熱伝導方程式の数値的解法

森田 太郎

目次

第 1 章

はじめに

熱伝導方程式に代表される偏微分方程式はコンピュータにおける数値計算によって計算されることが一般的である。熱伝導方程式を数値計算するためには熱伝導方程式の知識はもちろん、プログラミングについてもある程度の知識が必要である。熱伝導方程式をはじめとした偏微分方程式の数値的解法を扱った書籍は数多くあるが、いずれもある程度のプログラミング技能を前提として書かれているか、Fortran など現在ではメジャーではない言語で書かれていることがほとんどである。以上のような背景から本稿では比較的学習コストが低いと言われている Python を用いて熱伝導方程式を解くことを目的とする。また従来は Anaconda を使うのがベストプラクティスとされていたが、Python を web 開発などの他の業務で使う際はそうではない場合が多いため本稿では Anaconda を使わない[*1]。

[*1] 数値計算しかしない場合はベスト。Anaconda の numpy は IntelCPU に最適化されている。

第 2 章

開発環境の構築

2.1 Python のインストール

2.1.1 本体のインストール

プログラミング言語 Python のインストール方法を示す。既に Anaconda をインストールしている場合は次セクション「はじめのコード」まで呼び飛ばした方が良いかもしれない[*1]

1. `https://www.python.org/downloads/` にアクセスし、「Download」ボタンから Python インストーラーをダウンロードする。
2. ダウンロード後にインストーラーを起動し Python をインストール。
3. インストール時に「Add to PATH」にチェックを入れること（重要）

以上でプログラミング言語 Python のインストールは完了した。

2.1.2 パッケージのインストール

Python だけでもプログラムを書くことはできるが、Python の素晴らしさは先人が作ってきたプログラムをライブラリとして使用できる点である。Python は科学技術計算に強いと言われているが、それは Python には科学技術計算に必要な「Numpy、Scipy」といったライブラリやデータプロットの「Matplotlib」が使用できるからである。ここでは科学技術計算に必要なライブラリをインストールする。

1. windows のスタートから「Windows Powershell」を開く（検索ウィンドウから探すと早い)。
2. 「pip3 install matplotlib numpy scipy」と打ち込み Enter キーを押す。このとき Python インスートル時にパスを追加していないとエラーが出る。

2.2 Vusial studio code のインストール

2.2.1 本体のインストール

Python 自体のインストールはここまでで完了しているが、プログラムの内容（スクリプト）を編集するアプリケーションを用意するのが良い。数あるエディタから今回は比較的導入しやすい Visual Studio Code を紹介する。

1. `https://code.visualstudio.com/download` からインストーラーをダウンロードする。
2. インストールする。

以上で Visual Studio Code のインストールは完了した。

[*1] Anaconda と純 Python の共存はなにかと面倒が多い気がする（やったことないのでわからない）

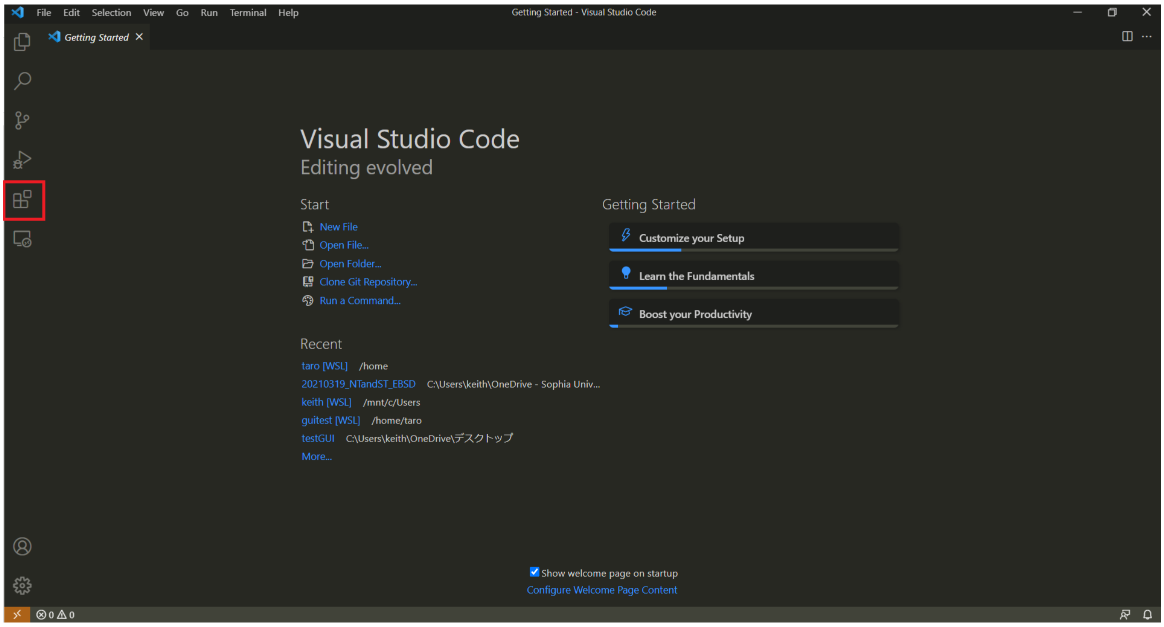

図 2.1　Visual studio code のメニュー画面

2.2.2　Visual studio code の設定

Visual Studio Code は標準アプリのメモ帳と比較して以下のようなメリットがある。

- コードがシンタックスハイライトされており、可読性が向上している
- 括弧やコードが自動で補完される（自動補完機能）
- Visual Studio Code 内でプログラムを実行できる
- デバック機能が付属している

また Visual Studio Code は Python に限らずほとんど全てのプログラミング言語を扱うことのできる汎用性に優れたアプリケーションである。

Visual studio code のインストールは以上で完了したが、Python を使うには若干の設定が必要である。

2.2.3　拡張機能のインストール

Visual studio code 上で Python を有効に使用するためには拡張機能をインストールする必要がある。

1. Visual studio code のメニュー画面左側の「Extensions」をクリックし、検索ウィンドウに「Python」と入力し Enter キーを押す。
2. 表一番上の拡張機能をインストールする（写真 2.2 参照）

これで Visual Studio Code 上で Python が使えるようになった。

2.3　はじめのコード

まず手始めに画面に「Hello world」と表示されるプログラムを書いてみる（プログラミング言語はまず Hello world から学習を始める）Visual Studio Code（以下 Vscode）の左上「File」ー「New File」からファイルを新規作成する。新しくタブが開かれるのですぐに「File」ー「Save」でファイル名をつけて保存する。このときファイル名を「hello.py」とすること。一般的にファイルの「.」以降を拡張子と呼び、取り扱うアプリケーションやプログラミング

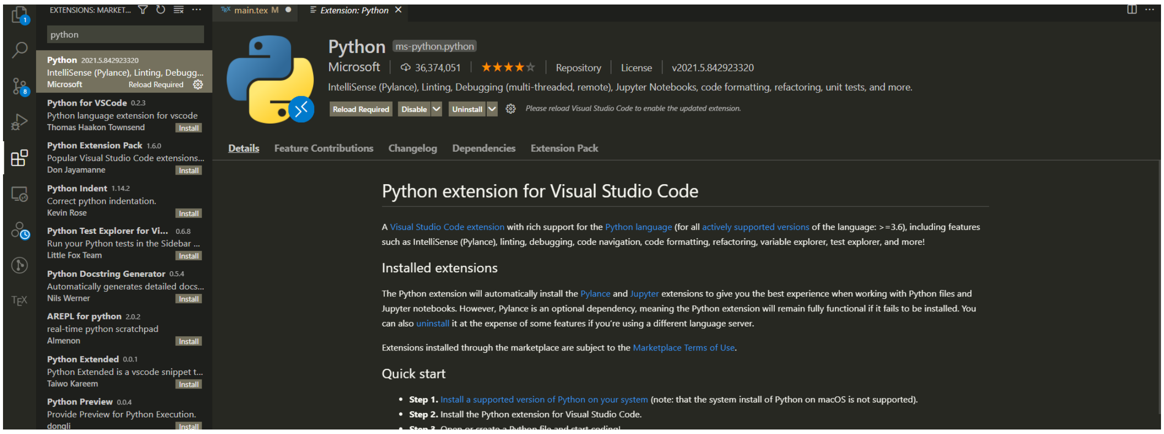

図 2.2　Vscode の拡張機能画面

言語によって異なる。Python ファイルの拡張子は「.py」である。ファイルの拡張子を「py」としないと Python 側でファイルを認識しないので注意する。ここではファイル名を「hello.py」として以下のコードを書いてみよう。

Listing 2.1　hello.py

```
print("Hello world")
```

コードを書いたらファイルを保存し、VScode 右上に表示されている緑色の矢印ボタン（再生ボタンみたいな）を押してコードを実行する。画面下に Powershell が開いてコードが実行されて、Hello world と表示されただろうか？もしエラーが表示されている場合はエラー内容を読んで、コードをもう一度見直してみよう。直したら上記の手順で再度実行しよう。おめでとう！　これで君は Python を使えるようになった！

2.4　次のステップへ

画面に Hello world を出力するだけでは退屈なので、もう一つステップアップしたプログラムを書こう。ここでは科学技術計算らしく、numpy と matplotlib を用いて Sin プロットを行うプログラムを書く。以下にコードを示す。

Listing 2.2　sin curve

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 x = np.linspace(0, 10, 1000)
5 y = np.sin(x)
6
7 plt.plot(x, y)
8 plt.show()
```

Sin カーブは出力されただろうか？ 次にコードについて説明する。1 行目、2 行目ではライブラリのインポートを行っている。ここでは数値計算ライブラリの Numpy とデータプロットライブラリの Matplotlib をインポートしている。4 行目では 0 から 10 まで等間隔に 1000 分割した配列（ndarray）を作成している。Numpy を使わずにこの機能を実装すると割と大変だが、Numpy によって 1 行のコードで実現できる。5 行目では作製した x を変数として Sin を適応した配列を y に代入している。7 行目では Maplotlib の plot 関数を用いて x vs y（Sin カーブ）を出力している。

このように数値計算ライブラリの Numpy を使うことで楽に Sin カーブをプロットすることができた。

第 3 章

Python で数値計算の基礎知識

3.1 変数

プログラミングでは「変数に値を代入する」という作業が多い。コードで書くと以下の通りである。

```
a = 1
```

これは「a という変数に 1 を代入する」を示す。我々は「a = 1」を目にすると「a は 1 に等しい！」と脊髄反射的に認識してしまうが、プログラミングの世界ではそうではなくあくまで「代入している」。言い換えれば「=」は「代入を示す演算子」である。

変数についてさらに詳しく記す。コンピュータ上にはメモリというデータを保管する装置が存在する。これは「横に連なるロッカー」のような構造をしており、ロッカーの一つ一つにデータを格納することができる。このロッカーのようなものをアドレスというが、変数とはこの「メモリ上のあるアドレス」についた名前を指す。つまり「変数 a に 1 を代入する」とは「メモリ上のあるアドレスに a という名前をつけてそこに 1 を格納する」ことである（図 3.1 参照）。

3.2 配列

どのプログラミング言語にも「配列」といったデータを一括に扱う方法が存在する。前述した変数では a = 1 のように 1 つの変数につき 1 つのデータしか格納することができない。この場合、例えば時刻のようなデータを格納する場合は不便であるが、配列を使うことによって多数のデータを 1 つの変数として使うことができる。以下に配列をつかったコードを示す。

Listing 3.1 array.py

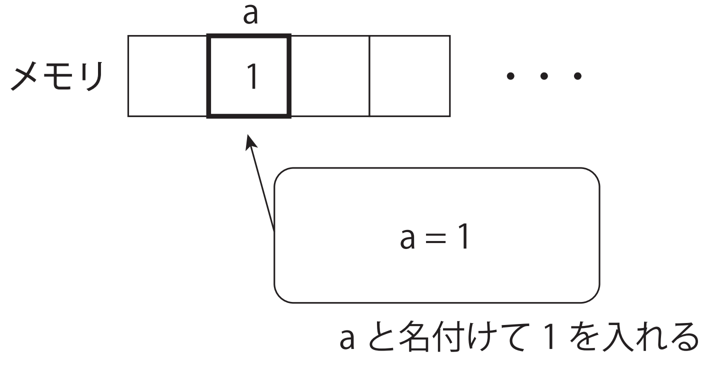

図 3.1 メモリと変数の関係の定性的な説明

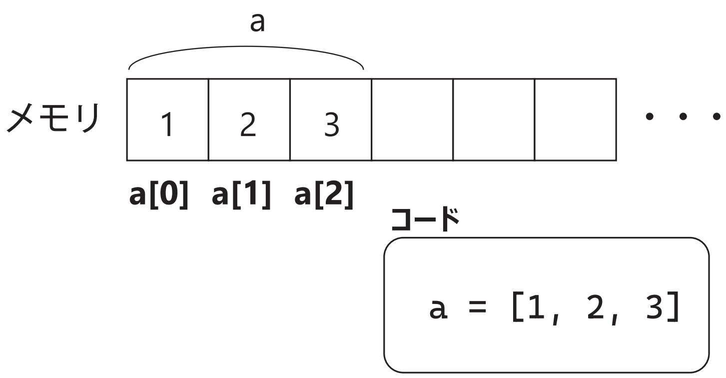

図 3.2　配列とメモリの関係の定性的な説明

```
x = [1, 2, 3]
print(x)
```

上記コードを実行すると「[1, 2, 3]」が出力される。このように配列を用いることで x に 1, 2, 3 の値を一括して格納することができた。

配列とメモリの関係を以下に示す。配列を作成するときはメモリ上で配列の大きさ分を確保し、その後に逐次 1, 2, 3 と代入する。つまり「a = [1, 2, 3]」ではまず配列の大きさである 3 マス分をメモリ上で確保しその領域を a と名付け、逐次 1，2、3 と代入する（図 3.2 参照）。

実際に配列を用いて計算するには配列から要素を取り出す必要がある。例えば「x から 1 をとりだして、それに 2 を加えたい」といった処理を行うためには x から 1 を取り出す必要がある。その処理を実現したコードを以下に示す。

Listing 3.2　array.py

```
x = [1, 2, 3]
ans = x[0] + 2
```

配列の要素を取り出すときは a[0] のようにして数字で要素を指定する。この数字をインデックスというが、配列の n 番目にある数値を取り出したい場合はインデックスとして $n-1$ を指定する。ここで注意すべきなのはインデックスの開始番号は 0 からであるということである[*1]。つまり 1 番目の要素を取り出したい場合はインデックスとして 0 を指定する。10 番目の要素を取り出したい場合はインデックスとして 9 を指定する。

また Python では配列に似た構造としてタプルやリストが存在するが、これらは混同しやすく混乱の元となるため本稿では numpy の ndarray を配列とする。numpy の配列 ndarray は以下のようにして定義することができる。

Listing 3.3　numpyarray.py

```
import numpy as np

x = np.array([1, 2, 3])
```

3.3　多次元配列

上述した配列はいわゆる 1 次元の配列である。特に数値計算では 2 次元以上の多次元配列を使う場合が多い。2 次元配列の定義の仕方を以下に示す。

[*1] 多くのプログラミング言語ではこの仕様。Matlab は異なる。

Listing 3.4 2darray.py

```
import numpy as np

x = np.array([[1, 2, 3], [4, 5, 6]])
print(x)

x1 = x[0, 0]
print(x1)
```

行列 $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ が出力された。2 次元配列の要素を取り出すには x[i, j] とインデックスを 2 つ指定する。i、j はそれぞれ行、列に対応する。

3.4 制御構文

プログラミングをするメリットはなんだろうか？ それは「退屈な繰り返し処理をパソコンに任せる」であると思う。ここではその繰り返し処理の基本について記す。

Hello と 10 回出力する処理の実装を考えてみよう。ここまでの知識では以下のようなコードが考えられよう。

Listing 3.5 hellloop.py

```
print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
```

非常に退屈なコードである。なぜならば、あなたは「print("hello")」を数え間違えないように細心の注意を払って 10 回書かなければならないからである。これでは処理をコンピュータに任せる意味がない。コンピュータにこのような「退屈な繰り返し処理」を任せる方法を記す。

3.4.1 while 構文

while は「～する限り」という意味がある。プログラミングでもその意味の通りの処理であり、「ある条件が成り立つ場合は処理を繰り返す」ことを示す。while 構文を使って hello を 10 回出力してみよう。

Listing 3.6 whileloop.py

```
count = 0

while count < 11:
    print("hello")
    count += 1
```

これこそプログラミングの醍醐味である。あなたは 10 も同じコードを書かずに済んだ！ while 構文について説明する。while のあとに count < 11 とあるが、これは「ループを繰り返す条件式」を示す。つまり count 変数が 11 未満である限り処理を繰り返すという意味である。その後のコロン（:）は区切り文字のようなものであり、お作法的に覚

えておいて問題ない。while で繰り返し処理する内容はインデント（コードがへこんでいるところ 4，5 行目）で表す。インデントを忘れると繰り返し処理されないので注意する。

繰り返し処理の内容について説明する。まず print("hello") を行う。その後、count += 1 を行い、count 変数に 1 を足す。そして 3 行目に戻り、count 変数の評価を行う。1 回ループした後では count は 1 であるため while ループの条件 count < 11 を満たしているため再びループ内容を処理これを繰り返して count が 11 になると while ループ条件を満たさなくなり、処理が終了する。

第 4 章

Numpy による科学技術計算の初歩

4.1　はじめに

熱伝導方程式の数値計算を始める前に、まず数値計算ライブラリの Numpy の基本的な使い方について記す。Numpy は Python 上で科学技術計算を容易に行うことを実現したライブラリである。行列計算や統計計算などが簡単なコマンドによって実行することができる。動作原理の知識がなくても扱うことができるが、ブラックボックス化されているためそれがかえってエラーやバグの元にもなる。最悪の場合は論文のデータねつ造につながる可能性がある。以下に配列の平均値を求めるコード比較を行う。n つの要素からなる $x_1, x_2, x_3 \cdots x_n$ の平均値 x_{ave} は

$$x_{\mathrm{ave}} = \frac{\sum_{k=1}^{n} x_k}{n} \tag{4.1}$$

コードは以下のようになる。

Listing 4.1　average.py

```
1 import numpy as np
2
3 x = np.array([1, 2, 3])
4 xave = (x[0] + x[1] + x[2]) / len(x)
```

4 行目の len() は配列の大きさを得るコマンドである。このコードでは Numpy は配列の定義のみに使用している。このコードでは x の要素数が 3 つであるため、それぞれを足し合わせることができているが要素数が 100 の場合はこのコードで平均値を求めることはできない。このように上記のコードは x の要素数が変わるだけで平均値を求めることができなくなり、普遍的なコードではない[*1]。

次に Numpy を活用した平均値算出のコードを示す。

Listing 4.2　averagenumpy.py

```
import numpy as np

x = np.array([1, 2, 3])
xave = np.average(x)
```

ぱっと見でわかりやすいコードになったと思う。さらにこのコードでは x の要素数が変化しても平均値を計算できるコードである。このように Numpy ライブラリを用いることで科学技術計算が楽になる。しかし、平均値の算出方法を知らないのに average() を使うのは良くないことがなんとなく想像できると思う。原理を理解した上で使うように心がけよう。

*1 このようなスパゲッティーコードは書くべきではない（ゴミのようなコードのことをスパゲッティーコードという）。

4.2　行列計算

第 5 章

熱伝導方程式の基礎

熱伝導方程式に代表される偏微分方程式の数値計算の解法は主に陽解法、陰解法、クランク・ニコルソン法の 3 つがある。計算の難易度は陽解法 < 陰解法 < クランク・ニコルソン法の順番で難しい。難易度順に説明していく。

5.1 陽解法

1 次元の熱伝導方程式は以下のように表される。

$$\frac{\partial\theta}{\partial t} = \kappa\frac{\partial^2\theta}{\partial x^2} \tag{5.1}$$

ここで、θ、t、x、κ はそれぞれ温度、時刻 (s)、距離 (mm)、熱拡散率（mm^2/s）を示す。微分の原理で書き下すと以下のようになる。

$$\lim_{\Delta t\to 0}\frac{\theta_{i,j+1}-\theta_{i,j}}{\Delta t} = \kappa\lim_{\Delta t\to 0}\frac{\theta_{i-1,j}-2\theta_{i,j}+\theta_{i+1,j}}{(\Delta x)^2} \tag{5.2}$$

ここで添え字の i、j はそれぞれ x 軸方向、時間方向を示している。式（5.2）は次のように近似できる。

$$\frac{\theta_{i,j+1}-\theta_{i,j}}{\Delta t} = \kappa\frac{\theta_{i-1,j}-2\theta_{i,j}+\theta_{i+1,j}}{(\Delta x)^2} \tag{5.3}$$

$$\theta_{i,j+1} = \left(\kappa\frac{\Delta t}{(\Delta x)^2}\right)(\theta_{i-1,j}-2\theta_{i,j}+\theta_{i+1,j})+\theta_{i,j} \tag{5.4}$$

ここで $\kappa\frac{\Delta t}{(\Delta x)^2} = r$ とすると、式（5.4）は

$$\theta_{i,j+1} = r\theta_{i-1,j}+(1-r)\theta_{i,j}+r\theta_{i+1,j} \tag{5.5}$$

と変形できる。この式の意味はなんだろうか。

図のように x 軸方向に n 個の点で $n-1$ 分割した 1 次元の棒を考えてみる。時刻 $t=k$ における x 軸上の格子点の温度は左から順番に

$$\theta_{0,k}, \theta_{1,k}, \theta_{2,k}\cdots\theta_{n-1,k} \tag{5.6}$$

となる。さらに時刻を $t=k+1$ とすると、棒状の格子点の温度は左から順に

$$\theta_{0,k+1}, \theta_{1,k+1}, \theta_{2,k+1}\cdots\theta_{n-1,k+1} \tag{5.7}$$

となる。例えば棒を 10 つの格子点で分割した場合を考える。$i = 1 \sim 8$ となる（詳しくは後述する）。$t=0$ とすると式（5.5）は

$$\begin{aligned}
\theta_{1,1} &= r\theta_{0,0}+(1-r)\theta_{1,0}+r\theta_{2,0}\\
\theta_{2,1} &= r\theta_{1,0}+(1-r)\theta_{2,0}+r\theta_{3,0}\\
\theta_{3,1} &= r\theta_{2,0}+(1-r)\theta_{3,0}+r\theta_{4,0}\\
&\vdots\\
\theta_{8,1} &= \theta_{7,0}+(1-r)\theta_{8,0}+r\theta_{9,0}
\end{aligned} \tag{5.8}$$

この式で $\theta_{0,1}$ と $\theta_{9,1}$ は棒の両端部である。数値解析ではこれらの端部の格子点の温度をあらかじめ設定しなければ計算を行うことができない。この端部における温度設定を**境界条件**といい、偏微分方程式の数値計算では不可欠な条件である。また式（5.8）中の

例えば、ある棒状の物体を 1 次元に 10 点の格子点で