计算天文期中报告

Student Xin Li, Junbao Ni, Zhiyuan Zheng
Student ID DZ20260004, DZ21260006 , DZ20260012

Q

Lecture 03 P64

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{1}$$

1 试任找若干种差分格式对模型方程及一定的初条进行模拟，并对不同格式下的结果进行比较。

2 找一种格式分析其稳定性条件。

3 找一种格式分析其精度。

A

1. 我们使用 Python 求解在波速大于 0($c > 0$) 情况下的单波方程。用代码实现了：a. 迎风格式，b. Lax-Friedrichs 格式，c. Lax-Wendroff 格式, d. 跳点格式, e. 全隐格式, f. Crack Nicholson 格式. 对该方程的数值求解。代码见附录：Code in Python，结果比较图见 PPT.

2. 我们以时间向前，空间中心差分格式为例，详细推导其 von Neumann 稳定性分析的过程。任意差分方程的本征模为：

$$u_j^n = \xi^n e^{ikj\Delta x} \tag{2}$$

若存在某一波数 k 使 $|\xi(k)| > 1$，则差分方程不稳定。时间向前，空间中心差分格式为：

$$u_j^{n+1} = u_j^n - \frac{c\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n) \tag{3}$$

将方程（2）带入方程（3）可得：

$$u_j^{n+1} = \xi^{n+1} e^{ikj\Delta x} \tag{4}$$

$$u_j^n = \xi^n e^{ikj\Delta x} \tag{5}$$

$$u_{j+1}^n = \xi^n e^{ik(j+1)\Delta x} \tag{6}$$

$$u_{j-1}^n = \xi^n e^{ik(j-1)\Delta x} \tag{7}$$

联立方程（3）-（7），并化简可得：

$$\xi^{n+1} e^{ijk\Delta x} = \xi^n e^{ijk\Delta x}\left[1 - \frac{c\Delta t}{2\Delta x}(e^{ik\Delta x} - e^{-ik\Delta x})\right] \tag{8}$$

也即：

$$\xi(k) = 1 - ic\frac{\Delta t}{\Delta x} sin(k\Delta x) \tag{9}$$

由（9）式可知，对时间向前，空间中心差分格式而言 $|\xi(k)| > 1$ 恒成立，因此无论怎么划分空间，时间格点，该格式均不稳定。

3. 我们使用 von Neumann 稳定性分析对对迎风格式和 Lax-Wendroff 格式的精度进行定性分析，遵循 2 中的推导过程。首先对迎风格式，其增长因子为：

$$\xi(k) = 1 - ar(1 - e^{-ik\Delta x}) \tag{10}$$

其中 $r = \frac{\Delta t}{\Delta x}$，由此可以计算增长因子的模为：

$$|\xi(k)| = sqrt(1 - 4ar(1 - ar)sin^2(\frac{k\Delta x}{2})) \tag{11}$$

辐角为：

$$arg(\xi(k)) = -tan^{-1}\left[\frac{arsin(k\Delta x)}{1 - ar + arcos(k\Delta x)}\right] = -ark\Delta x\left[1 - \frac{1}{6}(1 - ar)(1 - 2ar)(k\Delta x)^2 + ...\right] \tag{12}$$

对 Lax-Wendroff 格式，可以计算其增长因子为：

$$\xi(k) = 1 - iarsin(k\Delta x) - 2a^2r^2sin^2(\frac{k\Delta x}{2}) \tag{13}$$

由此可以计算增长因子的模为：

$$|\xi(k)| = 1 - 4a^2r^2(1 - a^2r^2)sin^4(\frac{k\Delta x}{2}) \tag{14}$$

辐角为：

$$arg(\xi(k)) = -tan^{-1}\left[\frac{arsin(k\Delta x)}{1 - 2a^2r^2sin^2(\frac{k\Delta x}{2})}\right] = -ark\Delta x\left[1 - \frac{1}{6}(1 - a^2r^2)(k\Delta x)^2 + ...\right] \tag{15}$$

增长因子的模和辐角则分别代表了该格式的耗散和色散。在 Lax-Wendroff 格式中，振幅衰减依旧存在，但每步的误差为 $O(\Delta x^4)$，相比于迎风格式中的误差阶数 $O(\Delta x^2)$，有很大的改善。而代价在于 Lax-Wendroff 格式存在相位延迟，虽然两种格式的相位误差都为 $O(\Delta x^2)$，但当波数取某些特定值（$ar = 1/2$）时，迎风格式的相位误差会穿过零点，即通过特殊的空间，时间格点选取可以避免在迎风格式中出现相位误差，而对 Lax-Wendroff 格式则不存在特殊的时间，空间格点选取，其相位误差恒为 $O(\Delta x^2)$。

Code in Python

```python
# packages and settings
import numpy as np
import time
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import animation
import warnings
warnings.filterwarnings("ignore")

# different methods
def upwind(r,X,T,U_init,U_left):
    U = np.zeros((len(X),len(T)))
    U[:,0] = U_init
    U[0,:] = U_left
    for n in np.arange(len(T)-1):
        U[1:, n+1] = U[1:, n] - r*(U[1:, n] - U[0:-1,n])
    return U
```

```python
def Lax_Friedrichs(r,X,T,U_init,U_left,U_right):
    U = np.zeros((len(X),len(T)))
    U[:,0] = U_init
    U[0,:] = U_left
    U[-1,:]= U_right
    for n in np.arange(len(T)-1):
        U[1:-1,n+1] = 0.5*(U[2:,n]+U[:-2,n])-0.5*r*(U[2:,n]-U[:-2,n])
    return U

def Lax_Wendroff(r,X,T,U_init,U_left,U_right):
    U = np.zeros((len(X),len(T)))
    U[:,0] = U_init
    U[0,:] = U_left
    U[-1,:]= U_right
    for n in np.arange(len(T)-1):
        U[1:-1,n+1]=U[1:-1,n]-0.5*r*(U[2:,n]-U[:-2,n])+0.5*r**2*(U[2:,n]-2*U[1:-1,n]+U[:-2,n])
    return U


def hopscotch(r,X,T,U_init,U_left,U_right):
    U = np.zeros((len(X),len(T)))
    U[:,0] = U_init
    U[0,:] = U_left
    for n in np.arange(len(T)-1):
        for i in np.arange(len(X)-2):
            if (n+i+1)%2==0:
                U[i+1,n+1] = U[i+1,n]-0.5*r*(U[i+2,n]-U[i,n])
        for i in np.arange(len(X)-2):
            if (n+i+1)%2!=0:
                U[i+1,n+1] = U[i+1,n]-0.5*r*(U[i+2,n+1]-U[i,n+1])
    return U


def Thomas(La, Mb, Uc, b):
    """
    Arguments:
        La -- [lower item for tri-diagonal matrix]
        Mb -- [mian item for tri-diagonal matrix]
        Uc -- [upper item for tri-diagonal matrix]
        b -- [AX = b, where A is the tri-diagonal matrix]
    """
    n = len(Mb)
    Uc[0] = Uc[0] / Mb[0]
    for i in range(1, n-1):
        Uc[i] = Uc[i] / (Mb[i] - La[i - 1] * Uc[i - 1])
    b[0] = b[0] / Mb[0]
    for i in range(1, n):
        b[i] = (b[i] - La[i-1]*b[i-1]) / (Mb[i] - La[i-1] * Uc[i-1])
    ls = list(range(n-1))[::-1]
    for i in ls:
        b[i] = b[i] - Uc[i]*b[i+1]
    return np.array(b)
```

```
def full_implicate(r,X,T,U_init,U_left,U_right):
    U = np.zeros((len(X),len(T)))
    U[:,0]  = U_init
    U[0,:]  = U_left
    U[-1,:]= U_right

    for n in range(len(T)-1):
        a = -0.5*r*np.ones((len(X)-3))#给出矩阵三条对角线的值
        b = np.ones((len(X)-2))
        C = 0.5*r*np.ones((len(X)-3))
        k = []
        for i in range(len(X)-2):
            k.append(U[i+1,n])

        k[0]   += 0.5*r*U[0,n+1]
        k[-1]  -= 0.5*r*U[-1,n+1]#计算Ax=b中的b

        U[1:-1,n+1] = Thomas(a,b,C,k)
    return U

def Crack_Nicholson(r,X,T,U_init,U_left,U_right):

    U = np.zeros((len(X),len(T)))
    U[:,0]  = U_init
    U[0,:]  = U_left
    U[-1,:]= U_right

    for n in range(len(T)-1):
        a = -0.25*r*np.ones((len(X)-3))
        b = np.ones((len(X)-2))
        C = 0.25*r*np.ones((len(X)-3))
        k = []
        for i in range(len(X)-2):
            k.append(U[i+1,n]-0.25*r*(U[i+2,n]- U[i,n]))

        k[0]   += 0.25*r*U[ 0,n+1]
        k[-1]  -= 0.25*r*U[-1,n+1]
        U[1:-1,n+1] = Thomas(a,b,C,k)
    return U

def solve(r,X,T,U_init,U_left,U_right,method):
    if method == 'upwind':
        U = upwind(r,X,T,U_init,U_left)
    elif method == 'Lax_Friedrichs':
        U = Lax_Friedrichs(r,X,T,U_init,U_left,U_right)
    elif method == 'Lax_Wendroff':
        U = Lax_Wendroff(r,X,T,U_init,U_left,U_right)
    elif method == 'hopscotch':
        U = hopscotch(r,X,T,U_init,U_left,U_right)
    elif method == 'full_implicate':
        U = full_implicate(r,X,T,U_init,U_left,U_right)
```

```
    elif method == 'Crack_Nicholson':
        U = Crack_Nicholson(r,X,T,U_init,U_left,U_right)
    return U

methods = np.array(['upwind','Lax_Friedrichs','Lax_Wendroff','hopscotch','full_implicate','Crack_Nicholson'])
[xl,xr] = [0,10]
[ti,tf] = [0,10]

# solving & plotting
## Case 1: 谐波

def analytic_function(x,t):
    return np.sin(2*np.pi*(x-c*t))

dx = 0.1
dt = 0.05
c  = 1
r  = c*dt/dx
T = np.arange(ti, tf+dt, dt)
X = np.arange(xl, xr+dx, dx)

U_init = analytic_function(X,ti)
U_left = analytic_function(xl,T)
U_right= analytic_function(xr,T)
tt,xx = np.meshgrid(T,X)
U_a = analytic_function(xx,tt)

for m, method in enumerate(methods):

    U_n = solve(r,X,T,U_init,U_left,U_right,method)

    fig = plt.figure(figsize = (6,4.5),dpi=100)
    plt.grid(ls='--')
    ani_a, = plt.plot(X,U_a[:,0],color='red', ls='-' ,label='analytic solution')
    ani_n, = plt.plot(X,U_n[:,0],color='blue',ls='--',label=method)
    plt.xlabel("X", fontsize = 16)
    plt.ylabel("Amptitude", fontsize = 16)
    plt.legend(loc='upper right',fontsize=8)
    text_ani = plt.text(0.05, 1, '',fontsize=12,color='black')
    plt.ylim(-1.8,1.8)

    def update(n):
        ani_a.set_data(X,U_a[:,n])
        ani_n.set_data(X,U_n[:,n])
        ti = dt*n
        text_ani.set_text('t=%.3f'%ti)
        return [ani_a,ani_n,text_ani]
    ani = animation.FuncAnimation(fig=fig, func = update, frames = np.arange(0,len(T)), interval = 100)
    ani.save('image/case1/'+method+'.gif')
    plt.show()

## Case 2: 矩形波，锯齿波，高斯波
```

```
def analytic_function(x,c,t):
    D = 0.4
    return np.exp(-(x- c * t - 7 ) ** 2 / D) \
        + np.heaviside(x - c * t- 2, 0.5) * np.heaviside(-(x- c * t - 3 ), 0.5) \
        + np.minimum(np.maximum(x - c * t - 4, 0), np.maximum(-(x- c * t - 5 ), 0)) * 2

U_init = analytic_function(X,c,0)
U_left = analytic_function(xl,c,T)
U_right= analytic_function(xr,c,T)
tt,xx = np.meshgrid(T,X)
U_a = analytic_function(xx,c,tt)


for m, method in enumerate(methods):

    U_n = solve(r,X,T,U_init,U_left,U_right,method)

    fig = plt.figure( figsize = (6,4.5),dpi=100)
    plt.grid(ls='--')
    ani_a, = plt.plot(X,U_a[:,0],color='red', ls='-' ,label='analytic solution')
    ani_n, = plt.plot(X,U_n[:,0],color='blue',ls='--',label=method)
    plt.xlabel("X", fontsize = 16)
    plt.ylabel("Amptitude", fontsize = 16)
    plt.legend(loc='upper right',fontsize=8)
    text_ani = plt.text(0.05, 1, '',fontsize=12,color='black')
    plt.ylim(-1.2,1.8)

    def update(n):
        ani_a.set_data(X,U_a[:,n])
        ani_n.set_data(X,U_n[:,n])
        ti = dt*n
        text_ani.set_text('t=%.3f'%ti)
        return [ani_a,ani_n,text_ani]
    ani = animation.FuncAnimation(fig=fig, func = update, frames = np.arange(0,len(T)), interval = 80)
    ani.save('image/case2/'+method+'.gif')
    plt.show()
```