有限差分法による拡散方程式の数値解法と Python プログラミング

東京農工大学 山中晃徳

2022 年 11 月吉日

1　数値シミュレーションの基礎

フェーズフィールドモデル（Phase-field model) の理論とその Python プログラミングを理解する基礎として, 拡散方程式（Diffusion equation）の定式とその Python プログラミングを説明する.

1.1　拡散方程式

ある溶媒原子 A と溶質原子 B からなる系を考え, 溶質原子 B が拡散する現象を想定する. 系に設定した直交デカルト座標 $\mathbf{r} = (x_1, x_2, x_3)$, 時刻 t において, 溶質原子 B の濃度変数 $c(\mathbf{r}, t)$ を考えると，拡散方程式は，次式で表される.

$$\frac{\partial c(\mathbf{r}, t)}{\partial t} = -\nabla \cdot J(\mathbf{r}, t) \tag{1}$$

ここで，$J(\mathbf{r}, t)$ は拡散流束であり，フィック（Fick）の第一法則より次式で与えられる.

$$J(\mathbf{r}, t) = -D(\mathbf{r}, t)\nabla c(\mathbf{r}, t) \tag{2}$$

ここで，$D(\mathbf{r}, t)$ は拡散係数である. 式 (1) に式 (2) を代入すると，次式が得られる.

$$\frac{\partial c(\mathbf{r}, t)}{\partial t} = \nabla \cdot (D(\mathbf{r}, t)\nabla c) \tag{3}$$

拡散係数 $D(\mathbf{r}, t)$ は, 座標と時間に寄らず一定値 D であると仮定すると，拡散方程式は次式のように書き下せる.

$$\begin{aligned}\frac{\partial c(\mathbf{r}, t)}{\partial t} &= \nabla \cdot (D(\mathbf{r}, t)\nabla c(\mathbf{r}, t)) \\ &= D\nabla^2 c(\mathbf{r}, t) \\ &= D\left(\frac{\partial^2 c(\mathbf{r}, t)}{\partial x_1^2} + \frac{\partial^2 c(\mathbf{r}, t)}{\partial x_2^2} + \frac{\partial^2 c(\mathbf{r}, t)}{\partial x_3^2}\right)\end{aligned} \tag{4}$$

次節では, 式 (4) を有限差分法（Finite Difference Method）で離散化し, 数値計算する方法を説明する.

1.2　有限差分法による拡散方程式の離散化

有限差分法は，偏微分方程式を数値解析する（コンピューターに近似的数値解を計算させる）基礎的な方法であり，フェーズフィールド法の数値計算方法として多用されている．ここでは，1.1 節で説明した拡散方

程式を例として，フェーズフィールドモデルを数値計算するために最低限必要となる前進差分法（オイラー（Euler）法とも呼ばれる）と中央差分法を解説する．これらの方法は，直感的で理解しやすいが，数値流体計算などの高精度な計算を必要とする場合には，精度が不十分である．より高精度な計算をするための有限差分法は, 他の文献も参考とされたい.

以下では，1.1 節と同様に, ある溶媒原子 A と溶質原子 B からなる系を考え, 溶質原子 B が拡散する現象を想定する．系に設定した 2 次元空間を考え，直交デカルト座標系 $\mathbf{r} = (x_1, x_2)$ と時間 t に依存する, 原子 B の濃度変数 $c(\mathbf{r}, t)$ の変化は, 次式の拡散方程式で表されるとする．

$$\begin{aligned}\frac{\partial c(\mathbf{r},t)}{\partial t} &= \nabla \cdot (D(\mathbf{r},t)\nabla c(\mathbf{r},t)) \\ &= D\nabla^2 c(\mathbf{r},t) \\ &= D\left(\frac{\partial^2 c(\mathbf{r},t)}{\partial x_1^2} + \frac{\partial^2 c(\mathbf{r},t)}{\partial x_2^2}\right)\end{aligned} \tag{5}$$

上式において, 拡散係数 $D(\mathbf{r}, t)$ は座標と時間に寄らず一定値 D であると仮定した.

式 (5) を有限差分法により数値計算するために, 時間の離散化を考える．いま時刻 $t = 0$ から $t = t_{end}$ までの溶質原子 B の拡散現象を考え, この時間を N_{step} に分割する．このとき, ある時刻 t は, $t = n\Delta t$ のように表すことができる．n は 時間ステップ数であり, 0 から $\frac{t_{end}}{\Delta t}$ の値をとりうる．Δt は, 1 つの時間ステップあたりの時間幅であり, 時間増分 (Time increment) である．

次に, 空間の離散化を考える．x_1 方向, x_2 方向の長さが, それぞれ L_x, L_y の 2 次元計算領域を考える．また, その計算領域を x_1 方向, x_2 方向に, それぞれ N_x, N_y の計算格子点（差分格子点）で分割することを考える．このとき, x_1 方向, x_2 方向の差分格子点の間隔（距離）は, それぞれ Δx と Δy である.

上記の時間と空間の離散化を考え, 式 (5) 左辺の時間微分を前進差分法で離散化すると，次式のように近似計算することができる.

$$\left.\frac{\partial c(\mathbf{r},t)}{\partial t}\right|_t \simeq \frac{c_{[i,j]}^{n+1} - c_{[i,j]}^{n}}{\Delta t} \tag{6}$$

ここで，$c_{[i,j]}^{n}$ は, 時間ステップ n での差分格子点 $[i, j]$ ($i = 1, 2, 3, \cdots, N_x$ および $j = 1, 2, 3, \cdots, N_y$) における濃度である．したがって，$c_{[i,j]}^{n+1}$ は, 時刻 t から時間増分 Δt 後の差分格子点 $[i, j]$ における溶質原子 B の濃度を意味する.

次に，式 (5) 右辺の空間微分の離散化を説明する．まず簡単のため, 1 次元空間を例にとり，濃度変数の座標 x に関する濃度変数 $c(x)$ の 2 階微分の有限差分法による離散化を説明する．$c(x)$ が, 微分可能であるとすれば，テイラー（Taylor）展開を利用して，次式が得られる.

$$c(x_{[i+1]}) = c(x) + \frac{dc(x)}{dx}\Delta x + \frac{1}{2}\frac{d^2c(x)}{dx^2}\Delta x^2 + \frac{1}{6}\frac{d^3c(x)}{dx^3}\Delta x^3 + O(\Delta x^4) \tag{7}$$

$$c(x_{[i-1]}) = c(x) + \frac{dc(x)}{dx}(-\Delta x) + \frac{1}{2}\frac{d^2c(x)}{dx^2}\Delta x^2 + \frac{1}{6}\frac{d^3c(x)}{dx^3}(-\Delta x)^3 + O(\Delta x^4) \tag{8}$$

ここで，例えば $O(\Delta x^4)$ は, 差分格子点の間隔 Δx の 4 次以上の項を表しており，Δx は全領域に比べて，小さく設定するため，無視できるほどの誤差となる．式 (7) と式 (8) の和を求めると，次式が得られる.

$$c(x_{[i+1]}) + c(x_{[i-1]}) = 2c(x_{[i]}) + \frac{d^2c(x)}{dx^2}\Delta x^2 + O(\Delta x^2) \tag{9}$$

よって，$c(x)$ の x に関する 2 階の空間微分は，次式のように近似的に計算できる.

$$\frac{d^2c(x)}{dx^2} \simeq \frac{c(x_{[i+1]}) - 2c(x_{[i]}) + c(x_{[i-1]})}{\Delta x^2} \tag{10}$$

これを中心差分法（Central difference method）と呼ぶ．ここで，式 (10) に存在するべき $O(\Delta x^2)$ を消去したため, この中央差分法には $O(\Delta x^2)$ の誤差を含む．同様の計算を, 2 次元空間についても同様に行うことで, 式 (5) 右辺の空間微分は, 時刻 t での値であることを考慮すれば，次式のように近似計算できる．

$$\left.\frac{\partial^2 c(\mathbf{r},t)}{\partial x_1^2}\right|_t \simeq \frac{c^n_{[i+1,j]} - 2c^n_{[i,j]} + c^n_{[i-1,j]}}{\Delta x^2} \tag{11}$$

$$\left.\frac{\partial^2 c(\mathbf{r},t)}{\partial x_2^2}\right|_t \simeq \frac{c^n_{[i,j+1]} - 2c^n_{[i,j]} + c^n_{[i,j-1]}}{\Delta y^2} \tag{12}$$

式 (6)，式 (11) および式 (12) を，式 (5) に代入すれば，各差分格子点での溶質原子 B の濃度変数 $c(\mathbf{r},t)$ の近似解を求める拡散方程式は次式のように与えられる．

$$\frac{c^{n+1}_{[i,j]} - c^n_{[i,j]}}{\Delta t} = D\left(\frac{c^n_{[i+1,j]} - 2c^n_{[i,j]} + c^n_{[i-1,j]}}{\Delta x^2} + \frac{c^n_{[i,j+1]} - 2c^n_{[i,j]} + c^n_{[i,j-1]}}{\Delta y^2}\right) \tag{13}$$

ここで，$c^{n+1}_{[i,j]}$ について整理し，$\Delta x = \Delta y$ を仮定すれば，次式が得られる．

$$c^{n+1}_{[i,j]} = c^n_{[i,j]} + D\Delta t\left(\frac{c^n_{[i+1,j]} + c^n_{[i-1,j]} + c^n_{[i,j+1]} + c^n_{[i,j-1]} - 4c^n_{[i,j]}}{\Delta x^2}\right) \tag{14}$$

式 (14) に示したように，差分格子点 $[i,j]$, 時刻 $t = (n+1)\Delta t$ における濃度は，時刻 $t = n\Delta t$ での差分格子点 $[i,j]$ とその上下左右 4 点（2 次元問題の場合）の差分格子点における濃度から求められる．次章では, 式 (14) を実際に計算する Python プログラムを説明する．

2　有限差分法による拡散方程式の数値計算の Python プログラミング

1.2 節で説明した，有限差分法で離散化した拡散方程式（式 (14)）を計算するサンプルプログラム diffusion-2d.ipynb を説明する．Python プログラミングは逐次処理を行うため，for 文による繰り返し処理を多用すると計算コストが増大する．差分格子点数が少ない場合には, それほど問題とならないが, 大きな計算領域で計算したい場合などは, 計算時間の増大が問題となる．そこで，for 文による繰り返し処理を使わないサンプルプログラム diffusion-2d-wo-for.ipynb との違いもあわせて説明する．特に説明しない限りは，2 つのサンプルプログラムは同じ記述である．

■STEP 1　必要となる Python のライブラリをインポートする．このプログラムでは，配列を用いた計算に便利な numpy と，グラフ描画に便利な matplotlib の 2 つのライブラリを主に使用する．また，for 文による繰り返し処理の有無による計算時間の違いを計測するために time ライブラリも使用する．

Listing 1　Python のライブラリのインポート

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from time import time
```

■STEP 2　x_1 方向と x_2 方向の差分格子点数，差分格子点の間隔，拡散係数 D, 計算ステップの総数 N_{steps}, 時間増分 Δt, 初期濃度 c_0 を設定する． 時間増分は, クーラン (Courant) 条件を満たすように設定する．

Listing 2　計算条件の設定

```
1 nx, ny = 128, 128
2 dx = dy = 0.5
3 D = 0.3
4 nsteps = 1000
5 dt = dx*dx*dy*dy/(2*D*(dx*dx+dy*dy))*0.5
6 c0 = 1.0
```

■STEP 3　時刻 $t = n\Delta t$ および $(n+1)\Delta t$ での各差分格子点での濃度変数の値 $c^n_{[i,j]}$ と $c^{n+1}_{[i,j]}$ を保存するための numpy 配列 (c と c_new) を定義する．配列の大きさは，x_1 軸と x_2 軸の方向の差分格子点数である nx × ny である．np.zeros と指定することで, 配列には初期値 0 が代入される．

Listing 3　濃度変数を保存する配列の定義

```
1 c = np.zeros((nx, ny))
2 c_new = np.zeros((nx, ny))
```

■STEP 4　式 (14) に示した離散化された拡散方程式を計算して $c^n_{[i,j]}$ から $c^{n+1}_{[i,j]}$ を求め，配列に保存する処理をするサブルーチン (Python では関数と呼ぶ) を定義する．ここでは，境界条件として，計算領域の端部では濃度変数の空間勾配がゼロであると仮定する, ゼロノイマン (Zero-Neumann) 条件を考える．

diffusion-2d.ipynb において, for 文による繰り返し処理を行う場合のプログラミング方法は次の通りである．16 行目において, 式 (14) を計算する．

Listing 4　for 文を用いた式 (14) の数値計算

```
1  def calc_diffusion(c, c_new):
2      for j in range(ny):
3          for i in range(nx):
4              ip = i + 1
5              im = i - 1
6              jp = j + 1
7              jm = j - 1
8              if ip > nx - 1:
9                  ip = nx - 1
10             if im < 0:
11                 im = 0
12             if jp > ny - 1:
13                 jp = ny - 1
14             if jm < 0:
15                 jm = 0
16             c_new[i,j] = c[i,j] + D*dt*(c[ip,j] + c[im,j] + c[i,jp] + c[i,jm] - 4*c[i
                   ,j])/dx/dx
17     c[:,:] = c_new[:,:]
```

次に，diffusion-2d-wo-for.ipynb において, for 文による繰り返し処理を行わない場合のプログラミング方法は次の通りである．2 行目が, 式 (14) を計算する箇所である．なお, 3-6 行目においては, 境界条件（本プログラムでは, ゼロノイマン条件）を満たすように, 計算領域の端での濃度変数を調整する必要がある．

Listing 5　for 文を用いない場合の式 (14) の数値計算

```
def calc_diffusion(c, c_new):
    c_new[1:-1, 1:-1] = c[1:-1, 1:-1] + D*dt* (c[2:, 1:-1] + c[:-2, 1:-1] + c
        [1:-1, 2:] + c[1:-1, :-2] - 4*c[1:-1, 1:-1]) /dx/dx
    c_new[0,:] = c_new[1,:]
    c_new[nx-1,:] = c_new[nx-2,:]
    c_new[:,0] = c_new[:,1]
    c_new[:,ny-1] = c_new[:,ny-2]
    c[:,:] = c_new[:,:]
```

■STEP 5　初期濃度分布を設定する．計算領域中心から半径 $5\Delta x$ の範囲内において, 初期濃度 $c = c_0$ とし, それ以外の領域では $c = 0$ する.

Listing 6　初期濃度分布の設定

```
r = 5.0
x0 = nx/2
y0 = ny/2
for i in range(nx):
    for j in range(ny):
        r2 = (i*dx-x0*dx)**2 + (j*dy-y0*dx)**2
        if r2 < r**2:
            c[i,j] = c0
```

■STEP 6　Step 4 で定義した関数を繰り返し呼び出す（Call する）ことで得られた $c^{n+1}_{[i,j]}$ と $c^{n}_{[i,j]}$ を入れ替える．本プログラムでは, 時間ステップを 100 進めるごとに, matplotlib ライブラリを用いて, 濃度変数の 2 次元分布を描画する．全計算ステップ数（1000）計算するために要した時間の計測結果を表示する.

Listing 7　拡散方程式を計算するメインルーチン

```
start = time()
for nstep in range(nsteps+1):
    calc_diffusion(c,c_new)

    if nstep % 100 == 0:
        print('nstep = ', nstep)
        plt.imshow(c, cmap='bwr')
        plt.title('concentration')
        plt.colorbar()
        plt.show()

end = time()
print("1000 steps of diffuse took", (end-start)*1000.0, "ms")
```

上記のプログラムを実行すると, 時間ステップ数の増加とともに, 濃度が高い領域が計算領域内で広がる（拡散する）様子が得られる．このとき, 2 つのプログラム（diffusion-2d.ipynb と diffusion-2d-wo-for.ipynb）は同じ計算結果となるが, for 文を用いた場合と用いない場合の計算速度の差は歴然である．Google Colaboratory

上で実行した場合には，前者の計算時間は 1 分強であるのに対し，後者は 3 秒弱である．しかしながら，for 文を使わないプログラミングは，必ずしも読みやすさや柔軟性が高くないため，for 文を用いてプログラミングしながらも計算の高速化を望む場合には，本書で使用する numba ライブラリの JIT コンパイラや画像描画装置（Graphics Processing Unit: GPU）を使った計算も有効である．