東京工業大学
Tokyo Institute of Technology

PARTIAL DIFFERENTIAL EQUATIONS FOR SCIENCE AND ENGINEERING

Final Report

DECEMBER 3, 2019

Chinchuthakun Worameth

18B00033

Table of Contents

Table of Figures

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). Use poisson equation by assigning external forcing (e.g. *g*).
2. Investigate various boundary conditions (non-steady state) by discussing its effects using animation:
   a. Dirichlet Boundary condition
   b. Neumann Boundary condition
   c. Mixed boundaries
3. Investigate what 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* for a stable model behavior? Why is this so?

A. Steady-State Condition of Possion Equation

Program

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d, Axes3D

#declare variables and initial conditions
alpha,g=1,9.81
xmax,ymax=100,100
dx=dy=2
nx=int(xmax/dx)
ny=int(ymax/dy)
x=np.linspace(0,nx*dx,nx)
y=np.linspace(0,ny*dy,ny)
T=np.zeros((nx,ny))
T[:,0]=1.;T[0,:]=1.;T[:,-1]=1.;T[-1,:]=1.

#construct and solve system of equations
A=np.zeros((nx*ny,nx*ny))
B=np.zeros(nx*ny)
for i in range(0,ny):
    for j in range(0,nx):
        if(i==0 or j==0 or i==ny-1 or j==nx-1):
            A[i*ny+j,i*ny+j]=1;B[i*ny+j]=T[i,j]
            continue
        A[i*ny+j,i*ny+j]=-4
        A[i*ny+j,i*ny+j+1]=1
        A[i*ny+j,i*ny+j-1]=1
        A[i*ny+j,i*ny+j-ny]=1
        A[i*ny+j,i*ny+j+ny]=1
        B[i*ny+j]=g/alpha
ans=np.linalg.solve(A,B)

#plot 3D graph
X,Y=np.meshgrid(x,y)
```

```
fig=plt.figure()
ax=fig.gca(projection='3d')
surface=ax.plot_surface(X,Y,ans.reshape(X.shape),cmap='Blues',vmin=np.amin(ans
),vmax=np.amax(ans))
ax.set_title('steady state')
ax.view_init(elev=10., azim=10.)
fig.colorbar(surface, shrink=1, aspect=5)
plt.savefig('1_1_final.png')
```

Discussion

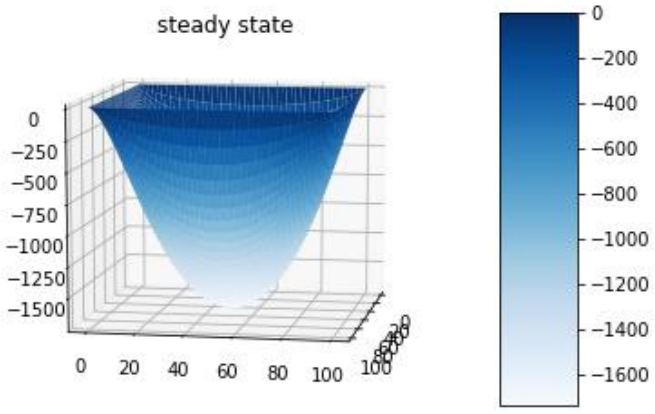

Figure 1 Steady-state of Possion equation (same fixed temperature, elev=10, azim=10)

The steady-state of the Possion equation represented by

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=g$$

with the Dirichlet boundary condition $f_0(0,x)=\begin{cases}1 & x=0\vee y=0\\ 0 & otherwise\end{cases}$ was investigated in the region $[0,100]\times[0,100]$.

The above equation represents an imaginary heat plate that has the temperature of 0^oC everywhere except at the boundaries which are always kept at 1^oC. The plate is exposed to an external heat source, $-g$ ^oC.; each point on the plate will receive the same amount of heat all the time. The steady-state of the heat plate is shown in figure 1 above. Note that because the boundaries must be kept at 1^oC all the time, the heat diffuses to neighbor points which eventually lead to the gradual decline of temperature with the minima at the center of the plate. If the boundaries are not fixed, every point in the plate must have the same amount of heat. Moreover, if the fixed temperatures at

each boundary are not equal, the location of the minima also shifts from the center as shown in figure 2 and figure 3.

We can also view this problem from another perspective; instead of a heat plate, we can view it as a curtain that the boundaries are hanged at the fixed height. The curtain is exposed to the effect of the gravity force; therefore, at the steady-state, it is depicted as shown in figure 1.

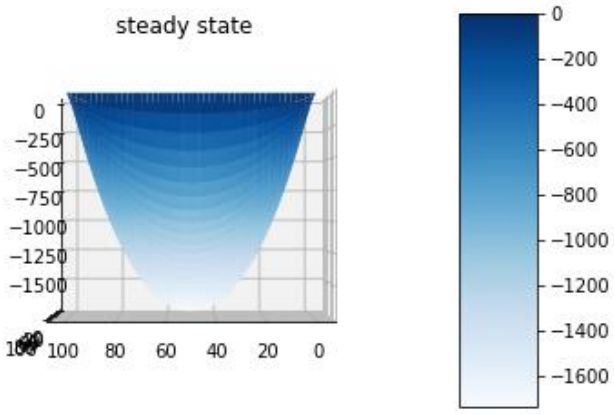

Figure 2 Steady-state of Possion equation (same fixed temperature, elev=0, azim=90)

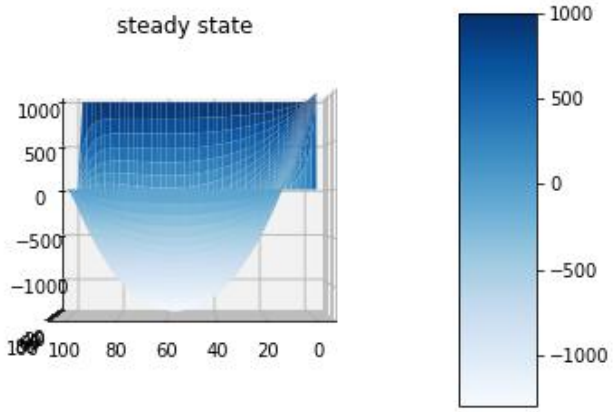

Figure 3 Steady-state of Possion equation (different fixed temperature, elev=0, azim=90)

B. Influence of Boundary conditions

Dirichlet Boundary condition

- Program

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  #declare variables and initial conditions
5.  d,alpha=0.2,10.
6.  xmax,ymax=100,100
7.  dx=dy=1
8.  loop=1000
9.  dt=d*(dx**2.)/alpha
10. nx=int(xmax/dx);ny=int(ymax/dy)
11. x=np.linspace(0,nx*dx,nx)
12. y=np.linspace(0,ny*dy,ny)
13. T=np.zeros((nx,ny))
14. T[:,0]=0.;T[0,:]=50.;T[:,-1]=0.;T[-1,:]=100.
15.
16. #Forward in time, Center in space
17. def fdm(T):
18.     T[1:ny-1,1:nx-1]=T[1:ny-1,1:nx-1]+d*(T[1:ny-1,2:nx]+T[1:ny-1,0:nx-2]-4*T[1
    :ny-1,1:nx-1]+T[2:ny,1:nx-1]+T[0:ny-2,1:nx-1])
19.     return T
20.
21. #time loop and plotting graph
22. cnt=0
23. for n in range(0,loop):
24.     plt.contourf(x,y,T,cmap='rainbow',vmin=0.,vmax=100)
25.     plt.colorbar()
26.     plt.title(n*dt)
27.     plt.savefig('2_dirichlet_%05.5d.png'%(cnt))
28.     plt.clf()
29.     plt.cla()
30.     T=fdm(T)
31.     cnt+=1
```

- Discussion

The steady-state of the Possion equation represented by

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = g$$

with the initial condition $T_0(x, y) = 0$, Dirichlet condition $T(0, y) = T(100, y) = 0$, $T(x,0) = 50$, and $T(x,100) = 100$ was investigated in region $[0,100] \times [0,100]$.

The equation above represents the simple diffusion of temperature generated by four heat sources with difference power located at the boundaries in the figure. According to the simulation, the heat generated by the high power heat source is transferred faster than the low power one because the difference between temperature is greater as shown the figure 4. Moreover, the heat distribution frontiers are cursive because of the left and right boundaries temperatures are kept at constant.

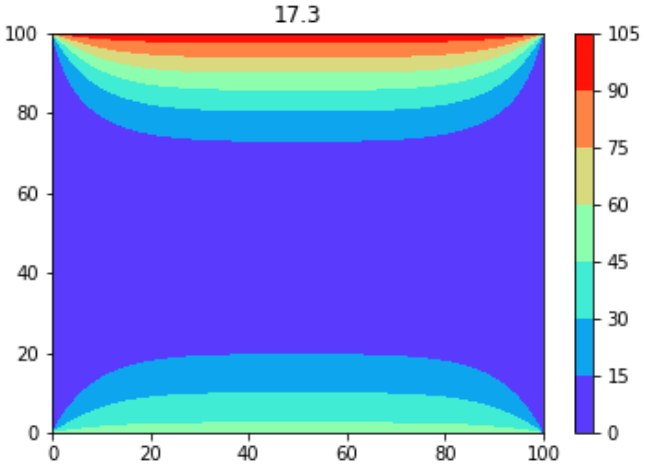

Figure 4 Heat distributions from two different sources at time 17.3

Neumann Boundary condition

- Program

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. #declare variables and initial conditions
5. d,alpha=0.2,10.
6. xmax,ymax=100,100
7. dx=dy=1
8. loop=1000
9. dt=d*(dx**2.)/alpha
10. nx=int(xmax/dx);ny=int(ymax/dy)
11. x=np.linspace(0,nx*dx,nx)
12. y=np.linspace(0,ny*dy,ny)
13. T=np.zeros((nx,ny))
14. T[1:ny-1,1:nx-1]=100.
15.
16. #Forward in time, Center in space
17. def fdm(T):
18.     T[1:ny-1,0]=T[1:ny-1,0]+d*(T[1:ny-1,1]+T[1:ny-1,2]-4*T[1:ny-1,0]+T[0:ny-2,
    0]+T[2:ny,0]-2*dx*(-10))
19.     T[1:ny-1,nx-1]=T[1:ny-1,nx-1]+d*(T[1:ny-1,nx-2]+T[1:ny-1,nx-2]-4*T[1:ny-1,
    nx-1]+T[0:ny-2,nx-1]+T[2:ny,nx-1]-2*dx*(8))
20.     T[ny-1,1:nx-1]=T[ny-1,1:nx-1]+d*(T[ny-1,2:nx]+T[ny-1,0:nx-2]-4*T[ny-1,1:nx
    -1]+T[ny-2,1:nx-1]+T[ny-2,1:nx-1]-2*dx*(6))
21.     T[0,1:nx-1]=T[0,1:nx-1]+d*(T[0,2:nx]+T[0,0:nx-2]-4*T[0,1:nx-1]+T[2,1:nx-1]
    +T[1,1:nx-1]-2*dx*(-10))
22.     T[1:ny-1,1:nx-1]=T[1:ny-1,1:nx-1]+d*(T[1:ny-1,2:nx]+T[1:ny-1,0:nx-2]-4*T[1
    :ny-1,1:nx-1]+T[2:ny,1:nx-1]+T[0:ny-2,1:nx-1])
23.     return T
24.
25. #time loop and plotting graph
26. cnt=0
27. for n in range(0,loop):
28.     plt.contourf(x,y,T,cmap='rainbow')
```

```
29.     plt.clim(-100,300)
30.     plt.colorbar()
31.     plt.title(n*dt)
32.     plt.savefig('2_neumann_%05.5d.png'%(cnt))
33.     plt.clf()
34.     plt.cla()
35.     T=fdm(T)
36.     cnt+=1
```

- Discussion

The steady-state of the Possion equation represented by

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=g$$

with the initial condition $T_0(x,y)=100$ and Neumann boundary condition $\frac{\partial}{\partial y}(100,y)=-8$ $\frac{\partial}{\partial y}(0,y)=\frac{\partial}{\partial x}(x,0)=10$, and $\frac{\partial}{\partial x}(x,100)=-6$ was investigated in region $[0,100]\times[0,100]$.

The above equation represents a control volume that has influx and outflux of a quantity along the x-axis and y-axis. In the case of the heat plate, that quantity is heat as the name suggested. At the coordinate $(100,0)$ (down right corner) and $(0,100)$ (upper left corner), the influx and outflux virtually cancel each other out; therefore, the temperature at those regions remain roughly the same as shown in figure 5. Note that because of the difference between outflux in the x-axis and y-axis, the cold region does not symmetrically expand about the line $y=x$. The effect is clearly contrasted with the equal influxes of heat; resulting in symmetrically expansion about $y=x$ of the hot region.

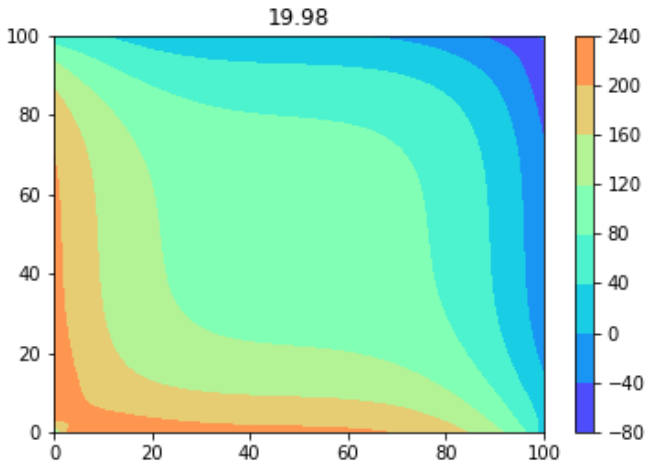

Figure 5 Expansion of hot and cold region at time 19.98

Mixed boundary

- Program

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. #declare variables and initial conditions
5. d,alpha=0.2,10.
6. xmax,ymax=100,100
7. dx=dy=1
8. loop=1000
9. dt=d*(dx**2.)/alpha
10. nx=int(xmax/dx);ny=int(ymax/dy)
11. x=np.linspace(0,nx*dx,nx)
12. y=np.linspace(0,ny*dy,ny)
13. T=np.zeros((nx,ny))
14. T[:,:]=20.
15. T[0,0:30]=35.
16.
17. #Forward in time, Center in space
18. def fdm(T):
19.     T[1:ny-1,0]=T[1:ny-1,0]+d*(T[1:ny-1,1]+T[1:ny-1,2]-4*T[1:ny-1,0]+T[0:ny-2,
    0]+T[2:ny,0]-2*dx*(0.2))
20.     T[1:ny-1,nx-1]=T[1:ny-1,nx-1]+d*(T[1:ny-1,nx-2]+T[1:ny-1,nx-2]-4*T[1:ny-1,
    nx-1]+T[0:ny-2,nx-1]+T[2:ny,nx-1]-2*dx*(0.2))
21.     T[ny-1,1:nx-1]=T[ny-1,1:nx-1]+d*(T[ny-1,2:nx]+T[ny-1,0:nx-2]-4*T[ny-1,1:nx
    -1]+T[ny-2,1:nx-1]+T[ny-2,1:nx-1]-2*dx*(0.2))
22.     T[0,30:nx-1]=T[0,30:nx-1]+d*(T[0,31:nx]+T[0,29:nx-2]-4*T[0,30:nx-1]+T[2,30
    :nx-1]+T[1,30:nx-1]-2*dx*(2))
23.     T[1:ny-1,1:nx-1]=T[1:ny-1,1:nx-1]+d*(T[1:ny-1,2:nx]+T[1:ny-1,0:nx-2]-4*T[1
    :ny-1,1:nx-1]+T[2:ny,1:nx-1]+T[0:ny-2,1:nx-1])
24.     return T
25.
26. #time loop and plotting graph
27. cnt=0
28. for n in range(0,loop):
```

```
29.     plt.contourf(x,y,T,cmap='rainbow',vmin=0.,vmax=40)
30.     plt.colorbar()
31.     plt.title(n*dt)
32.     plt.savefig('2_mixed_%05.5d.png'%(cnt))
33.     plt.clf()
34.     plt.cla()
35.     T=fdm(T)
36.     cnt+=1
```

- Discussion

The steady-state of the Possion equation represented by

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=g$$

with the initial condition $T_0(x,y)=20$, Dirichlet condition $T(x,0)=35, \forall x\in[0,30)$

Neumann boundary condition $\frac{\partial}{\partial y}(0,y)=\frac{\partial}{\partial x}(100,y)=\frac{\partial}{\partial x}(x,100)=0.2$ and

$\frac{\partial}{\partial x}(x,0)=2, \forall x\in[30,100]$ was investigated in region $[0,100]\times[0,100]$.

The above equation represents, in fact, a simplified version of the structure of my room. In this simplified version, one side of the room is the heater, occupied 30% of the space. Another 70% is the mirror with a high heat loss rate. Note that in the 3-dimensional world, there is an insulated wall below; however, for simplicity, the heat loss is ignored. The other 3 sides of the room are insulated walls with extremely low heat loss rate.

Assume that the temperature at the beginning in the room is uniform, after turn on the heater for a certain period of time, the heat distribution is acquired as shown in figure 7. In the region around the coordinate $(30,0)$, the collision between the hot and cold region occurs; therefore, the heat remains virtually the same. Another interesting observation is that the heat region expands rapidly at the beginning and gradually becomes slower as the time progress as shown in figure 6. Hence, it is possible that the power of the heater is not high enough to warm the room.

However, the model is still far from being perfect; there are many factors neglected in the model. One major factor is the heat loss rate of materials. In fact, the heat loss rate should be a function of the difference between inside and outside; in other words, it should be governed by the law of heat transfer. Because of the constant heat loss rates, a strange thing occurs as the time progress; below zero temperatures as shown in figure 7.

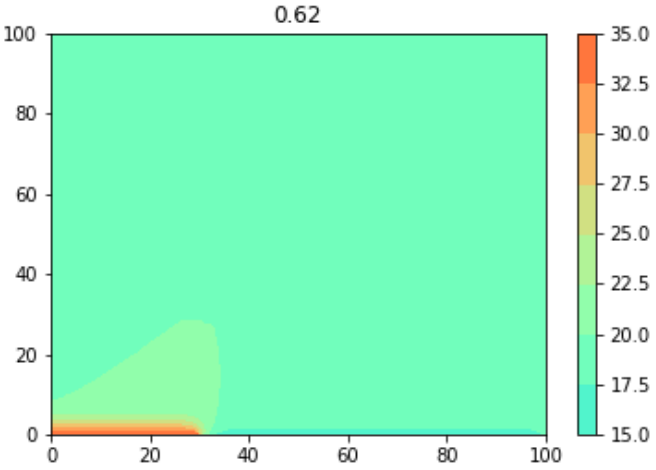

Figure 6 Heat distributions in the room at time 0.62

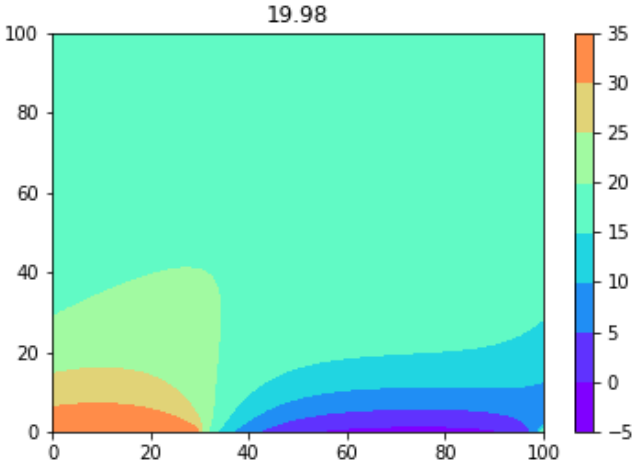

Figure 7 Heat distributions in the room at time 19.98

C. Influence of value of d

Program

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  #declare variables and initial conditions
5.  #d can be adjusted here
6.  d,alpha=0.26,10.
7.  xmax,ymax=100,100
8.  dx=dy=1
9.  loop=1000
10. dt=d*(dx**2.)/alpha
11. nx=int(xmax/dx);ny=int(ymax/dy)
12. x=np.linspace(0,nx*dx,nx)
13. y=np.linspace(0,ny*dy,ny)
14. T=np.zeros((nx,ny))
15. T[:,0]=0.;T[0,:]=50.;T[:,-1]=0.;T[-1,:]=100.
16.
17. #Forward in time, Center in space
18. def fdm(T):
19.     T[1:ny-1,1:nx-1]=T[1:ny-1,1:nx-1]+d*(T[1:ny-1,2:nx]+T[1:ny-1,0:nx-2]-4*T[1
    :ny-1,1:nx-1]+T[2:ny,1:nx-1]+T[0:ny-2,1:nx-1])
20.     return T
21.
22. #time loop and plotting graph
23. cnt=0
24. for n in range(0,loop):
25.     plt.contourf(x,y,T,cmap='rainbow',vmin=0.,vmax=100)
26.     plt.colorbar()
27.     plt.title(n*dt)
28.     plt.savefig('3_3_%05.5d.png'%(cnt))
29.     plt.clf()
30.     plt.cla()
31.     T=fdm(T)
32.     cnt+=1
```

Discussion

The threshold values for d can be derived by using Von Neumann stability analysis to the equation

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

by setting $\Delta x = \Delta y$. We obtain

$$\frac{\varepsilon_{ij}^{n+1} - \varepsilon_{ij}^{n}}{\Delta t} = \alpha\left(\frac{\varepsilon_{(i+1)j}^{n} + \varepsilon_{i(j+1)}^{n} - 4\varepsilon_{ij}^{n} + \varepsilon_{(i-1)j}^{n} + \varepsilon_{i(j-1)}^{n}}{\Delta x^2}\right)$$

By applying Fourier series in the similar manner with the case of 1-dimensional, we obtain

$$d = \frac{\alpha \Delta t}{\Delta x^2} \leq 0.25$$

The simulation was carried out to verify the analysis by running a program of sub-problem 2 which investigates the influence of Dirichlet boundary conditions and setting the value of d to 0.26 and 1.00. Naturally, both simulations yield strange results; however, the time required for each of them is different as shown in figure 8 and figure 9. Since the model works fine for the value of d less than 0.25 as shown in the previous discussions, it can be concluded that the threshold value for d is 0.25.

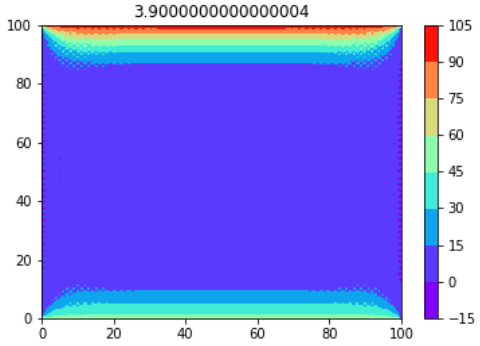

Figure 8 Strange behaviors at the boundaries at time 3.9 when d is 0.26

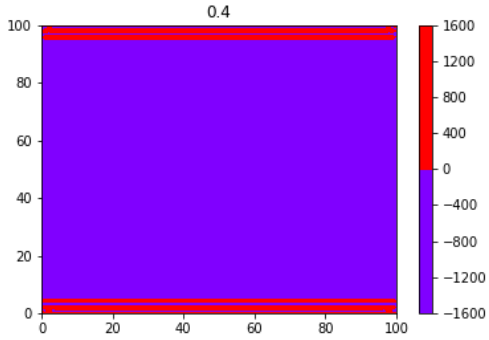

Figure 9 Strange behaviors at the boundaries at time 0.4 when d is 1.00

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 on your own initial conditions and values for *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 the discretization as an algebraic equation.
2. Investigate by modeling the differences when *a* (linear) is a constant and when *a* is *u* (non-linear) for a cyclic condition.
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 (no net flux)?

A. Discretization of the differential equation

By using forward-in-time and backward-in-space for the 1st derivative, and centered difference for the 2nd derivative, we can rewrite the equation

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

as

$$\frac{u_{ij}^{n+1}-u_{ij}^{n}}{\Delta t}+a\left(\frac{u_{ij}^{n}-u_{(i-1)j}^{n}}{\Delta x}+\frac{u_{ij}^{n}-u_{i(j-1)}^{n}}{\Delta y}\right)=v\left(\frac{u_{(i+1)j}^{n}-2u_{ij}^{n}+u_{(i-1)j}^{n}}{\Delta x^2}+\frac{u_{i(j+1)}^{n}-2u_{ij}^{n}+u_{i(j-1)}^{n}}{\Delta y^2}\right)$$

Then, by assuming $\Delta x=\Delta y$, we obtain

$$\frac{u_{ij}^{n+1}-u_{ij}^{n}}{\Delta t}+a\left(\frac{2u_{ij}^{n}-u_{(i-1)j}^{n}-u_{i(j-1)}^{n}}{\Delta x}\right)=v\left(\frac{u_{(i+1)j}^{n}+u_{i(j+1)}^{n}-4u_{ij}^{n}+u_{(i-1)j}^{n}++u_{i(j-1)}^{n}}{\Delta x^2}\right)$$

$$u_{ij}^{n+1}=u_{ij}^{n}+\Delta t\left[v\left(\frac{u_{(i+1)j}^{n}+u_{i(j+1)}^{n}-4u_{ij}^{n}+u_{(i-1)j}^{n}++u_{i(j-1)}^{n}}{\Delta x^2}\right)-a\left(\frac{2u_{ij}^{n}-u_{(i-1)j}^{n}-u_{i(j-1)}^{n}}{\Delta x}\right)\right]$$

B. Influence of the value of a

Program

- a is a constant (linear)

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  #declare variables and initial conditions
5.  v,a=0.1,0.1
6.  dt=0.1
7.  xmax,ymax=25,25
8.  dx=dy=1
9.  loop=750
10. nx=int(xmax/dx)
11. ny=int(ymax/dy)
12. x=np.linspace(0,nx*dx,nx)
13. y=np.linspace(0,ny*dy,ny)
14. T=np.zeros((nx,ny))
15. T[15,15]=1
```

```
16.
17. #Forward in time, Backward in Space, Center for 2nd Differentiation
18. def fdm(T):
19.     T=T+dt*(v*(np.roll(T,-1,axis=0)+np.roll(T,1,axis=0)+np.roll(T,-1,axis=1)+n
    p.roll(T,1,axis=1)-4*T)/(dx**2.)-a*(2*T-np.roll(T,1,axis=0)+np.roll(T,1,axis=1
    )/dx))
20.     return T
21.
22. #time loop and plotting graph
23. cnt=0
24. for n in range(0,loop):
25.     plt.contourf(x,y,T,cmap='rainbow',vmin=np.amin(T),vmax=np.amax(T))
26.     plt.colorbar()
27.     plt.title(n*dt)
28.     plt.savefig('burger_1_%05.5d.png'%(cnt))
29.     plt.clf()
30.     plt.cla()
31.     T=fdm(T)
32.     cnt+=1
```

- a is u (non-linear)

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. #declare variables and initial conditions
5. v,a=0.2,0.1
6. dt=0.1
7. xmax,ymax=15,15
8. dx=dy=1
9. loop=200
10. nx=int(xmax/dx)
11. ny=int(ymax/dy)
12. x=np.linspace(0,nx*dx,nx)
13. y=np.linspace(0,ny*dy,ny)
14. T=np.zeros((nx,ny))
15. T[8,8],T[8,9]=1,4
```

```
16.
17. #Forward in time, Backward in Space, Center for 2nd Differentiation
18. def fdm(T):
19.     T=T+dt*(v*(np.roll(T,-1,axis=0)+np.roll(T,1,axis=0)+np.roll(T,-1,axis=1)+n
    p.roll(T,1,axis=1)-4*T)/(dx**2.)-T*(2*T-np.roll(T,1,axis=0)+np.roll(T,1,axis=1
    )/dx))
20.     return T
21.
22. #time loop and plotting graph
23. cnt=0
24. for n in range(0,loop):
25.     plt.contourf(x,y,T,cmap='rainbow',vmin=np.amin(T),vmax=np.amax(T))
26.     plt.colorbar()
27.     plt.title(n*dt)
28.     plt.savefig('xburger_2x_%05.5d.png'%(cnt))
29.     plt.clf()
30.     plt.cla()
31.     T=fdm(T)
32.     cnt+=1
```

Discussion

From the convection-diffusion equation,

$$\frac{\partial c}{\partial t}+\nabla\cdot(ac)=\nabla\cdot(v\nabla c)+R$$

We can obtain the burger equation by assuming that there is no source or sink ($R=0$); considering only the effect of advection $\nabla\cdot(ac)$ and diffusion $\nabla\cdot(v\nabla c)$.

- ***a*** **is a constant**

The differential equation

$$\frac{\partial u}{\partial t}+\frac{1}{10}\left(\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}\right)=\frac{1}{10}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)$$

with the initial condition $u_0(x,y)=\begin{cases}1 & (x,y)=(15,15)\\ 0 & otherwise\end{cases}$ and cyclic boundary conditions was investigated in the region $[0,25]\times[0,25]$.

The above equation represents the diffusion of a quantity i.e. concentration, temperature subjected to a uniform velocity field. A real-life example is when we drop a sugar cube into a river with a

uniform velocity field. The sugar cube flows along with the water in the river while the sugar is diffused into the water. As a result, the shockwave pattern of concentration of sugar occurs as shown in figure 10. When visualizing the phenomena in the 3-dimensional, the height of the shockwave is gradually decreased which can be observed by comparing the color bar in figure 10 and 11. Note that the sugar cube is moving in the up-left direction in both figures.

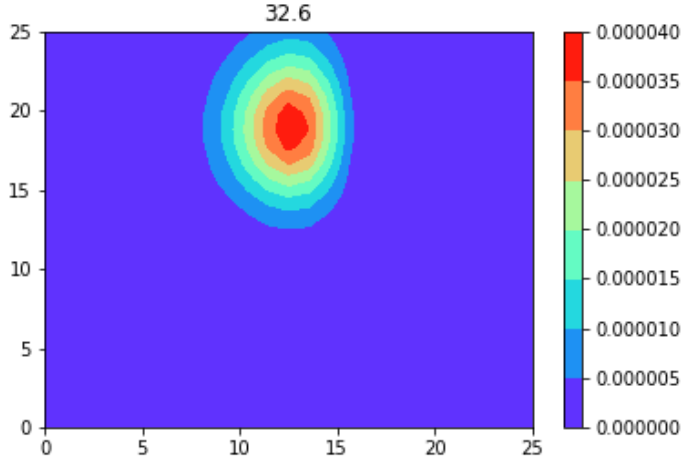

Figure 10 Shockwave pattern of concentration of sugar at time 32.6

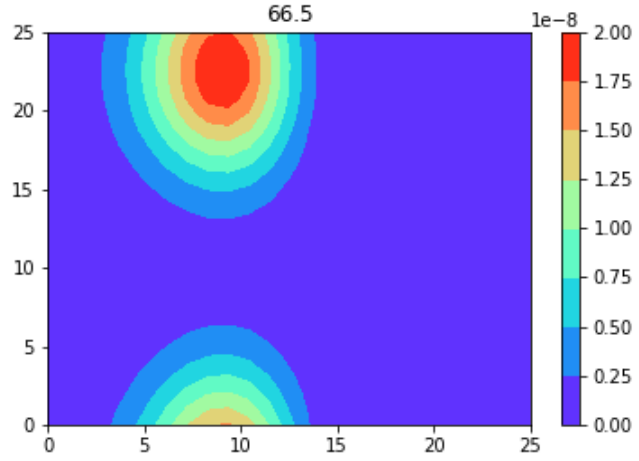

Figure 11 Shockwave pattern of concentration of sugar at time 66.5

- ***a* is *u***

The differential equation

$$\frac{\partial u}{\partial t}+u\left(\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}\right)=\frac{1}{5}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)$$

with the initial condition $u_0(x,y)=\begin{cases}4 & (x,y)=(8,9)\\ 1 & (x,y)=(8,8)\\ 0 & otherwise\end{cases}$ and cyclic boundary conditions was investigated in the region $[0,15]\times[0,15]$

We will continue to use the analogy of sugar cube to discuss this scenario as well; however, our sugar cube becomes polar sugar prism i.e. the concentration of sugar in both ends are not equal which aligned in the x-axis as shown in figure 12. Another major change is the velocity field is no longer uniform; it is now a function of the concentration of the sugar.

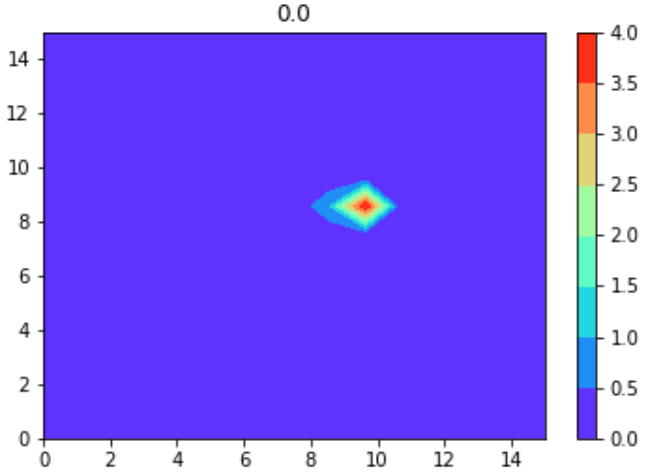

Figure 12 Concentration of sugar at time 0.0

While ignoring the effect of diffusion, because the concentration on the right end is much greater than the left end, the velocity field over the right end is greater than that of the left end. Therefore, as time progress, it can be observed that the concentration of both tails of the sugar prism is swapped as shown in figure 13. However, when taking the effect of diffusion into account, there is an additional force that drives the sugar in the right end toward the left end. As a result, the shape of sugar prism is changed drastically as shown in figure 14.

Because of the concentration of sugar at each point is extremely low, the effect of advection due to velocity field is negligible compared to the effect of diffusion. Therefore, in figure 15, the shockwave pattern is not observable.

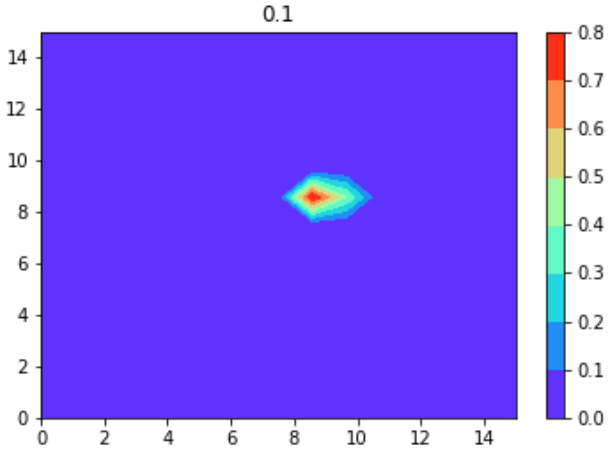

Figure 13 Concentration of sugar at time 0.1 (ignoring effect of diffusion)

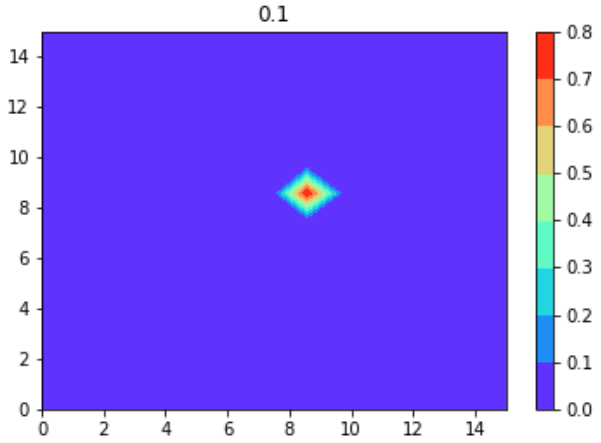

Figure 14 Concentration of sugar at time 0.1 (considering effect of diffusion)

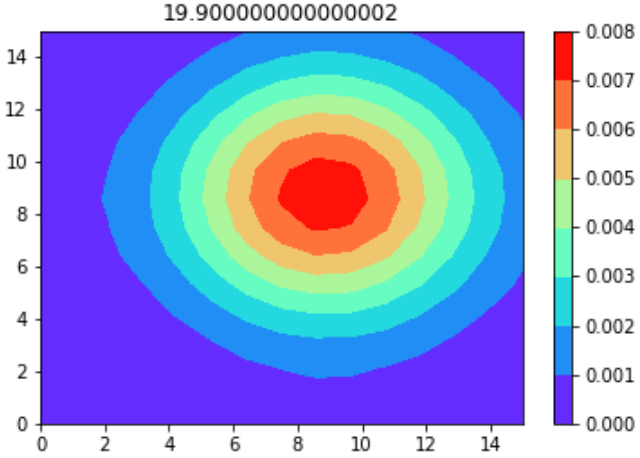

Figure 15 Concentration of sugar at time 19.9 (considering effect of diffusion)

C. Behavior of u

Program

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d, Axes3D

#declare variables and initial conditions
v,a=0.1,0.1
dt=0.1
xmax,ymax=25,25
dx=dy=1
loop=1000
nx=int(xmax/dx)
ny=int(ymax/dy)
x=np.linspace(0,nx*dx,nx)
y=np.linspace(0,ny*dy,ny)
T=np.zeros((nx,ny))
T[20,20]=1

#Forward in time, Backward in Space, Center for 2nd Differentiation
def fdm(T):
    T[0,:]=T[0,:]+dt*(v*(T[2,:]+T[1,:]+np.roll(T[0,:],-1,axis=0)+np.roll(T[0,:],1,axis=0)-4*T[0,:])/(dx**2.)-a*(2*T[0,:]-T[2,:]+np.roll(T[0,:],1,axis=0)/dx))
    T[ny-1,:]=T[ny-1,:]+dt*(v*(T[ny-2,:]+T[ny-2,:]+np.roll(T[ny-1,:],-1,axis=0)+np.roll(T[ny-1,:],1,axis=0)-4*T[ny-1,:])/(dx**2.)-a*(2*T[ny-1,:]-T[ny-2,:]+np.roll(T[ny-1,:],1,axis=0)/dx))
    T[1:ny-1,:]=T[1:ny-1,:]+dt*(v*(T[0:ny-2,:]+T[2:ny,:]+np.roll(T[1:ny-1,:],-1,axis=1)+np.roll(T[1:ny-1,:],1,axis=1)-4*T[1:ny-1,:])/(dx**2.)-a*(2*T[1:ny-1,:]-T[0:ny-2,:]+np.roll(T[1:ny-1,:],1,axis=1)/dx))
    return T

#time loop and plotting graph
cnt=0

```

```
X,Y=np.meshgrid(x,y)

for n in range(0,1):
    #2d plotting
    plt.contourf(x,y,T,cmap='rainbow',vmin=np.amin(T),vmax=np.amax(T))
    plt.colorbar()
    plt.title(n*dt)
    plt.savefig('burger_3_%05.5d.png'%(cnt))
    plt.clf()
    plt.cla()

    #3d plotting
    fig=plt.figure()
    ax=fig.gca(projection='3d')
    surface=ax.plot_surface(X,Y,T.reshape(X.shape),cmap='rainbow',vmin=np.amin
(T),vmax=np.amax(T))
    ax.set_title(n*dt)
    ax.view_init(elev=10., azim=45.)
    fig.colorbar(surface, shrink=1, aspect=5)
    plt.savefig('3d_burger_3_%05.5d.png'%(cnt))
    plt.cla()
    plt.clf()

    T=fdm(T)
    cnt+=1
```

Discussion

The differential equation

$$\frac{\partial u}{\partial t}+\frac{1}{10}\left(\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}\right)=\frac{1}{10}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)$$

with the initial condition $u_0(x,y)=\begin{cases}1 & (x,y)=(20,20)\\ 0 & otherwise\end{cases}$ and cyclic boundary conditions was investigated in the region $[0,25]\times[0,25]$.

The above equation represents the same situation with the sub-problem 2 of problem 2 when a is a constant; however, the fixed walls are added along the x-axis. When the sugar reaches the wall, the diffusion and advection in the wall direction are completely stopped since there is no net flux. Since the sugar cannot diffuse in the direction of the wall, the sugar diffuses in other directions instead; therefore, further accelerating the process of the diffusion as shown in figure 16 and 17.

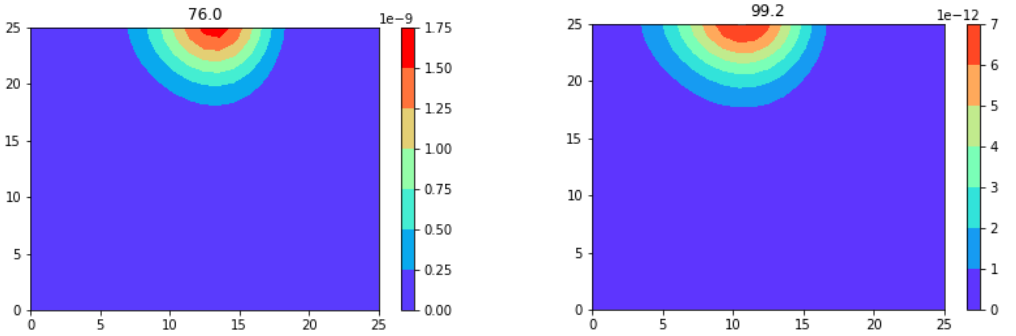

Figure 16 Concentration of sugar with fixed wall at time 76.0 and 99.2 (2-dimensional)

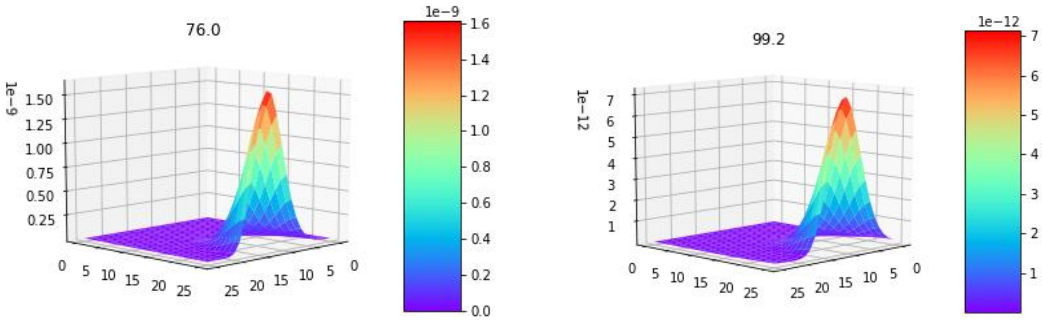

Figure 17 Concentration of sugar with fixed wall at time 76.0 and 99.2 (3-dimensional)

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,

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

and with a cyclic boundary, discuss using the following parameters: $u = 1.0, \Delta x = 1.0, 0 \le x \le 100$.

To answer the questions, decide on appropriate C values. Construct a model using (1) Upwind scheme, (2) Leith's Method, (3) CIP Method, and (4) analytical solution.

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

Note: Make sure the code is constructed neatly. Place comments using "#" character.

A. Error associated with each method

Program

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  #declare variables and choose Courant number
5.  c=.10
6.  dx,u=1.,1.
7.  dt=c*dx/u
8.  loop=int(600/dt+2)
9.  xmax=100
10. x=np.arange(0.,xmax+dx,dx)
11.
12. #initialize function / g is another function used in CIP method only.
13. f0=np.zeros(x.shape[0])
14. f0[(x>=40.)&(x<=60.)]=1.
15. f=np.zeros((4,x.shape[0]))
16. f[:]=f0
17. g=(f[3]-np.roll(f[3],1,axis=0))/dx
18.
19. #calculate iup and dxiup / xi is a variable used in CIP method only.
20. iup=-int(np.sign(u))
21. dxiup=iup*dx
22. xi=-1.*u*dt
23.
24. #analytical solution
25. def analytic(f0,t):
26.     f=np.roll(f0,int(u*t),axis=0)
27.     return f
28.
29. #upwind scheme
30. def upwind(f):
31.     f=f+u*dt*(np.roll(f,-1*iup,axis=0)-f)/dx
32.     return f
33.
```

```
#Leith method / 2nd order Lagrange Interpolation
def leith(f):
    a=(np.roll(f,-1,axis=0)+np.roll(f,1,axis=0)-2*f)/(2*dx**2.)
    b=(np.roll(f,-1,axis=0)-np.roll(f,1,axis=0))/(2*dx)
    c=f
    f=a*(u*dt)**2-b*u*dt+c
    return f

#CIP
def CIP(f,g):
    a=-2*(np.roll(f,-1*iup,axis=0)-f)/(dxiup)**3.+(g+np.roll(g,-1*iup,axis=0))/(dxiup**2.)
    b=-3*(f-np.roll(f,-1*iup,axis=0))/(dxiup)**2.-(2.*g+np.roll(g,-1*iup,axis=0))/dxiup
    f=a*xi**3.+b*xi**2.+g*xi+f
    g=3*a*xi**2.+2*b*xi+g
    return f,g

#information used when plotting graphs
cl=['black','red','green','blue']
method=['analytic','upwind','leith','CIP']
time=[10,200,400,600,loop*dt]
cnt=0

#time loop and plotting graph
for n in range(0,loop):
    if n*dt>=time[cnt]:
        plt.clf()
        plt.cla()
        fig=plt.figure()
        ax=fig.add_subplot(111)
        for i in range(0,4):
            ax.plot(x,f[i],color=cl[i],label=method[i])
        plt.ylim([0.,1.2])
        plt.xlim([0.,100.])
        plt.legend()
```

```
68.         plt.title('Time: %04.3f'%(n*dt))
69.         plt.savefig('advection_%04d.png'%(time[cnt]))
70.         cnt+=1
71.     f[0]=analytic(f0,n*dt)
72.     f[1]=upwind(f[1])
73.     f[2]=leith(f[2])
74.     f[3],g=CIP(f[3],g)
```

Discussion

The differential equation

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

with the initial condition $f_0(0,x)=\begin{cases}1 & 40\le x\le 60\\ 0 & otherwise\end{cases}$ and cyclic boundary conditions was investigated in the region $[0,100]$ by using 4 different approaches: upwind scheme, Leith's method, CIP method, and analytical solution with 4 different courant number: 0.1, 0.5, 0.9, and 1.0. Note that if the Courant number exceeds 1, the model becomes numerically unstable; therefore, there is no meaning to attempting so.

When the courant number is 1.0, all approaches work perfectly for long period of time as shown in figure 18. Note that solutions from 3 approaches are virtually the same; therefore, the lines overlap each other.

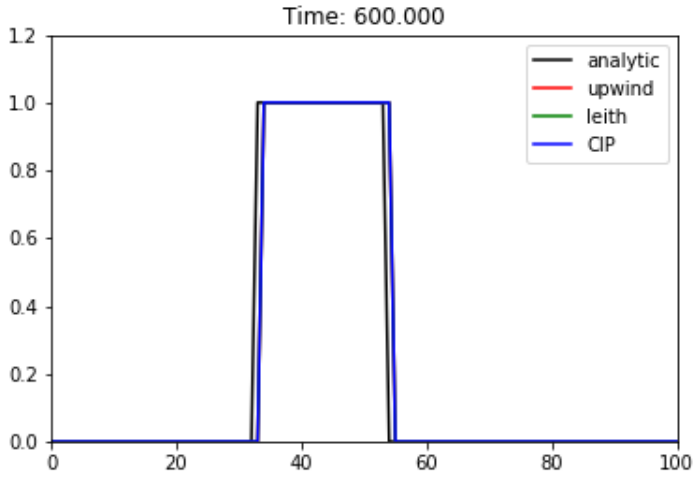

Figure 18 Result of simulations at time 600.000 (courant number is 1.0)

However, if the courant number is less than 1, there is always error presented in every approaches except the analytical one; though, different degrees. Approach can be ordered based on the presented errors; upwind scheme, Leith's method, and CIP from worst to best as shown in figure 19. Note that as time progress, the degree of errors increases for all approximation method as shown in figure 20.

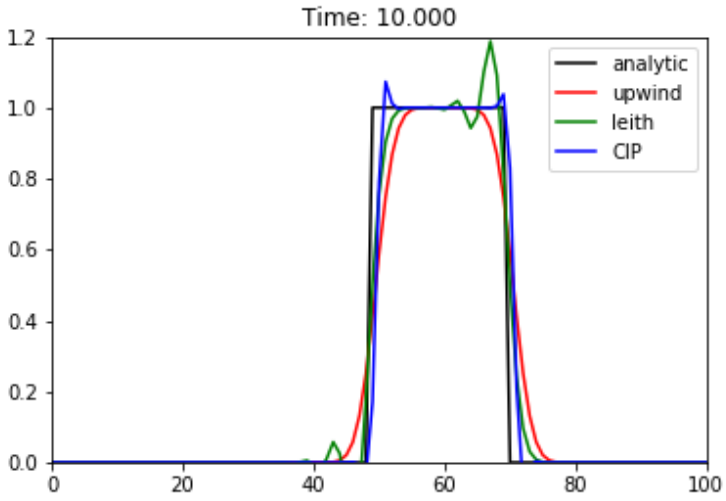

Figure 19 Result of simulations at time 10.000 (courant number is 0.50)

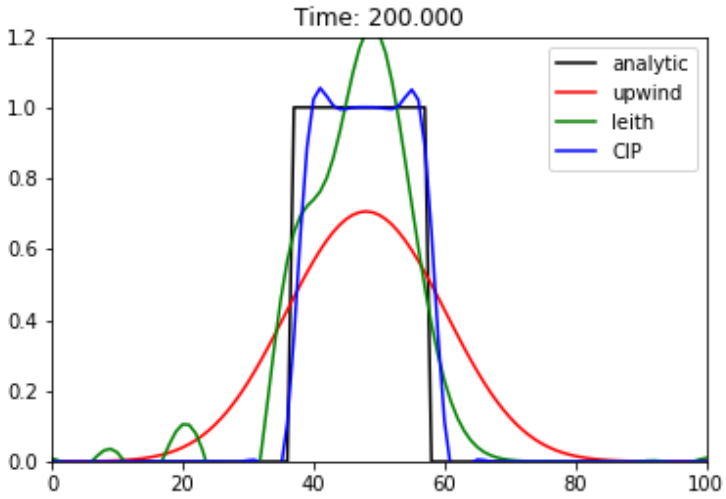

Figure 20 Result of simulations at time 200.97 (courant number is 0.50)

Another important observation is the degree of errors presented in the Leith's method and CIP method is largely different. Even if the courant number is extremely low, 0.1, the method still can approximate the solution relatively well compared to other approximation approaches.

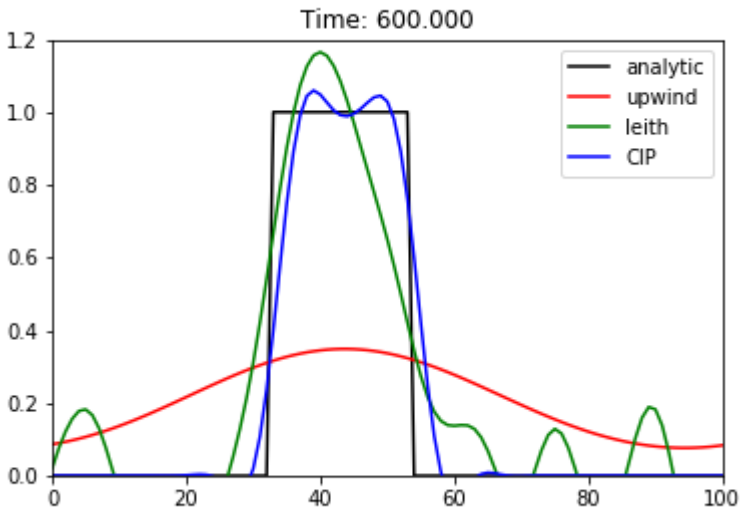

Figure 21 Result of simulations at time 600.000 (courant number is 0.10)