

## Popolo による熱負荷計算 version 1.0.0

本資料では、熱環境計算ライブラリである Popolo を使って建物の熱負荷計算を行う手順を示す。

### 内容

1. 熱負荷計算の意義 .....	2
2. 計算環境の導入 .....	2
2.1. 熱環境計算ライブラリ Popolo .....	2
1) Popolo.....	2
2) ライブラリ .....	2
2.2. Microsoft Visual Studio .....	3
1) インストール.....	3
2) 最初のプログラム .....	3
2.3. Popolo の導入.....	6
2.4. 湿り空気の飽和状態の計算例 .....	7
3. 熱負荷計算 .....	9
3.1. 計算対象の整理 .....	9
3.2. 建物モデルの作成 .....	10
3.3. 年間自然室温の計算 .....	13
3.4. 年間熱負荷の計算 .....	15
3.5. 年間熱負荷の計算 2 .....	18

## 1. 熱負荷計算の意義

空調の目的は空気の温湿度、気流、清浄度を操ることで室内の空気環境を快適な状態に維持することにある。この中で特に温度と湿度を制御するためには、部屋に対して外部から熱を供給したり、逆に部屋から外部に熱を取り去ったりする必要がある。冬季の暖房加湿は前者、夏の冷房除湿は後者にあたる。このように温度と湿度を制御するために必要となる熱を「熱負荷」と呼ぶ。

我々が建物の中で快適に過ごすためには、電力やガスといったエネルギーを消費して建物の内部の環境を制御しなければならない。このエネルギーの内、おおよそ半分は熱環境を制御するために費やされる。従って、建物の熱負荷の予測は建物のエネルギー性能を予測する上で大きな意味を持っている。

建物の熱負荷は、その断熱やガラスの仕様、壁や窓の面積、空調の時間、執務者たちの行動、などによって変化する。従って、建物の形状や運用方法とエネルギー消費の関係性を定量的に把握し、それらを改善する上でも熱負荷計算という技術は欠かすことができない。

## 2. 計算環境の導入

本章では具体的な計算の前提となる計算環境の導入手順を示す。

### 2.1. 熱環境計算ライブラリ Popolo

#### 1) Popolo

計算機による熱負荷計算は 1960 年代から 50 年を超える歴史を持つ技術であり、様々なソフトウェアが開発されている。本資料ではオープンソースの熱環境計算ライブラリである **Popolo** を用いる。同ライブラリは商用ソフトウェアではないため、グラフィカルなユーザーインターフェースは持たない。一方で、ソースコードレベルで操作できるように特殊で発展的な研究を目的とする場合には便利である。

「.NET core」に対応しているため、Windows, Mac, Linux のいずれでも動作する。また、無償公開された 800 ページ程度の詳細なマニュアルにはソースコードのすべてが解説されている。

→ <https://www.hvacsimulator.net/popolo>

#### 2) ライブラリ

計算機を使った処理を行う場合には、様々な場面で繰り返し登場する処理がある。それは、熱環境計算で言えば、湿り空気の物性計算であったり、熱交換器の計算であったりする。このような頻出する計算処理（プログラム）は、登場するたびに新たに一から書き直すのでは無駄に時間がかかるし、書き損じなどで間違いが発生する原因にもなる。そこで、このような問題を解決するために、できるだけ汎用性を高めた状態でこれらの頻出するプログラムをまとめておくことが多い。このようにすれば、必要に応じてこれらのプログラムを呼び出すという方法を取ることができるから手間が省けるし、共通のプログラムを繰り返し使用することで、多くのバグが取り除かれて堅牢なプログラムに育つ。このようなパッケージ化されたプログラム群を一般に「ライブラリ」と呼ぶ。

## 2.2. Microsoft Visual Studio

### 1) インストール

Popolo はライブラリであるから、これを使うために最低限の補足的なプログラムは自分で作成しなければならない。その方法はいくつもあるが **Visual Studio** を使って呼び出す方法が最も簡単だろう。**Visual Studio** は Microsoft 社によって開発されている統合開発環境で、C#以外にもいくつかの言語でプログラムを開発できる。以下の URL からダウンロードしてインストールできるが、詳細に関しては **Web** の情報を参照されたい。

→ <https://visualstudio.microsoft.com/ja/downloads>

### 2) 最初のプログラム

Visual Studio を起動すると図 2.1 が表示されるため、「新しいプロジェクトの作成」をクリックする。

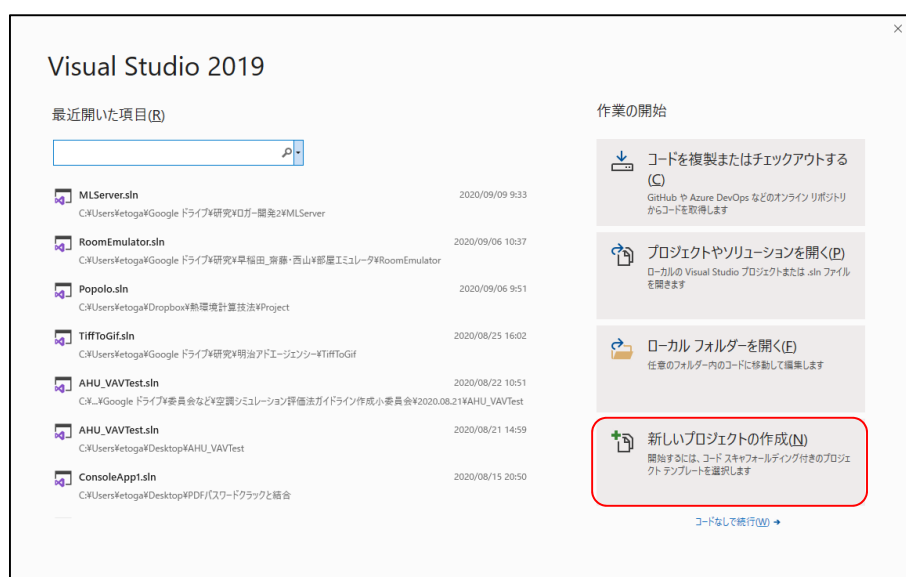


図 2.1 起動画面

Visual Studio では様々なプログラムが開発できるため、まずはどのような開発を行うのかを決定する必要がある。プログラム言語として「C#」、動作するプラットフォームとして「Windows」、プロジェクトの種類として「コンソール」を選択すると、この条件に合うプロジェクトが抽出される。表示された一覧の中から「コンソールアプリ (.NET core)」を選択する (図 2.2)。

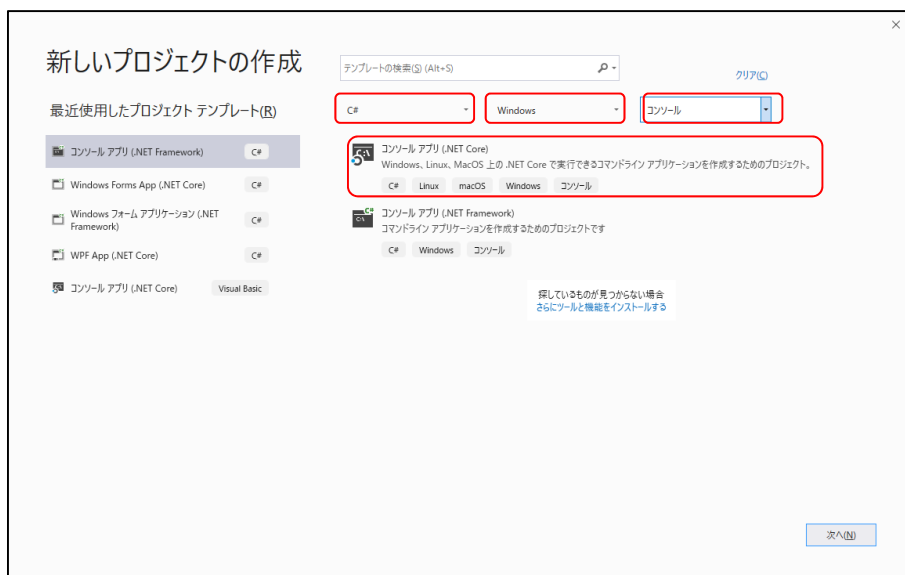


図 2.2 プロジェクトの種類の選択

続けてプロジェクトの名称と作業場所を指定する (図 2.3)。ここではプロジェクト名を「TestProject」、作業場所をデスクトップにする。



図 2.3 プロジェクトの保存

図 2.4 はコンソールプロジェクトの初期状態である。中央にはプログラムのソースコードが書かれており、この内容を修正することで計算処理の内容を変える。

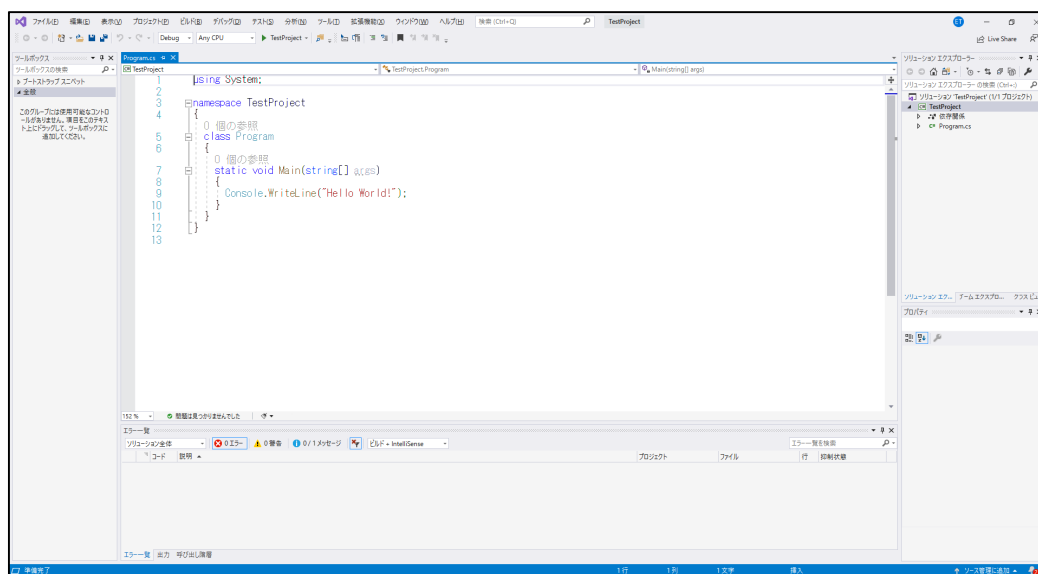


図 2.4 プロジェクトの初期状態

ごく単純な修正を加えて、プログラムを動かしてみる。プログラム 2.1 に修正したプログラムを示す。C#言語では、7~12 行に記載されるように **Main** に続く括弧「{ ... }」で挟まれた範囲に記載されたプログラムが、上から順に実行されていく。この例ではまず 9 行の「**Console.WriteLine**(“Hello World!”);」が実行される。この **Console.WriteLine** は画面にテキストを出力するための命令であり、このような命令のことを C#では「メソッド」と呼ぶ。このメソッドを実行するためには出力すべきテキストをメソッドに与えなければならない。この情報はメソッド名に続く括弧「( ... )」の中に記述する。ここでは“Hello World!”という文字列が与えられている。このように、メソッドが計算を実行するために必要とする情報を「引数」と呼ぶ。つまり 9 行は“Hello World!”という文字列を引数として受け取り、これを画面に出力するメソッドを呼び出しているということになる。

ここで 11 行目に「**Console.Read**();」を追加する。これはユーザーがキーボードで入力した内容を読み込むというメソッドであり、引数は持たない。

### プログラム 2.1 最初のプログラム

```
1 using System;
2
3 namespace TestProject
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10
11             Console.Read();
12         }
13     }
14 }
```

修正が終わったら、ツールバーにある「TestProject」をクリックすることでプログラムを実行させる（図 2.5）。実行結果は図 2.6 のようになる。Hello World! と出力されていることがわかる。現在、プログラムは 11 行まで進んでおり、ユーザーからの入力を待っている状態なので、適当な文字列を入力して Enter を押せば終了する。

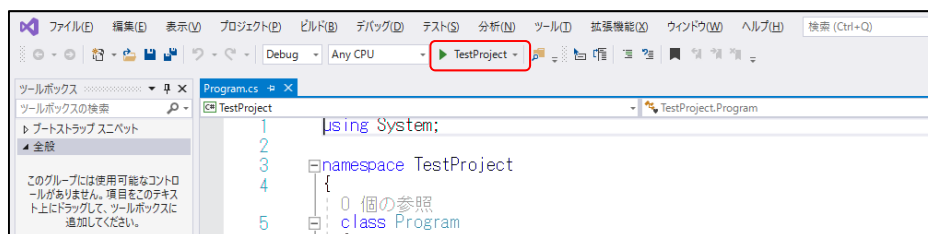


図 2.5 プロジェクトの実行

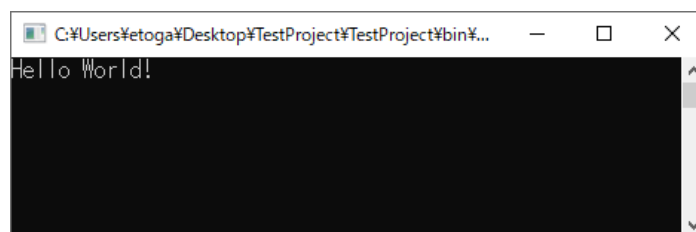


図 2.6 実行結果

### 2.3. Popolo の導入

Visual Studio から Popolo を使えるようにする。ソリューションエクスプローラーの依存関係を右クリックし、NuGet パッケージの管理を選択すると（図 2.7）、パッケージの検索画面が開く。参照タブを選択して Popolo で検索し、表示された Popolo を選択した後に、インストールボタンを押すとライブラリがダウンロードされてプロジェクトに追加される（図 2.8）。ソリューションエクスプローラーの依存関係に Popolo が追加されていることを確認する（図 2.9）。

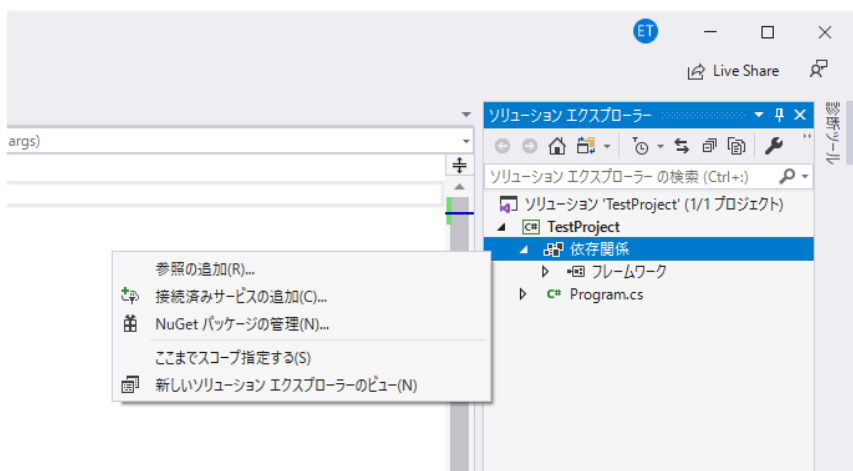


図 2.7 NuGet による Popolo の導入 1

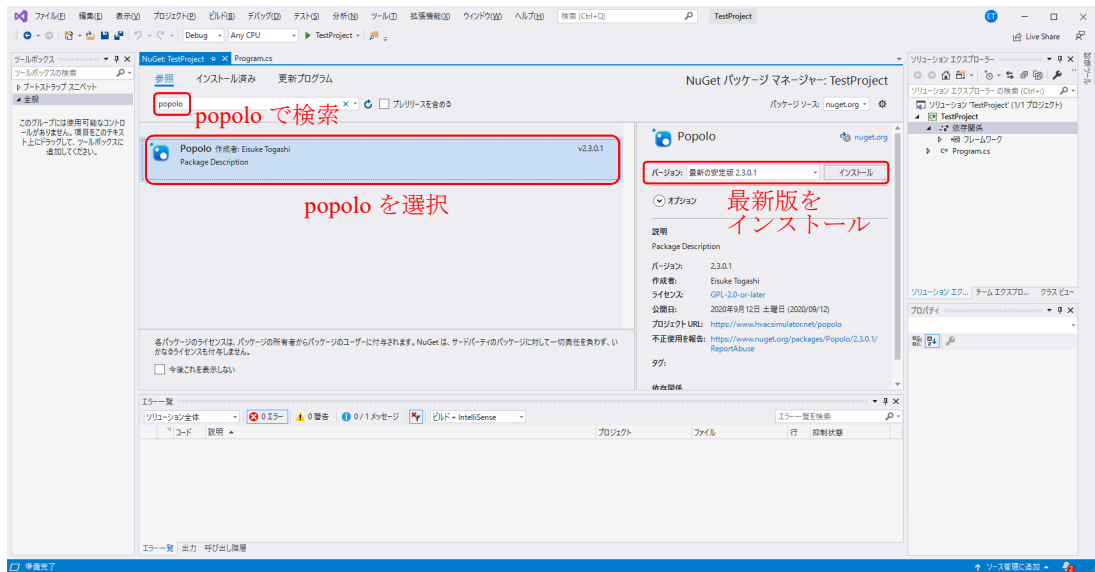


図 2.8 NuGet による Popolo の導入 2

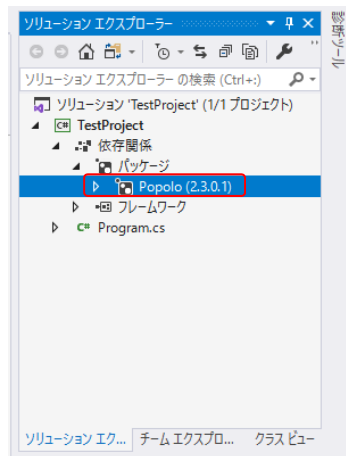


図 2.9 NuGet による Popolo の導入 3

## 2.4. 湿り空気の飽和状態の計算例

Popolo を使って湿り空気の飽和状態を計算するプログラムの例をプログラム 2.2 に示す。

ライブラリには様々なプログラムが登録されているが、通常は管理をしやすいようにいくつかのプログラムごとにグルーピングが行われている。このグループの名前を「名前空間」と言う。3 行は「Popolo.ThemophysicalProperty」という名前空間を用いることを宣言している。この名前空間は湿り空気や冷媒などの熱物性を計算するプログラム群が格納されている。この宣言をすると、以下のプログラムでは Popolo.ThemophysicalProperty に格納されたすべてのプログラムを使うことができるようになる。

このプログラムはユーザーから入力された乾球温度を手がかりに、飽和温度における絶対湿度と比エンタルピーを計算するプログラムである。11~23 行は繰り返し文で、`while (true){ ... }` で囲まれた内容が繰り返し実行される。ユーザーから乾球温度が入力されるたびに上記の計算処理を行う。

14 行で `ReadLine` メソッドを使ってユーザーの入力受けとり、これを `tmpS` に代入する。ただし、ユーザーから受け取った情報は文字列であるため、これを数値として扱うことをプログラムに伝えなければ

ならない。この処理は 15 行であり、変換した値は `tmp` に代入される。

17 行と 18 行が `Popolo` で用意されたメソッドで、17 行は飽和絶対湿度を計算するメソッド、18 行は飽和比エンタルピーを計算するメソッドである。いずれも計算のために乾球温度と大気圧の情報が必要であるため、引数としてこの値を渡す。乾球温度はユーザーから入力された `tmp` であり、大気圧は 101.325 kPa に固定している。

計算結果は 20 行と 21 行で出力する。`ToString()` は数値を文字列に変換するためのメソッドであり、引数の `"F $x$ "` は小数点以下  $x$  桁で表示するということを意味している。C# では文字列は「+」記号で連結する。

計算結果を図 2.10 に示す。図 2.10 は 20 を入力して Enter キーを押した場合の結果で、飽和絶対湿度は 14.7 g/kg、飽和比エンタルピーは 57.4 kJ/kg であることがわかる。

## プログラム 2.2 飽和状態の計算プログラム

```
1 using System;
2
3 using Popolo.ThermophysicalProperty;
4
5 namespace TestProject
6 {
7     class Program
8     {
9         static void Main(string[] args)
10         {
11             while (true)
12             {
13                 Console.WriteLine("Enter the dry-bulb temperature.");
14                 string tmpS = Console.ReadLine();
15                 double tmp = double.Parse(tmpS);
16
17                 double hrt = 1000 * MoistAir.GetSaturationHumidityRatioFromDryBulbTemperature(tmp, 101.325);
18                 double ent = MoistAir.GetSaturationEnthalpyFromDrybulbTemperature(tmp, 101.325);
19
20                 Console.WriteLine("Saturation humidity ratio = " + hrt.ToString("F1") + " g/kg");
21                 Console.WriteLine("Saturation enthalpy = " + ent.ToString("F1") + " kJ/kg");
22                 Console.WriteLine();
23             }
24         }
25     }
26 }
```

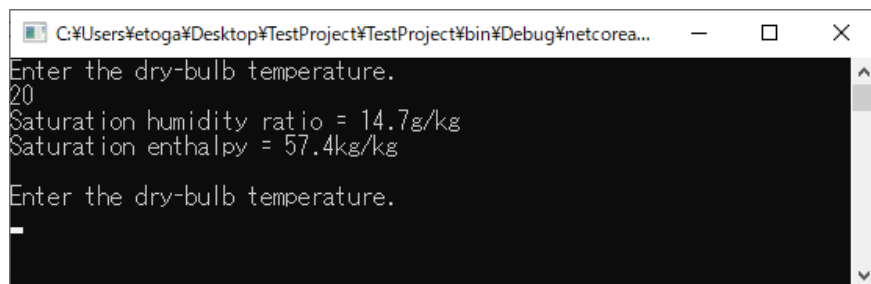


図 2.10 計算結果



### 3. 熱負荷計算

本章では、単純な形状の事務室を例にとって熱負荷計算を行う。ややプログラムは複雑になるため、一般的な事項に関しては適宜、C#自体の参考書を参照されたい。

#### 3.1. 計算対象の整理

計算対象の事務室を図 3.1 に示す。平面形状は 3m×7m の単純な長方形で、階高と天井高はそれぞれ 4m と 2.7m である。南側には 2.8m×1.8m の窓が設けられている。

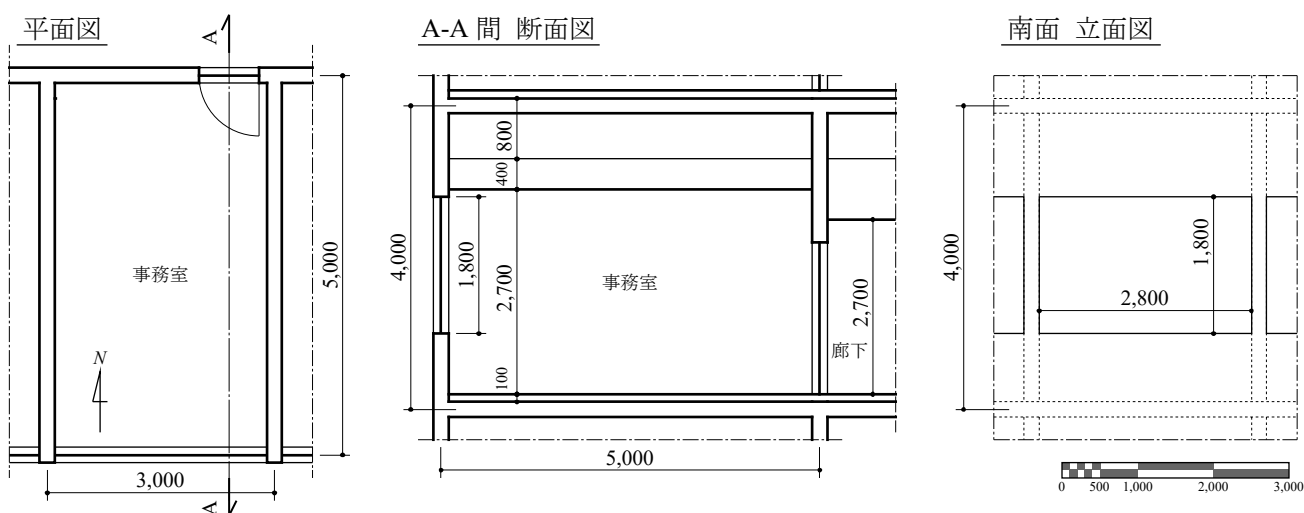


図 3.1 計算対象の事務室

熱負荷計算をするためには、図 3.1 のような図面をもとに、計算のためのモデルの情報を読み取るという作業が必要となる。いたずらに細かく捉えすぎではモデルを作成することが大変になるが、一方で粗すぎでは精度が保たれない。従って、現実の建物を適度に抽象化して捉える必要があり、この能力は計算の場数を踏むことによって鍛えられる。

事務室内の空気は、これを取り囲む上下東西南北の壁や窓を介して外部と熱交換をする。このため、これらの壁や窓の熱的な情報を整理する必要がある。表 3.1 に壁の構成と厚みを示す。窓は単層の透明フロートガラス（日射透過率 70%、日射反射率 10%）とする。

表 3.1 壁の構成と厚み

ID	部位	材料	熱伝導率 [W/(m・K)]	容積比熱 [kJ/(m <sup>3</sup> ・K)]	厚み [mm]
EW	南側外壁	石膏ボード	0.220	830	8
		コンクリート	1.600	2000	150
IW	内壁（東北西）	石膏ボード	0.220	830	8
		非密閉中空層	-	-	-
		石膏ボード	0.220	830	8
FL	床・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	8
		非密閉中空層	-	-	-
		コンクリート	1.600	2000	150
		非密閉中空層	-	-	-
		ビニル系床材	0.190	2000	3

モデル化にあたって迷いが生じる可能性がある点を以下に整理しておく。

よほど壁厚が大きくない限りは、通り芯で長さを取ろうが、内法で取ろうが、計算結果にほとんど影響はない。従って通常、面積は図面から読み取りやすい通り芯間の長さで計算することが多い。つまり、図 3.1 の床面積は  $3 \times 5 = 15 \text{ m}^2$ 、東西内壁の面積は  $5 \times 4 = 20 \text{ m}^2$  として扱う。

図 3.1 では北側に扉がついている。厳密に計算するのであれば、この部分だけ金属製の中空層の壁が存在するとみなしてもよいが、この違いもほとんど計算結果に影響が無い。そもそも現実には扉は開放されている可能性もあるし、その場合には空気の流入出によって、より大きな誤差が発生するのであるから、この扉を厳密にモデル化することに価値があるとは思えない。

天井裏を部屋とは別の空間として質点を設けて計算するか否かはやや検討が必要である。表 3.1 では、別の質点を設けず、床と天井を構成する空気層として取り扱っている。この設定が適切か否かは空調の方式にもよるだろう。

現在、かなり多くのオフィスビルでは吹出口を天井面に設ける一方で、吸い込みは照明器具に設けた空隙を利用する「天井リターンチャンバ」方式を採用している。この目的は主に還気ダクトを張り巡らすためのスペースやコストを節約するためである。この方式の場合には執務室の空気が天井裏を介して空調機に戻るため、執務室と天井裏とで空気状態に大きな違いは無い。また、照明器具から天井裏に放熱が合ったとしても執務室の空気と十分に混合される。従って、このような方式ではわざわざ天井裏に質点を設けて計算を複雑化させる利点は小さい。

一方で、天井面に吸込口を設けて天井裏空気と執務室の空気が混ざらない計画の場合には質点を分ける価値がある。照明器具から天井裏への放熱はそのまま天井裏に滞留するため、執務室と天井裏とでは空気状態にかなりの差が発生しうするためである。

本モデルでは天井裏は独立した質点を持たない方針とした。ただし外壁の面積を計算する際には天井裏部分も含めた面積にすることに注意する。つまり本例では  $3 \times 4 \text{ m} = 12 \text{ m}^2$  となる。

現実のスラブには構造上、大梁小梁が取り付く。これらは熱容量も大きく、計算結果への影響は小さくないが、これを反映するためには構造図面を入手する必要があり、作業もやや煩雑なため、無視することも多い。

## 3.2. 建物モデルの作成

プログラム 3.1 に建物モデルの作成処理を示す。

プログラム 3.1 建物モデルの作成

```
1 using Popolo.ThermalLoad;
2 using Popolo.Weather;
3
4 static BuildingThermalModel makeBuildingModel()
5 {
6     //傾斜面を作成
7     Incline incS = new Incline(Incline.Orientation.S, 0.5 * Math.PI);
8
9     //壁層を作成
10    WallLayer[] wIEW = new WallLayer[]
11    {
12        new WallLayer("石膏ボード", 0.220, 830, 0.008),
13        new WallLayer("コンクリート", 1.600, 2000, 0.150)
14    };
15    WallLayer[] wIIW = new WallLayer[]
```

```

16 {
17     new WallLayer("石膏ボード", 0.220, 830, 0.008),
18     new AirGapLayer("非密閉中空層", false, 0.050),
19     new WallLayer("石膏ボード", 0.220, 830, 0.008)
20 };
21 WallLayer[] wIFL = new WallLayer[]
22 {
23     new WallLayer("ロックウール化粧吸音板", 0.064, 290, 0.012),
24     new WallLayer("石膏ボード", 0.220, 830, 0.008),
25     new AirGapLayer("非密閉中空層", false, 0.050),
26     new WallLayer("コンクリート", 1.600, 2000, 0.150),
27     new AirGapLayer("非密閉中空層", false, 0.050),
28     new WallLayer("ビニル系床材", 0.190, 2000, 0.003)
29 };
30
31 //壁を作成
32 Wall[] walls = new Wall[4];
33 walls[0] = new Wall(3.0 * 5.0, wIFL); //床天井
34 walls[1] = new Wall(5.0 * 4.0, wIW); //東西内壁
35 walls[2] = new Wall(3.0 * 4.0, wIW); //北内壁
36 walls[3] = new Wall(3.0 * 4.0 - 2.8 * 1.8, wIEW); //南外壁
37
38 //窓を作成
39 Window[] windows = new Window[1];
40 windows[0] = new Window(2.8 * 1.8,
41     new double[] { 0.7 }, //透過率70%
42     new double[] { 0.1 }, //反射率10%
43     incS);
44
45 //ゾーンを作成
46 Zone[] zones = new Zone[1];
47 zones[0] = new Zone("計算対象室", 3.0 * 5.0 * 4.0 * 1.2, 3.0 * 5.0);
48 zones[0].HeatCapacity = zones[0].AirMass * 10;
49 zones[0].VentilationRate = zones[0].AirMass * 0.3 / 3600d;
50
51 //多数室を作成
52 MultiRooms mRoom = new MultiRooms(1, zones, walls, windows);
53 mRoom.AddZone(0, 0); //0番目のRoomに0番目のZoneを追加
54 mRoom.AddWindow(0, 0); //0番目のZoneに0番目のWindowを追加
55 mRoom.AddWall(0, 0, true); mRoom.AddWall(0, 0, false); //床天井・無限ループ
56 mRoom.AddWall(0, 1, true); mRoom.AddWall(0, 1, false); //東西内壁・無限ループ
57 mRoom.AddWall(0, 2, true); mRoom.UseAdjacentSpaceFactor(2, false, 0.4); //北内壁
58 mRoom.AddWall(0, 3, true); mRoom.SetOutsideWall(3, false, incS); //南外壁
59
60 BuildingThermalModel bModel = new BuildingThermalModel(new MultiRooms[] { mRoom });
61 bModel.InitializeAirState(24, 0.028);
62 bModel.TimeStep = 3600;
63
64 return bModel;
65 }

```

1~2 行は名前空間の導入である。熱負荷計算に関連するクラスは「Popolo.ThermalLoad」名前空間に属する。その他、気象関連のクラスを用いるために「Popolo.Weather」を導入する。

Popolo では BuildingThermalModel というクラスで建物モデルを管理する。4~65 行は BuildingThermalModel のインスタンスを出力する関数である。

太陽からの日射熱取得を計算する場合などには、受照面がどのように傾いているのかを指定する必要がある。今回のモデルでは、南側外壁や窓が日射熱を受けるため、その南側鉛直面を 7 行で定義している。Incline は傾斜面を表すクラスで、第 1 引数で方位、第 2 引数で垂直方向の傾きを設定する。

表 3.1 に示した壁のそれぞれの層の物性を定義する必要があり、これは壁層を表す WallLayer クラスの配列で表現する。10~14 行、15~20 行、21~29 行はそれぞれ外壁、内壁、床スラブの構成を表している。日本語の上では「床」は「壁」とは異なるが、水平か垂直かが異なるだけなので、熱流を計算の上では

同じだという点に注意する。WallLayer の第 1 引数は層の名前、第 2 引数は熱伝導率、第 3 引数は容積比熱、第 4 引数は厚み、である。特殊な壁層としては空気層がある。これを設定する場合には 18 行に示すように AirGapLayer を使う。第 1 引数は名前、第 2 引数は密閉されているか否か、第 3 引数は厚み、である。

31~36 行では、壁層配列（壁構成）と面積を組み合わせることで具体的な壁を作成する。このようにわざわざ壁構成を作ってから壁を作るような迂遠な処理を行う理由は、一般の建物では壁構成は同じだが面積が異なる壁が多数存在するためである。今回のモデルでは、南側外壁、東西内壁、北側内壁、床スラブ、の 4 枚の壁があるため、4 つの要素を持つ Wall 配列を作成する。Wall の第 1 引数は壁の面積、第 2 引数は壁層配列である。

窓は 1 枚しか無いため、窓を表す Window 配列は要素が 1 つのみである。Window の第 1 引数は面積、第 2 引数はガラスの日射透過率配列、第 3 引数は日射反射率配列、第 4 引数は窓の方位である。透過率と反射率を配列で設定する理由は、複層ガラスへ対応しているためである。今回は単層のガラスであるため、要素が 1 の配列を設定する。

45~49 行で、温湿度を計算するゾーンを作成する。ペリメータとインテリアが分かれたり、多数の小部屋に分かれている場合には複数のゾーンを作成するが、今回は単室のため Zone 配列の要素数は 1 である。Zone の第 1 引数は名称、第 2 引数は空気の重量[kg]、第 3 引数は床面積、である。48 行はゾーンにある家具などの什器の熱容量で、一般に空気の 10 倍程度を設定する。49 行は換気量 [kg/s]で、このモデルでは 0.3 回/h の隙間風が発生するとしている。

以上で用意した Wall 配列、Window 配列、Zone 配列を組み合わせることでモデルを構築するのだが、これは MultiRooms クラスを用いて行う。52 行が MultiRooms クラスの作成処理で、第 1 引数は部屋 (Room) の数、第 2 引数は Zone 配列、第 3 引数は Wall 配列、第 4 引数は Window 配列である。

MultiRooms は文字通り、複数の Room から構成される多数室を表す。ただし、今回は 1 つの Zone しか無いため、Room の数も 1 つである。53 行で 0 番目の Room に 0 番目の Zone を追加する。54 行は窓の設定処理で、0 番目の Zone に 0 番目の窓を追加する。

55~58 行は壁の設定処理である。壁は表面と裏面の 2 面があるため、いずれの面がどこに向いているのかを設定する必要がある。

AddWall メソッドの第 1 引数は壁を追加する Zone の番号、第 2 引数は壁の番号、第 3 引数は表側か否か、を表している。55 行では 0 番目の Wall の両方の面を 0 番目の Zone に向けて設定している。0 番目の Wall は床スラブで、このように両面を同じ Zone に向けるということは Zone と Wall が無限に繰り返されるということを意味する。同様に 56 行では東西面内壁を表す 1 番目の Wall の両面を 0 番目の Zone に設定しているから、東西に無限にモデルが繰り返されるということを意味する。

57 行は北側内壁の設定である。UseAdjacentSpaceFactor メソッドは隣室温度差係数を使う方法で、第 1 引数は Wall の番号、第 2 引数は表側か否か、第 3 引数は隣室温度差係数、である。

58 行は南側外壁の設定である。SetOutsideWall メソッドは屋外に面する壁の設定で、第 1 引数は Wall の番号、第 2 引数は表側か否か、第 3 引数は外壁の方位、である。

以上が MultiRooms (多数室) の作成処理であるが、現実の建物にはこのような多数室が複数、含まれる。そこで、建物全体を表す BuildingThermalModel は、60 行に示すように MultiRooms の配列を与えて初期化する。61 行は温湿度の初期化処理、62 行は計算のタイムステップの設定である。

### 3.3. 年間自然室温の計算

3.2 節で用意した建物モデルを使って年間自然室温を計算する処理をプログラム 3.2 に示す。

プログラム 3.2 年間自然室温の計算

```
1 using System.IO;
2 using Popolo.ThermalLoad;
3 using Popolo.Weather;
4
5 static void Main(string[] args)
6 {
7     simulateFreeFloat();
8 }
9
10 static void simulateFreeFloat()
11 {
12     //建物モデルを作成
13     BuildingThermalModel bModel = makeBuildingModel();
14
15     //1年間の気象データを作成
16     RandomWeather weather = new RandomWeather(0, RandomWeather.Location.Tokyo);
17     weather.MakeWeather(1, out double[] dbt, out double[] hmd, out double[] rad, out bool[] isFair);
18
19     //日時と太陽を作成
20     DateTime dTime = new DateTime(2019, 1, 1, 0, 0, 0);
21     Sun sun = new Sun(Sun.City.Tokyo);
22     ImmutableZone zone = bModel.MultiRoom[0].Zones[0]; //ゾーンを取得
23
24     //計算実行
25     using (StreamWriter sWriter = new StreamWriter("out.csv"))
26     {
27         //タイトル行
28         sWriter.WriteLine("Date time, Drybulb temperature [C], Humidity ratio [g/kg]");
29
30         //年間計算実行
31         for (int hour = 0; hour < dbt.Length; hour++)
32         {
33             //太陽位置を更新して直散分離する
34             sun.Update(dTime);
35             sun.SeparateGlobalHorizontalRadiation(rad[hour], Sun.SeparationMethod.Erbs);
36
37             //夜間放射[W/m2]を計算する
38             double vp = MoistAir.GetWaterVaporPartialPressureFromHumidityRatio(0.001 * hmd[hour], 101.325);
39             double noc = Sky.GetNocturnalRadiation(dbt[hour], isFair[hour] ? 0 : 10, vp);
40
41             //外気条件を設定する
42             bModel.UpdateOutdoorCondition(dTime, sun, dbt[hour], 0.001 * hmd[hour], noc);
43
44             //モデルを更新する
45             bModel.ForecastHeatTransfer();
46             bModel.ForecastWaterTransfer();
47             bModel.FixState();
48
49             //ゾーンの温湿度を書き出す
50             sWriter.WriteLine(
51                 dTime.ToString("MM/dd HH:mm") + "," +
52                 zone.Temperature.ToString("F2") + "," +
53                 (1000 * zone.HumidityRatio).ToString("F2"));
54
55             //時刻を1時間進める
56             dTime = dTime.AddHours(1);
57         }
58     }
59 }
```

10~59 行が具体的なプログラムで、7 行でこのプログラムを呼び出す。

13 行で 3.2 節のメソッドを呼び出して建物モデルを作成する。

年間の計算を実行するためには気象データが必要になり、一般的には標準年気象データを用いることが多い。ただし、この気象データは有償であるため、Popolo では典型的な気象データを自動で生成するクラスを用意している。16 行がこの処理を行う `RandomWeather` で、第 1 引数は乱数生成のためのシード値、第 2 引数は地点である。17 行で `MakeWeather` メソッドを呼び出しており、第 1 引数が生成する年数である。ここでは 1 年を指定しているため、合計で 8760 時間のデータが生成される。第 2、3、4、5 引数は出力で、それぞれ乾球温度、絶対湿度、水平面全天日射、晴れか曇りか、を表す配列である。

20 行の `DateTime` は日時を表すクラスで、年は何でも良いが、うるう年にしないように注意する。うるう年の時間数は 8760 時間ではなく 8784 時間のためである。

21 行は太陽位置を管理するクラスで、本モデルでは地点を東京としている。

22 行では `BuildingThermalModel` に含まれる `Zone` を取り出している。`ImmutableZone` の実体は `Zone` だが、書き込み処理を定義していない読み取り専用のクラスであるため、計算結果を外部から破壊する虞がない。

計算結果は CSV 形式で「out.csv」というファイルに書き出す。25 行で外部ファイルを開く。書き出し内容は日付、乾球温度、絶対湿度の 3 つで、28 行でタイトル行を書き出す。

31~57 行でループを回し、年間 8760 時間の計算を進める。

34 行で太陽に日時を与えて太陽位置を更新し、その太陽位置を前提に 35 行で直散分離を行う。`SeparateGlobalHorizontalRadiation` メソッドの第 1 引数は水平面全天日射、第 2 引数は直散分離の手法である。

38 と 39 行は夜間放射の推定である。38 行で絶対湿度と大気圧から水蒸気分圧を求め、39 行で水蒸気分圧と雲量から夜間放射を推定する。雲量は正確にわからないため、晴れの場合は 0、曇りの場合には 10 としている。

以上で求めた外気条件を 42 行で建物モデルに設定する。

建物モデルでは顕熱流と潜熱流を計算する。45 行と 46 行でそれぞれの 3600 秒後の予測を行う。細かな検討をする場合には、予測値を確認しながら計算条件を変えつつ予測を繰り返すこともあるが、今回は予測値をそのまま採用することとし、47 行で予測結果を確定させる。

49~53 行は計算結果の書き出し処理で、CSV 形式のため、コンマで区切って日時、乾球温度、絶対湿度を出力する。

最後に 56 行で時間を 1 時間進めてループの先頭に戻る。

計算結果をグラフにすると図 3.2 と図 3.3 が得られる。空調を行っていないため、外気の温湿度変動にほぼ連動した結果となる。

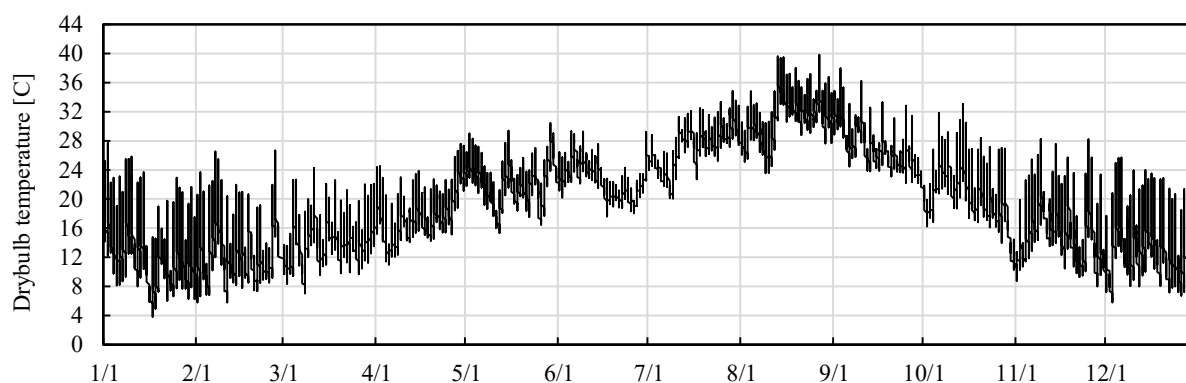


図 3.2 ゾーンの乾球温度の年間変動

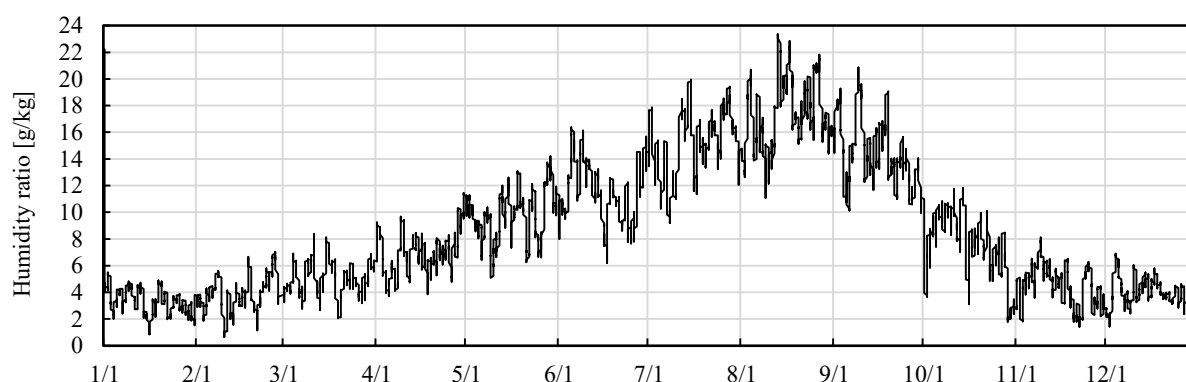


図 3.3 ゾーンの絶対湿度の年間変動

### 3.4. 年間熱負荷の計算

3.3 節の計算では空調を使わなかったため、温湿度は外気次第で変動した。現実の建物は冷暖房と除加湿を行い、エネルギーを消費する。このような空調に必要な熱負荷の計算をプログラム 3.3 に示す。

プログラム 3.3 年間熱負荷の計算

```

1 using System.IO;
2 using Popolo.ThermalLoad;
3 using Popolo.Weather;
4
5 static void Main(string[] args)
6 {
7     simulateHeatLoad();
8 }
9
10 static void simulateHeatLoad()
11 {
12     //建物モデルを作成
13     BuildingThermalModel bModel = makeBuildingModel();
14
15     //1年間の気象データを作成
16     RandomWeather weather = new RandomWeather(0, RandomWeather.Location.Tokyo);
17     weather.MakeWeather(1, out double[] dbt, out double[] hmd, out double[] rad, out bool[] isFair);
18
19     //日時と太陽を作成
20     DateTime dTime = new DateTime(2019, 1, 1, 0, 0, 0);
21     Sun sun = new Sun(Sun.City.Tokyo);
22     ImmutableZone zone = bModel.MultiRoom[0].Zones[0]; //ゾーンを取得

```

```

23
24 //計算実行
25 using (StreamWriter sWriter = new StreamWriter("out.csv"))
26 {
27     //タイトル行
28     sWriter.WriteLine(
29         "Date time, Drybulb temperature [C], Humidity ratio [g/kg], " +
30         "Sensible heat load [kW], Latent heat load [kW]");
31
32 //年間計算実行
33 for (int hour = 0; hour < dbt.Length; hour++)
34 {
35     //太陽位置を更新して直散分離する
36     sun.Update(dTime);
37     sun.SeparateGlobalHorizontalRadiation(rad[hour], Sun.SeparationMethod.Erbs);
38
39     //夜間放射[W/m2]を計算する
40     double vp = MoistAir.GetWaterVaporPartialPressureFromHumidityRatio(0.001 * hmd[hour], 101.325);
41     double noc = Sky.GetNocturnalRadiation(dbt[hour], isFair[hour] ? 0 : 10, vp);
42
43     //外気条件を設定する
44     bModel.UpdateOutdoorCondition(dTime, sun, dbt[hour], 0.001 * hmd[hour], noc);
45
46     //空調条件を設定する
47     bool weekDay = dTime.DayOfWeek != DayOfWeek.Saturday && dTime.DayOfWeek != DayOfWeek.Sunday;
48     if (weekDay && 8 <= dTime.Hour && dTime.Hour < 20)
49     {
50         //空調時間帯は25度、0.010kg/kgに制御
51         bModel.ControlDrybulbTemperature(0, 0, 25);
52         bModel.ControlHumidityRatio(0, 0, 0.010);
53     }
54     else
55     {
56         //非空調時間帯は熱供給は0
57         bModel.ControlHeatSupply(0, 0, 0);
58         bModel.ControlWaterSupply(0, 0, 0);
59     }
60
61     //モデルを更新する
62     bModel.ForecastHeatTransfer();
63     bModel.ForecastWaterTransfer();
64     bModel.FixState();
65
66     //ゾーンの温湿度を書き出す
67     sWriter.WriteLine(
68         dTime.ToString("MM/dd HH:mm") + "," +
69         zone.Temperature.ToString("F2") + "," +
70         (1000 * zone.HumidityRatio).ToString("F2") + "," +
71         (0.001 * zone.HeatSupply).ToString("F2") + "," +
72         (2500 * zone.WaterSupply).ToString("F2"));
73
74     //時刻を1時間進める
75     dTime = dTime.AddHours(1);
76 }
77 }
78 }

```

プログラム 3.2 と内容はほとんど変わらない。

大きな違いは、46~59 行で日時に応じた温湿度制御の処理を加えた点である。

まず 47 行で土日なのか平日なのかを判定する。平日かつ 8:00~20:00 の場合には空調を稼働させるために 49~53 行の処理を行う。逆に土日または夜間の場合には空調を停止させるために 54~59 行の処理を行う。

空調を稼働する場合には、設定温度および設定湿度が必要であり、51 行および 52 行のメソッドでこれ



を行う。第1引数は MultiRoom 番号、第2引数は Zone 番号で、第3引数に設定値を入力する。本例では 0 番目の MultiRoom に所属する 0 番目の Zone の設定温度を 25℃、設定湿度を 0.010 kg/kg としている。

空調を稼働させない場合には、顕熱と潜熱の供給量を直接に指定する。この場合には室温と湿度は成り行きとなる。57 行、58 行がこの処理で、第1引数は MultiRoom 番号、第2引数は Zone 番号で、第3引数に供給顕熱または潜熱を入力する。本例では熱供給を 0 としているため、結果的に自然室温が計算される。

本例では顕熱および潜熱負荷が計算されるため、71 行と 72 行で書き出し項目に熱負荷を追加する。潜熱は水分移動[kg/s]が算出されるため、水の蒸発潜熱 (2500 kJ/kg) を乗じることで単位を W に変換している。書き出し項目に合わせてヘッダ行も修正することを忘れないようにする (30 行)。

計算結果を図 3.4~図 3.7 に示す。

図 3.2 や図 3.3 に異なり、空調時間帯にはゾーンの乾球温度と絶対湿度が設定値である 25 °C、0.010 kg/kg に維持される。これに合わせて顕熱負荷と潜熱負荷が計上される。

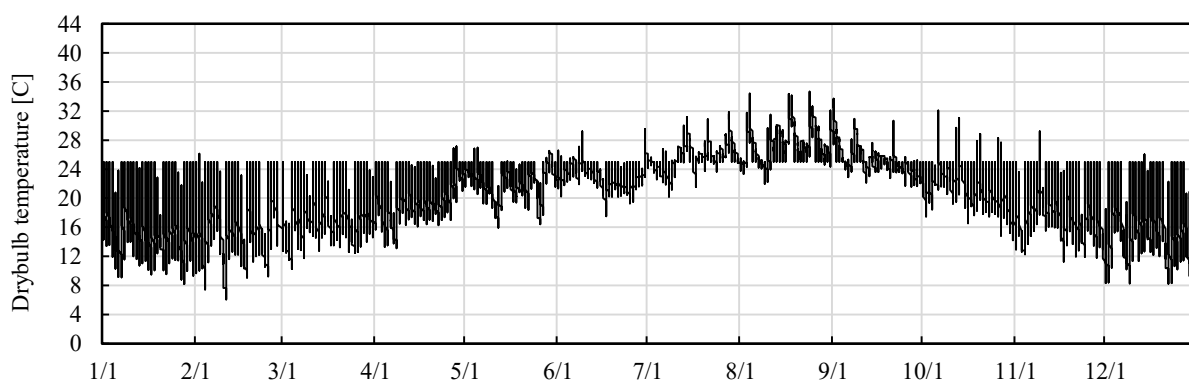


図 3.4 ゾーンの乾球温度の年間変動

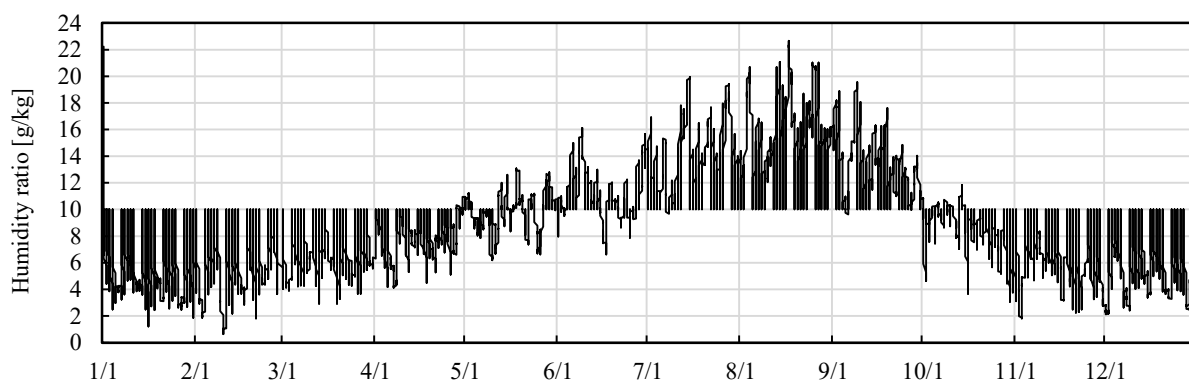


図 3.5 ゾーンの絶対湿度の年間変動

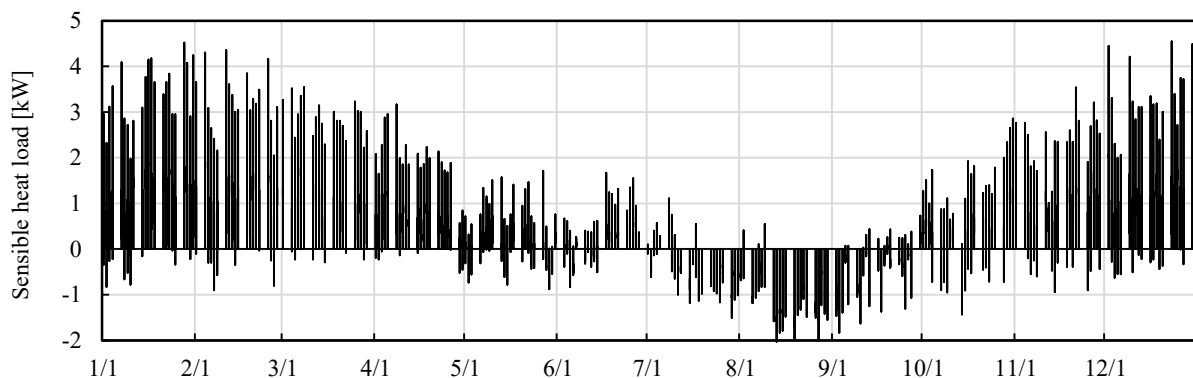


図 3.6 ゾーンの顕熱負荷の年間変動

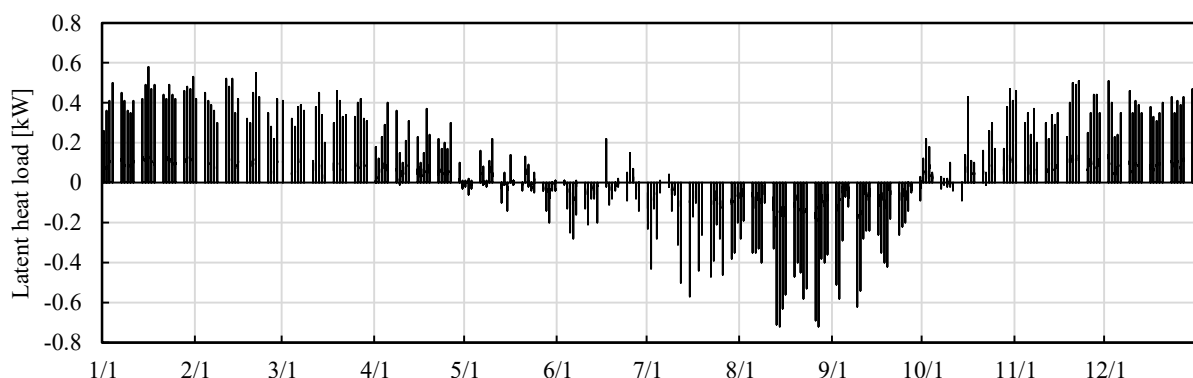


図 3.7 ゾーンの潜熱負荷の年間変動

### 3.5. 年間熱負荷の計算 2

3.4 節の計算結果にはやや不自然な点が 2 つある。

1 つ目は冬季に冷房負荷が発生したり、夏季に暖房負荷が発生したりしている点である。一般的に建物は、冬季には暖房加湿専用の運転となり、夏季には冷却除湿専用の運転となるため、どの季節でも自由に冷房も暖房できるというような運用をしない。

2 つ目は、特に冬季に非常に大きい負荷が発生している点である。最大で 5.0 kW 程度の暖房加湿負荷があるが、これは床面積あたりで 300 W/m<sup>2</sup> を超える。この負荷は夜間の蓄熱負荷の影響によって立ち上がり時に発生しているが、一般的にはこれほどの暖房能力で計画することは無い。このため、現実には即座に設定値まで室温を高めることができず、少し早めに運転を開始するという方法が取られる。

上記の問題が生じる理由は、標準のモデルでは無限大の熱供給能力をもった空調設備が仮定されているためである。このため、一年間のどの季節でもどのような大きさの冷房も暖房もできるという計算になってしまっている。そこでこの点を改良し、熱供給能力に制約をかけてみる。計算をプログラム 3.4 に示す。

#### プログラム 3.4 年間熱負荷の計算 2

```
1 using System.IO;
2 using Popolo.ThermalLoad;
3 using Popolo.Weather;
4
5 static void Main(string[] args)
6 {
```

```

7   simulateHeatLoad2();
8   }
9
10  static void simulateHeatLoad2()
11  {
12      //建物モデルを作成
13      BuildingThermalModel bModel = makeBuildingModel();
14
15      //1年間の気象データを作成
16      RandomWeather weather = new RandomWeather(0, RandomWeather.Location.Tokyo);
17      weather.MakeWeather(1, out double[] dbt, out double[] hmd, out double[] rad, out bool[] isFair);
18
19      //日時と太陽を作成
20      DateTime dTime = new DateTime(2019, 1, 1, 0, 0, 0);
21      Sun sun = new Sun(Sun.City.Tokyo);
22      ImmutableZone zone = bModel.MultiRoom[0].Zones[0]; //ゾーンを取得
23
24      //計算実行
25      using (StreamWriter sWriter = new StreamWriter("out.csv"))
26      {
27          //タイトル行
28          sWriter.WriteLine(
29              "Date time, Drybulb temperature [C], Humidity ratio [g/kg], " +
30              "Sensible heat load [kW], Latent heat load [kW]");
31
32          //年間計算実行
33          for (int hour = 0; hour < dbt.Length; hour++)
34          {
35              //太陽位置を更新して直散分離する
36              sun.Update(dTime);
37              sun.SeparateGlobalHorizontalRadiation(rad[hour], Sun.SeparationMethod.Erbs);
38
39              //夜間放射[W/m2]を計算する
40              double vp = MoistAir.GetWaterVaporPartialPressureFromHumidityRatio(0.001 * hmd[hour], 101.325);
41              double noc = Sky.GetNocturnalRadiation(dbt[hour], isFair[hour] ? 0 : 10, vp);
42
43              //外気条件を設定する
44              bModel.UpdateOutdoorCondition(dTime, sun, dbt[hour], 0.001 * hmd[hour], noc);
45
46              //冷暖房能力の制限を設定
47              if (5 <= dTime.Month && dTime.Month <= 10)
48              {
49                  bModel.SetCoolingCapacity(0, 0, 3000);
50                  bModel.SetDehumidifyingCapacity(0, 0, 1d / 2500d);
51                  bModel.SetHeatingCapacity(0, 0, 0);
52                  bModel.SetHumidifyingCapacity(0, 0, 0);
53              }
54              else
55              {
56                  bModel.SetCoolingCapacity(0, 0, 0);
57                  bModel.SetDehumidifyingCapacity(0, 0, 0);
58                  bModel.SetHeatingCapacity(0, 0, 3000);
59                  bModel.SetHumidifyingCapacity(0, 0, 1d / 2500d);
60              }
61
62              //空調条件を設定する
63              bool weekDay = dTime.DayOfWeek != DayOfWeek.Saturday && dTime.DayOfWeek != DayOfWeek.Sunday;
64              if (weekDay && 8 <= dTime.Hour && dTime.Hour < 20)
65              {
66                  //空調時間帯は25度、0.010kg/kgに制御
67                  bModel.ControlDrybulbTemperature(0, 0, 25);
68                  bModel.ControlHumidityRatio(0, 0, 0.010);
69              }
70              else
71              {
72                  //非空調時間帯は熱供給は0
73                  bModel.ControlHeatSupply(0, 0, 0);
74                  bModel.ControlWaterSupply(0, 0, 0);
75              }
76          }
77      }
78  }

```

```

76
77 //モデルを更新する
78 bModel.UpdateHeatTransferWithinCapacityLimit();
79
80 //ゾーンの温湿度を書き出す
81 sWriter.WriteLine(
82     dTime.ToString("MM/dd HH:mm") + "," +
83     zone.Temperature.ToString("F2") + "," +
84     (1000 * zone.HumidityRatio).ToString("F2") + "," +
85     (0.001 * zone.HeatSupply).ToString("F2") + "," +
86     (2500 * zone.WaterSupply).ToString("F2"));
87
88 //時刻を1時間進める
89 dTime = dTime.AddHours(1);
90 }
91 }
92 }

```

内容はプログラム 3.3 とほとんど変わらない。

追加・修正したプログラムは 46~60 行と 78 行である。1 年間で 5~10 月の冷房期間とその他の暖房期間に分け、それぞれの期間で暖房加湿および冷房除湿の最大能力を変えている。

48~53 行が冷房期間の能力設定処理である。SetCoolingCapacity は最大冷房能力を設定するメソッドで、同様に、SetDehumidifyingCapacity は最大除湿能力、SetHeatingCapacity は最大加熱能力、SetHumidifyingCapacity は最大加湿能力を設定する。第 1 引数は MultiRoom の番号、第 2 引数は Zone の番号、第 3 引数で能力を設定する。本例では、冷房期間は冷却能力を 3000 W、除湿能力を 1000 W、加熱加湿能力を 0W としている。ただし、加湿および除湿能力は kg/s で設定するため、水の蒸発潜熱である 2500 kJ/kg で除している。55~60 行は暖房期間の能力設定処理で内容は冷房期間と同様である。

78 行は、冷暖房能力の上限値を確認しながら、過負荷になった場合には上限値での運転に自動で変更するメソッドである。

計算結果を図 3.8~図 3.11 に示す。

図 3.4~図 3.7 と比較すると加熱加湿負荷が暖房期間にのみ、冷却除湿負荷が冷房期間のみにそれぞれ発生するように変化している。また冬季の立ち上げ時の大きな暖房負荷がなくなり、最大加熱能力である 3.0 kW で頭打ちになっている。このときの時刻の細かな計算結果を見ると、朝 8 時の瞬間には顕熱供給が 3.0 kW となり、その時間帯は設定室温の 24 °C が満たせていない。朝 9 時になるとようやく設定室温に到達する。

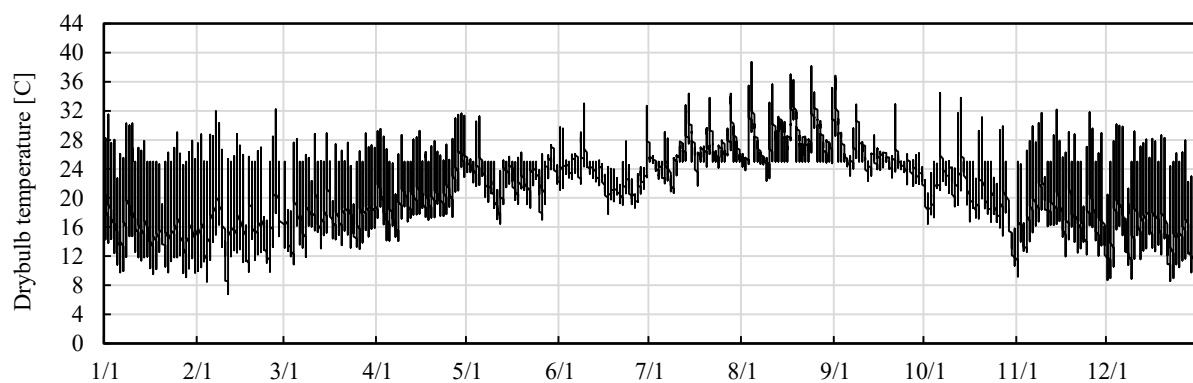


図 3.8 ゾーンの乾球温度の年間変動

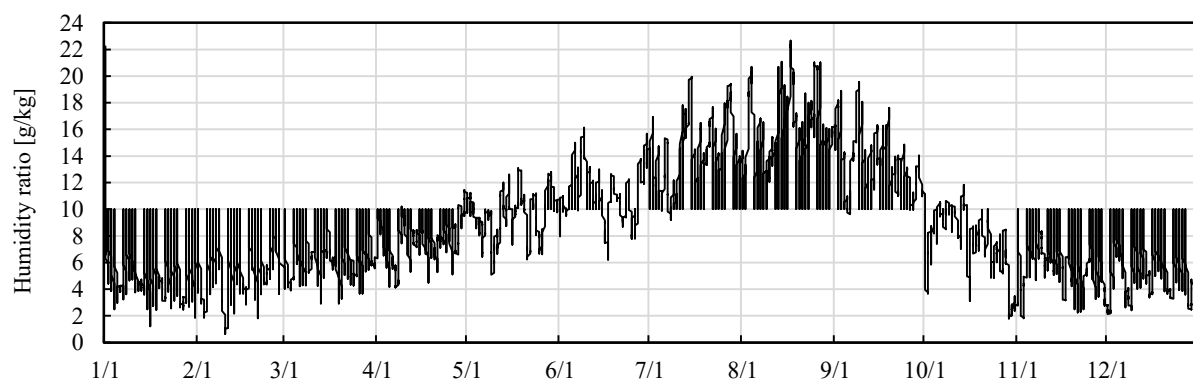


図 3.9 ゾーンの絶対湿度の年間変動

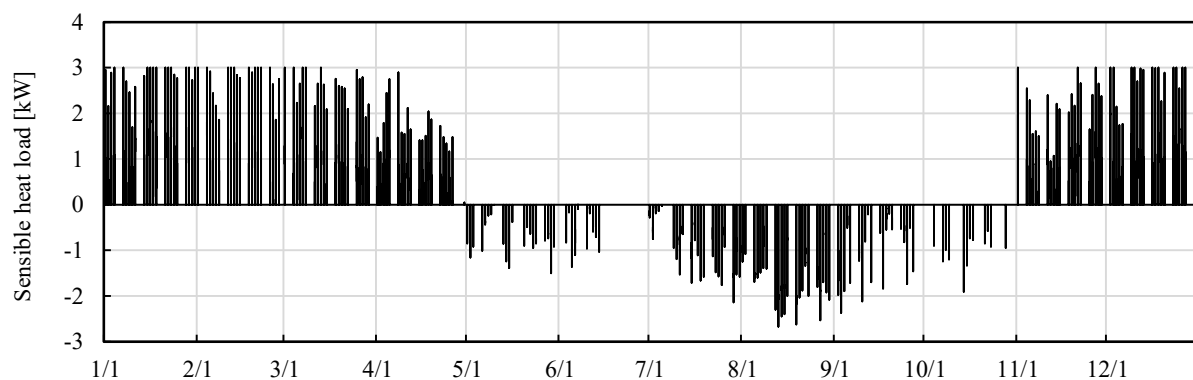


図 3.10 ゾーンの顕熱負荷の年間変動

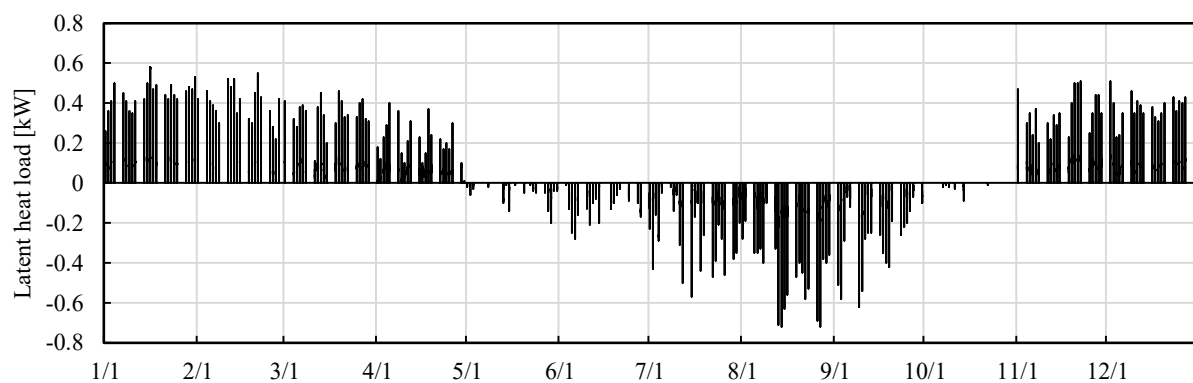


図 3.11 ゾーンの潜熱負荷の年間変動