popolo.HVAC.ThermalStorage	MultiConnectedWaterTank	連結完全混合型蓄熱槽	19
	MultipleStratifiedWaterTank	温度成層型蓄熱槽	19
Popolo.Numerics	CubicEquation	3 次方程式ソルバ	2
	CubicSpline	3 次スプライン補間処理クラス	2
	MersenneTwister	メルセンヌ・ツイスターによる擬似乱数製造機	2
	Minimization	1 変数非線形関数の最小化処理クラス	2
	MultiMinimization	多変数非線形関数の最小化処理クラス	2
	MultiRoots	多変数非線形関数の求根を行うクラス	2
	NormalRandom	正規分布に従う乱数系列を作成するクラス	2
	Roots	1 変数非線形関数の求根を行うクラス	2
Popolo.Numerics.MatrixOperation	IMatrix	行列インターフェース	2
	IVector	ベクトルインターフェース	2
	LinearAlgebra	線形代数処理に関する静的メソッドを提供する	2
	Matrix	行列	2
	MatrixView	部分行列	2
	SparseMatrix	疎行列	2
	Vector	ベクトル	2
	VectorView	部分ベクトル	2
Popolo.ThermalLoad	AirGapLayer	空気層	22
	BuildingThermalModel	建物の熱負荷計算モデル	25
	BuriedPipe	埋設配管	22
	IHeatGain	発熱要素インターフェース	25
	IShadingDevice	窓ガラス中空層の日射遮蔽物	23
	MultiRooms	多数室	25
	PCMWallLayer	潜熱蓄熱材料を用いた壁層	22
	SimpleHeatGain	発熱要素	25
	SunShade	日除け	23
	VenetianBlind	ベネシャンブラインド	23
	Wall	壁	22
	WallLayer	壁層	22
	wallWindowSurface	壁・窓表面	25
	Window	窓	24
	Zone	1 質点を持つ空間	25
Popolo.Thermophysical Property	LithiumBromide	臭化リチウム	15
	MoistAir	湿り空気	4
	Refrigerant	冷媒	3
	Water	水の物性計算に関する静的メソッドを提供する	5
Popolo.Weather	Incline	傾斜面	6
	RandomWeather	確率的気象計算に関する静的メソッドを提供する	7
	Sky	天空に関する静的メソッドを提供する	6
	Sun	太陽	6

索引

【著者略歴】

富樫 英介

1980.06　出生
2004.03　早稲田大学理工学部建築学科 卒業
2006.03　早稲田大学理工学研究科建築学専攻修士課程 修了
2009.03　早稲田大学理工学研究科建築学専攻博士課程 修了
2009.04-2009.12　早稲田大学創造理工学部 助手
2010.01-2015.03　株式会社日建設計
2015.04-現在　学校法人工学院大学 准教授

熱環境計算戯法

The Art of Thermal Environmental Computing

著　者　富樫 英介
出版者　工学院大学 建築学部建築学科 富樫研究室
表紙デザイン: 中川 純,　題字: 久保木 真俊

ISBN 978-4-9908908-1-0

Version 履歴

Version	日付	Popolo	内容
2.0.0	2016.12.31	2.0.0	一般公開
2.1.0	2017.10.01	2.1.0	熱負荷計算高速化、バグフィクス
2.2.0	2019.01.16	2.2.0	式記号などの軽微な誤植を修正