東京工業大学
Tokyo Institute of Technology

PARTIAL DIFFERENTIAL EQUATIONS FOR SCIENCE AND ENGINEERING

Final Report

NOVEMBER 30, 2020

TAERAKUL, Janat

19B60096

Contents

Table of Figures

Insructions:

1. Submit the following by 5PM, November 30, 2020:
    a. This report (pdf format)
    b. Separate figures (png of multiple snapshots). Note: Animation is optional
    c. Program codes (*_(student IDnumber).py)
    d. Compress all of the above into one ‘zip’ file (filename: ID.zip)

2. Plagiarism is illegal. All past submissions are in my possession.
3. Please find a purpose or motivation to learn how to conduct numerical modelling on your own. You are free to work with or consult with your friends; but copying codes from your peers is considered cheating.
4. This page is not necessary for the report. You may remove this page during submission.

Problem 1. Diffusion Equation

Construct a diffusion model for a 2-D heat plate with dimensions 100 m. by 100 m given the equation,

$$\frac{\partial T}{\partial t} - \alpha\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = 0$$

Implement the following:

1. Investigate steady-state condition given boundary conditions (Dirichlet) using “successive over-relaxation”. Use poisson equation by assigning external forcing (e.g. *g*). (10 points)
2. Investigate various boundary conditions (non-steady state) by discussing its effects using animation. Feel free to set the grid sizes, time-steps, internal parameters. Ensure stability by setting an appropriate *d*.
   a. Dirichlet Boundary condition (5 points)
   b. Neumann Boundary condition (5 points)
   c. Mixed boundaries (5 points)
3. Investigate the influence of $d = \frac{\alpha \Delta t}{\Delta x^2}$ by testing various values for *d* (0. to 1.0). What are the threshold values for *d* to simulate a stable model behavior? Why is *d* causing this effect? (10 points)

1. Successive Over-Relaxation method

- Program

```python
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm,trange

dx = dy = 1.0
span_x = 100.0
span_y = 100.0
g = 9.81
rho = 1.0
Tension = 1000.0
omega = 0.8
n = 10000
print("Total Iteration: ", n)

x = np.arange(0.0, span_x+dx, dx)
y = np.arange(0.0, span_y+dy, dy)
X, Y = np.meshgrid(x,y)

## Dirichlet Boundary Condition

T = np.zeros((y.shape[0],x.shape[0]))
# Dirichlet Boundary Condition
T[:,-1] = T [:,0] = T[0,:] = T[-1,:] = 0.0

for it in trange(n):

    # 3D image construction
    if it%10==0:
        ax = plt.axes(projection='3d')
        ax.plot_surface(X, Y, T, rstride=5, cstride=5, cmap='viridis', edgecolor='none')
        ax.set_xlabel('x')
        ax.set_ylabel('y')
```

```
            ax.set_zlabel('T')
            ax.set_zlim([-10.0,1.0])
            plt.title('omega = %.2f¥n%d'%(omega, it))
            plt.savefig('./pic2/%04d.png'%(it/10))
            plt.clf()
            plt.cla()
        for i in range(1,T.shape[0]-1):
            for j in range(1,T.shape[1]-1):
                R = 0.25*(T[i+1][j]+T[i-1][j]+T[i][j+1]+T[i][j-1]-
    dx*dy*g*rho/Tension)
                T[i][j] = (1-omega)*T[i][j]+omega*R
```

- Discussion

The Successive over-relaxation method is using numerical method to find the steady stead. The program determines the state based on the equation

$$T_{i,j}^{k+1} = (1-\omega)T_{i,j}^{k} + \frac{\omega}{4}\left\{T_{i+1}^{k} + T_{i-1,j}^{k+1} + T_{i,j+1}^{k} + T_{i,j-1}^{k+1} - \Delta x^2 g\left(\frac{\rho}{T}\right)\right\}$$

Since $T_{i,j}^{k+1}$ depends on the term $T_{i-1,j}^{k+1}$ and term $T_{i,j-1}^{k+1}$, so we cannot use the function np.roll() to determine the next T. However, the term $T_{i-1,j}^{k+1}$ and term $T_{i,j-1}^{k+1}$, are at the position that we already calculated ($i-1$ and $j-1$), hence we can use normal nested loop to calculate the next T. The behavior of the system is similar to a sheet of elastic clothe that is being pulled by the gravity while its border is being fixed. The center of the sheet is curved along the gravity. The valued of $g, \rho, and\ T$ are fixed to be gravity, 1.0 and 1000.0 respectively but I tried varied the value of ω to see what is changing. As shown in the figure 1, 2, 3, and 4, the bigger ω can transform the system into its steady state faster than smaller ω.

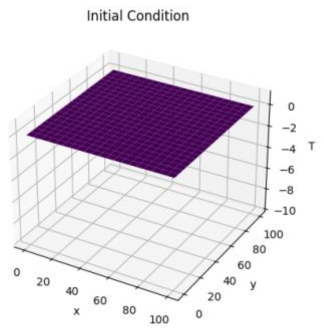


Figure 1 The Initial Condition of the System

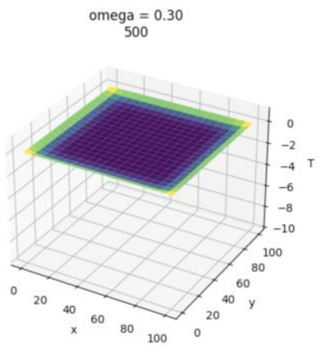


Figure 2 The system with ¥omega=0.3 after 500 iterations

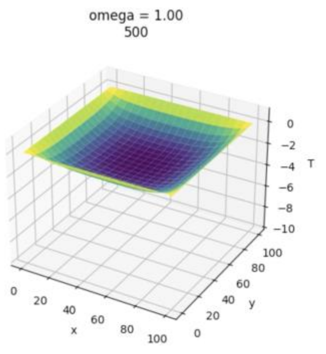


Figure 3 The system with $\omega = 1.0$ (Gauss-Seidel Method) after 500 iterations

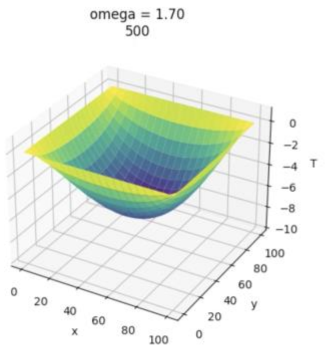


Figure 4 The system with $\omega = 1.7$ after 500 iterations

2. Various kind of Boundary Condition

- Dirichlet Boundary Condition
    - Program

The Dirichlet boundary condition is a boundary condition that has fixed value at the boundary. For my example, I set the value of the boundary to zero. Using the diffusion equation, I created 3D figures to see the behavior of the boundary. By observing the animation constructed from series of 3D figures, it seems like the heat is leaking through the border of the figure as shown in the figure 6. The value near the boundary is always near zero, despite of the amount of heat flowing into the area near border.

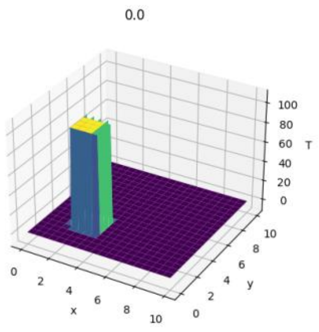


Figure 5 Diffusion Equation with Dirichlet Boundary Condition at the Initial Condition

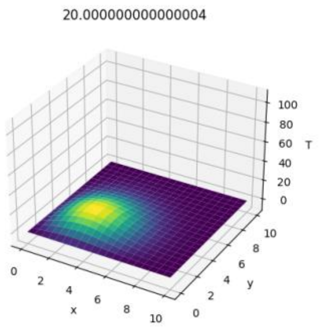


Figure 6 Diffusion Equation with Dirichlet Boundary Condition at $t = 20.0s$

- Neumann Boundary Condition

The Neumann boundary condition is a boundary condition that specify the value of derivative at the boundary. In this case, the value (G) is being set to 0.0 so the boundary act like a wall. The heat hits the boundary, but it cannot escape from the system. The value near the boundary remains there and spread to the side as shown in the figure 8.

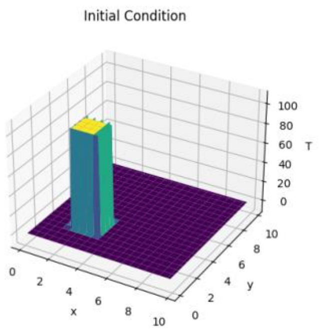


Figure 7 Diffusion Equation with Neumann Boundary Condition at the Initial Condition

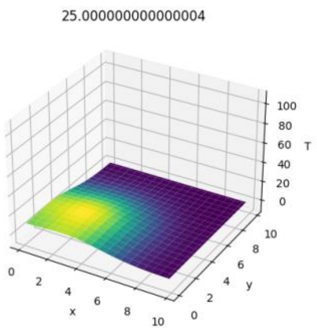


Figure 8 Diffusion Equation with Neumann Boundary Condition at $t = 25.0s$

- Mixed Boundary Condition

The mixed boundary condition is basically the Dirichlet boundary condition and the Neumann boundary condition combined. The boundary along x-axis is set to be Dirichlet boundary condition while the boundary along y-axis is set to be Neumann boundary condition. In this case, we can see the behavior of both boundary conditions at the boundary. The initial condition and the system at $t = 100.0s$ are shown in the figure 9 and 10. In the figure 10, the values at the boundaries along x-axis is near zero while the values at the boundaries along y-axis are not passing through the boundaries.

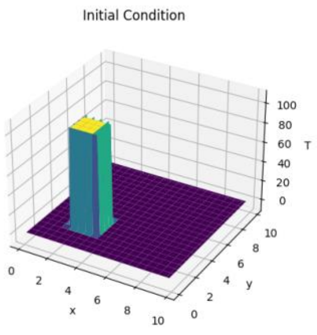


Figure 9 Diffusion Equation with Mixed Boundary Condition at the Initial Condition

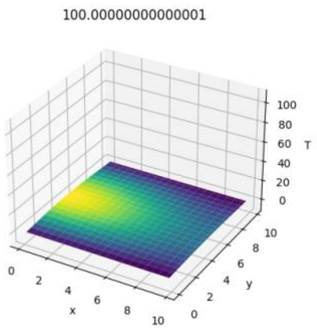


Figure 10 Diffusion Equation with Mixed Boundary Condition at $t = 100.0s$

3. Influence of $d = \frac{\alpha \Delta t}{\Delta x^2}$

As we discussed in the class, the term $d = \frac{\alpha \Delta t}{\Delta x^2}$ is determining the stability of the system. It is derived by using Fourier series. If the value of d surpass a certain threshold, the error that is created with numerical method will be amplified more and more as it iterates. The threshold is 0.5 for 1-D diffusion equation and 0.25 for 2-D diffusion equation. I tested the simulation of diffusion equation with Dirichlet boundary condition using several values of d. The results are as expected. The simulations with the value of d less than or equal to 0.25 can work without any problem, but as soon as the value of d surpass the threshold 0.25, the simulation starts to rumble and scattered away as shown in the figure15 and 16.

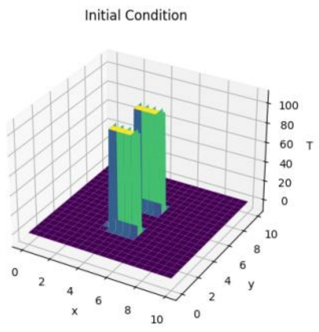


Figure 11 Initial Condition of the Diffusion Equation with Dirichlet Boundary Condition

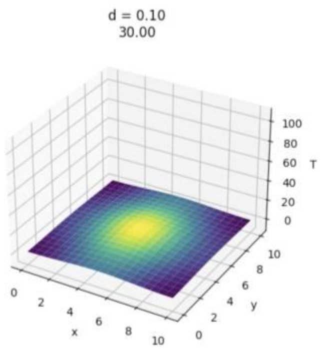


Figure 12 Diffusion Equation with Dirichlet Boundary Condition using $d = 0.1$ at $t = 30.00s$

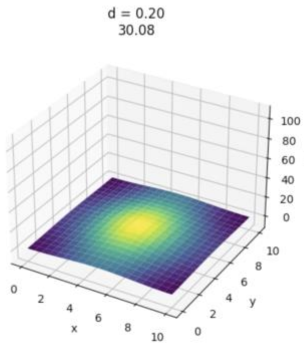


Figure 13 Diffusion Equation with Dirichlet Boundary Condition using $d = 0.2$ at $t = 30.08s$

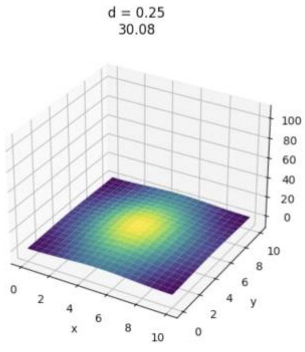


Figure 14 Diffusion Equation with Dirichlet Boundary Condition using $d = 0.25$ at $t = 30.08s$

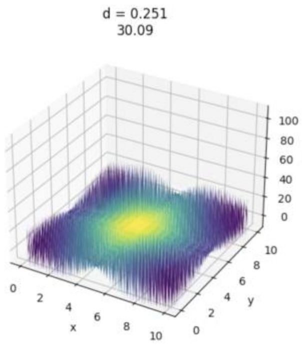


Figure 15 Diffusion Equation with Dirichlet Boundary Condition using $d = 0.251$ at $t = 30.09s$

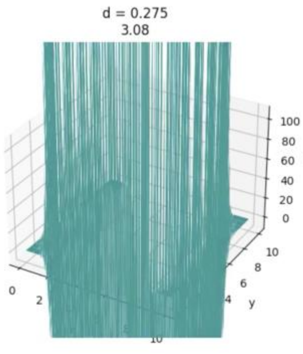


Figure 16 Diffusion Equation with Dirichlet Boundary Condition using $d = 0.275$ at $t = 3.08s$

Problem 2. Burger's Equation

Using forward-in-time and backward-in-space for the 1st derivative, and centered difference for the 2nd derivative, construct a numerical model for the Burger's equation. Decide your own initial conditions and internal parameter value, *v*.

$$\frac{\partial u}{\partial t} + a\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) = v\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

Discuss the following (add figures if necessary):

1. Derive and write (type or by hand) the discretized form of the equation above as an algebraic equation (10 points).
2. Investigate by modelling the differences when *a* (linear) is a constant and when *a* is *u* (non-linear). Use a cyclic boundary (10 points).
3. Investigate the behavior of *u* with time as you set a cyclic boundary at one axis and a close-wall boundary at another axis (i.e. no net flux) (10 points).

4. Derivation of the discretized form of the Burger's Equation

From the equation

$$\frac{\partial U}{\partial t} + a\left(\frac{\partial U}{\partial x} + \frac{\partial U}{\partial y}\right) = v\left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2}\right)$$

And these equations

$$\frac{\partial U}{\partial t} = \frac{U_{x,y}^{n+1} - U_{x,y}^{n}}{\Delta t}$$

$$\frac{\partial U}{\partial x} = \frac{U_{x,y}^{n} - U_{x-1,y}^{n}}{\Delta x}$$

$$\frac{\partial U}{\partial y} = \frac{U_{x,y}^{n} - U_{x,y-1}^{n}}{\Delta y}$$

$$\frac{\partial^2 U}{\partial x^2} = \frac{U_{x+1,y}^{n} + U_{x-1,y}^{n} - 2U_{x,y}^{n}}{\Delta x^2}$$

$$\frac{\partial^2 U}{\partial y^2} = \frac{U_{x,y+1}^{n} + U_{x,y-1}^{n} - 2U_{x,y}^{n}}{\Delta y^2}$$

We can derive the algebraic equation as below (assuming $\Delta x = \Delta y$)

$$U_{x,y}^{n+1} = U_{x,y}^{n} - \frac{a\Delta t}{\Delta x}\left(2U_{x,y}^{n} - U_{x-1,y}^{n} - U_{x,y-1}^{n}\right) + \frac{v\Delta \mathrm{t}}{\Delta x^2}\left(U_{x+1,y}^{n} + U_{x-1,y}^{n} + U_{x,y+1}^{n} + U_{x,y-1}^{n} - 4U_{x,y}^{n}\right)$$

If we substitute $\frac{a\Delta t}{\Delta x}$ with d_1 and $\frac{v\Delta t}{\Delta x^2}$ with d_2, the equation can be re-written as

$$U_{x,y}^{n+1} = U_{x,y}^{n} - d_1\left(2U_{x,y}^{n} - U_{x-1,y}^{n} - U_{x,y-1}^{n}\right) + d_2\left(U_{x+1,y}^{n} + U_{x-1,y}^{n} + U_{x,y+1}^{n} + U_{x,y-1}^{n} - 4U_{x,y}^{n}\right)$$

5. Modelling of Burger Equation using Cyclic Boundary Condition

The Burger equation is the combined of advection equation and diffusion equation. The coefficient of advection term $a\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right)$ is the term that we vary to see the difference. For a constant a, the equation is linear and for $a = u$ the equation is non-linear. The figures below show the behavior of burger equation with constant a and $a = u$. By observing, we can see that the behavior of burger equation is the combination of the characteristic of advection equation and the characteristic of diffusion equation. As the whole shape move to the right, it also diffuse into surrounded space. In the case that a is set to u, we can see that the peak moves faster when the value is higher. If we carefully look at the figure 20, we will see that the peak moves faster than its base, so the shape tilt to the right. As time passes, the value decreased due to diffusion, so the movement slows down. On the other hand, the peak in the figure with $a = u$ moves at constant velocity. As shown in the figure 21 and 23, the peak moves to the right and loop back to the left because of cyclic boundary condition.

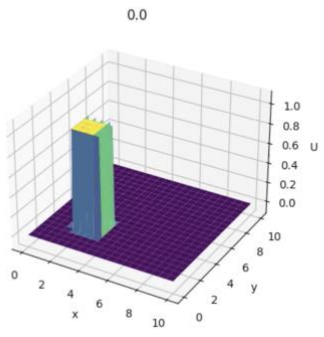


Figure 17 Burger Equation on Cyclic Boundary Condition with $a = 0.25$ at $t = 0.0s$

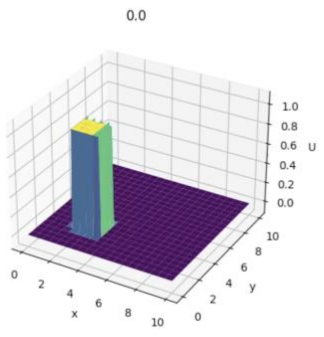


Figure 18 Burger Equation on Cyclic Boundary Condition with $a = u$ at $t = 0.0s$

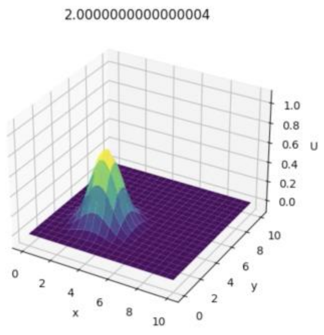


Figure 19 Burger Equation on Cyclic Boundary Condition with $a = 0.25$ at $t = 2.0s$

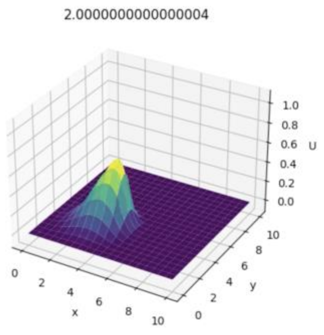


Figure 20 Burger Equation on Cyclic Boundary Condition with $a = u$ at $t = 2.0s$

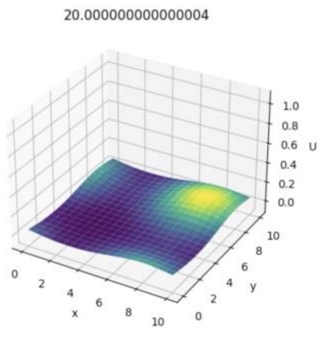


Figure 21 Burger Equation on Cyclic Boundary Condition with $a = 0.25$ at $t = 20.0s$

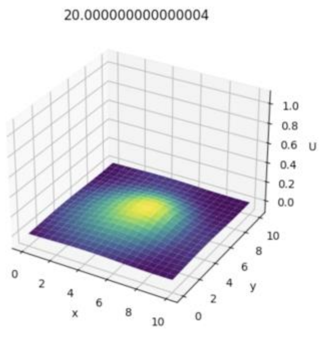
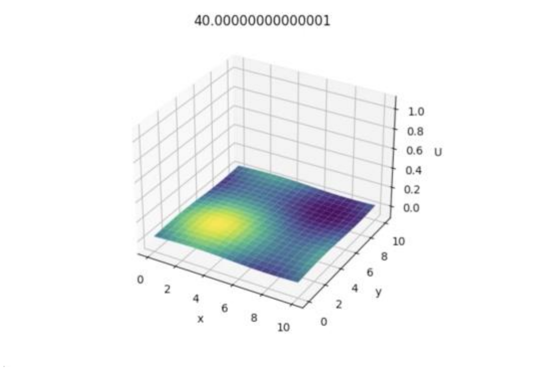


Figure 22 Burger Equation on Cyclic Boundary Condition with $a = u$ at $t = 20.0s$



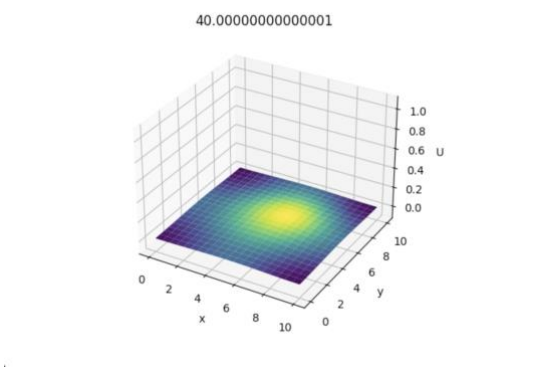


Figure 23 Burger Equation on Cyclic Boundary Condition with $a = 0.25$ at $t = 40.0s$

Figure 24 Burger Equation on Cyclic Boundary Condition with $a = u$ at $t = 40.0s$

6. Modelling of Burger Equation using Cyclic and Neumann Boundary Condition

Similar to the previous part, I implemented Burger equation using cyclic boundary condition along the x-axis and Neumann boundary condition along the y-axis. By using Neumann boundary condition along the y-axis, the net flux through the axis is equal to zero. The heat is moving in both x and y direction but cannot move through the Neumann boundary equation so after it hit the wall, it starts to move in x direction. Since the boundary on another axis is cyclic boundary condition, the heat can loop back to the other side of plane as shown in the figure 29. In the figure 31, there is a strip laying along y-axis. The strip is formed because the value of the peak decreases due to diffusion so the trail of the heat brighten up despite its small value.

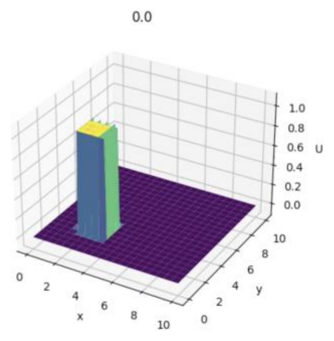


Figure 25 Burger Equation on Cyclic and Neumann Boundary Condition with $a = 0.25$ at $t = 0.0s$

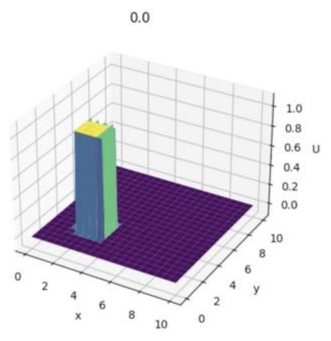


Figure 26 Burger Equation on Cyclic and Neumann Boundary Condition with $a = u$ at $t = 0.0s$

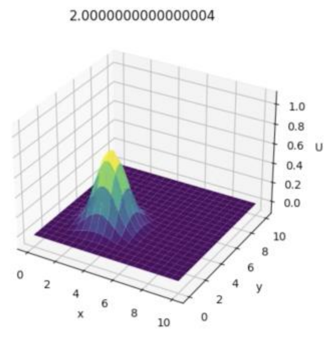


Figure 27 Burger Equation on Cyclic and Neumann Boundary Condition with $a = 0.25$ at $t = 2.0s$

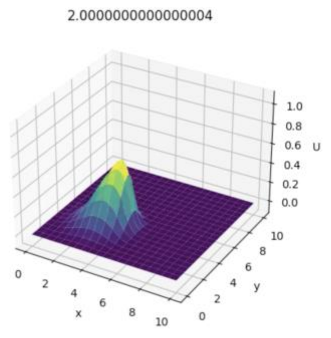


Figure 28 Burger Equation on Cyclic and Neumann Boundary Condition with $a = u$ at $t = 2.0s$

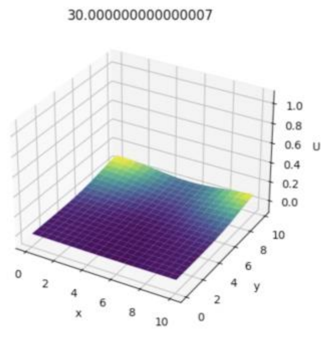


Figure 29 Burger Equation on Cyclic and Neumann Boundary Condition with $a = 0.25$ at $t = 30.0s$

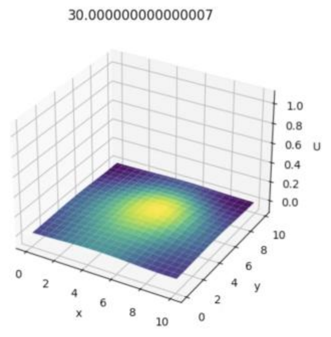


Figure 30 Burger Equation on Cyclic and Neumann Boundary Condition with $a = u$ at $t = 30.0s$

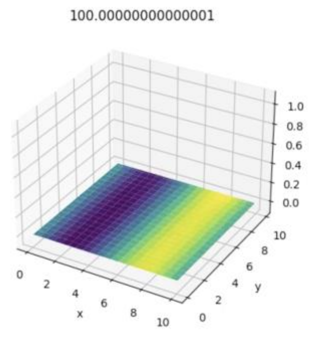


Figure 31 Burger Equation on Cyclic and Neumann Boundary Condition with $a = 0.25$ at $t = 100.0s$

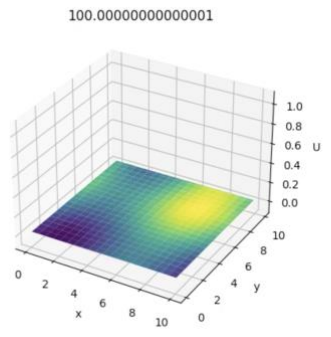


Figure 32 Burger Equation on Cyclic and Neumann Boundary Condition with $a = u$ at $t = 100.0s$

Problem 3. 1-D Advection Equation

Given the following 1-dimensional equation

$$\frac{\partial f}{\partial t} + u\left(\frac{\partial f}{\partial x}\right) = 0$$

At t=0,

$$f(0,x) = \begin{cases} 1 & 40 \leq x \leq 60 \\ 0 & \text{otherwise} \end{cases}$$

and with a cyclic boundary, discuss using the following parameters: $u = 1.0,\ \Delta x = 1.0,\ 0 \leq x \leq 100$. To answer the questions, decide on appropriate C values. Construct a model using (1) Upwind scheme (10 pts), (2) Leith's Method (10 pts), (3) CIP Method (10 pts), and (4) analytical solution (10 pts).

1. Construct f plots along x for t=10, 200, 400, 600. Compare the results of each method and discuss the errors accompanied by each method.

7. Construct a Model Using Upwind Scheme

- Program

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
from tqdm import tqdm
import os
os.system('mkdir pic')

def plot(x, y, t, path):
    fig, ax = plt.subplots()
    ax.set_xlim(0, 100)
    ax.set_ylim(-0.25, 1.50)
    ax.xaxis.set_major_locator(MultipleLocator(20))
    ax.yaxis.set_major_locator(MultipleLocator(0.25))
    ax.grid(which='major', color='#CCCCCC', linestyle='--')
    ax.plot(x, y, '-k')
    ax.set_xlabel('x')
    ax.set_ylabel('f')
    plt.title('t='+t)
    plt.savefig(path)
    plt.close()

C = 0.1
u = 1.0
dx = 1.0
span_x = 100.0
dt = C*dx/u
print("dt: ", dt)
end_time = 600.0
print("Total Time Intervals: ", end_time/dt)

x = np.arange(0.0, span_x+dx, dx)
f = np.zeros_like(x)
f[40:61] = 1.0
```

```
n = 0
for i in tqdm(np.arange(0.0,end_time+dt, dt)):
        if n%10==0 and i <= 100.0:
                plot(x,f, '%.2f'%(i),'./pic/%04.0f.png'%(i) )
        n+=1
        f = f + C*(np.roll(f,1,axis=0)-f)

```

- Discuss

The equation for upwind scheme is $f_i^{n+1} = f_i^n + \frac{u\Delta t}{\Delta x}(f_{i-1}^n - f_i^n)$. I constructed a function for plotting figure to clean the code.

8. Construct a Model Using Leith's Method

- Program

```
c = f
b = 0.5/dx*(np.roll(f,-1,axis=0)-np.roll(f,1,axis=0))
a = 0.5/dx/dx*(np.roll(f,-1,axis=0)+np.roll(f,1,axis=0)-2.0*f)
f = a*(u*dt)**2-b*(u*dt)+c
```

- Discuss

The source code of the modelling of Leith's method is mostly the same as upwind scheme. The only difference is the equation used for determining next term. The equation of Leith's method are as following:

$$f_i^{n+1} = a_i(u\Delta t)^2 - b_i(u\Delta t) + c_i$$

Where

$$a_i = \frac{1}{2\Delta x^2}(f_{i+1}^n + f_{i-1}^n - 2f_i^n),$$

$$b_i = \frac{1}{2\Delta x}(f_{i+1}^n - f_{i-1}^n),$$

and

$$c_i = f_i^n$$

9. Construct a Model Using CIP Method

- Program

```
signu = np.int(np.sign(u))
xiiup = -signu*dx
E = -u*dt
a = -2*(np.roll(f,signu,axis=0)-
f)/xiiup**3+(g+np.roll(g,signu,axis=0))/xiiup**2
b = -3*(f-np.roll(f,signu,0))/xiiup**2-
(2*g+np.roll(g,signu,0))/xiiup
f = a*E**3+b*E**2+g*E+f
g = 3*a*E**2+2*b*E+g
```

- Discuss

The modelling for CIP method is also the same as two methods above. The equation that are used to calculate the next term are the following:

$$f_i^{n+1} = a_i\xi^3 + b_i\xi^2 + g_i^n\xi + f_i^n$$

$$g_i^{n+1} = 3a_i\xi^2 + 2b_i\xi + g_i^n$$

Where

$$a_i = -\frac{2\left(f_{iup}^n - f_i^n\right)}{\Delta x_{i\to iup}^3} + \frac{\left(g_i^n + g_{iup}^n\right)}{\Delta x_{i\to iup}^2}$$

$$b_i = -\frac{3\left(f_i^n - f_{iup}^n\right)}{\Delta x_{i\to iup}^2} - \frac{\left(2g_i^n + g_{iup}^n\right)}{\Delta x_{i\to iup}}$$

and

$$\xi = -u\Delta t$$

10. Construct a Model Using Analytic Solution

- Program

```
u = 1.0
dx = 1.0
dt = 1.0
span_x = 100.0
print("dt: ", dt)
end_time = 600.0
```

```python
print("Total Time Intervals: ", end_time/dt)
x = np.arange(0.0, span_x+dx, dx)
f = np.zeros_like(x)
f[40:61] = 1.0

for i in tqdm(np.arange(0.0,end_time+dt, dt)):
        if i <= 100.0:
                plot(x,f, '%.2f'%(i),'./pic/%04.0f.png'%(i) )
        f = np.roll(f,1,axis=0)
```

- Discuss

The equation for analytic solution of advection equation is basically $f_i^{n+1} = f_{i-u\Delta x}^n$. Therefore, I let $\Delta x = 1.0$ to make the code easier to implement. Therefore $f_i^{n+1} = f_{i-1}^n$

11. Construct f plots along x for $t = 10, 200, 400, 600$ for each Model

- Discussion

Comparing the four methods of modelling, the obvious difference is the error of each method. The upwind scheme can move the shape accordingly to the value of u, but it cannot preserve the shape of the function, instead, the function behaves similar to diffusion. The Leith's method can do better in preserving the main feature or the peak of the figure as shown in the figure 40. However, the error grows into noises and make the figure look like a chaos. The CIP method is the best method among the numerical methods. The error occurs at the corners of the function but it still can preserve the important features of the function as shown in the figure 44. It is the most ideal method to use with digital signal which looks just like the function. Lastly, the analytical solution can predict the exact value of the function respect to the time without having any error.

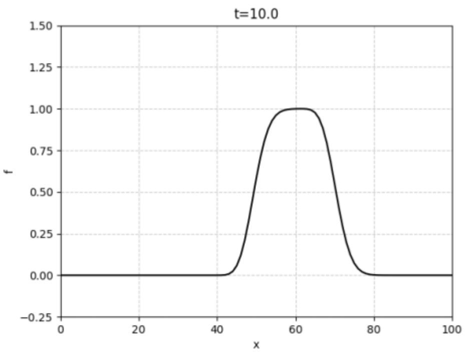


Figure 33 Advection Upwind Scheme at t=10.0s

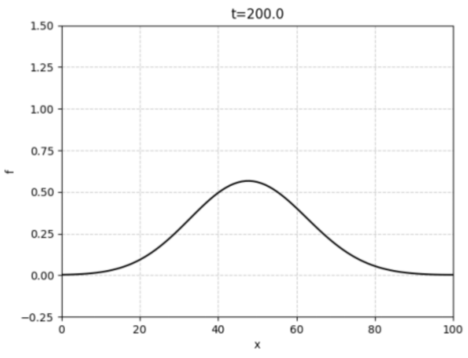


Figure 34 Advection Upwind Scheme at t=200.0s

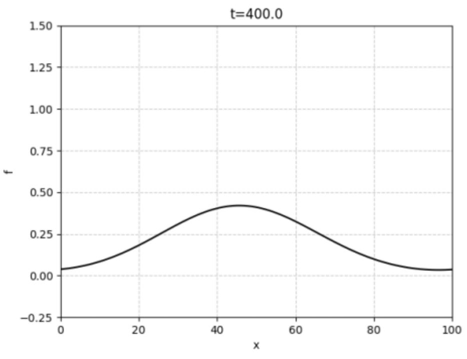


Figure 35 Advection Upwind Scheme at t=400.0s

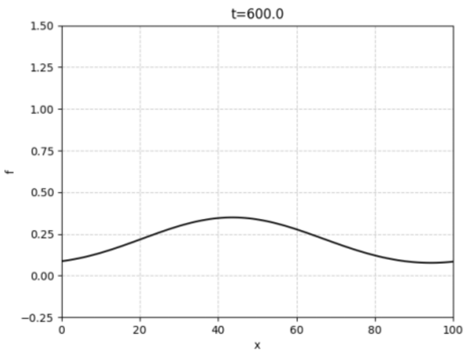


Figure 36 Advection Upwind Scheme at t=600.0s

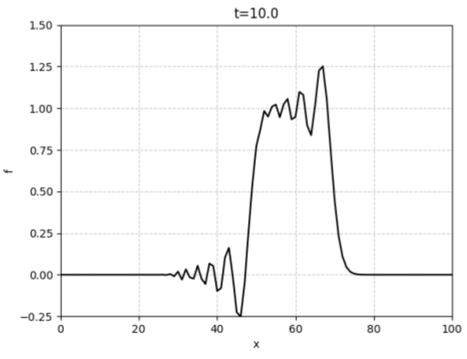


Figure 37 Advection Leith's Method at t=10.0s

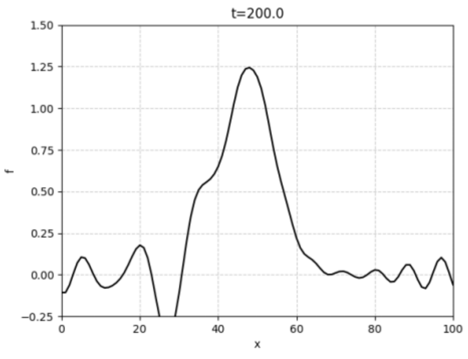


Figure 38 Advection Leith's Method at t=200.0s

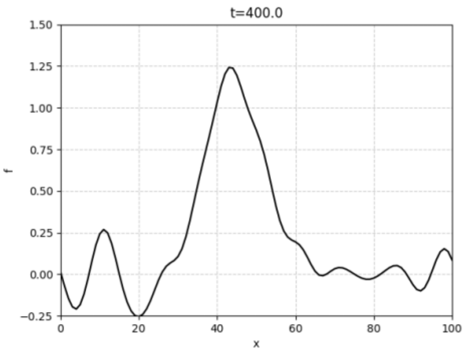


Figure 39 Advection Leith's Method at t=400.0s

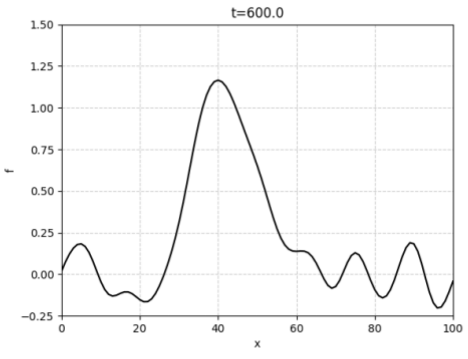


Figure 40 Advection Leith's Method at t=600.0s

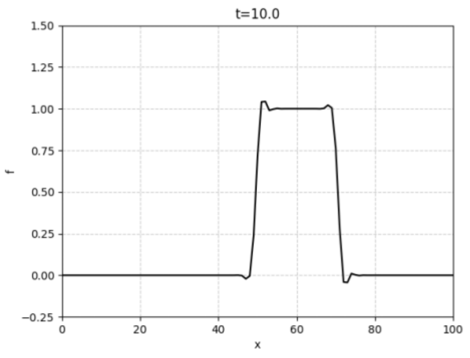


Figure 41 Advection CIP Method at t=10.0s

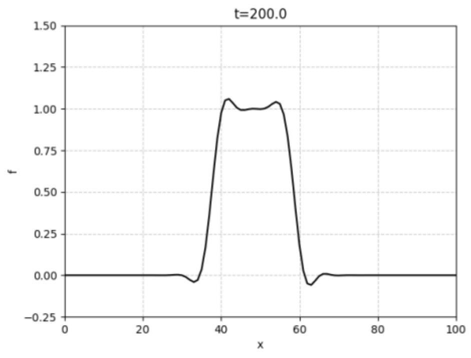


Figure 42 Advection CIP Method at t=200.0s

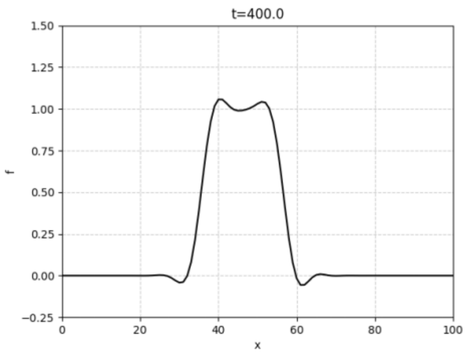


Figure 43 Advection CIP Method at t=400.0s

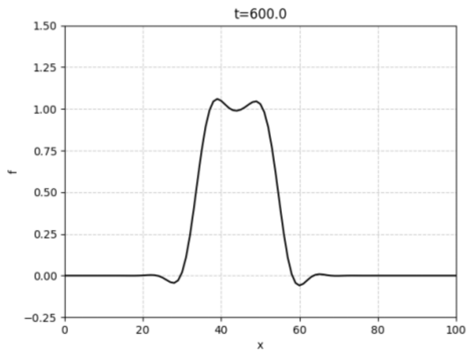


Figure 44 Advection CIP Method at t=600.0s

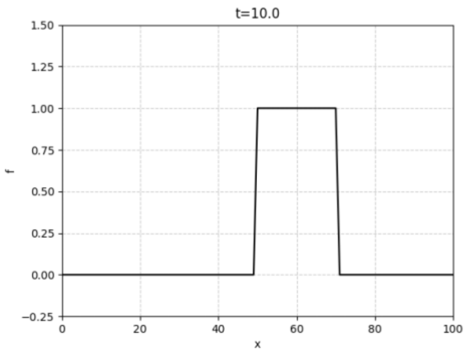


Figure 45 Advection Analytical Solution at t=10.0s

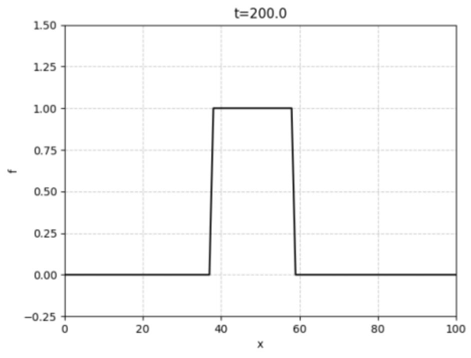


Figure 46 Advection Analytical Solution at t=200.0s

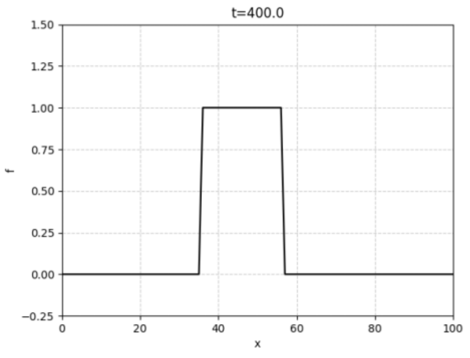


Figure 47 Advection Analytical Solution at t=400.0s

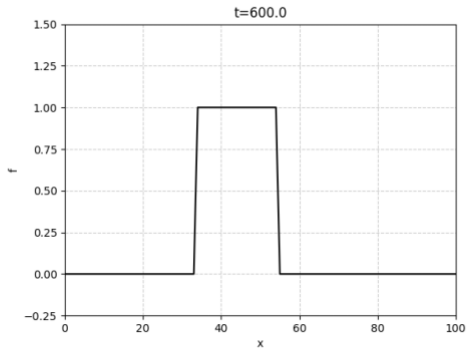


Figure 48 Advection Analytical Solution at t=600.0s