对流方程的数值求解

对于对流方程

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

初始条件分别为

$$u_0(x) = \begin{cases} 1 - \cos(2\pi x) & \text{如果 } x \in [0, 1], \\ 0 & \text{否则,} \end{cases} \tag{2}$$

和

$$u_0(x) = \begin{cases} 1 & \text{如果 } x \in [0.4, 0.6], \\ 0 & \text{否则.} \end{cases} \tag{3}$$

在数值求解中，我们离散化空间和时间，把连续的偏微分方程转换为差分方程。

离散化

设空间步长为 (\Delta x) 和时间步长为 (\Delta t)，网格点 (x_i = i\Delta x) 和 (t_n = n\Delta t)。然后我们使用以下格式：

1. 迎风格式（**Upwind Scheme**）：

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) \tag{4}$$

2. **Lax-Friedrichs** 格式:

$$u_i^{n+1} = \frac{1}{2}(u_{i+1}^n + u_{i-1}^n) - \frac{\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) \tag{5}$$

3. **Lax-Wendroff** 格式：

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) + \frac{\Delta t^2}{2\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \tag{6}$$

4. **Beam-Warming** 格式：

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{2\Delta x}(3u_i^n - 4u_{i-1}^n + u_{i-2}^n) + \frac{\Delta t^2}{2\Delta x^2}(u_i^n - 2u_{i-1}^n + u_{i-2}^n) \tag{7}$$

实现细节

我们将用Python实现这些格式，并比较它们在初始条件下的数值解。

Python代码实现

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

# 设置参数
L = 2.0  # 计算区域长度
T = 1.0  # 计算时间
Nx = 100  # 空间网格数
Nt = 200  # 时间步数
dx = L / Nx  # 空间步长
dt = T / Nt  # 时间步长
x = np.linspace(0, L, Nx+1)  # 空间坐标
alpha = dt / dx  # CFL数

# 初始条件
def initial_condition1(x):
    return np.where((0 <= x) & (x <= 1), 1 - np.cos(2 * np.pi * x),
0)

def initial_condition2(x):
    return np.where((0.4 <= x) & (x <= 0.6), 1, 0)

# 迎风格式
def upwind(u):
    u_new = np.copy(u)
    for n in range(Nt):
        u_new[1:] = u_new[1:] - alpha * (u_new[1:] - u_new[:-1])
        u_new[0] = 0  # 边界条件
    return u_new

# Lax-Friedrichs 格式
def lax_friedrichs(u):
    u_new = np.copy(u)
    for n in range(Nt):
```

```
        u_new[1:-1] = 0.5 * (u_new[2:] + u_new[:-2]) - 0.5 * alpha
* (u_new[2:] - u_new[:-2])
        u_new[0] = 0  # 边界条件
        u_new[-1] = 0  # 边界条件
    return u_new

# Lax-Wendroff 格式
def lax_wendroff(u):
    u_new = np.copy(u)
    for n in range(Nt):
        u_new[1:-1] = u_new[1:-1] - 0.5 * alpha * (u_new[2:] -
u_new[:-2]) + 0.5 * alpha**2 * (u_new[2:] - 2*u_new[1:-1] +
u_new[:-2])
        u_new[0] = 0  # 边界条件
        u_new[-1] = 0  # 边界条件
    return u_new

# Beam-Warming 格式
def beam_warming(u):
    u_new = np.copy(u)
    for n in range(Nt):
        u_new[2:] = u_new[2:] - 0.5 * alpha * (3*u_new[2:] -
4*u_new[1:-1] + u_new[:-2]) + 0.5 * alpha**2 * (u_new[2:] -
2*u_new[1:-1] + u_new[:-2])
        u_new[0] = 0  # 边界条件
        u_new[1] = 0  # 边界条件
    return u_new

# 初始条件
u0_1 = initial_condition1(x)
u0_2 = initial_condition2(x)

# 数值解
u_upwind_1 = upwind(u0_1)
u_lax_friedrichs_1 = lax_friedrichs(u0_1)
u_lax_wendroff_1 = lax_wendroff(u0_1)
u_beam_warming_1 = beam_warming(u0_1)

u_upwind_2 = upwind(u0_2)
u_lax_friedrichs_2 = lax_friedrichs(u0_2)
```

```
u_lax_wendroff_2 = lax_wendroff(u0_2)
u_beam_warming_2 = beam_warming(u0_2)

# 画图
plt.figure(figsize=(14, 10))

# 初始条件1的结果
plt.subplot(2, 2, 1)
plt.plot(x, u_upwind_1, label='Upwind')
plt.plot(x, u_lax_friedrichs_1, label='Lax-Friedrichs')
plt.plot(x, u_lax_wendroff_1, label='Lax-Wendroff')
plt.plot(x, u_beam_warming_1, label='Beam-Warming')
plt.title('Initial Condition 1: $u_0(x) = 1 - \cos(2\pi x)$')
plt.xlabel('x')
plt.ylabel('u')
plt.legend()

# 初始条件2的结果
plt.subplot(2, 2, 2)
plt.plot(x, u_upwind_2, label='Upwind')
plt.plot(x, u_lax_friedrichs_2, label='Lax-Friedrichs')
plt.plot(x, u_lax_wendroff_2, label='Lax-Wendroff')
plt.plot(x, u_beam_warming_2, label='Beam-Warming')
plt.title('Initial Condition 2: $u_0(x) = 1 (0.4 \leq x \leq
0.6)$')
plt.xlabel('x')
plt.ylabel('u')
plt.legend()

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

在上面的代码中，我们定义了四种数值格式，并针对两种初始条件进行了计算和比较。最终结果将以图形形式展示，使我们可以清晰地看到不同数值格式的效果。通过比较这些结果，可以分析每种格式的优缺点和适用范围。