1. Problem description

Develop a computer program to solve the two-dimensional viscous scalar transport equation

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

Where u and v are the two Cartesian velocities and c is the scalar to be transported. Γ is the diffusivity. Use two values of Γ of 0.0. For both cases the initial scalar is given by

$$c(x,y,0)=\sin(5\pi(x-0.9))\sin(5\pi(y-.9))$$

for

$$0.9<x<1.10;0.90<y<1.10$$

Otherwise it equals zero at all other positions. For both cases u = 0.5 v = 0.5.

Integrate the above equation with four differencing schemes:

a) Upwind
b) MacCormack scheme
c) Leap frog
d) Lax scheme

For each scheme, plot the evolution of the surface (three-dimensional plot) of the scalar at four equally positioned time values. Consider a domain of 4 units x 4 units. On this, the signal will take 4 time units to go from x = 1.0 to x = 3.0 with the velocity of 0.5 units/sec. Use a grid of 321 x 321 finite difference nodes.

2. Numerical Scheme

i. Upwind Scheme

In the Upwind scheme, we use backward difference for $\frac{\partial c}{\partial x}$ and $\frac{\partial c}{\partial y}$ (for u>0, v>0):

$$u\frac{\partial c}{\partial x} \approx u\frac{c_{i,j}^n - c_{i-1,j}^n}{\Delta x}$$
$$v\frac{\partial c}{\partial y} \approx v\frac{c_{i,j}^n - c_{i,j-1}^n}{\Delta y}$$

Centered difference for $\frac{\partial^2 c}{\partial x^2}$ and $\frac{\partial^2 c}{\partial y^2}$:

$$\frac{\partial^2 c}{\partial x^2} \approx \frac{c_{i+1,j}^n - 2c_{i,j}^n + c_{i-1,j}^n}{\Delta x^2}$$
$$\frac{\partial^2 c}{\partial y^2} \approx \frac{c_{i,j+1}^n - 2c_{i,j}^n + c_{i,j-1}^n}{\Delta y^2}$$

Forward difference for $\frac{\partial c}{\partial t}$:

$$\frac{\partial c}{\partial t} \approx \frac{c_{i,j}^{n+1} - c_{i,j}^n}{\Delta t}$$

Thus the discretized equation is:

$$\frac{c_{i,j}^{n+1} - c_{i,j}^n}{\Delta t} + u\frac{c_{i,j}^n - c_{i-1,j}^n}{\Delta x} + v\frac{c_{i,j}^n - c_{i,j-1}^n}{\Delta y} = \Gamma\left(\frac{c_{i+1,j}^n - 2c_{i,j}^n + c_{i-1,j}^n}{\Delta x^2} + \frac{c_{i,j+1}^n - 2c_{i,j}^n + c_{i,j-1}^n}{\Delta y^2}\right)$$

Stability Condition for Upwind scheme is:

$$\Delta t \leq \frac{1}{\frac{u}{\Delta x} + \frac{v}{\Delta y} + \frac{2\Gamma}{\Delta x^2} + \frac{2\Gamma}{\Delta y^2}}$$

ii. MacCormack Scheme

MacCormack scheme uses a kind of predictor-corrector strategy:

Predictor:

$$\bar{c}_{i,j}^{n+1} = c_{i,j}^n - u\Delta t\frac{c_{i+1,j}^n - c_{i,j}^n}{\Delta x} - v\Delta t\frac{c_{i,j+1}^n - c_{i,j}^n}{\Delta y} + \Gamma\Delta t\left(\frac{c_{i+1,j}^n - 2c_{i,j}^n + c_{i-1,j}^n}{\Delta x^2} + \frac{c_{i,j+1}^n - 2c_{i,j}^n + c_{i,j-1}^n}{\Delta y^2}\right)$$

Corrector:

$$c_{i,j}^{n+1} = \frac{c_{i,j}^n + \bar{c}_{i,j}^{n+1}}{2} - \frac{u\Delta t}{2}\cdot\frac{\bar{c}_{i,j}^{n+1} - \bar{c}_{i-1,j}^{n+1}}{\Delta x} - \frac{v\Delta t}{2}\cdot\frac{\bar{c}_{i,j}^{n+1} - \bar{c}_{i,j-1}^{n+1}}{\Delta y}$$

Stability Condition for MacCormack scheme is following CFL stability criterion:

$$|\nu| = \left|\Delta t\left(\frac{u}{\Delta x} + \frac{v}{\Delta y}\right)\right| \leq 1$$

iii. Leap Frog Scheme

Leap Frog scheme is a 3 level time step scheme:

$$c_{i,j}^{n+1} = c_{i,j}^{n-1} + 2\Delta t\left[-u\frac{c_{i+1,j}^{n} - c_{i-1,j}^{n}}{2\Delta x} - v\frac{c_{i,j+1}^{n} - c_{i,j-1}^{n}}{2\Delta y} + \Gamma\left(\frac{c_{i+1,j}^{n} - 2c_{i,j}^{n} + c_{i-1,j}^{n}}{\Delta x^2} + \frac{c_{i,j+1}^{n} - 2c_{i,j}^{n} + c_{i,j-1}^{n}}{\Delta y^2}\right)\right]$$

In addition we assume that $c_{i,j}^{1} = c_{i,j}^{0}$.

Stability Condition for Leap Frog scheme is unconditionally unstable.

iv. Lax Scheme

In Lax scheme, $c_{i,j}^{n}$ is replaced by the spatial average of $c_{i,j}^{n}$ taken over the neighboring grid points:

$$c_{i,j}^{n} = \frac{c_{i+1,j}^{n} + c_{i-1,j}^{n} + c_{i,j+1}^{n} + c_{i,j-1}^{n}}{4}$$

Thus the discretized equation is:

$$\frac{c_{i,j}^{n+1} - \frac{c_{i+1,j}^{n} + c_{i-1,j}^{n} + c_{i,j+1}^{n} + c_{i,j-1}^{n}}{4}}{\Delta t} + u\frac{\left(c_{i+1,j}^{n} - c_{i-1,j}^{n}\right)}{2\Delta x} + v\frac{\left(c_{i,j+1}^{n} - c_{i,j-1}^{n}\right)}{2\Delta y} = \Gamma\left(\frac{c_{i+1,j}^{n} - 2c_{i,j}^{n} + c_{i-1,j}^{n}}{\Delta x^2} + \frac{c_{i,j+1}^{n} - 2c_{i,j}^{n} + c_{i,j-1}^{n}}{\Delta y^2}\right)$$

Stability Condition for Lax scheme is following CFL stability criterion:

$$|\nu| = \left|\Delta t\left(\frac{u}{\Delta x} + \frac{v}{\Delta y}\right)\right| \leq 1$$

3. Results

Based on given initial scalar, we can plot the initial condition of domain. In the following part, the configurations of plot are all same as the figure 1, x,y axis range was [0,1] and z axis range was set to [-1,1].

It should be noticed that although the color map of color bar in following plot are the same ("jet" scheme), however, the range of color bar is not uniform, which means that **the elevation of surface** of following plots is standardized and referenceable, but the **color of surface** of following plots is not standardized thus not referenceable.

Figure 1 shows that the initial condition (t=0s) of the surface of the scalar.

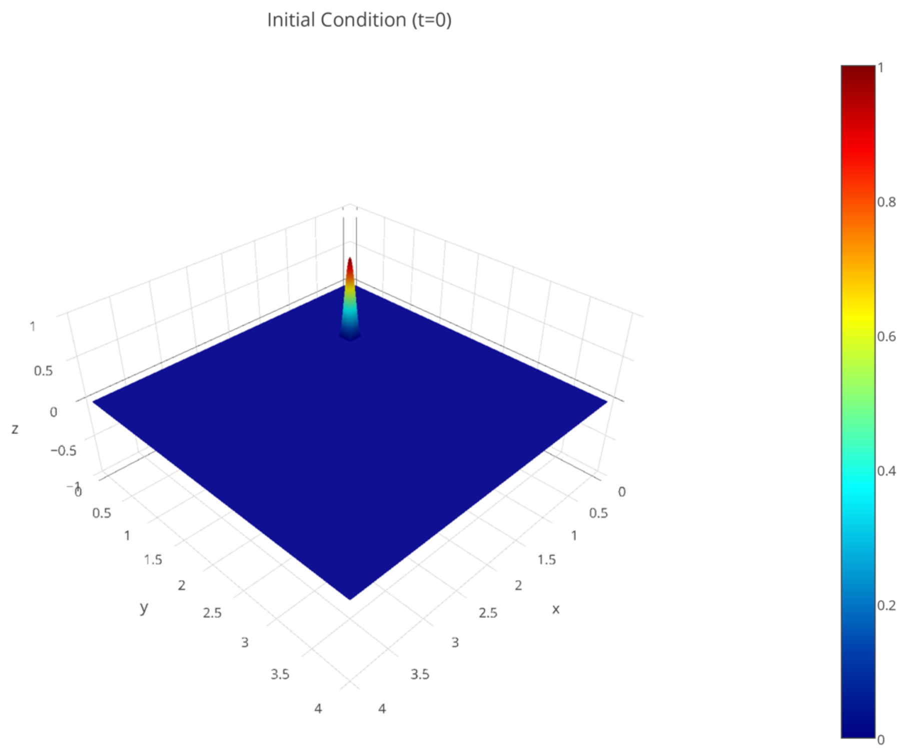
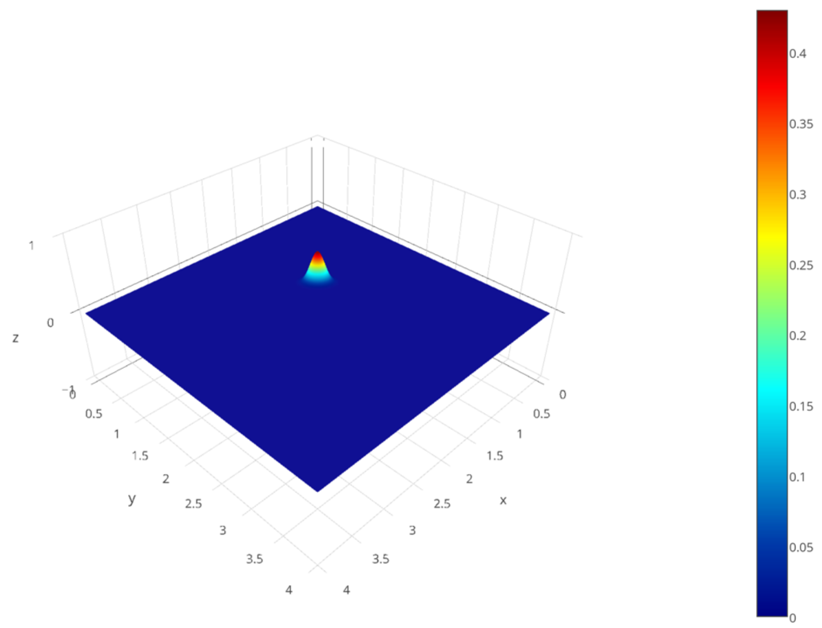


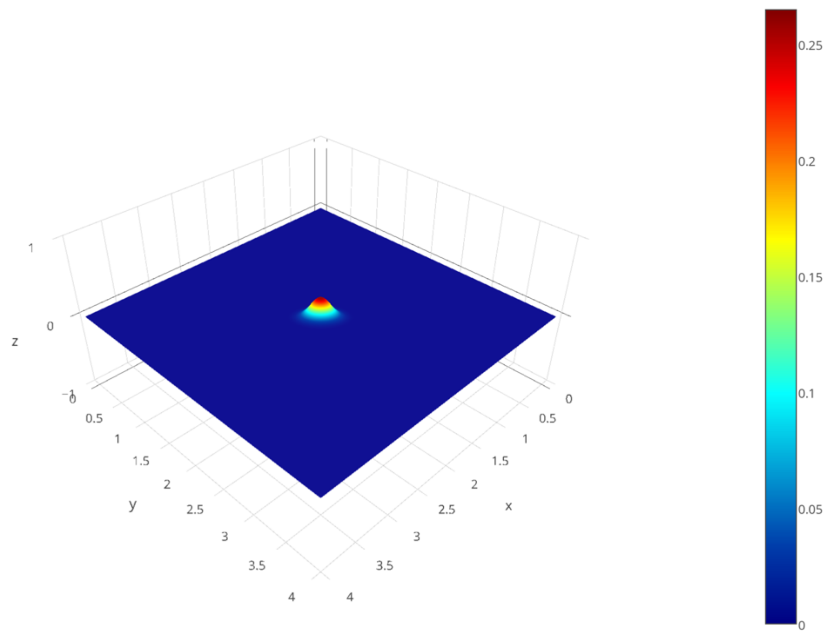
Figure 1. Initial condition of the surface of the scalar in domain (t=0s)

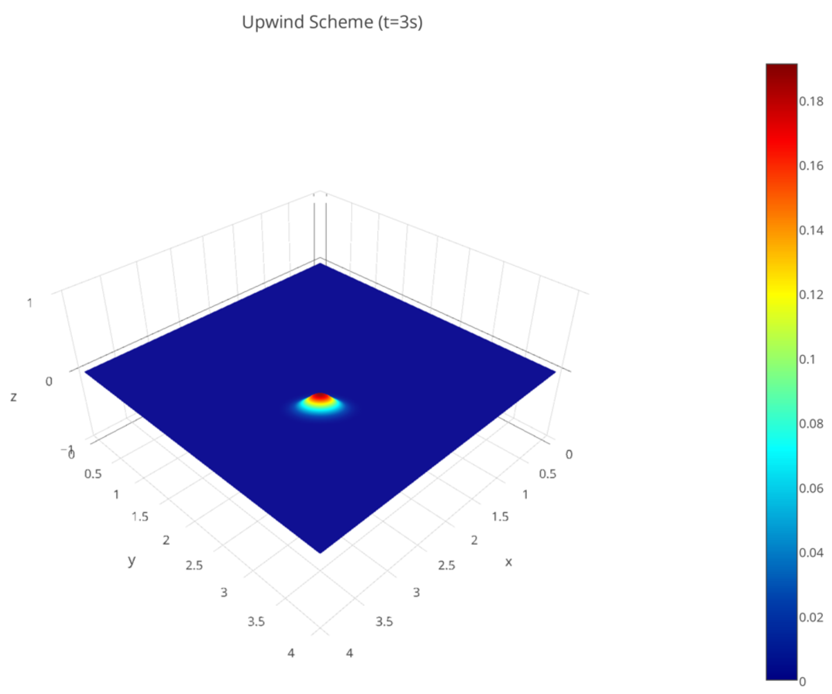
Figure 2 shows that the evolution of the surface of the scalar computed by using Upwind scheme. Based on stability analysis, the critical temporal step is 0.0125s for grid of 321×321 nodes. Time step is set to 0.01s so that stability condition is satisfied.

Upwind Scheme (t=1s)



Upwind Scheme (t=2s)





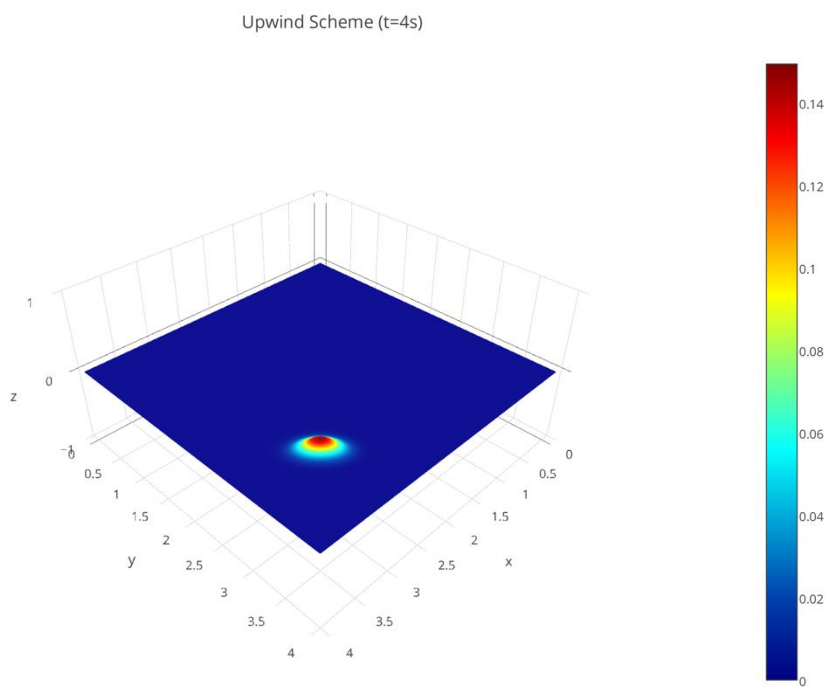
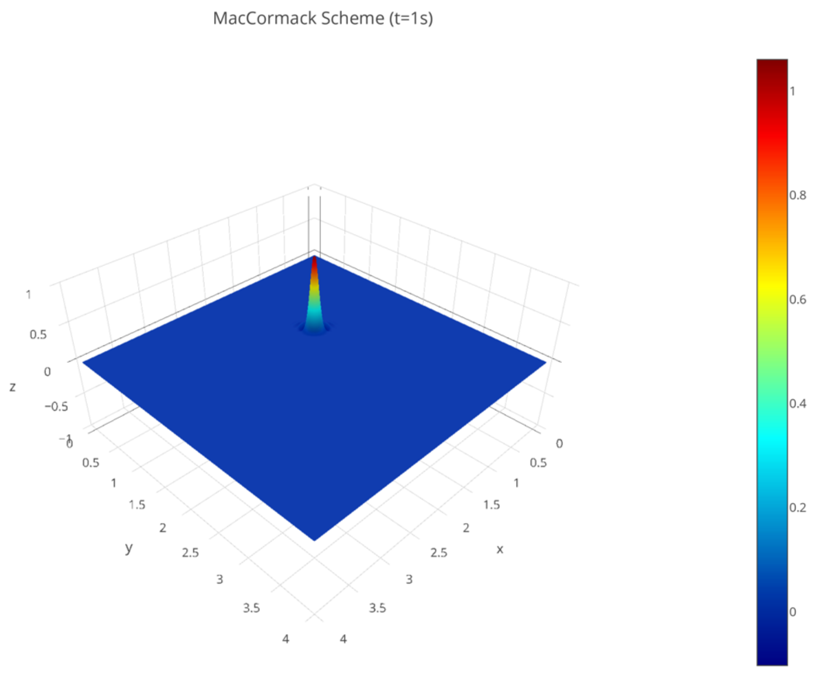
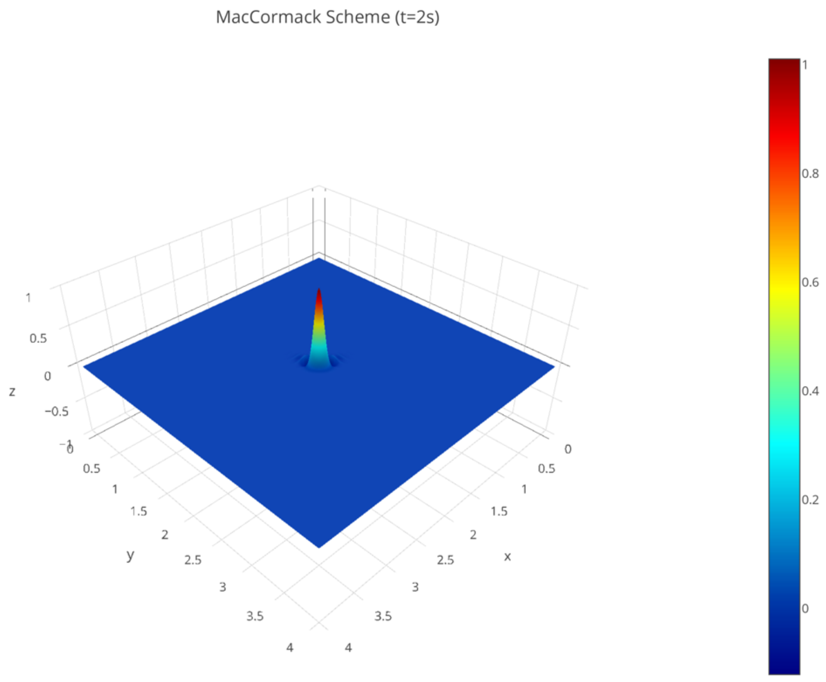
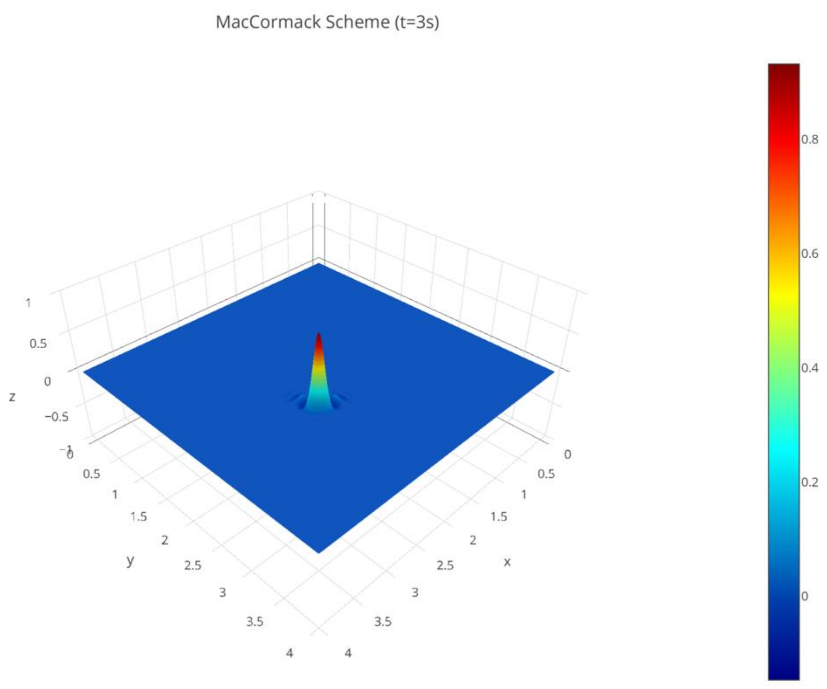


Figure 2. Evolution of the surface of the scalar computed by using Upwind Scheme

Figure 3 shows that the evolution of the surface of the scalar computed by using MacCormack scheme. Based on stability analysis, the critical temporal step is 0.0125s for grid of 321×321 nodes. Time step is set to 0.01s so that stability condition is satisfied.







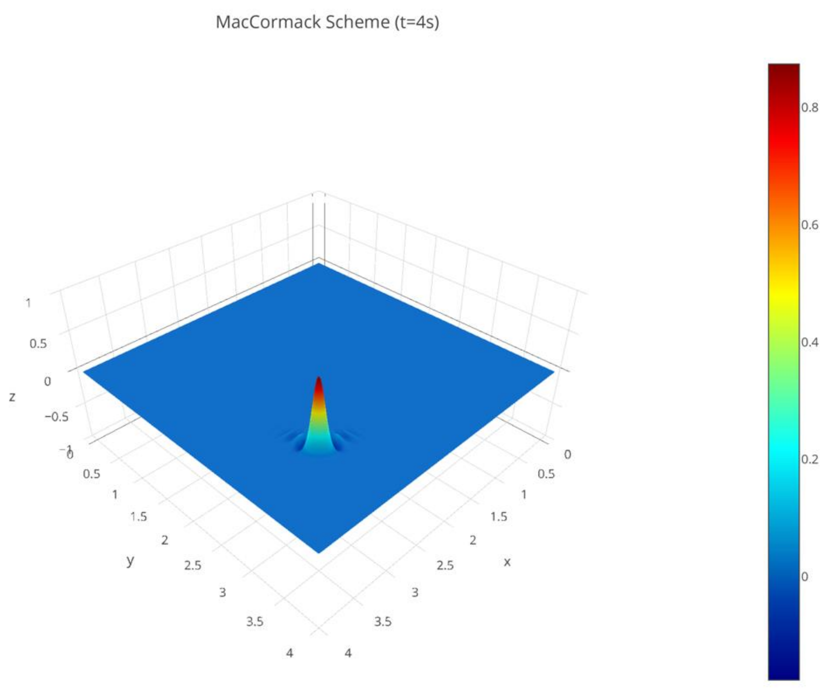
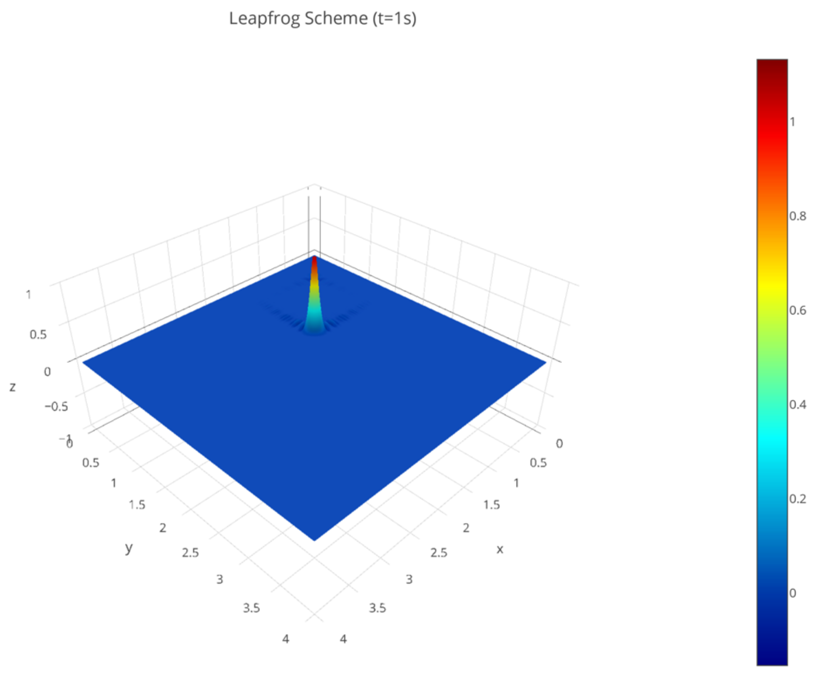
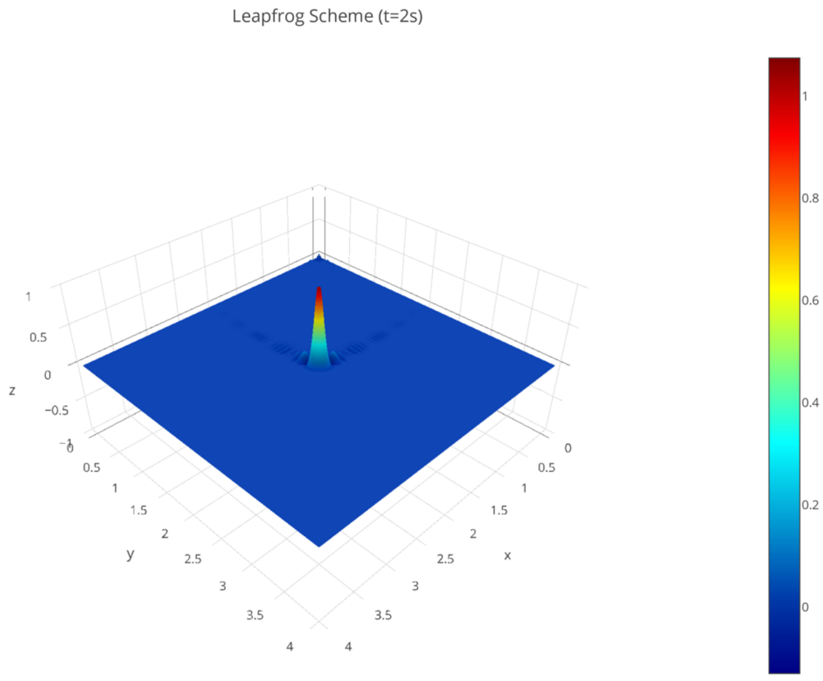
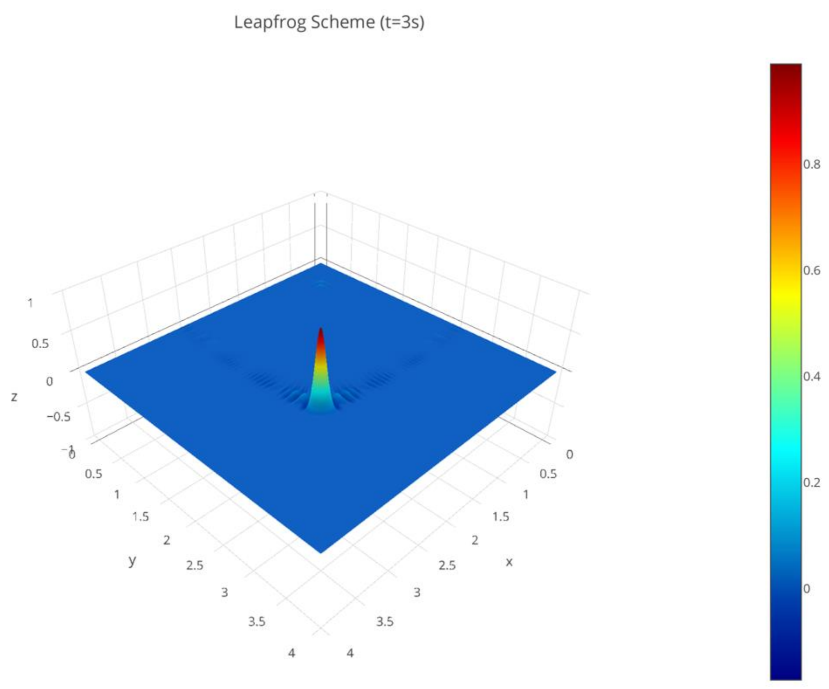


Figure 3. Evolution of the surface of the scalar computed by using MacCormack Scheme

Figure 4 shows that the evolution of the surface of the scalar computed by using Leap Frog scheme. Based on stability analysis, the scheme is unstable no matter what the time step is. We took the time step of 0.01s for the convenience of comparison with other schemes.







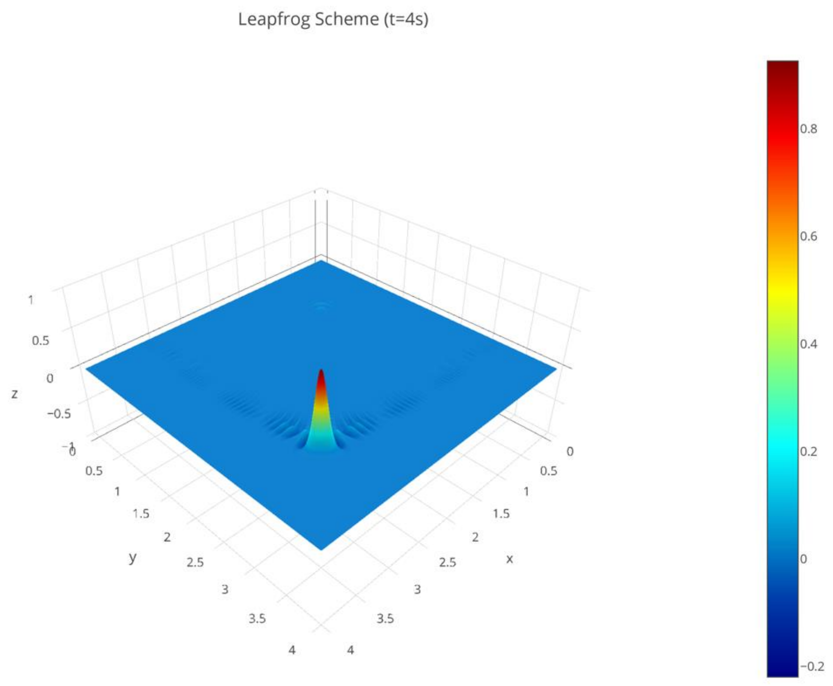
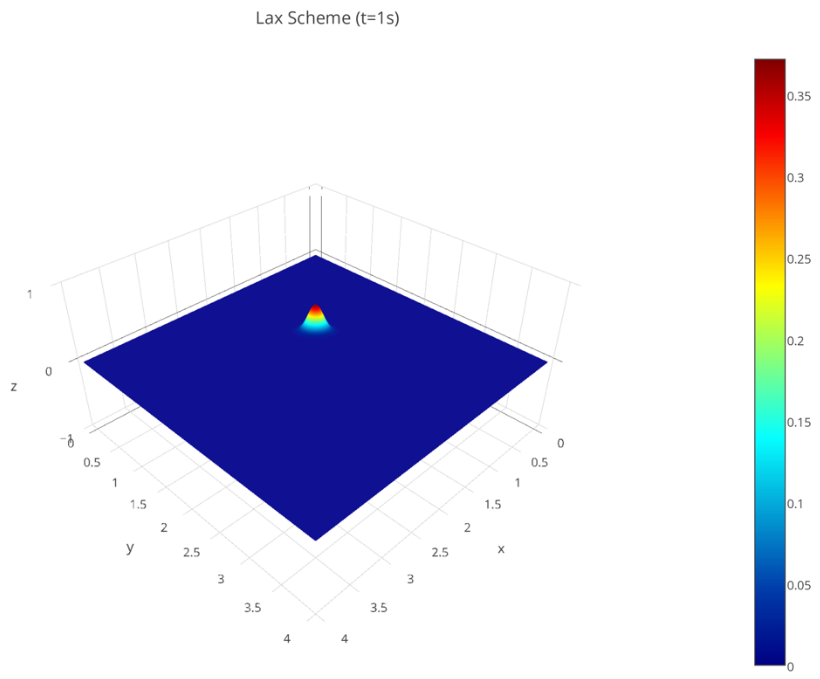
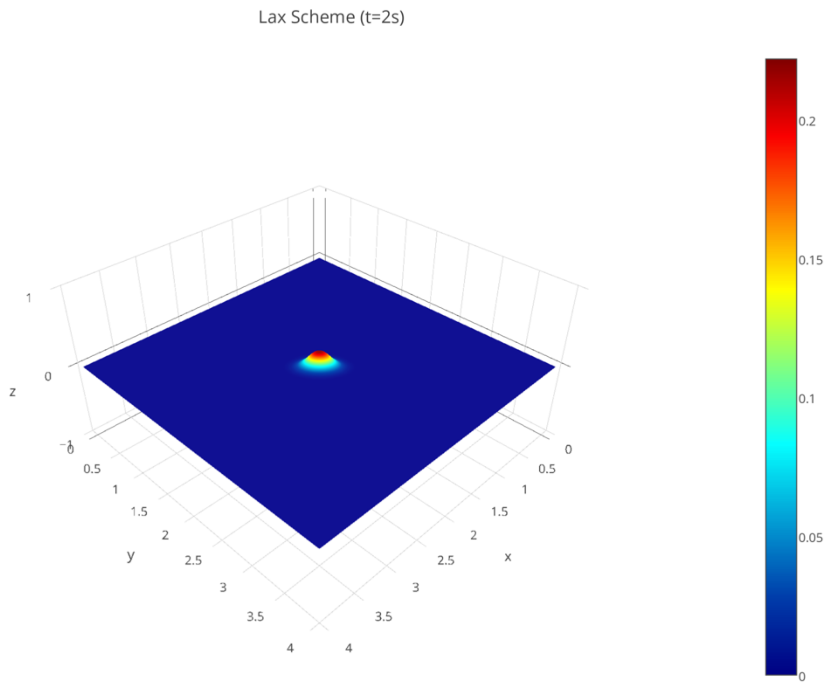
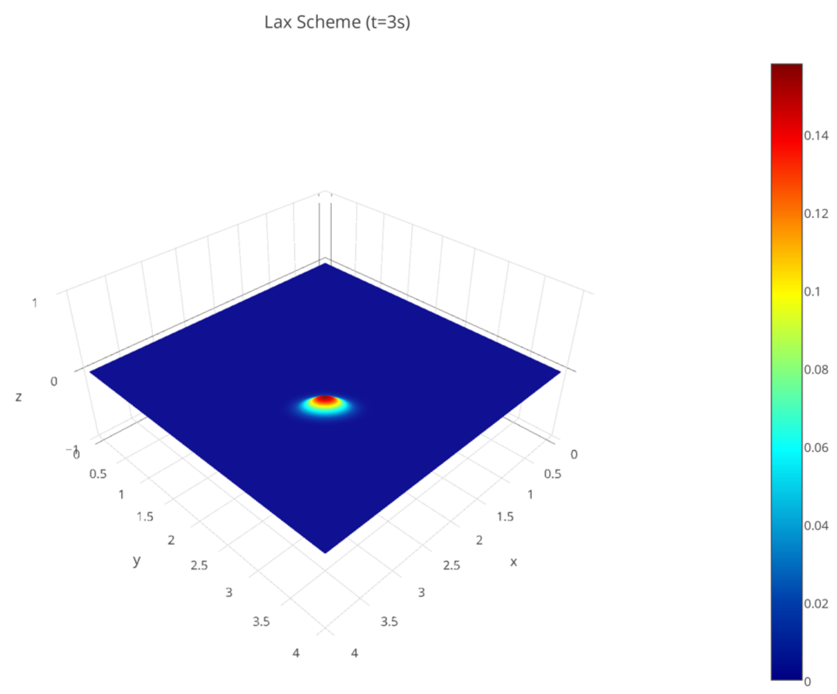


Figure 4. Evolution of the surface of the scalar computed by using Leap Frog Scheme

Figure 5 shows that the evolution of the surface of the scalar computed by using Lax scheme. Based on stability analysis, the critical temporal step is 0.0125s for grid of 321×321 nodes. Time step is set to 0.01s so that stability condition is satisfied.







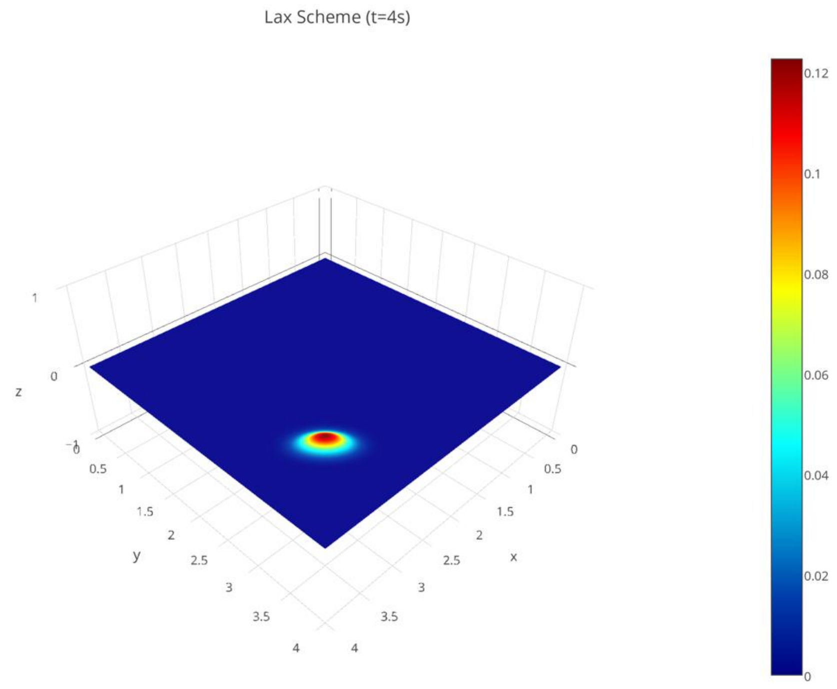


Figure 5. Evolution of the surface of the scalar computed by using Lax Scheme

4. Discussion

After observation of figure 1-5, it can be found that the magnitude of signal was gradually attenuated under all of schemes, however, the signal was significantly attenuated (large dissipative error) under Upwind scheme and Lax scheme, but was only attenuated a little under MacCormack scheme and Leap Frog scheme.

The magnitude of signal attenuation under each scheme was as following: Lax > Upwind > MacCormack > Leap Frog. Under ideal circumstances, since the diffusivity $\Gamma = 0$, the scalar should remain constant even after a much large number of time step. One assumption made here is that the reason of attenuation of scalar is truncation error of difference scheme.

It also can be observed that during the time span of 4s, the scalar was not going negative under Upwind scheme and Lax scheme (no dispersive error) but was going negative under MacCormack scheme and Leap Frog scheme. The magnitude of negative scalar value was as following: Leap Frog > MacCormack > Lax = Upwind. If we consider that the negative scalar value was caused by the fluctuation of scalar, and this fluctuation is related with the instability of difference scheme, we can draw this conclusion: when temporal step size is Δt = 0.01s, Leap Frog and MacCormack scheme are not stable, Lax and Upwind scheme are stable. However, based on stability analysis conducted in previous part, MacCormack scheme should be also stable. Thus there must be other reason which contribute to the negative value of scalar. One assumption made here is that the fluctuation of scalar is dissipative error of difference scheme. Also, it could be noticed in plot of 3,4, that the region affected by fluctuation of scalar under Leap Frog scheme was much larger than MacCormack scheme. We could say that the stability of MacCormack scheme is better than Leap Frog scheme. When considering both the attenuation and fluctuation of scalar, the MacCormack has a best performance among 4 difference schemes of scalar transport equation.

Based on CFL stability criterion, if we choose the time step size Δt of 0.0125s to make Courant number equal 1, the MacCormack scheme and Lax scheme could be more stable, which may partially improve the result under MacCormack scheme.

5. Conclusions

In this project, we developed a program to solve two-dimensional viscous scalar transport equation in a square domain by using four different differencing schemes. Evolution of the surface of the scalar computed by different schemes under both zero and nonzero diffusivity conditions are provided. It is observed that MacCormack scheme and Leap Frog scheme have smaller dissipative error than that of Upwind scheme and Lax scheme, but they also generate dispersive error which does not exist under Upwind scheme and Lax scheme. Appropriate time step size is necessary to guarantee stability, and sometimes large diffusivity will also cause instability. In this project, we don’t need analyze the computational cost of each scheme, but if we investigate comprehensively by combining the stability consideration and computational cost consideration, the result would be much interesting.

Appendix A: Source Code

Upwind.py

```
# 2D-viscous scalar transport equation

import numpy as np
import scipy as sc
import scipy.sparse as sparse
import scipy.sparse.linalg

#==========================================
# using matplotlib kit
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

#==========================================
# using plotly kit
import plotly.plotly as py
import plotly.tools as tls
tls.set_credentials_file(username='king_yin3613', api_key='cELFEragb4loopx5xzxx')
import plotly.graph_objs as go
import pandas as pd

# initial condition: scalar is is 0 everywhere in a 4*4 rectangular domain except x in (0.9,1,1)
and y(0.9,1.1),
# where: c(x,y,0) = sin(5pi(x-0.9))*in(5pi(y-0.9))
#============================================================
# differencing scheme is "Upwind", 321 * 321 grids.

# parameters
u = 0.5
v = 0.5

# two diffusivity
gamma_1 = 0.0
gamma_2 = 0.0001

#spatial step-size
x_min = 0.0
```

```
x_max = 4.0
x_node = 321

y_min = 0.0
y_max = 4.0
y_node = 321

dx = (x_max - x_min)/(x_node - 1)
dy = (y_max - y_min)/(y_node - 1)

#number of internal points
N = (x_max-x_min)/dx - 1

x_grid = np.arange(x_min, x_max+dx, dx)
y_grid = np.arange(y_min, y_max+dy, dy)

# limit of temporal step-size
def stable_time_step(gamma):
    dt = 1/(u/dx+v/dy+2*gamma/dx**2+2*gamma/dy**2)
    return dt
dt_1 = stable_time_step(gamma_1)-0.0025
print(dt_1)
dt_2 = stable_time_step(gamma_2)
print(dt_2)
# time span
t_i = 0.0
t_f = 4.0
time_steps_1 = (t_f-t_i)/dt_1
print(time_steps_1)
time_steps_2 = (t_f-t_i)/dt_2
print(time_steps_2)

#initial conditions
def c_ini(x_grid,y_grid):

    c = np.zeros([len(x_grid),len(y_grid)]) #initialize

    for i in range(0,len(x_grid)):
        for j in range(0,len(y_grid)):
            if i*dx <= 0.9:
                c[i,j] = 0
            elif i*dx < 1.1:
                    if j*dy <= 0.9:
                        c[i,j] = 0
                    elif j*dy < 1.1:
                        c[i,j] = np.sin(5*np.pi*(i*dx-0.9))*np.sin(5*np.pi*(j*dy-0.9))
                    else:
                        c[i,j] = 0
                else:
                    c[i,j] = 0
    return c
c_0 = c_ini(x_grid,y_grid)


# Visualize initial state

mat_fig = plt.figure()
ax = mat_fig.gca(projection='3d')
surf = ax.plot_surface(x_grid, y_grid, c_0, cmap='jet',
                                      linewidth=0)
ax.set_zlim(-1.00, 1.00)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
mat_fig.colorbar(surf, shrink=0.5, aspect=10)
plt.show()

#=============
# Upwind Scheme
# Backward Difference Scheme for dc/dx and dc/dy
# Centered Difference Scheme for d2c/dx2 and d2c/dy2
def Upwind(u,v,gamma,ini_matrix,dt,time_steps):
    c_old = np.copy(ini_matrix)
    c_new = np.zeros([len(x_grid),len(y_grid)])
    for n in range(1,int(time_steps+1)): # here n is the time_step
        for i in range(1, c_old.shape[0]-1):
            for j in range(1, c_old.shape[1]-1):
                #dcdx = (c[i,j]-c[i-1,j])/dx
                #dcdy = (c[i,j]-c[i,j-1])/dy
                #d2cdx2 = (c[i+1,j] - 2*c[i,j] + c[i-1,j])/(dx**2)
                #d2cdy2 = (c[i,j+1] - 2*c[i,j] + c[i,j-1])/(dy**2)
                #c[i,j] = c[i,j] + dt*(gamma*(d2cdx2+d2cdy2)-u*dcdx-v*dcdy)
                c_new[i,j] = (1-u*dt/dx-v*dt/dy-2*gamma*dt/dx**2-
2*gamma*dt/dy**2)*c_old[i,j]+(gamma*dt/dx**2+u*dt/dx)*c_old[i-
1,j]+(gamma*dt/dy**2+v*dt/dy)*c_old[i,j-
1]+(gamma*dt/dx**2)*c_old[i+1,j]+(gamma*dt/dy**2)*c_old[i,j+1]
                        c_old = c_new
      return c_new

'''
c_1 = Upwind(u,v,gamma_1,c_0,dt_1,80)
z_data_1 = c_1
data = [go.Surface(x=x_grid, y=y_grid,z=z_data_1,colorscale='Jet')]
#data = [go.Surface(z=z_data.as_matrix())]
layout = go.Layout(title='Initial Condition (t=1)', autosize=True)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='Initial Condition (t=1)')
'''

def Visualization(time):
      time_step = time/dt_1
      print(time_step)
      figure_index = 1
      for t in time_step:

            c_data = Upwind(u,v,gamma_1,c_0,dt_1,t)

            mat_fig = plt.figure()
            ax = mat_fig.gca(projection='3d')
            surf = ax.plot_surface(x_grid, y_grid, c_data, cmap='jet',
                                   linewidth=0, antialiased=False)
            mat_fig.colorbar(surf, shrink=0.5, aspect=10)
            plt.show()

            z_data = c_data
            data = [go.Surface(x=x_grid, y=y_grid,z=z_data,colorscale='Jet')]
            layout = go.Layout(title='Upwind Scheme (t=%s)' % figure_index, autosize=True)
            fig = go.Figure(data=data, layout=layout)
            py.iplot(fig, filename='Upwind Scheme (t=%s)' % figure_index)
            figure_index += 1


Visualization(np.array([1,2,3,4]))
```

Maccormack.py

```
# 2D-viscous scalar transport equation

import numpy as np
import scipy as sc
import scipy.sparse as sparse
import scipy.sparse.linalg

#=========================================
# using plotly kit
import plotly.plotly as py
import plotly.tools as tls
tls.set_credentials_file(username='king_yin3613', api_key='cELFEragb4loopx5xzxx')
import plotly.graph_objs as go
import pandas as pd
# initial condition: scalar is is 0 everywhere in a 4*4 rectangular domain except x in (0.9,1,1)
and y(0.9,1.1),
# where: c(x,y,0) = sin(5pi(x-0.9))*in(5pi(y-0.9))
#============================================================
# differencing scheme is "MacCormack", 321 * 321 grids.

# parameters
u = 0.5
v = 0.5

# two diffusivity
gamma_1 = 0.0
gamma_2 = 0.0001

#spatial step-size
x_min = 0.0
x_max = 4.0
x_node = 321

y_min = 0.0
y_max = 4.0
y_node = 321

dx = (x_max - x_min)/(x_node - 1)
dy = (y_max - y_min)/(y_node - 1)

#number of internal points
N = (x_max-x_min)/dx - 1
```

```
x_grid = np.arange(x_min, x_max+dx, dx)
y_grid = np.arange(y_min, y_max+dy, dy)

# limit of temporal step-size
# MacCormack Scheme: absolute value of Courant number is less 1
# abs(v) = abs(u*dt/dx + v*dt/dy) < 1
def stable_time_step(gamma):
    dt = 1/(u/dx+v/dy+2*gamma/dx**2+2*gamma/dy**2)
    dt_max = 1/(u/dx+v/dy)
    return dt,dt_max

print('dt is in (0,%s)' % stable_time_step(gamma_1)[1])
print('dt is in (0,%s)' % stable_time_step(gamma_2)[1])
dt_1 = stable_time_step(gamma_1)[0]
dt_2 = stable_time_step(gamma_2)[0]

# time span
t_i = 0.0
t_f = 4.0
time_steps_1 = (t_f-t_i)/dt_1
print(time_steps_1)
time_steps_2 = (t_f-t_i)/dt_2

#initial conditions
def c_ini(x_grid,y_grid):

    c = np.zeros([x_node,y_node]) #initialize

    for i in range(0,len(x_grid)):
        for j in range(0,len(y_grid)):
            if i*dx <= 0.9:
                c[i,j] = 0
            elif i*dx < 1.1:
                if j*dy <= 0.9:
                    c[i,j] = 0
                elif j*dy < 1.1:
                    c[i,j] = np.sin(5*np.pi*(i*dx-0.9))*np.sin(5*np.pi*(j*dy-0.9))
                else:
                    c[i,j] = 0
            else:
                c[i,j] = 0
```

```
	return c
c_0 = c_ini(x_grid,y_grid)

# MacCormack Scheme
# Forward Difference Scheme for dc/dx and dc/dy
# Centered Difference Scheme for d2c/dx2 and d2c/dy2
# Predictor-corrector strategy
def MacCormack(u,v,gamma,ini_matrix,dt,time_steps):
	c = np.copy(ini_matrix)
	predict_c = np.zeros([x_node,y_node])
	for n in range(1,int(time_steps+1)): # here n is the time_step
		for i in range(1, c.shape[0]-1):
			for j in range(1, c.shape[1]-1):
				predict_c[i,j] = c[i,j] - u*dt*(c[i+1,j]-c[i,j])/dx - v*dt*(c[i,j+1]-c[i,j])/dy +
gamma*dt*((c[i+1,j]-2.0*c[i,j]+c[i-1,j])/dx**2+(c[i,j+1]-2.0*c[i,j]+c[i,j-1])/dy**2)
		for i in range(1, c.shape[0]-1):
			for j in range(1, c.shape[1]-1):
				c[i,j] = (c[i,j]+predict_c[i,j])/2-((u*dt*(predict_c[i,j]-predict_c[i-1,j]))/(2*dx))-
((v*dt*(predict_c[i,j]-predict_c[i,j-1]))/(2*dy))+gamma*dt/2*((predict_c[i+1,j]-
2*predict_c[i,j]+predict_c[i-1,j])/(dx**2)+(predict_c[i,j+1]-2*predict_c[i,j]+predict_c[i,j-
1])/(dy**2))
	return c
'''
c_1 = MacCormack(u,v,gamma_1,c_0,dt_1,80)
z_data_1 = c_1
data = [go.Surface(x=x_grid, y=y_grid,z=z_data_1,colorscale='Jet')]
#data = [go.Surface(z=z_data.as_matrix())]
layout = go.Layout(title='MacCormack Scheme (t=1)', autosize=True)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='MacCormack Scheme (t=1)')
'''
def Visualization(time):
	time_step = time/dt_1
	print(time_step)
	figure_index = 1
	for t in time_step: # here t in time_step is increment, idk why
		c_data = MacCormack(u,v,gamma_1,c_0,dt_1,t)
		z_data = c_data
		data = [go.Surface(x=x_grid, y=y_grid,z=z_data,colorscale='Jet')]
		layout = go.Layout(title='MacCormack Scheme (t=%s)' % figure_index,
autosize=True)
		fig = go.Figure(data=data, layout=layout)
```

```
            py.iplot(fig, filename='MacCormack Scheme (t=%s)' % figure_index)
            figure_index += 1

Visualization(np.array([1,2,3,4]))
```

Leapfrog.py

```
# 2D-viscous scalar transport equation

import numpy as np
import scipy as sc
import scipy.sparse as sparse
import scipy.sparse.linalg

#========================================
# using plotly kit
import plotly.plotly as py
import plotly.tools as tls
tls.set_credentials_file(username='king_yin3613', api_key='cELFEragb4loopx5xzxx')
import plotly.graph_objs as go
import pandas as pd
# initial condition: scalar is is 0 everywhere in a 4*4 rectangular domain except x in (0.9,1,1)
and y(0.9,1.1),
# where: c(x,y,0) = sin(5pi(x-0.9))*in(5pi(y-0.9))
#==========================================================
# differencing scheme is "Leap Frog", 321 * 321 grids.

# parameters
u = 0.5
v = 0.5

# two diffusivity
gamma_1 = 0.0
gamma_2 = 0.0001

#spatial step-size
x_min = 0.0
x_max = 4.0
x_node = 321

y_min = 0.0
y_max = 4.0
y_node = 321
```

```
dx = (x_max - x_min)/(x_node - 1)
dy = (y_max - y_min)/(y_node - 1)

#number of internal points
N = (x_max-x_min)/dx - 1

x_grid = np.arange(x_min, x_max+dx, dx)
y_grid = np.arange(y_min, y_max+dy, dy)

# limit of temporal step-size
# Leap Frog Scheme: unconditionally unstable
def stable_time_step(gamma):
	dt = 1/(u/dx+v/dy+2*gamma/dx**2+2*gamma/dy**2)
	dt_max = 1/(u/dx+v/dy)
	return dt,dt_max

print('dt is in (0,%s)' % stable_time_step(gamma_1)[1])
print('dt is in (0,%s)' % stable_time_step(gamma_2)[1])
dt_1 = stable_time_step(gamma_1)[0]
dt_2 = stable_time_step(gamma_2)[0]

# time span
t_i = 0.0
t_f = 4.0
time_steps_1 = (t_f-t_i)/dt_1
print(time_steps_1)
time_steps_2 = (t_f-t_i)/dt_2

#initial conditions
def c_ini(x_grid,y_grid):

	c = np.zeros([len(x_grid),len(y_grid)]) #initialize

	for i in range(0,len(x_grid)):
		for j in range(0,len(y_grid)):
			if i*dx <= 0.9:
				c[i,j] = 0
			elif i*dx < 1.1:
				if j*dy <= 0.9:
					c[i,j] = 0
				elif j*dy < 1.1:
```

```
                    c[i,j] = np.sin(5*np.pi*(i*dx-0.9))*np.sin(5*np.pi*(j*dy-0.9))
                else:
                    c[i,j] = 0
            else:
                c[i,j] = 0
    return c
c_0 = c_ini(x_grid,y_grid)

# Leap Frog Scheme
# Centered Difference Scheme for dc/dx and dc/dy
# Centered Difference Scheme for d2c/dx2 and d2c/dy2
# Centered Difference Scheme for dc/dt (3 level scheme)
# We assume that c^1 = c^0
def LeapFrog(u,v,gamma,ini_matrix,dt,time_steps):
    c_old = np.copy(ini_matrix)
    c_inter = np.copy(ini_matrix)
    c_new = np.zeros([len(x_grid),len(y_grid)])
    for n in range(2,int(time_steps+1)): # here n is the time_step
        for i in range(1, c_new.shape[0]-1):
            for j in range(1, c_new.shape[1]-1):
                c_new[i,j] = c_old[i,j] + 2*dt*(-u*(c_inter[i+1,j]-c_inter[i-1,j])/(2.0*dx)-
v*(c_inter[i,j+1]-c_inter[i,j-1])/(2.0*dy)+gamma*((c_inter[i+1,j]-2.0*c_inter[i,j]+c_inter[i-
1,j])/dx**2+(c_inter[i,j+1]-2.0*c_inter[i,j]+c_inter[i,j-1])/dy**2))
                c_old = c_inter
                c_inter = c_new
    return c_new

def Visualization(time):
    time_step = time/dt_1
    print(time_step)
    figure_index = 1
    for t in time_step:
        c_data = LeapFrog(u,v,gamma_1,c_0,dt_1,t)
        z_data = c_data
        data = [go.Surface(x=x_grid, y=y_grid,z=z_data,colorscale='Jet')]
        layout = go.Layout(title='LeapFrog Scheme (t=%s)' % figure_index, autosize=True)
        fig = go.Figure(data=data, layout=layout)
        py.iplot(fig, filename='LeapFrog Scheme (t=%s)' % figure_index)
        figure_index += 1


Visualization(np.array([1,2,3,4]))
```

Lax.py

```
# 2D-viscous scalar transport equation

import numpy as np
import scipy as sc
import scipy.sparse as sparse
import scipy.sparse.linalg

#========================================
# using plotly kit
import plotly.plotly as py
import plotly.tools as tls
tls.set_credentials_file(username='king_yin3613', api_key='cELFEragb4loopx5xzxx')
import plotly.graph_objs as go
import pandas as pd
# initial condition: scalar is is 0 everywhere in a 4*4 rectangular domain except x in (0.9,1,1)
and y(0.9,1.1),
# where: c(x,y,0) = sin(5pi(x-0.9))*in(5pi(y-0.9))
#==========================================================
# differencing scheme is "Lax", 321 * 321 grids.

# parameters
u = 0.5
v = 0.5

# two diffusivity
gamma_1 = 0.0
gamma_2 = 0.0001

#spatial step-size
x_min = 0.0
x_max = 4.0
x_node = 321

y_min = 0.0
y_max = 4.0
y_node = 321

dx = (x_max - x_min)/(x_node - 1)
dy = (y_max - y_min)/(y_node - 1)
```

```
#number of internal points
N = (x_max-x_min)/dx - 1

x_grid = np.arange(x_min, x_max+dx, dx)
y_grid = np.arange(y_min, y_max+dy, dy)

# limit of temporal step-size
# Lax Scheme: absolute value of Courant number is less 1
# abs(v) = abs(u*dt/dx + v*dt/dy) < 1
def stable_time_step(gamma):
	dt = 1/(u/dx+v/dy+2*gamma/dx**2+2*gamma/dy**2)
	dt_max = 1/(u/dx+v/dy)
	return dt,dt_max

print('dt is in (0,%s)' % stable_time_step(gamma_1)[1])
print('dt is in (0,%s)' % stable_time_step(gamma_2)[1])
dt_1 = stable_time_step(gamma_1)[0]
dt_2 = stable_time_step(gamma_2)[0]

# time span
t_i = 0.0
t_f = 4.0
time_steps_1 = (t_f-t_i)/dt_1
print(time_steps_1)
time_steps_2 = (t_f-t_i)/dt_2

#initial conditions
def c_ini(x_grid,y_grid):

	c = np.zeros([len(x_grid),len(y_grid)]) #initialize

	for i in range(0,len(x_grid)):
		for j in range(0,len(y_grid)):
			if i*dx <= 0.9:
				c[i,j] = 0
			elif i*dx < 1.1:
				if j*dy <= 0.9:
					c[i,j] = 0
				elif j*dy < 1.1:
					c[i,j] = np.sin(5*np.pi*(i*dx-0.9))*np.sin(5*np.pi*(j*dy-0.9))
				else:
					c[i,j] = 0
```

```
                else:
                        c[i,j] = 0
        return c
c_0 = c_ini(x_grid,y_grid)

# Lax Scheme
# Centered Difference Scheme for dc/dx and dc/dy
# Centered Difference Scheme for d2c/dx2 and d2c/dy2
# Spatial average of c_{i,j} taken over the neighboring grids
def Lax(u,v,gamma,ini_matrix,dt,time_steps):
        c_old = np.copy(ini_matrix)
        c_new = np.zeros([len(x_grid),len(y_grid)])
        for n in range(1,int(time_steps+1)): # here n is the time_step
                for i in range(1, c_old.shape[0]-1):
                        for j in range(1, c_old.shape[1]-1):
                                c_new[i,j] = (1/4.0-
u*dt/(2*dx)+gamma*dt/dx**2)*c_old[i+1,j]+(1/4.0+u*dt/(2*dx)+gamma*dt/dx**2)*c_old[i-
1,j]+(1/4.0-
v*dt/(2*dy)+gamma*dt/dy**2)*c_old[i,j+1]+(1/4.0+v*dt/(2*dy)+gamma*dt/dy**2)*c_old[i,j-1]-
(2*gamma*dt/(dx**2)+2*gamma*dt/(dy**2))*c_old[i,j]
                                c_old = c_new
        return c_new

def Visualization(time):
        time_step = time/dt_1
        print(time_step)
        figure_index = 1
        for t in time_step:
                c_data = Lax(u,v,gamma_1,c_0,dt_1,t)
                z_data = c_data
                data = [go.Surface(x=x_grid, y=y_grid,z=z_data,colorscale='Jet')]
                layout = go.Layout(title='Lax Scheme (t=%s)' % figure_index, autosize=True)
                fig = go.Figure(data=data, layout=layout)
                py.iplot(fig, filename='Lax Scheme (t=%s)' % figure_index)
                figure_index += 1


Visualization(np.array([1,2,3,4]))
```