熱伝導方程式の差分法による解法

平成 27 年 12 月 31 日

1　目的

湯気のシミュレーションにおいて熱伝導方程式を解く必要がある。熱伝導方程式の差分法による解法をまとめる。

2　熱伝導方程式

一次元の熱伝導方程式は次式で表される。

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial^2 x} \tag{1}$$

T は温度、α は温度伝導率であり、次式より求まる。

$$\alpha = \frac{\lambda}{\rho c_p} \tag{2}$$

λ は熱伝導率、ρ は密度、c_p は定圧比熱である。

3　偏導関数の近似

差分法では偏導関数を前進差分、後退差分で次のように近似する。

$$\frac{\partial f}{\partial x} = \frac{f_{i+1} - f_i}{\Delta x} \tag{3}$$

$$\frac{\partial f}{\partial x} = \frac{f_i - f_{i-1}}{\Delta x} \tag{4}$$

2 階の偏導関数は 1 階の偏導関数であるから前進差分と後退差分を用いれば次式を得られる。

$$\frac{\partial^2 f}{\partial^2 x} = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} \tag{5}$$

4　陽解法

前進差分を用いて数値解を求める方法は陽解法と呼ばれる。$T_{x,t}$ として陽解法による差分近似式は次式になる。

$$\frac{T_{i,j+1}-T_{i,j}}{\Delta t}=\alpha\frac{T_{i+1,j}-2T_{i,j}+T_{i-1,j}}{\Delta x^2} \tag{6}$$

$T_{i,j+1}$ について解くと次式になる。ここで γ は $\frac{\Delta t}{\Delta x^2}$ となる。

$$T_{i,j+1}=\alpha\gamma T_{i+1,j}+(1-2\alpha\gamma)T_{i,j}+\alpha\gamma T_{i-1,j} \tag{7}$$

5　陰解法

後退差分を用いて数値解を求める方法は陰解法と呼ばれる。差分近似式は次式となる。

$$\frac{T_{i,j+1}-T_{i,j}}{\Delta t}=\alpha\frac{T_{i+1,j+1}-2T_{i,j+1}+T_{i-1,j+1}}{\Delta x^2} \tag{8}$$

これを整理すると次式になる。

$$-\alpha\gamma T_{i+1,j+1}+(1+2\alpha\gamma)T_{i,j+1}-\alpha\gamma T_{i-1,j+1}=T_{i,j} \tag{9}$$

陽解法とは異なり $T_{i,j}$ から、次の時刻での $T_{i-1,j+1}, T_{i,j+1}, T_{i+1,j+1}$ の 3 点を求める。前式を行列で表すと次の式のようになる。

$$\begin{pmatrix} 1+2\alpha\gamma & -\alpha\gamma & 0 & \dots & 0 \\ -\alpha\gamma & 1+2\alpha\gamma & -\alpha\gamma & \dots & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & \dots & -\alpha\gamma & 1+2\alpha\gamma & -\alpha\gamma \\ 0 & \dots & 0 & -\alpha\gamma & 1+2\alpha\gamma \end{pmatrix} \begin{pmatrix} T_{1,j+1} \\ T_{2,j+1} \\ \vdots \\ T_{M-1,j+1} \\ T_{M,j+1} \end{pmatrix} = \begin{pmatrix} T_{1,j} \\ T_{2,j} \\ \vdots \\ T_{M-1,j} \\ T_{M,j} \end{pmatrix}$$

前式の連立一次方程式を数値解法によって解く。このとき行列の $T_{i,j+1}$ と $T_{M,j+1}$ には境界条件を設定する。

6　連立一次方程式の数値解法

熱伝導方程式を陰解法で解くためには連立一次方程式を解く必要がある。連立一次方程式を解く方法は数多くあるが、ここでは反復法のヤコビ反復法とガウス・サイデル法について記載する。

以下の三元連立一次方程式を考える。

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{10}$$

前式の行列は次式に書き下せる。

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases} \tag{11}$$

この時、x_i について次式が導かれる。

$$\begin{cases} x_1^{(k+1)} = \dfrac{b_1 - a_{12}x_2^{(k)} - a_{12}x_3^{(k)}}{a_{11}} \\ x_2^{(k+1)} = \dfrac{b_2 - a_{21}x_1^{(k)} - a_{22}x_3^{(k)}}{a_{22}} \\ x_3^{(k+1)} = \dfrac{b_3 - a_{31}x_1^{(k)} - a_{31}x_3^{(k)}}{a_{33}} \end{cases} \tag{12}$$

$x_i^{(0)}$ を初期値として与え、上式に基づき $x_i^{(0)}, x_i^{(1)}, x_i^{(2)}, ...$ と計算する。反復回数を k とし、k を十分大きくすると,$x_i^{(k)}$ は解に収束し、n 元連立一次方程式の場合は次式により計算する。

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{j=1, j\neq i}^{n} \frac{a_{ij}}{a_{ii}} x_j \tag{13}$$

このように解を求める方法がヤコビ反復法である。解が収束したかどうかは $x_i^{(k)}$ と $x_i^{(k+1)}$ の値を比べてその差が許容誤差内に収まっているかどうかで判断する。

ヤコビ反復法では k ステップの x だけを使って $k+1$ の値を求める。ガウス・サイデル法では $x_1, x_2 ... x_n$ と順番に計算した場合、x_2^{k+1} を求める時には x_1^{k+1} がすでに計算されており、これらの最新の値を $k+1$ の値の計算で使う。

7　プログラム例

陽解法の場合、CFL 条件によりタイムステップ幅に厳しい条件を付ける必要がある。陰解法の場合はタイムステップ幅により解が不安定になることはない。ここでは陰解法を用いてガウスサイデル法により一次元の熱伝導方程式を解いた Python のプログラム例とその結果を以下に示す。

```
#coding: utf-8
```

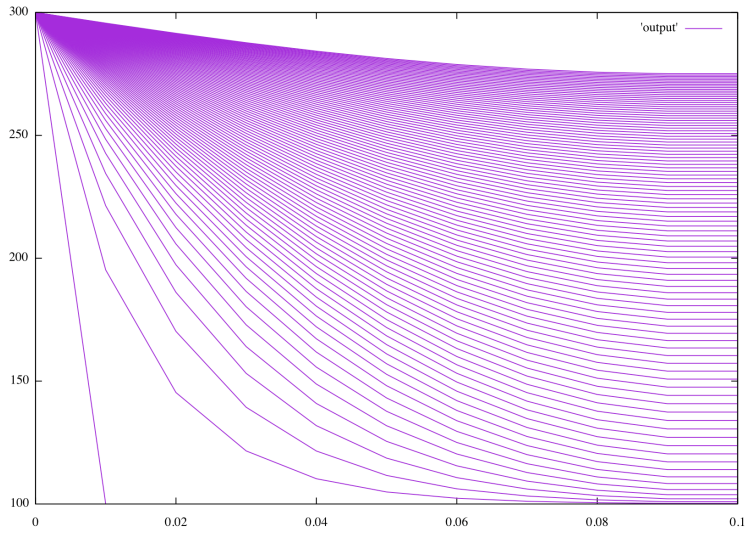
```
L=0.1 #長さ
M=10 #分割数
dx=L/M #空間刻み
dt=1.0 #時間刻み
N=100 #時間ステップ数
alpha=80.2/(7874.0*440.0) #熱伝導率
gamma=dt/dx**2
a=alpha*gamma
T=[273.15 for i in range(M+1)]
A=[[0.0 for i in range(M+1)] for j in range(M+1)]
EPS=1e-15

def gaussseidel(T,preT,A):
        while True:
                d=0.0
                for i in range(M+1):
                        sum=0.0
                        for j in range(M+1):
                                if i != j:
                                        sum+=(A[i][j]/A[i][i])*T[j]
                        n=preT[i]/A[i][i]-sum
                        d+=abs(n-T[i])
                        T[i]=n
                if d < EPS:
                        break

if __name__=='__main__':
        for i in range(1,M):
                A[i][i-1] = -a
                A[i][i] = 1+2*a
                A[i][i+1] = -a
        A[0][0]=1.0
        A[M][M]=1.0

        f = open('output','w')
        for i in range(1,N):
                T[0]=300.0
                T[M]=T[M-1]
                preT = list(T)
```

```
            gaussseidel(T,preT,A)
            f.write('# t=%ss\n'%(i*dt))
            for j in range(M+1):
                    f.write('%s, %s\n'%(j*dx,T[j]))
            f.write('\n\n')
f.close()
```



8　実践的な解法

式 (6) を $T_{i,j+1}$ について整理すると次式になる。

$$T_{i,j+1} = \frac{T_{i,j} + \alpha\gamma(T_{i+1,j+1} + T_{i-1,j+1})}{1 + 2\alpha\gamma} \tag{14}$$

上式を線形反復式として解くこともできる。この場合は行列を意識することがなく簡潔になる。プログラム例は以下を参照。

```
#coding: utf-8

L=0.1 #長さ
M=10 #分割数
dx=L/M #空間刻み
dt=1.0 #時間刻み
N=100 #時間ステップ数
alpha=80.2/(7874.0*440.0) #熱伝導率
```

```
gamma=dt/dx**2
a=alpha*gamma
T=[273.15 for i in range(M+1)]
EPS=1e-15

def gaussseidel(T,preT):
        while True:
                d=0.0
                for i in range(1,M):
                        n=(preT[i]+a*(T[i+1]+T[i-1]))/(1+2*a)
                        d+=abs(n-T[i])
                        T[i]=n
                if d < EPS:
                        break

if __name__=='__main__':

        f = open('output','w')
        for i in range(1,N):
                T[0]=300.0
                T[M]=T[M-1]
                preT = list(T)
                gaussseidel(T,preT)
                f.write('# t=%ss\n'%(i*dt))
                for j in range(M+1):
                        f.write('%s, %s\n'%(j*dx,T[j]))
                f.write('\n\n')
        f.close()
```