# 数值天气预报课程作业

吴浩杰 20221170212

## 1 实习五

一维线性平流方程的两时间层积分方案 1

### 1.1 实习目的

通过编程实现一维线性平流方程的离散化，使学生掌握前差格式的计算方法以及前差格式稳定性分析的方法。

### 1.2 实习内容

(1) 采用如下初始条件 u(x,0)=sin( x),0 x 1 c=1.5，dx=0.05，dt=0.004 设置一维线性平流方程的边界条件。根据上述边界条件，设置 x 方向格点数 (nlon)、x 方向格距 (dx)、积分时间步长 dt 和常数 c 的值，对运动速度进行初始化。

(2) 采用前差格式积分线性平流方程，空间差分采用中央差格式。循环积分所有格点上的运动速度 u，并求出相应的动能。

(3) 分析差分格式的稳定性。

Listing 1: 实习内容

```
import numpy as np
import matplotlib.pyplot as plt
from 实习内容 import
    magnification_factor_and_coriolis_parameter
import numpy as np
import matplotlib.pyplot as plt

# 设置matplotlib支持中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示
    中文标签
plt.rcParams['axes.unicode_minus'] = False    # 用来正常显示
    负号

```

```
11      def time_integration_forward(u0, c, dx, dt, nsteps):
12          """
13          显式前向差分（时间）+ 中心差分（空间）积分一维线性平流方
                程
14           u/ t + c  u/ x = 0
15          边界条件：固定边界 u[0]=u[0], u[-1]=u[-1]
16
17          输入：
18            u0        -- 初始一维场，长度 nlon
19            c         -- 平流速度常数
20            dx        -- 空间步长
21            dt        -- 时间步长
22            nsteps    -- 积分步数
23          返回：
24            u         -- 始终保存所有时刻的 u，shape=(nlon, nsteps)
25            uk        -- 每步的总动能 time series，shape=(nsteps,)
26          """
27          nlon = u0.size
28          u = np.zeros((nlon, nsteps))
29          uk = np.zeros(nsteps)
30
31          # 初始条件
32          u[:, 0] = u0
33          uk[0] = np.sum(u[:, 0]**2) / 2.0
34
35          # 时间积分
36          for n in range(1, nsteps):
37              # 前向 Euler 时间 + 中心差分空间
38              for i in range(1, nlon - 1):
39                  u[i, n] = (u[i, n-1]
40                             - 0.5 * c * dt / dx
41                             * (u[i+1, n-1] - u[i-1, n-1]))
42              # 固定边界条件
43              u[0, n]  = u[0,   n-1]
44              u[-1, n] = u[-1,  n-1]
45
46              # 累计动能
47              uk[n] = np.sum(u[:, n]**2) / 2.0
48
49          return u, uk
50
51      def main():
52          # 参数设置
```

```
    nlon  = 20        # 空间格点数
    ntime = 3000      # 时间步数
    dx    = 0.05      # 空间步长
    dt    = 0.004     # 时间步长
    c     = 1.5       # 平流速度

    # 初始场 u(x,0) = sin(pi x), x = i*dx, i=1..nlon
    x  = np.arange(1, nlon+1) * dx
    u0 = np.sin(np.pi * x)

    # 时间积分
    u, uk = time_integration_forward(u0, c, dx, dt, ntime)

    # 绘图：动能随积分步数变化
    plt.figure(figsize=(6, 4))
    plt.plot(np.arange(ntime), uk, '--k', linewidth=1.5)
    plt.ylim(0, 200)

    # 坐标轴标签和标题，设置字体大小
    plt.xlabel('积分步数', fontsize=14)
    plt.ylabel(r'动能 $\sum_i u_i^2/2$', fontsize=14)
    plt.title('显式前向+中心差分方案动能演变', fontsize=16)

    # 刻度字体大小
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    # 添加网格和加粗坐标轴线
    plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
    ax = plt.gca()
    for spine in ax.spines.values():
        spine.set_linewidth(1.2)

    plt.tight_layout()
    plt.show()

if __name__ == '__main__':
    main()
```

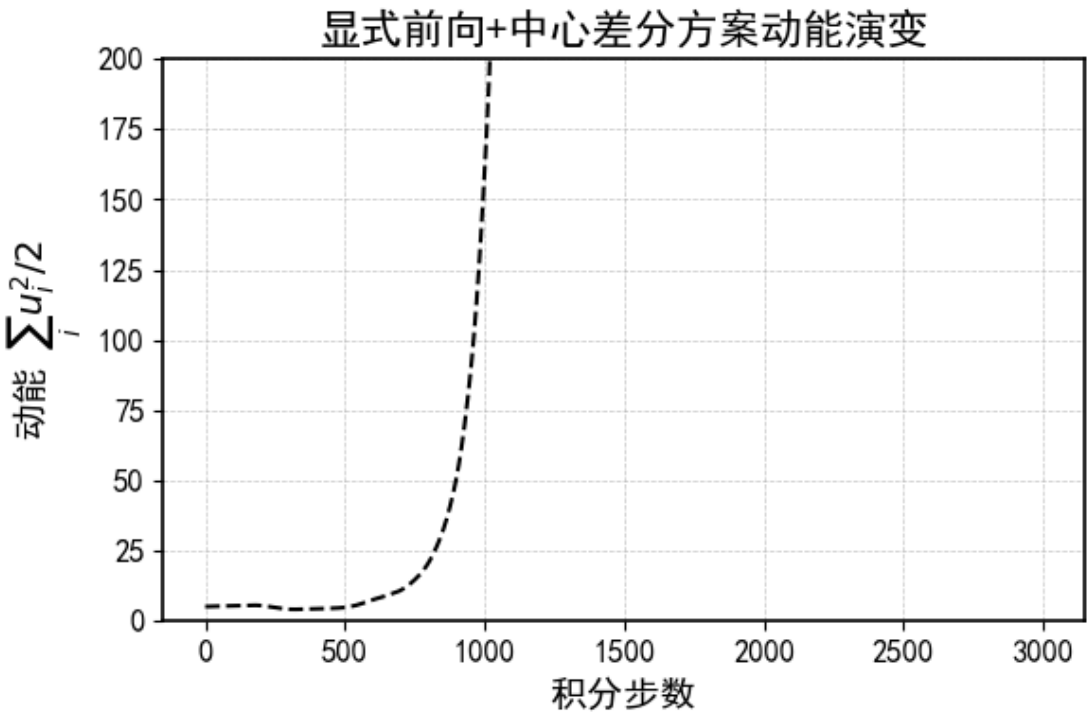


Figure 1: 实习内容

## 1.3 习题一

分析时间层采用后差格式，空间层分别采用中央差分和前差格式的一维线性平流差分方程的稳定性。

Listing 2: 习题一

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

# 设置matplotlib支持中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False    # 用来正常显示负号

def time_integration_backward_central(u0, c, dx, dt, nsteps):
    """
    时间层后差 + 空间层中央差分格式积分一维线性平流方程
     u/ t + c  u/ x = 0
    边界条件：固定边界 u[0]=u[0], u[-1]=u[-1]
    """
    nlon = u0.size
```

```
15          u = np.zeros((nlon, nsteps))
16          uk = np.zeros(nsteps)
17
18          # 初始条件
19          u[:, 0] = u0
20          uk[0] = np.sum(u[:, 0]**2) / 2.0
21
22          # 时间积分
23          for n in range(1, nsteps):
24              # 后差时间 + 中心差分空间
25              for i in range(1, nlon - 1):
26                  u[i, n] = u[i, n-1] - 0.5 * c * dt / dx * (u[i
                        +1, n] - u[i-1, n])
27
28              # 固定边界条件
29              u[0, n] = u[0, n-1]
30              u[-1, n] = u[-1, n-1]
31
32              # 累计动能
33              uk[n] = np.sum(u[:, n]**2) / 2.0
34
35          return u, uk
36
37      def time_integration_backward_forward(u0, c, dx, dt, nsteps)
            :
38          """
39          时间层后差 + 空间层前差分格式积分一维线性平流方程
40           u/ t + c  u/ x = 0
41          边界条件: 固定边界 u[0]=u[0], u[-1]=u[-1]
42          """
43          nlon = u0.size
44          u = np.zeros((nlon, nsteps))
45          uk = np.zeros(nsteps)
46
47          # 初始条件
48          u[:, 0] = u0
49          uk[0] = np.sum(u[:, 0]**2) / 2.0
50
51          # 时间积分
52          for n in range(1, nsteps):
53              # 后差时间 + 前差分空间
54              for i in range(1, nlon - 1):
55                  u[i, n] = u[i, n-1] - c * dt / dx * (u[i+1, n] -
```

```
                    u[i, n])
56
57              # 固定边界条件
58              u[0, n] = u[0, n-1]
59              u[-1, n] = u[-1, n-1]
60
61              # 累计动能
62              uk[n] = np.sum(u[:, n]**2) / 2.0
63
64          return u, uk
65
66      def main():
67          # 参数设置
68          nlon = 20          # 空间格点数
69          ntime = 3000       # 时间步数
70          dx = 0.05          # 空间步长
71          dt = 0.004         # 时间步长
72          c = 1.5            # 平流速度
73
74          # 初始场 u(x,0) = sin(pi x), x = i*dx, i=1..nlon
75          x = np.arange(1, nlon+1) * dx
76          u0 = np.sin(np.pi * x)
77
78          # 时间积分
79          u1, uk1 = time_integration_backward_central(u0, c, dx,
                dt, ntime)
80          u2, uk2 = time_integration_backward_forward(u0, c, dx,
                dt, ntime)
81
82          # 创建图形窗口
83          plt.figure(figsize=(16, 5))
84
85          # 绘制中央差分结果（只显示前35步）
86          plt.subplot(1, 2, 1)
87          plt.plot(np.arange(35), uk1[:35], '--k', linewidth=1.5)
88          plt.ylim(0, 200)
89          plt.xlabel('积分步数', fontsize=14)
90          plt.ylabel(r'动能 $\sum_i u_i^2/2$', fontsize=14)
91          plt.title('时间层后差，空间层中央差分', fontsize=16)
92          plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
93
94          # 绘制前差分结果
95          plt.subplot(1, 2, 2)
```

```
        plt.plot(np.arange(ntime), uk2, '--k', linewidth=1.5)
        plt.ylim(0, 10)
        plt.xlabel('积分步数', fontsize=14)
        plt.ylabel(r'动能 $\sum_i u_i^2/2$', fontsize=14)
        plt.title('时间层后差，空间层前差分', fontsize=16)
        plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)

        # 调整布局并显示
        plt.tight_layout()
        plt.show()

if __name__ == '__main__':
        main()
```

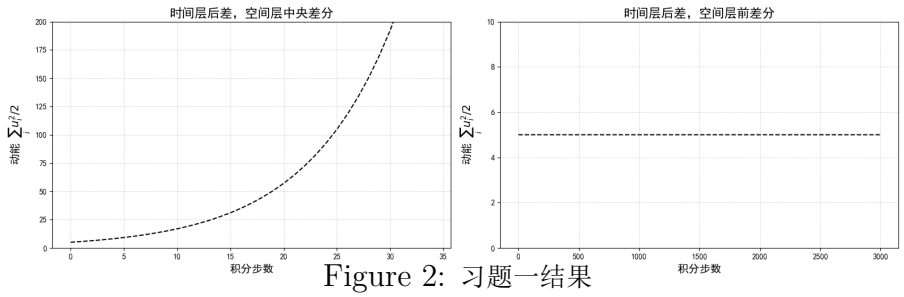


Figure 2: 习题一结果

可以看出，中央差依然是不稳定的，计算溢出非常快。而空间层前差是稳定的，动能并不会随积分步数的增加而增加。

Listing 3: 习题二

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

# 设置matplotlib支持中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False    # 用来正常显示负号

def time_integration_backward_central(u0, c, dx, dt, nsteps):
    """
    时间层后差 + 空间层中央差分格式积分一维线性平流方程
     u/ t + c  u/ x = 0
    边界条件：固定边界 u[0]=u[0], u[-1]=u[-1]
    """
    nlon = u0.size
    u = np.zeros((nlon, nsteps))
```

```
16          uk = np.zeros(nsteps)
17
18          # 初始条件
19          u[:, 0] = u0
20          uk[0] = np.sum(u[:, 0]**2) / 2.0
21
22          # 时间积分
23          for n in range(1, nsteps):
24              # 后差时间 + 中心差分空间
25              for i in range(1, nlon - 1):
26                  u[i, n] = u[i, n-1] - 0.5 * c * dt / dx * (u[i
                        +1, n] - u[i-1, n])
27
28              # 固定边界条件
29              u[0, n] = u[0, n-1]
30              u[-1, n] = u[-1, n-1]
31
32              # 累计动能
33              uk[n] = np.sum(u[:, n]**2) / 2.0
34
35          return u, uk
36
37      def time_integration_backward_forward(u0, c, dx, dt, nsteps)
            :
38          """
39          时间层后差 + 空间层前差分格式积分一维线性平流方程
             u/ t + c  u/ x = 0
41          边界条件: 固定边界 u[0]=u[0], u[-1]=u[-1]
42          """
43          nlon = u0.size
44          u = np.zeros((nlon, nsteps))
45          uk = np.zeros(nsteps)
46
47          # 初始条件
48          u[:, 0] = u0
49          uk[0] = np.sum(u[:, 0]**2) / 2.0
50
51          # 时间积分
52          for n in range(1, nsteps):
53              # 后差时间 + 前差分空间
54              for i in range(1, nlon - 1):
55                  u[i, n] = u[i, n-1] - c * dt / dx * (u[i+1, n] -
                        u[i, n])
```

```
56
57              # 固定边界条件
58              u[0, n] = u[0, n-1]
59              u[-1, n] = u[-1, n-1]
60
61              # 累计动能
62              uk[n] = np.sum(u[:, n]**2) / 2.0
63
64          return u, uk
65
66      def main():
67          # 参数设置
68          nlon = 20          # 空间格点数
69          ntime = 3000       # 时间步数
70          dx = 0.05          # 空间步长
71          dt = 0.004         # 时间步长
72          c = 1.5            # 平流速度
73
74          # 初始场 u(x,0) = sin(pi x) + 1.5, x   [0,1]
75          x = np.linspace(0, 1, nlon)
76          u0 = np.sin(np.pi * x) + 1.5
77
78          # 时间积分
79          u1, uk1 = time_integration_backward_central(u0, c, dx,
                dt, ntime)
80          u2, uk2 = time_integration_backward_forward(u0, c, dx,
                dt, ntime)
81
82          # 创建图形窗口
83          plt.figure(figsize=(16, 5))
84
85          # 绘制中央差分结果（只显示前9步）
86          plt.subplot(1, 2, 1)
87          plt.plot(np.arange(9), uk1[:9], '--k', linewidth=1.5)
88          plt.ylim(0, 200)
89          plt.xlabel('积分步数', fontsize=14)
90          plt.ylabel(r'动能 $\sum_i u_i^2/2$', fontsize=14)
91          plt.title('时间层后差，空间层中央差分', fontsize=16)
92          plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
93
94          # 绘制前差分结果（显示所有3000步）
95          plt.subplot(1, 2, 2)
96          plt.plot(np.arange(ntime), uk2, '--k', linewidth=1.5)
```

```
        plt.ylim(0, 200)  # 修改为与左图相同的范围
        plt.xlabel('积分步数', fontsize=14)
        plt.ylabel(r'动能 $\sum_i u_i^2/2$', fontsize=14)
        plt.title('时间层后差，空间层前差分', fontsize=16)
        plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)

        # 调整布局并显示
        plt.tight_layout()
        plt.show()

if __name__ == '__main__':
    main()
```

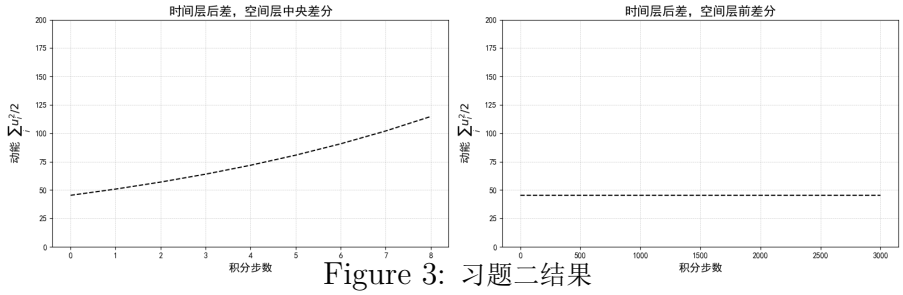


Figure 3: 习题二结果

首先在稳定度上，中央差仍然不稳定，前差仍然稳定；其次相比习题 (1) 可以发现初始条件的改变对最终积分结果会有影响，习题 (2) 所得结果相比习题 (1) 同等积分步长动能均有所增大。