project1

20989977 Zhang Mingtao

2024/3/19

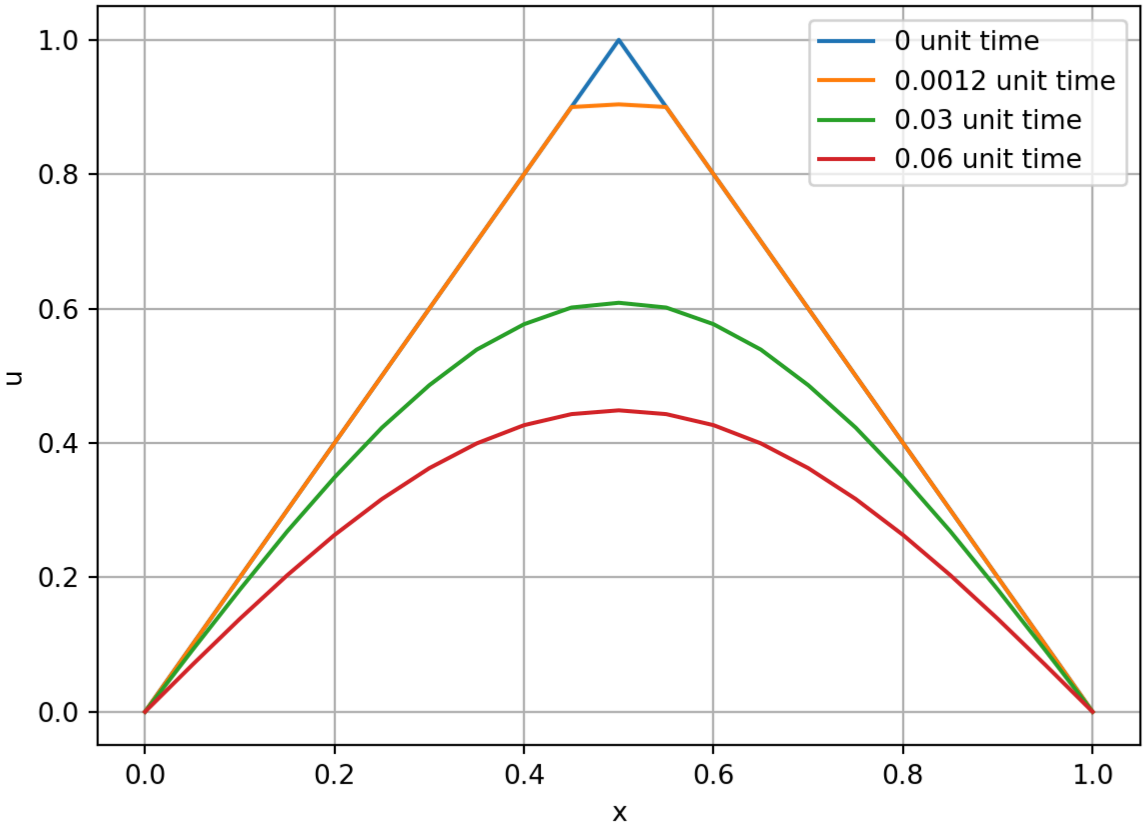
0.

```
library(reticulate)
```

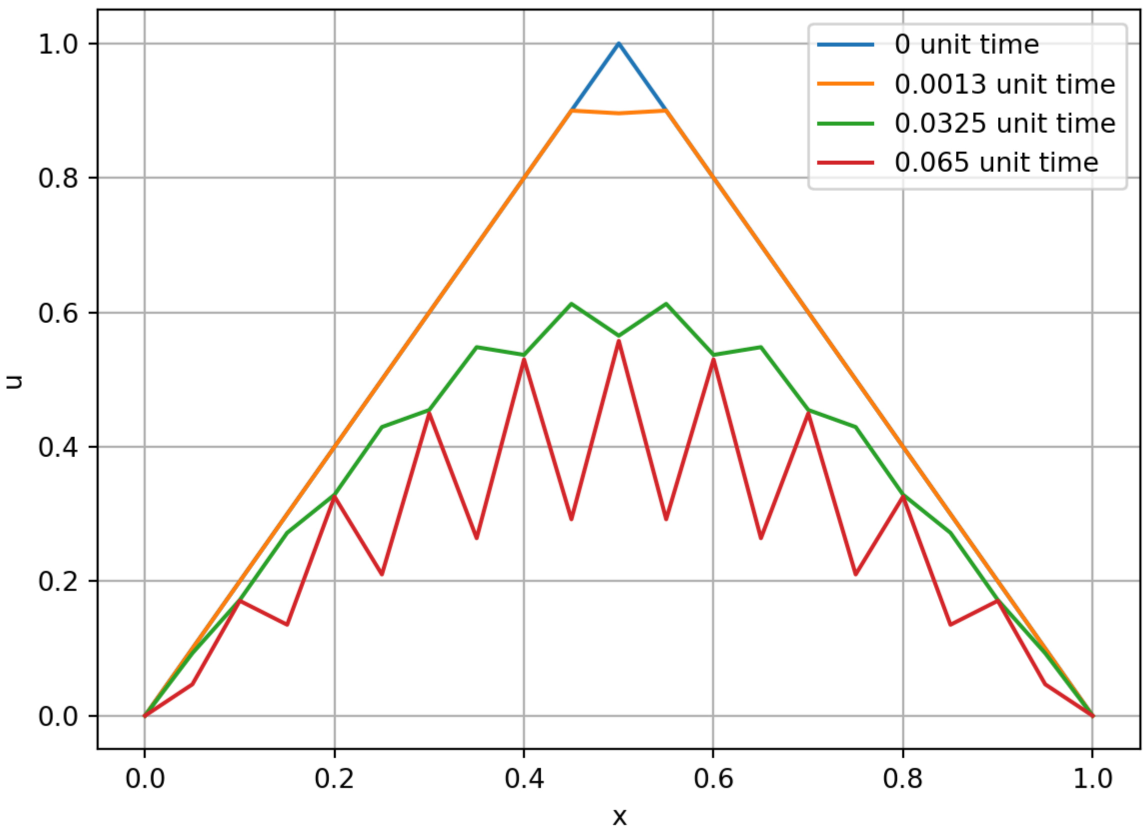
1.

```python
import numpy as np
import matplotlib.pyplot as plt
################################### 1 ###################################
### (1)
def u0(x):
    return np.where(x<=0.5, 2*x, 2-2*x)

def method(J, dx, dt, tlist):
    # x-J;t-n
    x = np.linspace(0, 1, J+1)
    t = np.arange(0, tlist[-1]+dt, dt)

    # μ
    mu = dt / (dx**2)

    # 初始化
    U = np.zeros((len(t), len(x)))
    U[:, 0] = 0
    U[:, -1] = 0

    # 设置初始条件
    U[0, :] = u0(x)

    # explicit scheme
    for n in range(len(t) - 1):
        for j in range(1, J):
            U[n+1, j] = U[n, j] + mu * (U[n, j+1] - 2 * U[n, j] + U[n, j-1])

    # 绘制数值解
    fig, ax = plt.subplots()
    for i, plot_time in enumerate(tlist):
        n = int(plot_time / dt)
        ax.plot(x, U[n, :], label=f"{plot_time} unit time")

    ax.set_xlabel('x')
    ax.set_ylabel('u')
    ax.legend()
    ax.grid(True)
    plt.show()

# 参数
J = 20
dx = 0.05
dt1 = 0.0012
dt2 = 0.0013
tlist = [0, dt1, 25*dt1, 50*dt1]
```

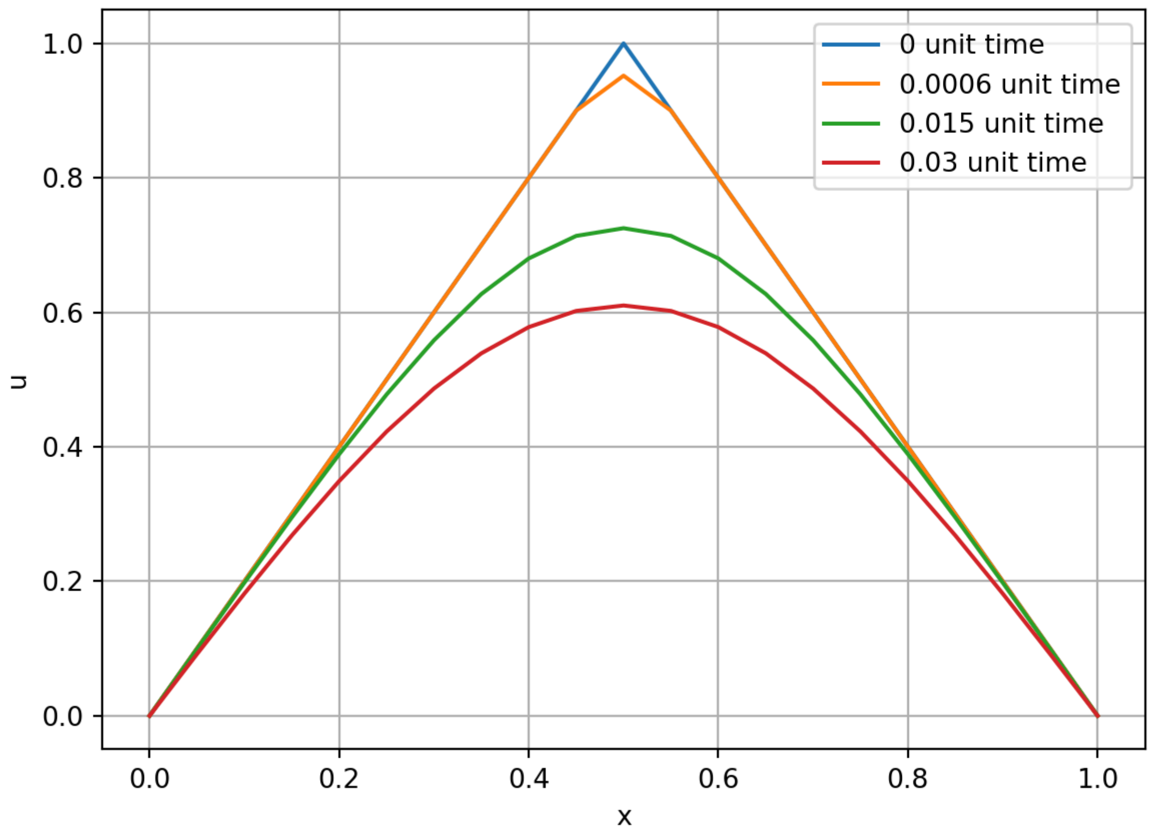
```python
# (i)
method(J, dx, dt1, tlist)
```




```
# (ii)
tlist2 = [0, dt2, 25*dt2, 50*dt2]
method(J, dx, dt2, tlist2)
```

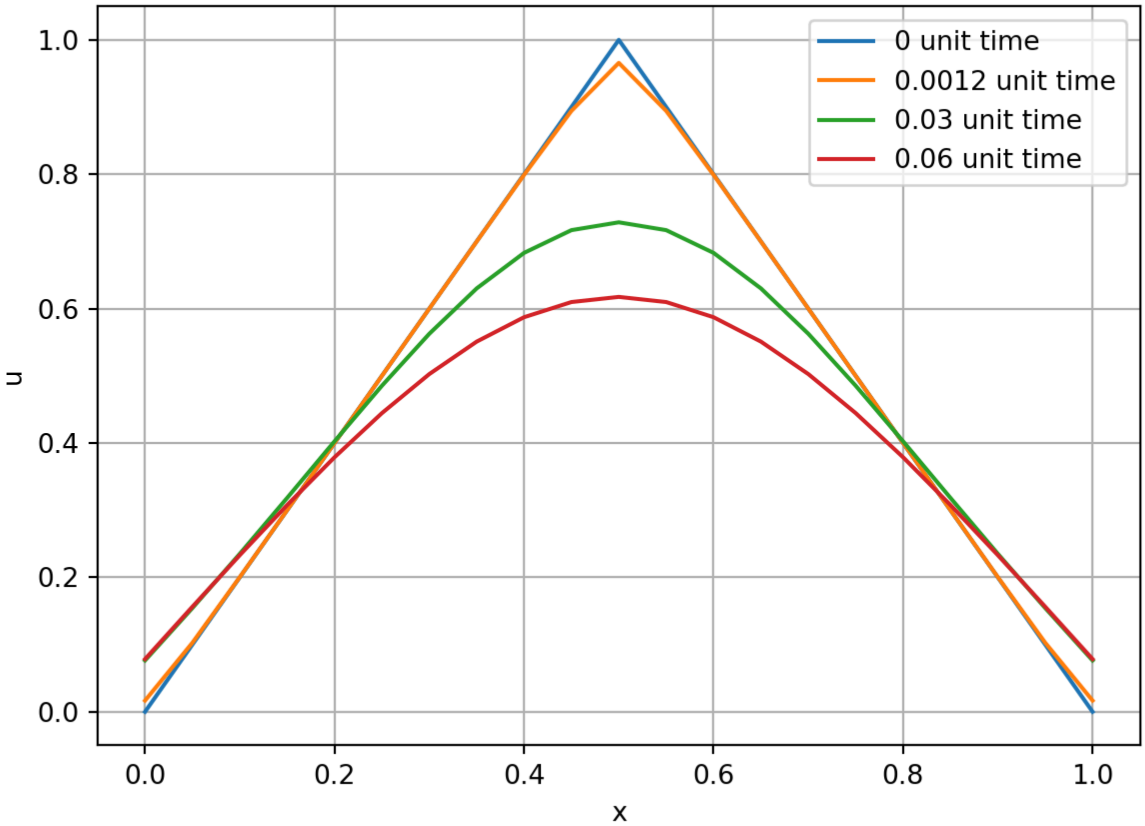



```
### (2)

dt3 = 0.0006
tlist3 = [0, dt3, 25*dt3, 50*dt3]

method(J, dx, dt3, tlist3)
```

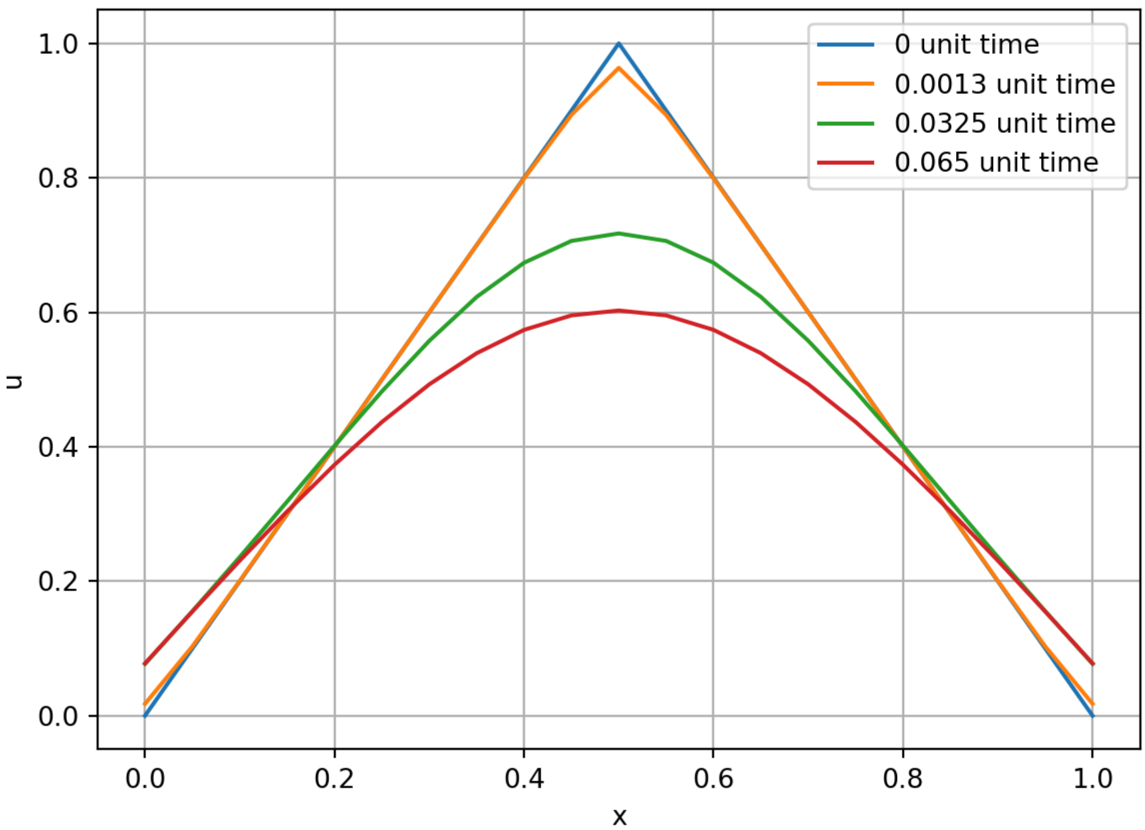

0 unit time
0.0006 unit time
0.015 unit time
0.03 unit time
1.0
0.8
0.6
0.4
0.2
0.0
u
0.0
0.2
0.4
0.6
0.8
1.0
x

```
### (3)
from scipy.sparse import diags

def Crank_Nicolson_method(J, dx, dt, tlist):
    # x-J;t-n
    x = np.linspace(0, 1, J+1)
    t = np.arange(0, tlist[-1]+dt, dt)

    # μ
    mu = 0.5*dt/(dx**2)

    # 初始化
    U = np.zeros((len(t), len(x)))
    U[0, :] = u0(x)
    U[:, 0] = 0
    U[:, -1] = 0

    # 系数矩阵
    A = diags([-mu, 1+2*mu, -mu], [-1, 0, 1], shape=(J+1, J+1)).toarray()

    # 迭代计算数值解
    for n in range(len(t) - 1):
        U[n+1, :] = np.linalg.solve(A, U[n, :])

    # 绘制数值解
    fig, ax = plt.subplots()
    for i, plot_time in enumerate(tlist):
        n = int(plot_time / dt)
        ax.plot(x, U[n, :], label=f"{plot_time} unit time")

    ax.set_xlabel('x')
    ax.set_ylabel('u')
    ax.legend()
    ax.grid(True)
    plt.show()


# 参数
J = 20
dx = 0.05
dt4 = 0.0012
```

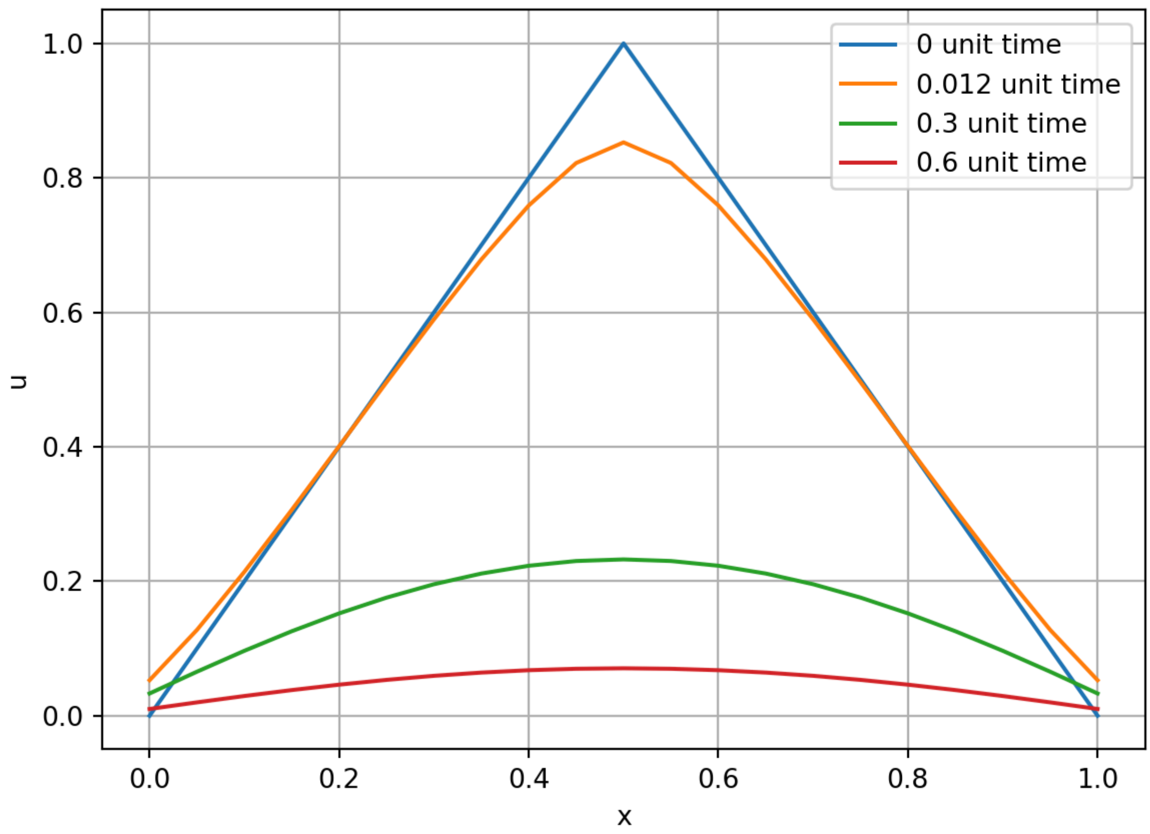
```
# (i)
dt4 = 0.0012
tlist4 = [0, dt4, 25*dt4, 50*dt4]
Crank_Nicolson_method(J, dx, dt4, tlist4)
```




```
# (ii)
dt5 = 0.0013
tlist5 = [0, dt5, 25*dt5, 50*dt5]
Crank_Nicolson_method(J, dx, dt5, tlist5)
```

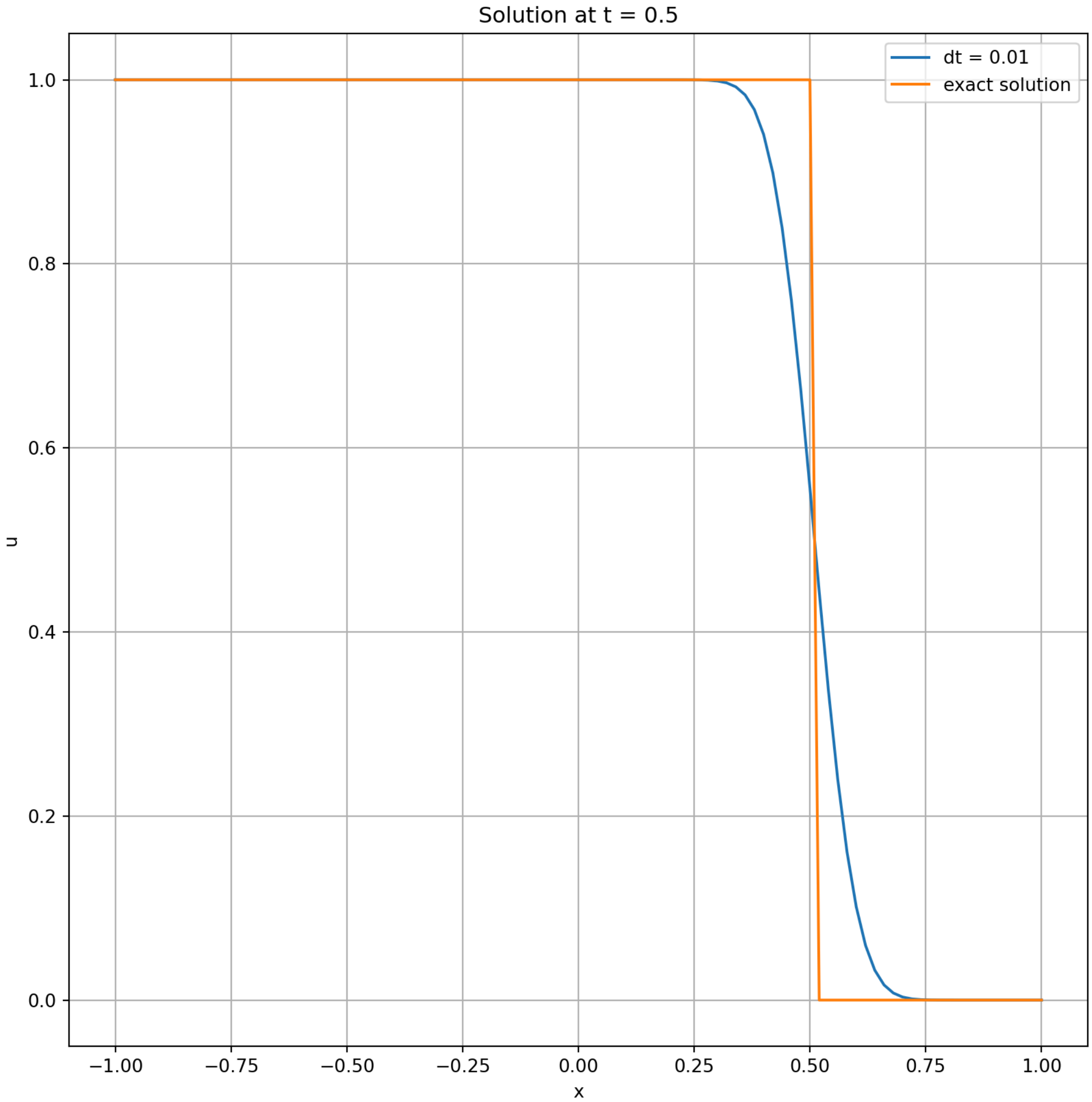



```
# (iii)
dt6 = 0.012
tlist6 = [0, dt6, 25*dt6, 50*dt6]
Crank_Nicolson_method(J, dx, dt6, tlist6)
```

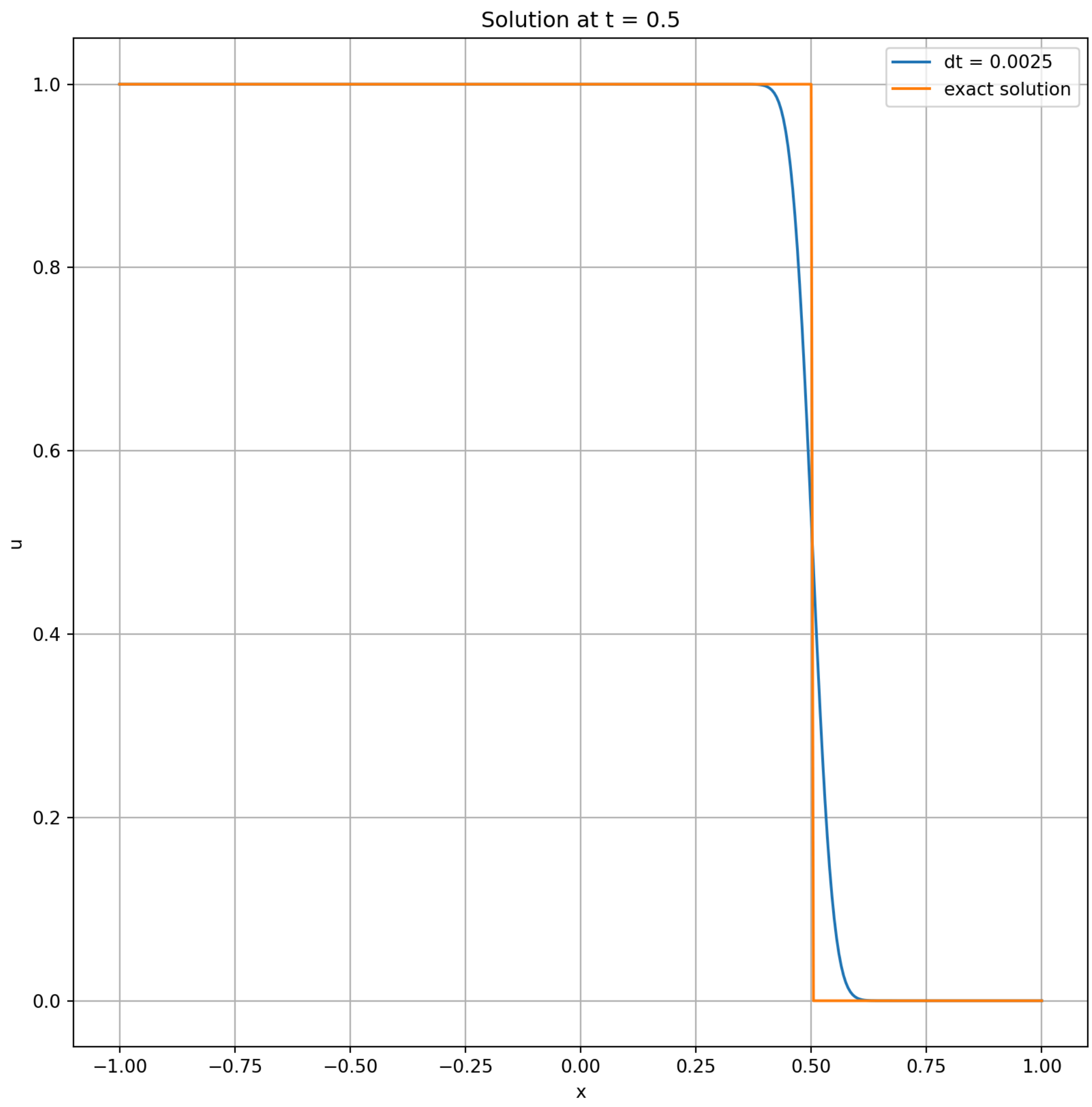

1.0
0.8
0.6
0.4
0.2
0.0
u
0.0
0.2
0.4
0.6
0.8
1.0
x
0 unit time
0.012 unit time
0.3 unit time
0.6 unit time

2.

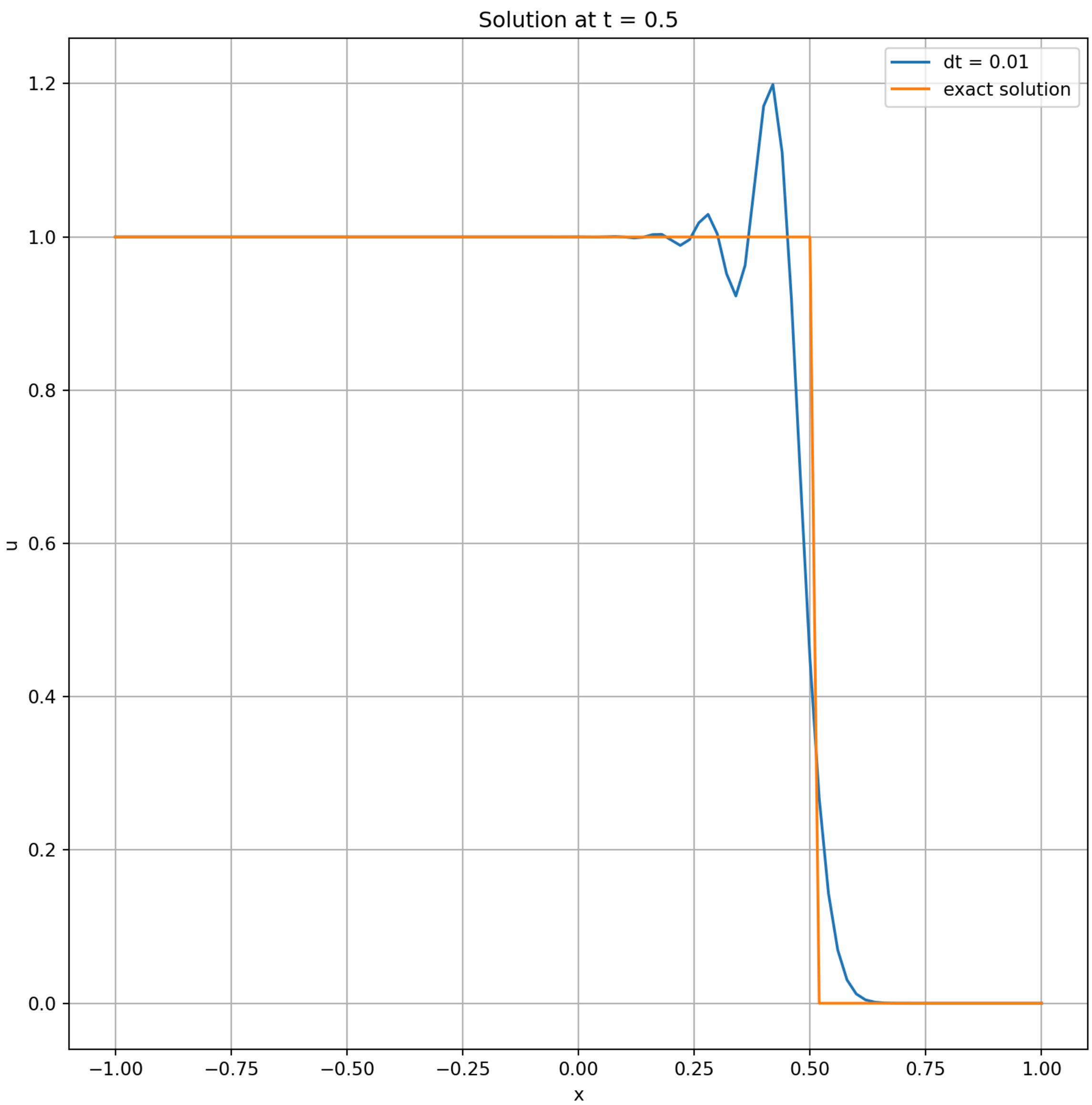
```
################################### 2 ###################################
### (1)

def u0(x):
    return np.where(x<=0, 1, 0)

def method(dx, dt, a=1, v=0.5, t_final=0.5):

    nt = int(t_final / dt)
    nx = int(2/dx)
    x = np.linspace(-1, 1, nx+1)
    t = np.arange(0, t_final+dt, dt)

    # v
    v = dt / dx

    # 初始化
    U = np.zeros((len(t), len(x)))

    # 设置初始和边界条件
    U[0, :] = u0(x)
    U[:, 0] = 1
    U[:, -1] = 0

    # upwind method
    for n in range(len(t)-1):
        for j in range(1, len(x)-1):
            U[n+1, j] = U[n, j] - a * v * (U[n, j] - U[n, j-1])

    # 绘制数值解
    plt.figure(figsize=(10, 10))
    plt.plot(x, U[nt, :], label=f"dt = {dt}")
    plt.plot(x, u0(x-0.5), label='exact solution')
    plt.xlabel("x")
    plt.ylabel("u")
    plt.title("Solution at t = 0.5")
    plt.legend()
    plt.grid(True)
    plt.show()

# 设置模拟区域和时间步长
v = 0.5
a = 1
dt1 = 0.01
dt2 = 0.0025
dx1 = dt1/v
dx2 = dt2/v

method(dx1, dt1)
```



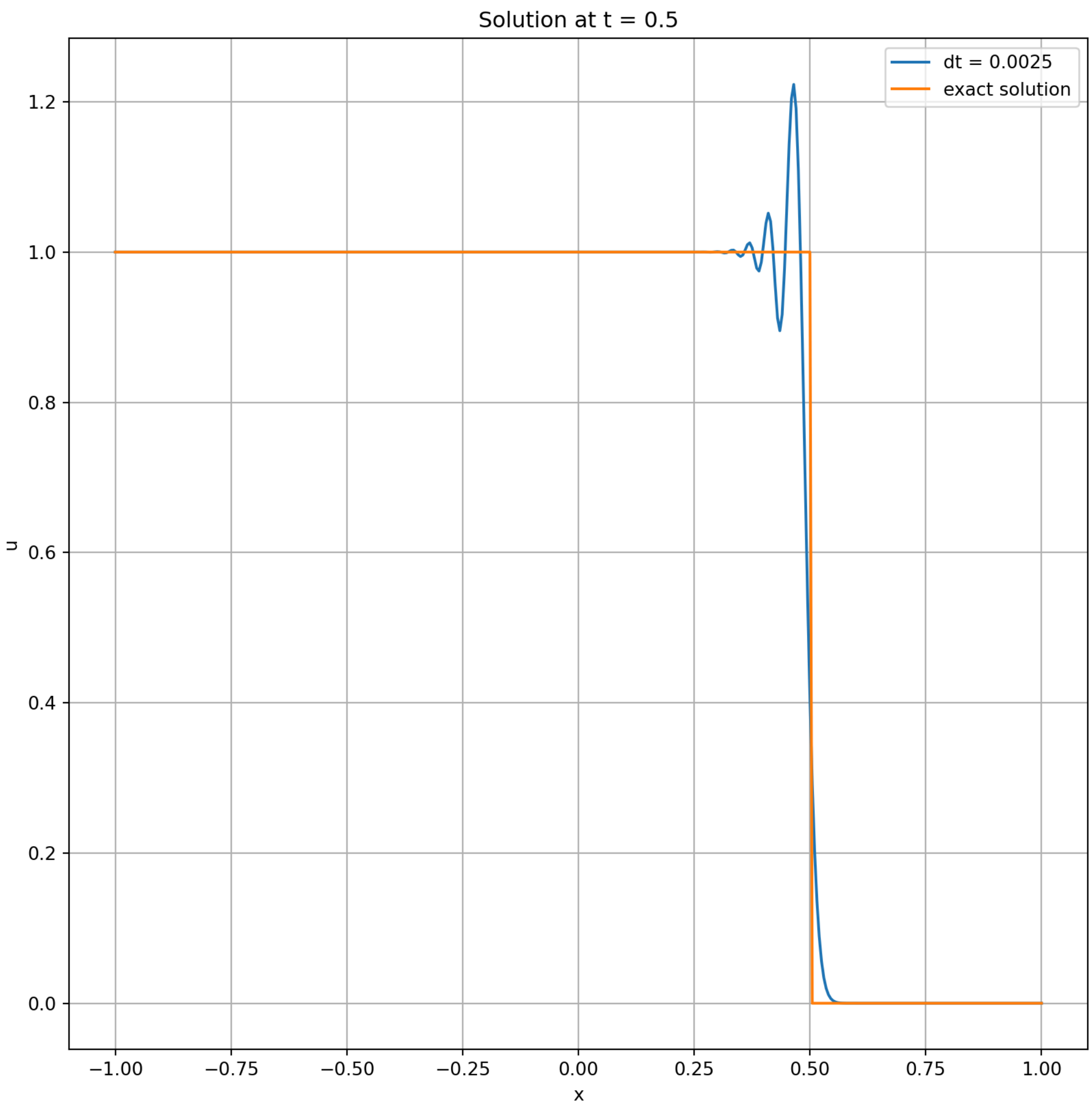

```
method(dx2, dt2)
```


Solution at t = 0.5
dt = 0.0025
exact solution
1.0
0.8
0.6
0.4
0.2
0.0
u
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

```
### (2)

def Lax_Wendroff_method(dx, dt, a=1, v=0.5, t_final=0.5):

    nt = int(t_final / dt)
    nx = int(2/dx)
    x = np.linspace(-1, 1, nx+1)
    t = np.arange(0, t_final+dt, dt)

    # v
    v = dt / dx

    # 初始化
    U = np.zeros((len(t), len(x)))

    # 设置初始和边界条件
    U[0, :] = u0(x)
    U[:, 0] = 1
    U[:, -1] = 0

    # upwind method
    for n in range(len(t)-1):
        for j in range(1, len(x)-1):
            U[n+1, j] = U[n, j] - 0.5 * a * v * (U[n, j+1] - U[n, j-1]) + 0.5 * a**2 * v**2 *
(U[n, j+1] - 2 * U[n, j] + U[n, j-1])

    # 绘制数值解
    plt.figure(figsize=(10, 10))
    plt.plot(x, U[nt, :], label=f"dt = {dt}")
    plt.plot(x, u0(x-0.5), label='exact solution')
    plt.xlabel("x")
    plt.ylabel("u")
    plt.title("Solution at t = 0.5")
    plt.legend()
    plt.grid(True)
    plt.show()

# 设置模拟区域和时间步长
v = 0.5
a = 1
dt1 = 0.01
dt2 = 0.0025
dx1 = dt1/v
dx2 = dt2/v

Lax_Wendroff_method(dx1, dt1)
```




```
Lax_Wendroff_method(dx2, dt2)
```




```
### (3)
# As in (1) and (2), upwind scheme has significant smoothing of the edges of the pulse compared with the exact solution.
# Lax Wendroff method maintains the height and width of the pulse better than the upwind scheme, but generates oscillations.
```