数值天气预报课程作业

吴浩杰 20221170212

1 实习六

一维线性平流方程的两时间层积分方案 2

1.1 实习目的

通过编程实现用欧拉后差格式积分一维线性平流方程，使学生掌握欧拉后差格式的计算方法以及欧拉后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：

$$u(x,0) = \sin(\pi x), \quad 0 \le x \le 1 \tag{1}$$

其中，平流速度 $c = 1.5$，空间步长 $dx = 0.05$，时间步长 $dt = 0.004$。空间差分采用中央差格式，用欧拉后差格式积分线性平流方程。其中，速度 u 的初始化过程和 5.4 节相同，时间积分公式采用欧拉后差格式（6.47），循环积分所有格点的速度 u，并求出相应的 u 风动能 $\frac{(u_i^n)^2}{2}$。

(2) 分析欧拉后差格式的稳定性。

Listing 1: 实习内容

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

# 让 Matplotlib 正确显示中文和负号
plt.rcParams["font.sans-serif"] = ["SimHei"]        # 显示中文
plt.rcParams["axes.unicode_minus"] = False          # 正确显示负号

def time_integration(u, c, dx, dt, difference_scheme="backward")
    :
    """
```

```
    用欧拉格式（显式）或欧拉后插 / Lax-Wendroff 格式积分一维线性平流方程，
    并计算每步累计动能 uk。

    参数
    ----
    u : 2-D ndarray, shape (nlon, ntime)
        储存积分结果，u[:,0] 必须已填入初值。
    c : float
        平流速度
    dx : float
        空间分辨率
    dt : float
        时间步长
    difference_scheme : {'forward','backward'}
        'forward' - 前向欧拉 + 空间中心差分
        'backward' - 欧拉后插 / Lax-Wendroff (与您给出的 MATLAB 公式一致)

    返回
    ----
    u  : ndarray  同输入，积分后的场
    uk : 1-D ndarray, shape (ntime,)
        每一步全场动能的累加值
    """
    nlon, ntime = u.shape
    uk = np.zeros(ntime)
    uk[0] = 0.5 * np.sum(u[:, 0] ** 2)

    if difference_scheme.lower() == "forward":
        for step in range(1, ntime):
            for i in range(1, nlon - 1):
                u[i, step] = (u[i, step - 1]
                              - 0.5 * c * dt / dx
                              * (u[i + 1, step - 1] - u[i - 1,
                                 step - 1]))
                uk[step] += 0.5 * u[i, step] ** 2

            u[0, step] = u[0, step - 1]
            u[-1, step] = u[-1, step - 1]
            uk[step] += 0.5 * (u[0, step] ** 2 + u[-1, step] **
                2)

```

```
    elif difference_scheme.lower() == "backward":
        coef1 = 0.5 * c * dt / dx
        coef2 = coef1 ** 2
        for step in range(1, ntime):
            for i in range(2, nlon - 2):
                u[i, step] = (u[i, step - 1]
                              - coef1 * (u[i + 1, step - 1] - u[
                                  i - 1, step - 1])
                              + coef2 * (u[i + 2, step - 1]
                                         - 2 * u[i, step - 1]
                                         + u[i - 2, step - 1]))
                uk[step] += 0.5 * u[i, step] ** 2

            for k in range(2):
                u[k, step] = u[k, step - 1]
                u[-(k + 1), step] = u[-(k + 1), step - 1]
                uk[step] += 0.5 * (u[k, step] ** 2 + u[-(k + 1),
                    step] ** 2)
    else:
        raise ValueError("difference_scheme must be 'forward' or
            'backward'")

    return u, uk

# -------------------- 主程序 --------------------
if __name__ == "__main__":
    nlon = 20
    dx   = 0.05
    c    = 1.5

    # 初值: u(x,0) = sin(pi x), 其中 x = (1:nlon) * dx
    u0 = np.sin(np.arange(1, nlon + 1) * np.pi * dx)

    # -------- 实验 1 : dt = 0.004, 共 3000 步积分 ----------
    dt1    = 0.004
    ntime1 = 3000
    u1     = np.zeros((nlon, ntime1))
    u1[:, 0] = u0
    _, uk1 = time_integration(u1, c, dx, dt1, "backward")

    # -------- 实验 2 : dt = 0.04, 仅 40 步积分 ----------
    dt2    = 0.04
    ntime2 = 40
```

```
    u2      = np.zeros((nlon, ntime2))
    u2[:, 0] = u0
    _, uk2 = time_integration(u2, c, dx, dt2, "backward")

    # ---------------- 绘图 ----------------
    fig, axes = plt.subplots(1, 2, figsize=(11, 4),
        constrained_layout=True)

    axes[0].plot(np.arange(ntime1), uk1, "--k", linewidth=1.5)
    axes[0].set_ylim(-1, 100)
    axes[0].set_xlabel("积分步数", fontsize=12)
    axes[0].set_ylabel("动能",    fontsize=12)
    axes[0].text(-0.05, 1.05, "(a) dt = 0.004, 3000 步",
        transform=axes[0].transAxes)

    axes[1].plot(np.arange(ntime2), uk2, "--k", linewidth=1.5)
    axes[1].set_ylim(-1, 100)
    axes[1].set_xlabel("积分步数", fontsize=12)
    axes[1].set_ylabel("动能",    fontsize=12)
    axes[1].text(-0.05, 1.05, "(b) dt = 0.04, 40 步", transform=
        axes[1].transAxes)

    plt.show()
```

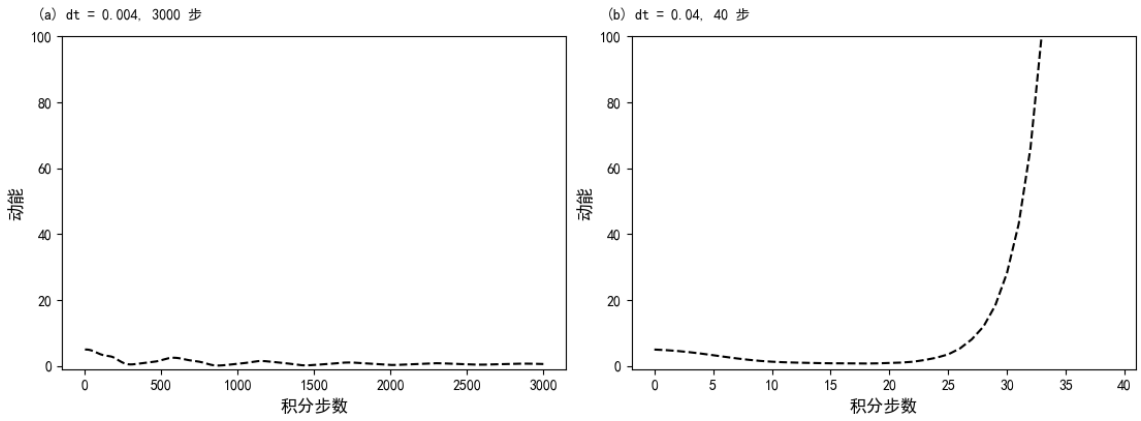

Figure 1: 实习内容

理论分析可知 $|\beta| \leq \frac{1}{2}$ 时差分格式稳定，而 $c = 1.5$，$dx = 0.05$，$dt = 0.004$ 时 $\beta = 0.06$，格式稳定，而 dt 扩大 10 倍时 $\beta = 0.6$，格式不稳定。从积分结果中看与理论分析相同。

1.3 习题一

分析一维线性平流方程的稳定性，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。

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

# ———————————————————— 参数设置 ————————————————————
nlon = 20          # 空间格点数
ntime = 300        # 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

# 无量纲参数
lambda_ = 0.5 * c * dt / dx     # 用于中央差分的系数
r = c * dt / dx                 # 用于前差格式的系数

# ———————————————————— 初始化场和动能数组 ————————————————————
# 初始条件 u(i,0) = sin(i * pi * dx)
x = np.arange(1, nlon+1) * dx
u_central = np.zeros((nlon, ntime))
u_forward = np.zeros((nlon, ntime))
u_central[:, 0] = np.sin(x * np.pi)
u_forward[:, 0] = u_central[:, 0].copy()

# 动能：uk[n] = sum(u[:,n]**2 / 2)
uk_central = np.zeros(ntime)
uk_forward = np.zeros(ntime)
uk_central[0] = np.sum(u_central[:, 0]**2) / 2
uk_forward[0] = uk_central[0]

# ———————————————————— 数值积分主循环
```

```
    # ——————————————————
for n in range(1, ntime):
    # —— 时间层后差，空间层中央差 ——
    u_old = u_central[:, n-1]
    u_new = np.zeros(nlon)
    # 边界不变
    u_new[0] = u_old[0]
    u_new[-1] = u_old[-1]
    # 内部点
    for i in range(1, nlon-1):
        u_new[i] = u_old[i] - lambda_ * (u_old[i+1] - u_old[
            i-1])
    u_central[:, n] = u_new
    uk_central[n] = np.sum(u_new**2) / 2

    # —— 时间层后差，空间层前差 ——
    u_old = u_forward[:, n-1]
    u_new = np.zeros(nlon)
    # 边界不变
    u_new[0] = u_old[0]
    u_new[-1] = u_old[-1]
    # 内部点
    for i in range(1, nlon-1):
        u_new[i] = u_old[i] - r * (u_old[i+1] - u_old[i])
    u_forward[:, n] = u_new
    uk_forward[n] = np.sum(u_new**2) / 2

# ———————————————————— 绘图
    ——————————————————
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(range(ntime), uk_central, '--k', linewidth=1.5)
plt.title("时间层后差，空间层中央差")
plt.xlabel("积分步数")
plt.ylabel("动能 $\\sum u^2/2$")
plt.ylim(0, 10)

plt.subplot(1, 2, 2)
plt.plot(range(40), uk_forward[:40], '--k', linewidth=1.5)
   # 只绘制前40步
plt.title("时间层后差，空间层前差")
plt.xlabel("积分步数")
```

```
    plt.ylabel("动能 $\\sum u^2/2$")
    plt.ylim(0, 100)

    plt.tight_layout()
    plt.show()
```

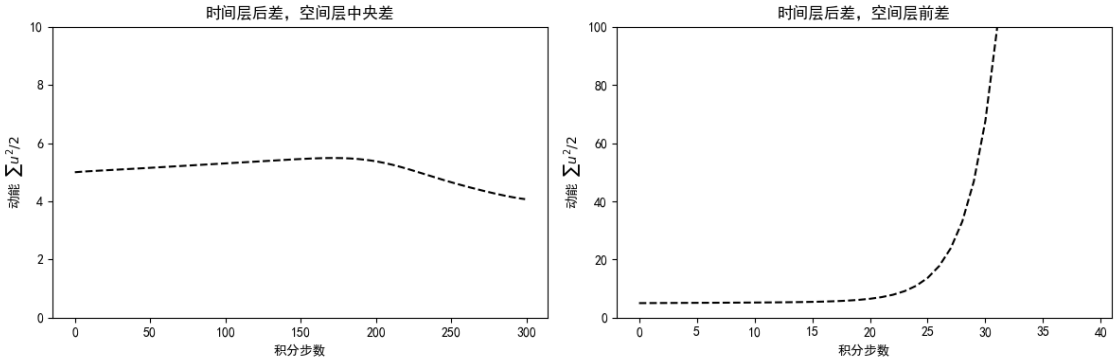

Figure 2: 习题一结果

该习题与实习 5 的习题 1 一致，但认为此次结果正确。此次积分结果与理论分析结果一致，即时间层后差、空间层中央差时差分格式绝对稳定，时间层后差、空间层前差时差分格式不稳定。

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

    # ———————————————————— 参数设置 ————————————————————
    nlon = 20          # 空间格点数
    ntime = 300        # 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

    # ———————————————————— 初始化场和动能数组 ————————————————————
    # 初始化数组
    u = np.zeros((nlon, ntime))
```

```
    # 设置初始条件
    u[:, 0] = np.sin(np.arange(1, nlon+1) * np.pi * dx)
    # 初始化动能数组
    uk = np.zeros(ntime)
    uk[0] = np.sum(u[:, 0]**2 / 2)

    # ———————————————————— 数值积分主循环 ————————————————————
    for step_id in range(1, ntime):
        # 计算内部点
        for lon_id in range(1, nlon-1):
            # Lax-Wendroff格式
            u[lon_id, step_id] = u[lon_id, step_id-1] - \
                0.5 * c * dt / dx * (u[lon_id+1, step_id-1] - u[lon_id-1, step_id-1]) + \
                (c * dt / dx)**2 * (u[lon_id+1, step_id-1] - 2*u[lon_id, step_id-1] + u[lon_id-1, step_id-1])
            uk[step_id] += u[lon_id, step_id]**2 / 2

        # 边界条件保持不变
        u[nlon-1, step_id] = u[nlon-1, step_id-1]
        u[0, step_id] = u[0, step_id-1]

        # 添加边界点的动能贡献
        uk[step_id] += u[nlon-1, step_id]**2 / 2 + u[0, step_id]**2 / 2

    # ———————————————————— 绘图 ————————————————————
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, ntime+1), uk, '--k', linewidth=1.5)
    plt.xlabel('积分步数', fontsize=14)
    plt.ylabel('动能', fontsize=14)
    plt.title('Lax-Wendroff格式', fontsize=14)
    plt.grid(True)
    plt.show()
```

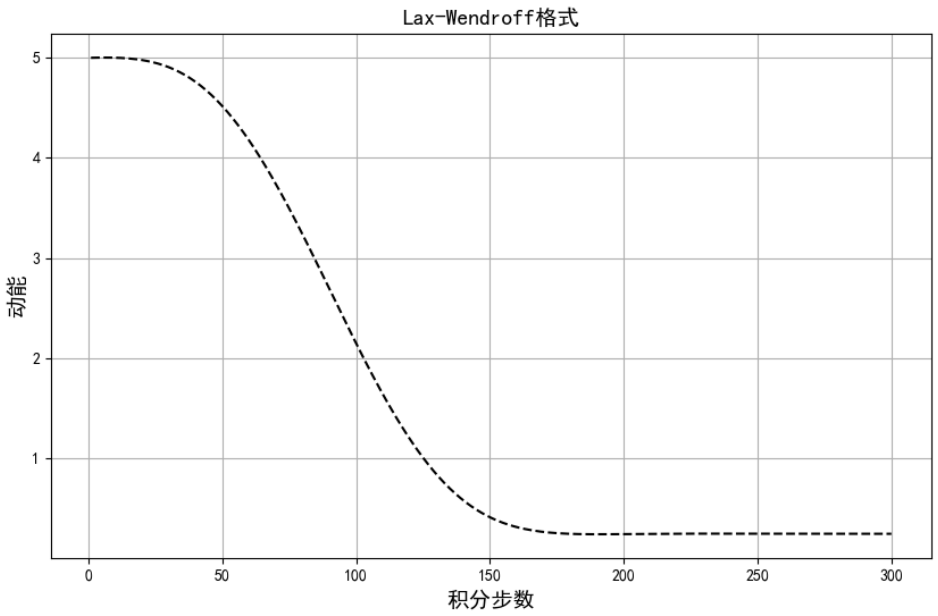

Figure 3: 习题二结果

经过理论分析可知，对于 Lax-Wendroff 差分格式，其放大因子

$$G = 1 + 2\beta^2(\cos k\Delta x - 1) - i\beta \sin k\Delta x \tag{2}$$

其模的平方为

$$\begin{aligned} |G|^2 &= (1-2\beta^2)(1-2\beta^2+4\beta^2\cos k\Delta x) \\ &\quad + \beta^2(4\beta^2-1)\cos^2 k\Delta x \\ &\le (1-2\beta^2)(1-2\beta^2+4\beta^2) + \beta^2(4\beta^2-1) \\ &= 1-\beta^2 \le 1 \end{aligned} \tag{3}$$

故此格式绝对稳定，这与采用与实习同样初始条件的积分结果也相似。