计算物理第五次作业

梁旭民*

Cuiying Hornors College, Lanzhou University

liangxm15@lzu.edu.cn

Abstract

本次作业学习了*Crank-Nicolson* 差分格式求解含时微分方程的解法，并运用该方法求解了初始条件为*Gauss*函数的扩散方程，并画出了该扩散方程随时间的演化。

I. 第一个问题

A. 问题描述

已知一维扩散方程形式为

$$\frac{\partial u(x,t)}{\partial t} = D\frac{\partial^2 u(x,t)}{\partial x^2}$$
$$u(x,t)|_{t=0} = u_0(x)$$
$$a_1 u + b_1\frac{\partial u}{\partial n} = c_1 \quad (x=0)$$
$$a_2 u + b_2\frac{\partial u}{\partial n} = c_2 \quad (x=a_0)$$

例：考虑f=0,D=1,边界条件为$\phi(0) = \phi(1) = 0$。设给定的初始条件是中心在$x = \frac{1}{2}$的一个Gauss函数，即：

$$\phi(x,t=0) = \mathrm{e}^{-20(x-\frac{1}{2})^2} - \mathrm{e}^{-20(x-\frac{3}{2})^2} - \mathrm{e}^{-20(x+\frac{1}{2})^2}$$

求随后各个时刻的ϕ。

若设$\tau = 1+80t$，则解析解为:

$$\phi(x,t) = \tau^{-\frac{1}{2}}[\mathrm{e}^{-20(x-\frac{1}{2})^2} - \mathrm{e}^{-20(x-\frac{3}{2})^2} - \mathrm{e}^{-20(x+\frac{1}{2})^2}]$$

上例可以用标准的Crank-Nicolson方法求解。

B. 问题分析

Crank-Nicolson是一种数值分析的有限差分法，可用于数值求解热方程以及类似形式的偏微分方程[2]。它在时间方向上是隐式的二阶方法，可以写成隐式的Runge-Kutta法，数值稳定。该方法诞生于20世纪，由John Crank与Phyllis Nicolson发展[3]。

可以证明Crank-Nicolson方法对于扩散方程(以及许多其他方程)是无条件稳定[4]。但是，如果时间步长Δt乘以热扩散率，再除以空间步长平方Δt^2的值过大(根据Von Neumann稳定性分析，以大于$\frac{1}{2}$为准），近似解中将存在虚假的振荡或衰减。基于这个原因，当要求大时间步或高空间分辨率的时候，往往会采用数值精确较差的后向欧拉法进行计算，这样即可以保证稳定，又避免了解的伪振荡。

1. 一维扩散方程Crank-Nicolson格式

已知一维扩散方程形式为

$$\begin{cases} \dfrac{\partial u(x,t)}{\partial t} = D\dfrac{\partial^2 u(x,t)}{\partial x^2} \\ u(x,t)|_{t=0} = u_0(x) \\ a_1 u + b_1\dfrac{\partial u}{\partial n} = c_1 \qquad (x=0) \\ a_2 u + b_2\dfrac{\partial u}{\partial n} = c_2 \qquad (x=a_0) \end{cases}$$

取$\Delta x = h, \Delta t = \tau$进行离散化，节点坐标为

$$\begin{cases} x_i = (i-1)h \qquad (i=1,2,...,N) \\ t_k = k\tau \qquad (k=1,2,...,k_{max}) \end{cases}$$

节点处的函数为

$$u(x-i, t_k) = u_i^k$$

*指导老师：齐新老师

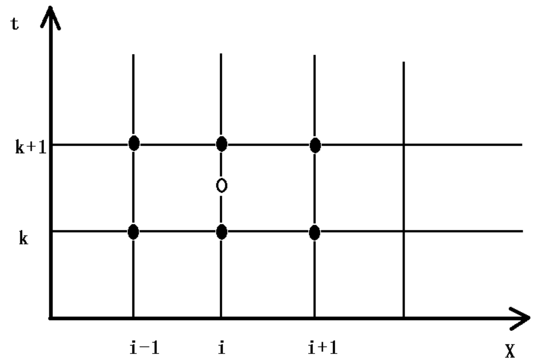

Figure 1: 一维扩散方程的隐式六点差分格式

利用如下差分格式

$$f'(0) \approx \frac{f_1 - f_{-1}}{2h}$$
$$\frac{\partial u}{\partial t} = \frac{u_i^{k+1} - u_i^k}{\tau}$$
$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{2}\left[\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}\right)^k + \left(\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}\right)^{k+1}\right]$$
$$\frac{u_i^{k+1} - u_i^k}{\tau} = \frac{D}{2h^2}\left[\left(u_{i+1} - 2u_i + u_{i-1}\right)^k + \left(u_{i+1} - 2u_i + u_{i-1}\right)^{k+1}\right]$$

令$P = \frac{\tau D}{h^2}, P_1 = \frac{1}{P} + 1, P_2 = \frac{1}{P} - 1$，则可以得到隐式六点差分格式(Crank-Nicolson格式)

$$\left(-u_{i-1} + 2P_1 u_{i+1} - u_{i+1}\right)^{k+1} = \left(u_{i-1} + 2P_2 u_i + u_{i+1}\right)^k$$

2. 边界条件的处理

$$\begin{cases} a_1 u + b_1 \dfrac{\partial u}{\partial n} = c_1 & (x = 0) \\ a_2 u + b_2 \dfrac{\partial u}{\partial n} = c_2 & (x = a_0) \end{cases}$$

设置虚格点$i = 0, i = N + 1$，利用中心差商公式有

$$\begin{cases} a_1 u_1 + b_1 \dfrac{u_0 - u_2}{2h} = c_1 & (x = 0) \\ a_2 u_n + b_2 \dfrac{u_{n+1} - u_{n-1}}{2h} = c_2 & (x = a_0) \end{cases}$$

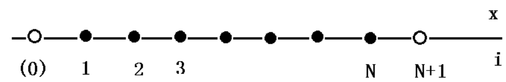

Figure 2: *1*维扩散方程的边界条件

解出u_0, u_{n+1}代入$i = 1, i = N$的Crank-Nicolson差分格式

$$\left[\left(b_1 P_1 + h a_1\right) u_1 - b_1 u_2\right]^{k+1} = \left[\left(b_1 P_2 - h a_1\right) u_1 + b_1 u_2\right]^k + 2hc_1$$
$$\left[-b_2 u_{N-1} + \left(b_2 P_1 + h a_2\right) u_N\right]^{k+1} = \left[b_2 u_{N-1} + \left(b_2 P_2 - h a_2\right) u_N\right]^k + 2hc_2$$

差分方程组及其求解:

将上述差分公式和边界条件结合起来，得到差分线性方程组，其形式为:

$$\mathbf{AU} = \mathbf{R}$$

其中，
$\mathbf{U} = (u_1, u_2, ..., u_n)$是未知量组成的矢量。
$\mathbf{R} = (R_1, R_2, ..., R_n)$是有前一时刻的u值组成的矢量。

则有

$$R_1 = (b_1 P_2 - h a_1) u_1 + b_1 u_2 + 2hc_1$$
$$R_i = u_{i-1} + 2P_2 u_i + u_{i+1} \qquad i = 2, 3, ..., N-1$$
$$R_N = b_2 u_{N-1} + (b_2 P_2 - h a_2) u_N + 2hc_2$$

系数矩阵A是三对角的

$$\begin{bmatrix} b_1 P_1 + h a_1 & -b_1 & 0 & \cdots & 0 & 0 \\ -1 & 2P_1 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2P_1 & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & -1 & 2P_1 & -1 & 0 \\ 0 & \cdots & 0 & -1 & 2P_1 & -1 \\ 0 & 0 & \cdots & 0 & -b_2 & b_2 P_1 + h a_2 \end{bmatrix}$$

这是一个三对角问题，应用追赶法即可得到u_i^{k+1}，而不需要对矩阵直接求逆。

C. 计算结果

运行结果见压缩包中的figure文件夹中的动画1dDiffusion.gif，以下只展示截图

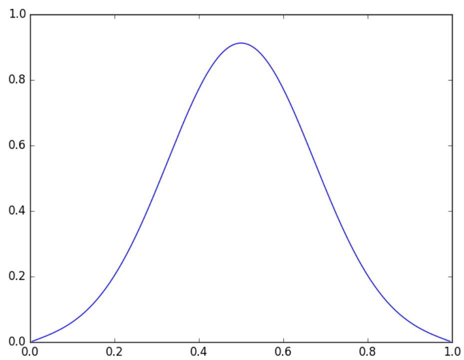

Figure 3: *1维扩散方程瞬时截图*

II. 第二个问题

A. 问题表述

试将第一个问题拓展成二维，求解二维扩散方程随时间的演化并画图(初始条件、边界条件自设，你可以选择二维Crank-Nicolson方法，也可以选择简单Eular法，也可以选择上一章提到的迭代法)。

B. 问题分析

1. 二维扩散方程

将扩散问题延伸到二维的Cartesian网络，我们有类似的推导:

二维的扩散方程形式为

$$\begin{cases} \dfrac{\partial u}{\partial t} = D\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right) & \begin{pmatrix} 0 < x < a_0 \\ 0 < y < b_0 \\ 0 < t < t_{max} \end{pmatrix} \\ u(x,y,0) = u_0(x,y) \end{cases}$$

取$\Delta x = \Delta y = h$的正方形网格覆盖x-y平面，并取$\Delta t = \tau$，节点坐标为

$$\begin{cases} x_i = (i-1)h & (i = 1, 2, \cdots, N) \\ y_i = (i-1)h & (j = 1, 2, \cdots, M) \\ t_k = k\tau & (k = 1, 2, \cdots, k_{max}) \end{cases}$$

节点处的函数为$u(x_i, y_j, t_k) = u_{ij}^k$

在$(i, j, k+\frac{1}{2})$点，将$\frac{\partial u}{\partial t}$用$k$时中心差商代替，将$\frac{\partial^2 u}{\partial x^2}$用$k+1$时的中心差商代替，将$\frac{\partial^2 u}{\partial y^2}$用$k$时的中心差商代替，则二维扩散方程变为

$$\frac{u_{ij}^{k+1} - u_{ij}^k}{\tau} = \frac{D}{h^2}[(u_{i+1,j} - 2u_{ij} + u_{i-1,j})^{k+1} + (u_{i,j+1} - 2u_{ij} + u_{i,j-1})^k]$$

在$(i, j, k+\frac{3}{2})$点，将$\frac{\partial u}{\partial t}$用$k+1$时中心差商代替，将$\frac{\partial^2 u}{\partial x^2}$用$k+1$时的中心差商代替，将$\frac{\partial^2 u}{\partial y^2}$用$k+2$时的中心差商代替，则二维扩散方程变为

$$\frac{u_{ij}^{k+2} - u_{ij}^{k+1}}{\tau} = \frac{D}{h^2}[(u_{i+1,j} - 2u_{ij} + u_{i-1,j})^{k+1} + (u_{i,j+1} - 2u_{ij} + u_{i,j-1})^{k+2}]$$

令$P = \frac{\tau D}{h^2}, P_1 = \frac{1}{P} + 1, P_2 = \frac{1}{P} - 1$，则有

$$\begin{aligned} &(-u_{i-1,j} + 2P_1 u_{ij} - u_{i+1,j})^{k+1} \\ &= (u_{i,j-1} + 2P_2 u_{ij} + u_{i,j+1})^k \qquad \cdots (a) \\ &(-u_{i-1,j} + 2P_1 u_{ij} - u_{i+1,j})^{k+2} \\ &= (u_{i,j-1} + 2P_2 u_{ij} + u_{i+1,j})^{k+1} \qquad \cdots (b) \end{aligned}$$

说明:
1.求解方法与一维情形类似。
2.k为奇数时用(b)式沿y方向计算，k为偶数时用(a)式沿x方向计算，并且只输出后一结果，以减少因为时间上的不一致引起的误差。
3.该方法是无条件稳定的。

2. 边界条件的处理

边界条件的差分格式

$$\begin{cases} a_i u + b_i \dfrac{\partial u}{\partial n} = c_i(y,t) & (1)(2)\text{边界}(i = 1, 2) \\ a_i u + b_i \dfrac{\partial u}{\partial n} = c_i(x,t) & (3)(4)\text{边界}(i = 3, 4) \end{cases}$$

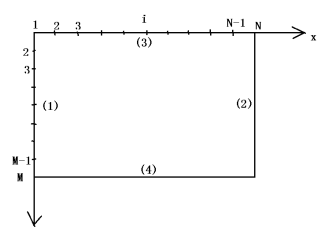

Figure 4: *2维扩散方程边界条件*

设置虚节点:
在(1)(2)边界上有

$$\begin{cases} b_1 u_{0j} = 2hc_1(y_i,t) - 2ha_1 u_{1j} + b_1 u_{2j} & (1) \\ b_2 u_{n+1,j} = 2hc_2(y_i,t) - 2ha_1 u_{nj} + b_2 u_{n-1,j} & (2) \end{cases}$$

在(3)(4)边界上有

$$\begin{cases} b_3 u_{i0} = 2hc_3(x_i,t) - 2ha_3 u_{i1} + b_3 u_{i2} & (3) \\ b_4 u_{i,m+1} = 2hc_4(x_i,t) - 2ha_4 u_{im} + b_4 u_{i,m-1} & (4) \end{cases}$$

解出$u_{i0}, u_{i,m+1}$ 结果会是带形矩阵的方程式。

C. 计算结果

运行结果见压缩包中的figure文件夹中的动画2dDiffusion.mp4，以下只展示截图

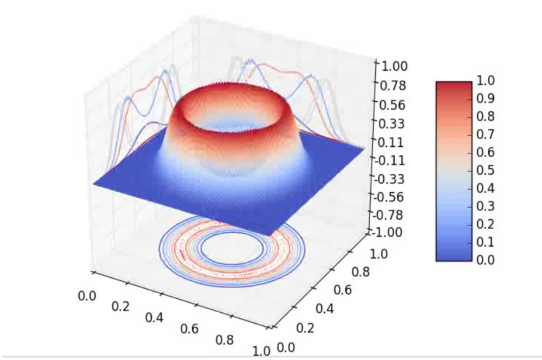

Figure 5: *2*维扩散方程动画瞬时截图

Appendices

A. 使用Crank-Nicolson方法求解一维扩散方程

Here is the program to solve the one dimensional diffusion equation by Crank-Nicolson method in Python programming language.

Input Python source:

```python
"""
This program solves the heat equation
    u_t = u_xx
with dirichlet boundary condition
    u(0,t) = u(1,t) = 0
with the Initial Conditions
    u(x,0) = exp(-20*(x-0.5)**2)-exp(-20*(x-1.5)**2)-exp(-20*(x+0.5)**2)))
over the domain x = [0, 1]

    The program solves the heat equation using a finite difference method
where we use a center difference method in space and Crank-Nicolson in time.
"""
import scipy as sc
import scipy.sparse as sparse
import scipy.sparse.linalg
import numpy as np
import matplotlib.pyplot as plt
# Number of internal points
N=200
# Calculate Spatial Step-Size
h=1/(N+1.0)
# Create Temporal Step-Size, TFinal, Number of Time-Steps
k=h/2
TFinal=1
NumOfTimeSteps=int(TFinal/k)
# Create grid-points on x axis
x=np.linspace(0,1,N+2)
x=x[1:-1]
# Initial Conditions
u=np.transpose(np.mat(np.exp(-20*(x-0.5)**2)-np.exp(-20*(x-1.5)**2)-np.exp(-20*(x+0.5)**2)))
# Second-Derivative Matrix
data=np.ones((3,N))
data[1]=-2*data[1]
diags=[-1,0,1]
D2=sparse.spdiags(data,diags,N,N)/(h**2)
# Identity Matrix
I=sparse.identity(N)
# Data for each time-step
data=[]
for i in range(NumOfTimeSteps):
    # Solve the System: (I-k/2*D2) u_new=(I+k/2*D2)*u_old
    A=(I-k/2*D2)
    b=(I+k/2*D2)*u
    u=np.transpose(np.mat(sparse.linalg.spsolve(A,b)))
    # Save Data
    data.append(u)
# Define of the Create Movie Function
def CreateMovie(plotter,numberOfFrames,fps=10):
    import os,sys
    import matplotlib.pyplot as plt

    for i in range(numberOfFrames):
        plotter(i)
        fname='_tmp%05d.png'%i
```

```

        plt.savefig(fname)
        plt.clf()
# Define the Frame Speed and Movie Length
FPS=60
MovieLength=10
# Function to plot any given Frame
def plotFunction(frame):
    plt.plot(x,data[int(NumOfTimeSteps*frame/(FPS*MovieLength))])
    plt.axis((0,1,0,1.0))
# Generate the movie
CreateMovie(plotFunction,int(MovieLength*FPS),FPS)
```

B. 一维扩散方程动画

Here is the program to merge png pictures together to gif animation in Python programming language.

Input Python source:

```
import matplotlib.pyplot as plt
import os,imageio
images = []
filenames=sorted((fn for fn in os.listdir('.') if fn.endswith('.png')))
for filename in filenames:
    images.append(imageio.imread(filename))
imageio.mimsave('1dDiffusion.gif', images,duration=1/60)
```

C. 求解二维扩散方程

Here is the program to two dimensional diffusion equation in Python programming language.

Input Python source:

```
import scipy as sp
import time
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.animation as animation

dx=0.01
dy=0.01
a=0.5
timesteps=500
t=0.0

nx = int(1/dx)
ny = int(1/dy)

dx2=dx**2
dy2=dy**2

dt=dx2*dy2/(2*a*(dx2+dy2))

ui = sp.zeros([nx,ny])
u = sp.zeros([nx,ny])

for i in range(nx):
```

```
    for j in range(ny):
        if (((i*dx-0.5)**2+(j*dy-0.5)**2 <= 0.1)
            &((i*dx-0.5)**2+(j*dy-0.5)**2>=0.05)):
                ui[i,j] = 1
def evolve_ts(u, ui):
    u[1:-1,1:-1] = ui[1:-1,1:-1]+a*dt*(
                    (ui[2:,1:-1]-2*ui[1:-1,1:-1]+ui[:-2,1:-1])/dx2+
                    (ui[1:-1,2:]-2*ui[1:-1,1:-1]+ui[1:-1,:-2])/dy2)

def data_gen(framenumber,Z,surf):
    global u
    global ui
    evolve_ts(u,ui)
    ui[:]=u[:]
    Z=ui

    ax.clear()
    plotset()
    surf=ax.plot_surface(X,Y,Z,rstride=1,cstride=1,cmap=cm.coolwarm,
                       linewidth=0,antialiased=False,alpha=0.7)
    return surf,

fig=plt.figure()
ax=fig.add_subplot(111, projection='3d')

X = sp.arange(0,1,dx)
Y = sp.arange(0,1,dy)
X,Y= sp.meshgrid(X,Y)

Z = ui

def plotset():
    ax.set_xlim3d(0.0,1.0)
    ax.set_ylim3d(0.0,1.0)
    ax.set_zlim3d(-1.0,1.0)
    ax.set_autoscalez_on(False)
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
    cset=ax.contour(X,Y,Z,zdir='x',offset=0.0,cmap=cm.coolwarm)
    cset=ax.contour(X,Y,Z,zdir='y',offset=1.0,cmap=cm.coolwarm)
    cset=ax.contour(X,Y,Z,zdir='z',offset=-1.0,cmap=cm.coolwarm)

plotset()
surf=ax.plot_surface(X,Y,Z,rstride=1,cstride=1,cmap=cm.coolwarm,
                       linewidth=0,antialiased=False,alpha=0.7)


fig.colorbar(surf,shrink=0.5,aspect=5)

ani=animation.FuncAnimation(fig,data_gen,fargs=(Z,surf),frames=1000,interval=30,blit=False)
ani.save("2dDiffusion.mp4", bitrate=1024)
```

参考文献

[1] Tuncer Cebeci. Convective Heat Transfer. Springer. 2002. ISBN 0-9668461-4-1.

[2] Crank, J.; Nicolson, P. A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type. Proc. Camb. Phil. Soc. 1947, 43 (1): 50–67. doi:10.1007/BF02127704.

[3] Thomas, J. W. Numerical Partial Differential Equations: Finite Difference Methods. Texts in Applied Mathematics 22. Berlin, New York: Springer-Verlag. 1995. ISBN 978-0-387-97999-1.. Example 3.3.2 shows that Crank–Nicolson is unconditionally stable when applied to$u_t = au_{xx}$

[4] Wilmott, P.; Howison, S.; Dewynne, J. The Mathematics of Financial Derivatives: A Student Introduction. Cambridge Univ. Press. 1995. ISBN 0-521-49789-2.