さわって、うごかして理解する
フェーズフィールド法 ＜基礎編＞

東京農工大学 山中晃徳

2022 年 11 月吉日

1 Allen-Cahn 方程式を用いた移動境界問題のフェーズフィールドシミュレーション

1.1 Allen-Cahn 方程式の確認

次式の Allen-Cahn 方程式を数値解析し, 移動する境界を追跡することを説明する.

$$\frac{\partial \phi}{\partial t} = -M_\phi \left(\frac{\partial p(\phi)}{\partial \phi}(g_B - g_A) + W \frac{\partial q(\phi)}{\partial \phi} - a^2 \nabla^2 \phi \right) \tag{1}$$

ここで, W, a および M_ϕ は, ダルブウェルポテンシャルのエネルギー障壁高さ, 勾配エネルギー係数, フェーズフィールドモビティーと呼ばれるパラメータであり, 界面エネルギー σ と界面モビリティー M および拡散界面幅 δ と次式のように関係付けられる.

$$W = \frac{6\sigma\delta}{b} \tag{2}$$

$$a = \sqrt{\frac{3\sigma b}{\delta}} \tag{3}$$

$$M_\phi = \frac{\sqrt{2W}}{6a} M \tag{4}$$

ここで, $b = 2\tanh^{-1}(1-2\lambda)$ であり, 界面領域の幅を決めるパラメータ λ によって値が決まる. 次節で, 式 (1) の Allen-Cahn 方程式を有限差分法を用いて離散化し, それを Python プログラミングすることを説明しよう.

1.2 有限差分法を用いた離散化

式 (1) の Allen-Cahn 方程式には, 秩序変数 ϕ の 1 階の時間微分と 2 階の空間微分が含まれる. ここでは, 1 次元空間を考え, x 軸の方向に N_x の差分格子点で離散化しよう. このように空間を離散化した場合, 時刻 t における差分格子点 $[i]$ での秩序変数は $\phi^t_{[i]}$ と表記する.

時間微分は次式で与えられる 1 次精度の前進差分法を用いることとしよう. つまり, 次式で近似計算する.

$$\frac{\partial \phi}{\partial t} \simeq \frac{\phi^{t+\Delta t}_{[i]} - \phi^t_{[i]}}{\Delta t} \tag{5}$$

ここで，Δt は有限差分法における 1 ステップあたりの時間増分である．一方，ϕ の 2 階の空間微分は，2 次精度の中央差分法を用いると

$$\begin{aligned}\nabla^2\phi &= \frac{\partial^2\phi}{\partial x^2} \\ &\simeq \frac{\phi_{[i+1]}^{t+\Delta t} + 2\phi_{[i]}^{t} - \phi_{[i-1]}^{t}}{\Delta x^2}\end{aligned} \tag{6}$$

と計算する．ここで，Δx は x 軸の方向の差分格子点の間隔である．以上の離散式を用いれば，式 (1) の離散式は，次式のように表される．

$$\phi_{[i]}^{t+\Delta t} = \phi_{[i]}^{t} + M_\phi f\left(\phi_{[i]}^{t}\right)\Delta t + a^2\nabla^2\phi_{[i]}^{t}\Delta t \tag{7}$$

ここで，

$$f\left(\phi_{[i]}^{t}\right) = 4W\phi_{[i]}^{t}\left(1-\phi_{[i]}^{t}\right)\left(\phi_{[i]}^{t} - \frac{1}{2} + \frac{3}{2W}(g_A - g_B)\right) \tag{8}$$

である．次節では，式 (8) の離散式を計算する Python プログラムを説明する．

1.3 有限差分法での Python プログラミング

1.2 節で離散化した 1 次元での Allen-Cahn 方程式を計算するサンプルプログラム Allen-Cahn-1d-FDM.ipynb を説明する．このサンプルプログラムでは，A 相を母相として単一の B 相が一定の駆動力のもとで成長する現象を計算する．以下に，サンプルプログラムの主なポイントを解説する．

■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，numpy，matplotlib，math の 3 つのライブラリを使用する．

Listing 1　Python のライブラリのインポート

```
import numpy as np
import matplotlib.pyplot as plt
import math
```

■STEP 2　N_x の差分格子点，Δx の差分格子点の間隔，A 相と B 相の界面での界面エネルギー σ，拡散界面の幅 δ，界面のモビリティー M, 定数 b の値を決めるパラメータ λ を設定する．

Listing 2　物性値や定数の設定

```
nx = 64 # number of computational grids
dx = 0.5e-6 # spacing of computational grid [m]
eee = 5.e+5 # magnitude of driving force of growth of phase B [J/m3] = [Pa]
sigma = 1.0 # ineterfacial energy [J/m2]
delta = 4.*dx # interfacial thickness [m]
amobi = 4.e-14 # interfacial mobilitiy [m4/(Js)]
ram = 0.1 # paraneter which deternines the interfacial area
bbb = 2.*np.log((1.+(1.-2.*ram))/(1.-(1.-2.*ram)))/2. # The constant b = 2.1972
```

■STEP 3　フェーズフィールドパラメータである W，a および M_ϕ を計算する．

Listing 3　フェーズフィールドパラメータの計算

```
aaa = np.sqrt(3.*delta*sigma/bbb) # gradient energy coefficient "a"[(J/m)^(1/2)]
www = 6.*sigma*bbb/delta # potential height W [J/m3]
pmobi = amobi*math.sqrt(2.*www)/(6.*aaa) # mobility of phase-field [m3/(Js)]
```

■STEP 4　数値計算を安定に行うため，時間増分 Δt をクーラン条件よりも小さい値（本プログラムでは 1/2）とし，時間ステップの総数 N_{step} を設定する．

Listing 4　時間増分と計算ステップ数の設定

```
dt = dx*dx/(5.*pmobi*aaa*aaa)/2 # time increment for a time step [s]
nsteps = 1000 # total number of time step
```

■STEP 5　時刻 t での各差分格子点での秩序変数の値を保存する配列を numpy 配列として定義する．配列の大きさは，x 軸の方向の差分格子点数である．なお，Allen-Cahn 方程式を計算して，時刻での秩序変数の値が得られるたびに，この配列を上書きして使用する．また, このプログラムでは, Allen-Cahn 方程式の右辺の駆動力項と勾配項が界面移動にどう寄与しているかを理解するために, 各差分格子点での駆動力項と勾配項を保存する配列 driv と grad を定義する．

Listing 5　秩序変数を保存する配列の定義

```
p = np.zeros((nx)) # phase-field variable at time t
p_new = np.zeros((nx)) # phase-field variable at time t + dt
driv = np.zeros((nx)) # array for saving driving force term (only for visualization)
grad = np.zeros((nx)) # array for saving gradient force term (only for visualization)
```

■STEP 6　B 相の秩序変数の初期分布を設定する．計算領域の原点から半径 $r_0 = 5\Delta x$ の領域は，$\phi = 1$ であり，それ以外の領域は $\phi = 0$ となるように設定する．ただし，拡散界面領域では，秩序変数の平衡分布を表す次式を用いて，初期分布を設定する．

$$\phi = \frac{1}{2}\left(1 - \tanh\frac{\sqrt{2W}}{2a}(r - r_0)\right) \tag{9}$$

ここで，r は原点からの距離を表す．

Listing 6　秩序変数の初期分布の設定

```
r_nuclei = 10.*dx # length of the initial B phase
for i in range(0,nx):
        r = i*dx - r_nuclei
        p[i] = 0.5*(1.-np.tanh(np.sqrt(2.*www)/(2.*aaa)*r))
```

■STEP 7　Allen-Cahn 方程式を計算して $\phi^t_{[i]}$ から $\phi^{t+\Delta t}_{[i]}$ を求め (16 行目)，配列を上書きする処理を n_{step} 回繰り返す．本プログラムでは，境界条件として，秩序変数の空間勾配がゼロとするゼロノイマン条件を考える．また，時間ステップを 100 回繰り返すごとに，matplotlib を用いて秩序変数 ϕ の分布を描画する．

Listing 7　Allen-Cahn 方程式の計算

```
for t in range(nsteps+1):
    for i in range(nx):
        ip = i + 1
        im = i - 1
        if ip > nx - 1:
            ip = nx -1
        if im < 0:
            im = 0
        p_new[i] = p[i] + pmobi * ( 4.*www*p[i]*(1.-p[i])*(p[i]-0.5+3./(2.*www)*eee)
                                    + aaa*aaa*((p[ip] - 2*p[i] + p[im])/dx/dx) ) * dt
        driv[i] = pmobi*(4.*www*p[i]*(1.-p[i])*(p[i]-0.5+3./(2.*www)*eee))
        grad[i] = pmobi*(aaa*aaa*(p[ip] - 2*p[i] + p[im])/dx/dx)
    p = p_new

    if t%100 == 0:
        print('nstep = ', t)
        fig = plt.figure(figsize=(7,9))
        fig.set_dpi(100)
        plt.subplots_adjust(hspace=0.5)
        plt.subplot(4, 1, 1)
        plt.ylim([0,1])
        plt.xlim([0,nx])
        plt.xlabel('Grid number')
        plt.ylabel('Phase-field variable')
        plt.grid(True)
        plt.plot(p[:],color="b",marker="o")
       ....
        plt.show()
```

上記のサンプルプログラムを実行すると，原点付近に設置した B 相（$\phi = 1$ の領域）が時間とともに増大する．また，Step 2 で設定したように，A 相と B 相の間の拡散界面は差分格子点 4 つほどの幅をもつこともわかる．

■演習 1

- 駆動力の大きさを変更して, 計算結果（界面移動）の変化を確認してみよう.
- 界面幅 $\delta = 4\Delta x$ を $2\Delta x$ などに変更して, 計算結果（界面移動）の変化を確認してみよう.

2 Cahn-Hilliard 方程式を用いたスピノーダル相分解のフェーズフィールドシミュレーション

2.1 Cahn-Hilliard 方程式の確認

Cahn-Hilliard 方程式を計算し, スピノーダル相分解のフェーズフィールドシミュレーションを行うことを説明する. Cahn-Hilliard 方程式を次式に示す.

$$\frac{\partial c}{\partial t} = \nabla \cdot \left(M_c \nabla \frac{\delta G}{\delta c} \right) = M_c \nabla^2 \mu \tag{10}$$

ここで，M_c は拡散モビリティーであり, 場所に依存しない定数と仮定した. μ は拡散ポテンシャルであり，次式で与えられる.

$$\mu = \frac{\delta G}{\delta c} = \mu_{chem} + \mu_{grad} \tag{11}$$

ここで，μ_{chem} と μ_{grad} は, それぞれ次式で与えられる.

$$\mu_{chem} = \frac{\partial f_{chem}(c)}{\partial c} = 2Ac(1-c)^2 - 2Ac^2(1-c) \tag{12}$$

$$\mu_{grad} = -\nabla \cdot \frac{\partial f_{grad}}{\partial(\nabla c)} = -a\nabla^2 c \tag{13}$$

次節では，式 (10) の Cahn-Hilliard 方程式を有限差分法を用いて離散化し，それをプログラミングすることを説明する.

2.2 有限差分法を用いた Cahn-Hilliard 方程式の離散化

式 (10) の Cahn-Hilliard 方程式には，秩序変数 c の 1 階の時間微分と 2 階の空間微分が含まれる．ここでは，2 次元空間を考え，x 軸と y 軸の方向にそれぞれ N_x と N_y の差分格子点で離散化する．時刻 t における差分格子点 $[i,j]$ での秩序変数は $c^t_{[i,j]}$ と表記する．時間微分は次式で与えられる 1 次精度の前進差分法を用いる．つまり，

$$\frac{\partial c}{\partial t} \simeq \frac{c^{t+\Delta t}_{[i,j]} - c^t_{[i,j]}}{\Delta t} \tag{14}$$

である．ここで，Δt は有限差分法における 1 ステップあたりの時間増分である．これを式 (10) に代入し，$c^{t+\Delta t}_{[i,j]}$ について整理すれば，次式が得られる.

$$c^{t+\Delta t}_{[i,j]} = c^t_{[i,j]} + \Delta t M_c \nabla^2 \mu^t_{[i,j]} \tag{15}$$

ここで，$\mu^t_{[i,j]}$ の 2 階の空間微分に対して，2 次精度の中央差分法を用いると

$$\nabla^2 \mu^t_{[i,j]} \simeq \frac{\mu^{t+\Delta t}_{[i+1,j]} + 2\mu^t_{[i,j]} - \mu^t_{[i-1,j]}}{\Delta x^2} + \frac{\mu^{t+\Delta t}_{[i,j+1]} + 2\mu^t_{[i,j]} - \mu^t_{[i,j-1]}}{\Delta y^2} \tag{16}$$

と計算できる．ここで，Δx と Δy は x 軸と y 軸の方向の差分格子点の間隔である．その他，式 (16) を計算するために, 次式を示しておこう.

$$\mu^t_{[i,j]} = \mu^t_{chem[i,j]} + \mu^t_{grad[i,j]} \tag{17}$$

$$\mu^t_{chem[i,j]} = 2Ac^t_{[i,j]}\left(1-c^t_{[i,j]}\right)^2 - 2A(c^t_{[i,j]})^2\left(1-c^t_{[i,j]}\right) \tag{18}$$

$$\mu^t_{grad[i,j]} = -a\left\{\frac{c^t_{[i+1,j]}-2c^t_{[i,j]}+c^t_{[i-1,j]}}{\Delta x^2} + \frac{c^t_{[i,j+1]}-2c^t_{[i,j]}+c^t_{[i,j-1]}}{\Delta y^2}\right\} \tag{19}$$

次節では，これらの離散式を用いて，Cahn-Hilliard 方程式を計算するサンプルプログラムを説明しよう．

2.3 有限差分法での Python プログラミング

2.2 節で離散化した 2 次元での Cahn-Hilliard 方程式を計算するサンプルプログラム Cahn-Hilliard-2d-FDM.ipynb を説明する．このサンプルプログラムでは，A-B2 元合金で生じるスピノーダル分解を計算する．以下に，サンプルプログラムの主なポイントを解説しよう．

■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，numpy，matplotlib および numba ライブラリの JIT コンパイラを使用する．

Listing 8　Python のライブラリのインポート

```
import numpy as np
import matplotlib.pyplot as plt
from time import time
from numba import jit
```

■STEP 2　N_x や N_y の差分格子点，Δx や Δy の差分格子点の間隔，計算ステップの総数 N_{step} を設定する．

Listing 9　差分格子点数やステップ数などの設定

```
nx = ny = 128
dx = dy = 1.0
total_step = 2000
```

■STEP 3　フェーズフィールドパラメータである濃度勾配エネルギー係数 a_c，拡散モビリティー M_c, 時間増分 Δt, 初期濃度 c_0 を設定する．

Listing 10　拡散係数や時間増分などの設定

```
dt = 1.0e-2
A = 2.0
mobility = 1.0
grad_coef = 1.0
c0 = 0.5
```

■STEP 4　時刻 t および $t+\Delta t$ での各差分格子点での秩序変数の値 $c^t_{[i,j]}$ を保存する配列を numpy 配列として定義する．配列の大きさは，x 軸と y 軸の方向の差分格子点数である $N_x \times N_y$ である．また, 初期濃度場を設定する．このとき, STEP 3 で設定した平均値 c_0 に対して, 乱数を用いてゆらぎを与える．なお, 全ての差分格子点において，時刻 $t+\Delta t$ での秩序変数の値 $c^{t+\Delta t}_{[i,j]}$ が得られた後は，2 つの配列の中身を入れ替える．

Listing 11　時刻 t および $t+\Delta t$ での秩序変数を保存する配列の定義

```
con = np.zeros([nx, ny])
con_new = np.zeros([nx, ny])
con = c0 + 0.01 * (0.5 - np.random.rand(nx, ny))
```

■STEP 5　Cahn-Hilliard 方程式を計算して $c^t_{[i,j]}$ から $c^{t+\Delta t}_{[i,j]}$ を求め，配列に保存する処理をするための関数（サブルーチン）を定義する．この部分は長いため，部分的に説明しよう．まず，差分格子点 $[i,j]$ を中心として，$[i\pm1, j\pm1]$，$[i\pm2, j]$，$[i, j\pm2]$ での濃度変数を求め，変数に代入する．このとき，境界条件として，周期境界条件を考える．

Listing 12　各差分格子点での秩序変数を変数に代入する

```
@jit(nopython=True)
def update(con, con_new):
  for j in range(ny):
    for i in range(nx):

      ip = i + 1
      im = i - 1
      jp = j + 1
      jm = j - 1
      ipp = i + 2
      imm = i - 2
      jpp = j + 2
      jmm = j - 2

      if ip > nx-1:
        ip = ip - nx
      if im < 0:
        im = im + nx
      if jp > ny-1:
        jp = jp - ny
      if jm < 0:
        jm = jm + ny
      if ipp > nx-1:
        ipp = ipp - nx
      if imm < 0:
        imm = imm + nx
      if jpp > ny-1:
        jpp = jpp - ny
      if jmm < 0:
        jmm = jmm + ny

      cc = con[i,j] # at (i,j) "centeral point"
      ce = con[ip,j] # at (i+1.j) "eastern point"
      cw = con[im,j] # at (i-1,j) "western point"
```

```
35        cs = con[i,jm] # at (i,j-1) "southern point"
36        cn = con[i,jp] # at (i,j+1) "northern point"
37        cse = con[ip,jm] # at (i+1, j-1)
38        cne = con[ip,jp]
39        csw = con[im,jm]
40        cnw = con[im,jp]
41        cee = con[ipp,j] # at (i+2, j)
42        cww = con[imm,j]
43        css = con[i,jmm]
44        cnn = con[i,jpp]
```

続いて，式 (18) に基づき, 差分格子点 $[i, j]$ を中心として上下左右の差分格子点での拡散ポテンシャルの化学的自由エネルギーに由来する成分 $\mu^t_{chem[i\pm1,j\pm1]}$ を計算する．

Listing 13　各差分格子点での $\mu^t_{chem[i,j]}$ の計算

```
1        mu_chem_c = 2.*A*cc*(1.-cc)**2 - 2.*A*cc**2*(1.-cc)
2        mu_chem_w = 2.*A*cw*(1.-cw)**2 - 2.*A*cw**2*(1.-cw)
3        mu_chem_e = 2.*A*ce*(1.-ce)**2 - 2.*A*ce**2*(1.-ce)
4        mu_chem_n = 2.*A*cn*(1.-cn)**2 - 2.*A*cn**2*(1.-cn)
5        mu_chem_s = 2.*A*cs*(1.-cs)**2 - 2.*A*cs**2*(1.-cs)
```

次に，式 (19) に基づき, 拡散ポテンシャルの勾配エネルギーに由来する成分 $\mu^t_{grad[i\pm1,j\pm1]}$ を計算する．

Listing 14　各差分格子点での $\mu^t_{grad[i,j]}$ の計算

```
1        mu_grad_c = -grad_coef*((ce - 2.0*cc + cw)/dx/dx + (cn - 2.0*cc + cs)/dy/dy)
2        mu_grad_w = -grad_coef*((cc - 2.0*cw + cww)/dx/dx + (cnw - 2.0*cw + csw)/dy/dy)
3        mu_grad_e = -grad_coef*((cee - 2.0*ce + cc)/dx/dx + (cne - 2.0*ce + cse)/dy/dy)
4        mu_grad_n = -grad_coef*((cne - 2.0*cn + cnw)/dx/dx + (cnn - 2.0*cn + cc)/dy/dy)
5        mu_grad_s = -grad_coef*((cse - 2.0*cs + csw)/dx/dx + (cc - 2.0*cs + css)/dy/dy)
```

その後, 拡散ポテンシャル $\mu^t_{[i,j]}$ を計算する．さらに，式 (16) に従って, 拡散ポテンシャルの 2 階微分を計算する．

Listing 15　各差分格子点での化学ポテンシャルの 2 階の空間微分の計算

```
1        laplace_mu = (mu_w - 2.0*mu_c + mu_e)/dx/dx + (mu_n - 2.0 *mu_c + mu_s)/dy/dy
```

以上を用いて，式 (15) の通りに，$c^{t+\Delta t}_{[i,j]}$ を求める．

Listing 16　Cahn-Hilliard 方程式の計算

```
1        con_new[i,j] = con[i,j] + mobility*laplace_mu*dt
```

■STEP 5　Step 4 で定義した関数を繰り返し呼び出すことで得られた $c^{t+\Delta t}_{[i,j]}$ と $c^t_{[i,j]}$ を入れ替える．また，ここでは所定 (100) の計算ステップ数ごとに濃度変数の 2 次元分布を描画する．

Listing 17　Cahn-Hilliard 方程式を計算するメインルーチン

```
1 start = time()
```

```
for nstep in range(total_step+1):
    update(con, con_new)
    con = con_new # swap c at time t and c at time t+dt

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

end = time()
print("It takes ", (end-start)*1000.0, "ms")
```

上記のサンプルプログラムを実行すると，スピノーダル分解挙動が再現される．

■演習 2

- 初期の平均濃度を変更して, スピノーダル分解の挙動の変化を調べてみよう．
- 秩序変数の空間勾配がゼロとするゼロノイマン境界条件に変更してみよう．