中国矿业大学数学学院

实验报告

实验课名称	微分方程数值实验
学生班级	数学 *15-1* 班
学生姓名	魏弋凡
学生学号	10154085

成绩

目录

实验一：初始值问题的 Runge-Kutta 方法数值实验

一、实验目的

通过实验，使得学生掌握求解初值问题的四阶、三阶以及二阶的 Runge-Kutta 方法，并通过编程实现算法。

二、实验内容

求解如下微分方程的数值解

$$\begin{cases} \dfrac{dy}{dx} = y \\ y(0) = 1 \end{cases}$$

其中计算区间为 [0,1] 解析解为 $y(x) = e^x$。

分别用二阶、三阶和四阶 R-K 方法求解该问题。取空间步长为 $h = 1/40, 1/20, 1/10, 1/5$。然后将每一种步长下的数值解分别于解析解相比较（计算绝对误差），给出收敛阶（RR1=LOG(ER1/ERROR1)/LOG(2.0)）并列表显示。分析计算结果。

三、实验算法理论

二阶 R-K 方法 $$\begin{cases} y_i = y_{i-1} + \frac{h}{2}(K_1 + K_2) \\ K_1 = f(x_{i-1}, y_{i-1}) \\ K_2 = f(x_i, y_{i-1} + hK_1) \\ y_0 = y(x_0) \end{cases}$$

三阶 R-K 方法 $$\begin{cases} y_i = y_{i-1} + \frac{h}{6}(K_1 + 4K_2 + K_3) \\ K_1 = f(x_{i-1}, y_{i-1}) \\ K_2 = f(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2}K_1) \\ K_3 = f(x_{i-1} + h, y_{i-1} + h(2K_2 - K_1)) \\ y_0 = y(x_0) \end{cases}$$

四阶 R-K 方法 $\begin{cases} y_i = y_{i-1} + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ K_1 = f(x_{i-1}, y_{i-1}) \\ K_2 = f(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2}K_1) \\ K_3 = f(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2}K_2) \\ K_4 = f(x_{i-1} + h, y_{i-1} + hK_3) \\ y_0 = y(x_0) \end{cases}$

四、实验程序

```
from math import exp, log

# R-K 2
def rk2(h):
    N=int(1/h)
    x=[i*h for i in range(N+1)]
    y=[1]
    for i in range(N):
        y0=y[-1]
        K1=y0
        K2=y0+h*K1
        y1=y0+(K1+K2)*h/2
        y.append(y1)

    a=list(map(exp,x))

    return [a, y, [abs(a[i]-y[i]) for i in range(N+1)]]

# R-K 3
def rk3(h):
    N=int(1/h)
    x=[i*h for i in range(N+1)]
    y=[1]
    for i in range(N):
        y0=y[-1]
        K1=y0
        K2=y0+K1*h/2
        K3=y0+h*(2*K2-K1)
        y1=y0+(K1+4*K2+K3)*h/6
        y.append(y1)

    a=list(map(exp,x))

```

```
    return [a, y, [abs(a[i]-y[i]) for i in range(N+1)]]

# R-K 4
def rk4(h):
    y=[1]
    N=int(1/h)
    x=[i*h for i in range(N+1)]
    for i in range(N):
        y0=y[-1]
        K1=y0
        K2=y0+K1*h/2
        K3=y0+K2*h/2
        K4=y0+h*K3
        y1=y0+(K1+2*K2+2*K3+K4)*h/6
        y.append(y1)

    a=list(map(exp,x))

    return [a, y, [abs(a[i]-y[i]) for i in range(N+1)]]

# 收敛阶计算
def err(func, h):
    e1=max(func(h)[2])
    e2=max(func(h/2)[2])

    return log(e1/e2)/log(2)
```

五、实验结果及分析

下表为采用二阶 R-K 方法、步长取 1/20 的结果。

```
  x(i)       解析解     数值解     绝对误差
0.00000    1.00000    1.00000    0.0000000000
0.05000    1.05127    1.05125    0.0000210964
0.10000    1.10517    1.10513    0.0000443556
0.15000    1.16183    1.16176    0.0000699439
0.20000    1.22140    1.22130    0.0000980390
0.25000    1.28403    1.28390    0.0001288307
0.30000    1.34986    1.34970    0.0001625215
0.35000    1.41907    1.41887    0.0001993279
0.40000    1.49182    1.49159    0.0002394806
0.45000    1.56831    1.56803    0.0002832261
0.50000    1.64872    1.64839    0.0003308272
0.55000    1.73325    1.73287    0.0003825641
0.60000    1.82212    1.82168    0.0004387359
0.65000    1.91554    1.91504    0.0004996612
```

```
0.70000    2.01375    2.01319    0.0005656798
0.75000    2.11700    2.11636    0.0006371538
0.80000    2.22554    2.22483    0.0007144689
0.85000    2.33965    2.33885    0.0007980363
0.90000    2.45960    2.45871    0.0008882937
0.95000    2.58571    2.58472    0.0009857075
1.00000    2.71828    2.71719    0.0010907741

最大误差    0.0010907741041608077
```

经计算，三种方法对应的收敛阶分别为

```
         R-K 2                   R-K 3                R-K 4
1.9864339103319741  2.9855816878063317  3.984981344915041
```

实验二：经典一阶差分格式数值实验

一、实验目的

通过实验，使得学生掌握经典的 FTCS，FTBS，FTFS 数值格式，并通过编程实现算法。

二、实验内容

求解如下偏微分方程的数值解

$$\begin{cases} u_t + u_x = 0, & x \in [0,2] \\ u(x,0) = \begin{cases} 1, & 0.5 < x < 1.5 \\ 0, & 0 \le x \le 0.5, \quad 1.5 \ge x \le 2 \end{cases} \end{cases}$$

左右两端采用周期性边界条件，将 [0,2] 等分成 20 个网格，通过 MATLAB 程序，我们使用 FTFS 格式、FTCS 格式和 FTBS 格式这三种一阶精度的差分格式。分别取 $\lambda = 0.5, 0.8, 1.2$，算至 $T = 2$ 的各数值模拟。

三、实验算法理论

考虑方程 $\begin{cases} \dfrac{\partial u}{\partial t} + a\dfrac{\partial u}{\partial x} = 0, & x \in \mathbb{R},\ t > 0 \\ u(x,0) = f(x), & x \in \mathbb{R} \end{cases}$

FTFS 格式: $u_j^{n+1} = u_j^n - \frac{\tau}{h}(u_{j+1}^n - u_j^n)$

FTCS 格式: $u_j^{n+1} = u_j^n - \frac{\tau}{2h}(u_{j+1}^n - u_{j-1}^n)$

FTBS 格式: $u_j^{n+1} = u_j^n - \frac{\tau}{h}(u_j^n - u_{j-1}^n)$

四、实验程序

```python
# Define initial value (as a function)
def u(x):
    if x>=0 and x <=2:
        if x > 0.5 and x < 1.5:
            return 1
        else:
            return 0

    else:
        return u(x%2)

N=20
init=[[j*2/N for j in range(N+1)],[u(j*2/N) for j in range(N+1)]]


# FTFS Method
def showFS(l, N):
    NT=int(N/l)

    # FS
    fs0 = [u(j*2/N) for j in range(N+NT+2)]

    n=0
    while n < NT:
        fs1 = []
        j=0
        while j+1<len(fs0):
            # 迭代
            fs1.append(fs0[j]-l*(fs0[j+1]-fs0[j]))
            j=j+1

        fs0 = fs1[:]
        n=n+1


    # 算至确切的T=2
    fs1 = []
    j=0
    l=(2-NT*l*2/N)/(2/N)
    while j+1<len(fs0):
```

```
        fs1.append(fs0[j]-l*(fs0[j+1]-fs0[j]))
        j=j+1

    return [[j*2/N for j in range(N+1)],fs1]


# FTCS Method
def showCS(l, N):
    NT=int(N/l)

    # CS
    cs0 = [u(j*2/N) for j in range(-NT-1, N+NT+2)]

    n=0
    while n < NT:
        cs1 = []
        j=0
        while j+2<len(cs0):
            # 迭代
            cs1.append(cs0[j+1]-l*(cs0[j+2]-cs0[j])/2)
            j=j+1

        cs0 = cs1[:]
        n=n+1


    # 算至确切的T=2
    cs1 = []
    j=0
    l=(2-NT*l*2/N)/(2/N)
    while j+2<len(cs0):
        cs1.append(cs0[j+1]-l*(cs0[j+2]-cs0[j])/2)
        j=j+1

    return [[j*2/N for j in range(N+1)],cs1]


# FTBS Method
def showBS(l, N):
    NT=int(N/l)

    # BS
    bs0 = [u(j*2/N) for j in range(-NT-1,N+1)]

    n=0
    while n < NT:
```

```
        bs1 = []
        j=0
        while j+1<len(bs0):
            # 迭代
            bs1.append(bs0[j+1]-l*(bs0[j+1]-bs0[j]))
            j=j+1

        bs0 = bs1[:]
        n=n+1


    # 算至确切的T=2
    bs1 = []
    j=0
    l=(2-NT*l*2/N)/(2/N)
    while j+1<len(bs0):
        bs1.append(bs0[j+1]-l*(bs0[j+1]-bs0[j]))
        j=j+1

    return [[j*2/N for j in range(N+1)],bs1]
```

五、实验结果及分析

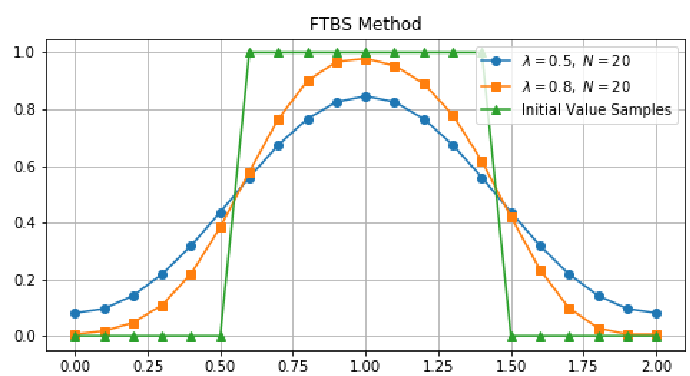


图 1: 收敛的结果

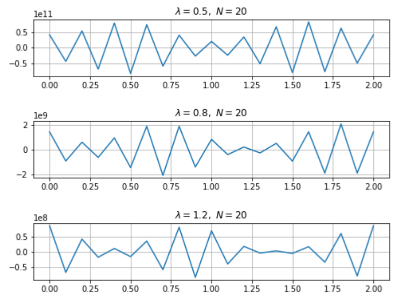


图 2: 发散的 FTFS 结果

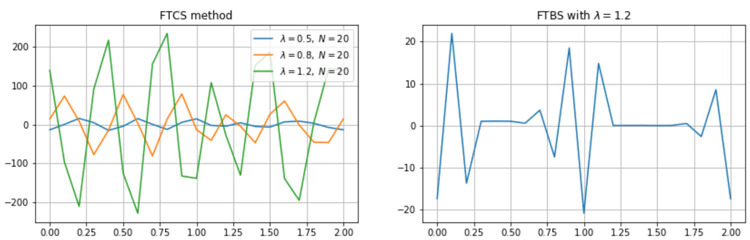


图 3: 其他发散结果

实验三：经典二阶差分格式数值实验

一、实验目的

通过实验，使得学生掌握经典的 FTCS，FTBS，FTFS 数值格式，并通过编程实现算法。

二、实验内容

求解如下偏微分方程的数值解

$$\begin{cases} u_t + u_x = 0, \qquad x \in [0,2] \\ u(x,0) = \begin{cases} 1, & 0.5 < x < 1.5 \\ 0, & 0 \le x \le 0.5, \quad 1.5 \ge x \le 2 \end{cases} \end{cases}$$

左右两端采用周期性边界条件，将 $[0,2]$ 等分成 20 个网格，通过 MATLAB 程序，我们使用 Lax-Friedrichs 格式、Lax-Wendroff 格式和 MacCormack 格式这三种二阶精度的格式，分别取 $\lambda = 0.5, 0.8, 1.2$，算至 $T = 2$ 的各数值模拟。

三、实验算法理论

考虑方程 $\begin{cases} \dfrac{\partial u}{\partial t} + a\dfrac{\partial u}{\partial x} = 0, & x \in \mathbb{R},\ t > 0 \\ u(x,0) = f(x), & x \in \mathbb{R} \end{cases}$

Lax-Friedrichs 格式:

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) - \frac{\tau a}{2h}(u_{j+1}^n - u_{j-1}^n)$$

Lax-Wendroff 格式:

$$u_j^{n+1} = u_j^n - \frac{a\tau}{2h}(u_{j+1}^n - u_{j-1}^n) + \frac{a^2}{2}\frac{\tau^2}{h^2}(u_{j+1}^n - 2u_j^n + u_{j-1}^n)$$

MacCormack 格式:

$$\begin{aligned} u_j^* &= u_j^n - a\frac{\tau}{h}(u_{j+1}^n - u_j^n) \\ u_j^{**} &= u_j^n - a\frac{\tau}{h}(u_j^* - u_{j-1}^*) \\ u_j^{n+1} &= \frac{u_j^* + u_j^{**}}{2} \end{aligned}$$

由于在线性系统中 MacCormack 格式与 Lax-Friedrichs 格式等价，本实验不需要做 MacCormack 格式。

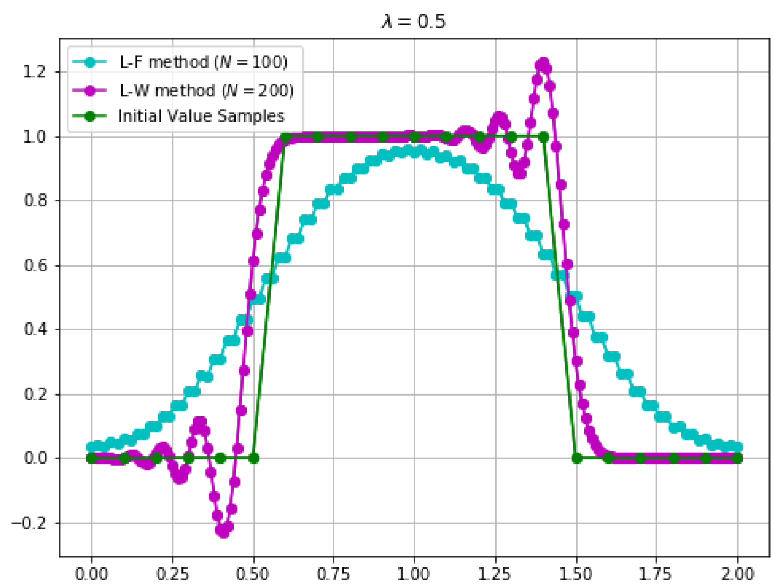
四、实验程序

```
# Define initial value (as a function)
def u(x):
    if x>=0 and x <=2:
        if x > 0.5 and x < 1.5:
            return 1
        else:
            return 0

    else:
        return u(x%2)

```

```
N=20
init=[[j*2/N for j in range(N+1)],[u(j*2/N) for j in range(N+1)]]

# Lax-Friedrichs Method
def showLF(l, N):
    NT=int(N/l)

    # LF
    cs0 = [u(j*2/N) for j in range(-NT-1, N+NT+2)]

    n=0
    while n < NT:
        cs1 = []
        j=0
        while j+2<len(cs0):
            cs1.append((cs0[j+2]+cs0[j])/2-l*(cs0[j+2]-cs0[j])/2)
            j=j+1

        cs0 = cs1[:]
        n=n+1


    cs1 = []
    j=0
    l=(2-NT*l*2/N)/(2/N)
    while j+2<len(cs0):
        cs1.append((cs0[j+2]+cs0[j])/2-l*(cs0[j+2]-cs0[j])/2)
        j=j+1

    return [[j*2/N for j in range(N+1)],cs1]

# Lax-Wendroff Method
def showLW(l, N):
    NT=int(N/l)

    # LW
    cs0 = [u(j*2/N) for j in range(-NT-1, N+NT+2)]
    n=0
    while n < NT:
        cs1 = []
        j=0
        while j+2<len(cs0):
            cs1.append(cs0[j+1]-l*(cs0[j+2]-cs0[j])/2+l*l*(cs0[j+2]-2*cs0[j
+1]+cs0[j])/2)
            j=j+1

```

```
            cs0 = cs1[:]
            n=n+1


    cs1 = []
    j=0
    l=(2-NT*l*2/N)/(2/N)
    while j+2<len(cs0):
        cs1.append(cs0[j+1]-l*(cs0[j+2]-cs0[j])/2+l*l*(cs0[j+2]-2*cs0[j+1]+
    cs0[j])/2)
        j=j+1

    return [[j*2/N for j in range(N+1)],cs1]
```

五、实验结果及分析



实验四：经典差分格式对非线性方程的数值试验

一、实验目的

通过实验，使得学生掌握经典的 FTCS，FTBS，FTFS，Lax-Friedrichs 格式、Lax-Wendroff 格式和 MacCormack 格式数值格式，并通过编程实现对非线性方程的算法。

二、实验内容

求解如下偏微分方程的数值解

$$\begin{cases} u_t + uu_x = 0, \qquad x \in [0,2] \\ u(x,0) = 0.5 + \sin(\pi x) \end{cases}$$

同样的，我们采用左右两端周期性边界条件，将 $[0,2]$ 等分成 20 个网格，通过 MATLAB 程序，分别使用上面用到的 6 种格式，取 $\lambda = 0.8$，分别算至 $T = 0.5/\pi$，$1.0/\pi$，$1.5/\pi$ 时的各自数值结果。

三、实验算法理论

FTFS 格式: $u_j^{n+1} = u_j^n - \frac{\tau}{h}\left(\left(\frac{u^2}{2}\right)_{j+1}^n - \left(\frac{u^2}{2}\right)_j^n\right)$

FTCS 格式: $u_j^{n+1} = u_j^n - \frac{\tau}{2h}\left(\left(\frac{u^2}{2}\right)_{j+1}^n - \left(\frac{u^2}{2}\right)_{j-1}^n\right)$

FTBS 格式: $u_j^{n+1} = u_j^n - \frac{\tau}{h}\left(\left(\frac{u^2}{2}\right)_j^n - \left(\frac{u^2}{2}\right)_{j-1}^n\right)$

Lax-Friedrichs 格式:

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) - \frac{\tau}{2h}\left(\left(\frac{u^2}{2}\right)_{j+1}^n - \left(\frac{u^2}{2}\right)_{j-1}^n\right)$$

Lax-Wendroff 格式:

$$u_j^{n+1} = u_j^n - \frac{\tau}{2h}\left(\left(\frac{u^2}{2}\right)_{j+1}^n - \left(\frac{u^2}{2}\right)_{j-1}^n\right) + \frac{1}{2}\frac{\tau^2}{h^2}\left(\left(\frac{u^2}{2}\right)_{j+1}^n - 2\left(\frac{u^2}{2}\right)_j^n + \left(\frac{u^2}{2}\right)_{j-1}^n\right)$$

MacCormack 格式:

$$
\begin{aligned}
u_j^* &= u_j^n - \frac{\tau}{h}\left(\left(\frac{u^2}{2}\right)_{j+1}^n - \left(\frac{u^2}{2}\right)_j^n\right) \\
u_j^{**} &= u_j^n - \frac{\tau}{h}\left(\left(\frac{u^2}{2}\right)_j^* - \left(\frac{u^2}{2}\right)_{j-1}^*\right) \\
u_j^{n+1} &= \frac{u_j^* + u_j^{**}}{2}
\end{aligned}
$$

四、实验程序

```
import numpy as np
from numpy import pi

# Initial value
def u(x):
    return 0.5+np.sin(pi*x)

N=40
init=[[j*2/N for j in range(N+1)],[u(j*2/N) for j in range(N+1)]]

# FTFS
def FTFS(N, t):
    l=0.8
    NT=int(t*N/2/l)

    # FS
    fs0 = [u(j*2/N) for j in range(N+NT+2)]

    n=0
    while n < NT:
        fs1 = []
        j=0
        while j+1<len(fs0):
            fs1.append(fs0[j]-l*(fs0[j+1]**2/2-fs0[j]**2/2))
            j=j+1

        fs0 = fs1[:]
        n=n+1


    fs1 = []
    j=0
    l=(t-NT*l*2/N)/(2/N)
    while j+1<len(fs0):
```

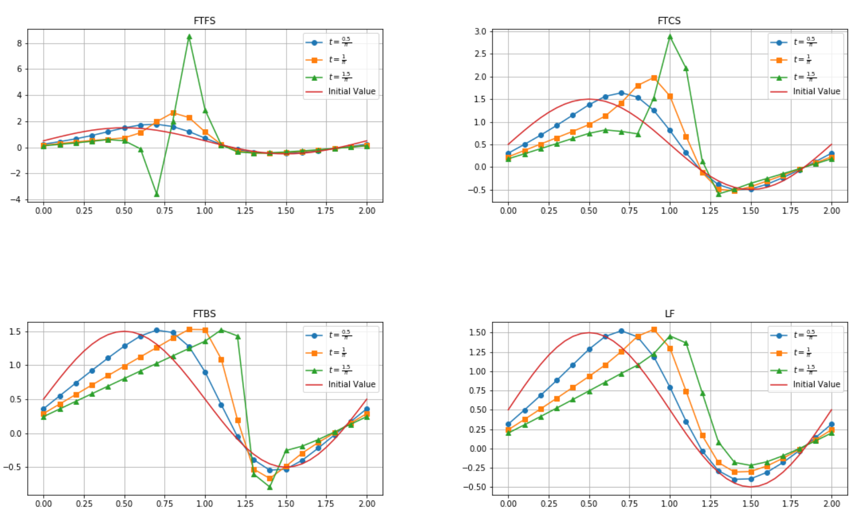
```
        fs1.append(fs0[j]-l*(fs0[j+1]**2/2-fs0[j]**2/2))
        j=j+1

    return [[j*2/N for j in range(N+1)],fs1]

# FTCS Method
def FTCS(N,t):
    l=0.8
    NT=int(t*N/2/l)

    # CS
    cs0 = [u(j*2/N) for j in range(-NT-1, N+NT+2)]

    n=0
    while n < NT:
        cs1 = []
        j=0
        while j+2<len(cs0):
            cs1.append(cs0[j+1]-l*(cs0[j+2]**2/2-cs0[j]**2/2)/2)
            j=j+1

        cs0 = cs1[:]
        n=n+1


    cs1 = []
    j=0
    l=(t-NT*l*2/N)/(2/N)
    while j+2<len(cs0):
        cs1.append(cs0[j+1]-l*(cs0[j+2]**2/2-cs0[j]**2/2)/2)
        j=j+1

    return [[j*2/N for j in range(N+1)],cs1]

# FTBS Method
def FTBS(N, t):
    l=0.8
    NT=int(t*N/2/l)

    # BS
    bs0 = [u(j*2/N) for j in range(-NT-1,N+1)]

    n=0
    while n < NT:
        bs1 = []
        j=0
```

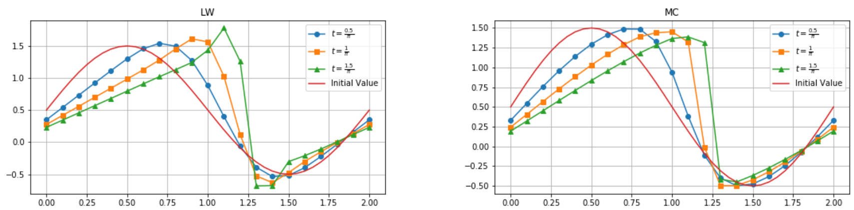
```
        while j+1<len(bs0):
            bs1.append(bs0[j+1]-l*(bs0[j+1]**2/2-bs0[j]**2/2))
            j=j+1

        bs0 = bs1[:]
        n=n+1


    bs1 = []
    j=0
    l=(t-NT*l*2/N)/(2/N)
    while j+1<len(bs0):
        bs1.append(bs0[j+1]-l*(bs0[j+1]**2/2-bs0[j]**2/2))
        j=j+1

    return [[j*2/N for j in range(N+1)],bs1]

# Lax-Friedrichs Method
def LF(N, t):
    l=0.8
    NT=int(t*N/2/l)

    # LF
    cs0 = [u(j*2/N) for j in range(-NT-1, N+NT+2)]

    n=0
    while n < NT:
        cs1 = []
        j=0
        while j+2<len(cs0):
            cs1.append((cs0[j+2]+cs0[j])/2-l*(cs0[j+2]**2/2-cs0[j]**2/2)/2)
            j=j+1

        cs0 = cs1[:]
        n=n+1


    cs1 = []
    j=0
    l=(t-NT*l*2/N)/(2/N)
    while j+2<len(cs0):
        cs1.append((cs0[j+2]+cs0[j])/2-l*(cs0[j+2]**2/2-cs0[j]**2/2)/2)
        j=j+1

    return [[j*2/N for j in range(N+1)],cs1]

```

```
# Lax-Wendorff Method
def LW(N, t):
    l=0.8
    NT=int(t*N/2/l)

    # LW
    cs0 = [u(j*2/N) for j in range(-NT-1, N+NT+2)]
    n=0
    while n < NT:
        cs1 = []
        j=0
        while j+2<len(cs0):
            cs1.append(cs0[j+1]-l*(cs0[j+2]**2/2-cs0[j]**2/2)/2+l*l*(cs0[j+2]**2/2-2*cs0[j+1]**2/2+cs0[j]**2/2)/2)
            j=j+1

        cs0 = cs1[:]
        n=n+1


    cs1 = []
    j=0
    l=(t-NT*l*2/N)/(2/N)
    while j+2<len(cs0):
        cs1.append(cs0[j+1]-l*(cs0[j+2]**2/2-cs0[j]**2/2)/2+l*l*(cs0[j+2]**2/2-2*cs0[j+1]**2/2+cs0[j]**2/2)/2)
        j=j+1

    return [[j*2/N for j in range(N+1)],cs1]

# MacCormack Method
def MC(N, t):
    l=0.8
    NT=int(t*N/2/l)

    # LW
    cs0 = [u(j*2/N) for j in range(-NT-1, N+NT+2)]
    n=0
    while n < NT:
        cs1 = []
        j=0
        while j+1<len(cs0):
            cs1.append(cs0[j]-l*(cs0[j+1]**2/2-cs0[j]**2/2))
            j=j+1

        j=0
```

```
        cs2 = []
        while j+1<len(cs1):
            cs2.append((cs0[j+1]+cs1[j+1])/2-l/2*(cs1[j+1]**2/2-cs1[j]**2/2))
            j=j+1

        cs0=cs2[:]
        n=n+1


    cs1 = []
    j=0
    l=(t-NT*l*2/N)/(2/N)
    while j+1<len(cs0):
        cs1.append(cs0[j]-l*(cs0[j+1]**2/2-cs0[j]**2/2))
        j=j+1

    j=0
    cs2 = []
    while j+1<len(cs1):
        cs2.append((cs0[j+1]+cs1[j+1])/2-l/2*(cs1[j+1]**2/2-cs1[j]**2/2))
        j=j+1

    return [[j*2/N for j in range(N+1)],cs2]
```

五、实验结果及分析





L-F 格式的精度测试

实验程序

```
import numpy as np
from numpy import pi

# Initial Value (which is actually the exact value when t=2)
def u(x):
    return np.sin(pi*x)

# Machine precision
eps = 7./3-4./3-1

# Lax-Friedrichs Method Errors
def LF(N, l):
    NT=int(N/l)

    # LF
    cs0 = [u(j*2/N) for j in range(-NT-1, N+NT+2)]

    n=0
    while n < NT:
        cs1 = []
        j=0
        while j+2<len(cs0):
            cs1.append((cs0[j+2]+cs0[j])/2-l*(cs0[j+2]-cs0[j])/2)
            j=j+1

        cs0 = cs1[:]
        n=n+1


    cs1 = []
    j=0
    l=(2-NT*l*2/N)/(2/N)
```

```
    while j+2<len(cs0):
        cs1.append((cs0[j+2]+cs0[j])/2-l*(cs0[j+2]-cs0[j])/2)
        j=j+1

    init = [u(j*2/N) for j in range(N+1)]
    err = [abs(init[j]-cs1[j]) for j in range(N+1)]
    return max([eps, max(err)])

errlist=[LF(8*2**k,0.8) for k in range(5,9)]

r = []
for j in range(len(errlist)-1):
    r.append(np.log(errlist[j]/errlist[j+1])/np.log(2))

print(r)
```

实验结果及分析

结果:

```
[0.99351807455171526, 0.99681647030443343, 0.99842252636116657]
```

结果接近于 1 而不是 2 的原因是该格式关于空间变量的迭代是 1 阶的，该效果起了主导性的作用。