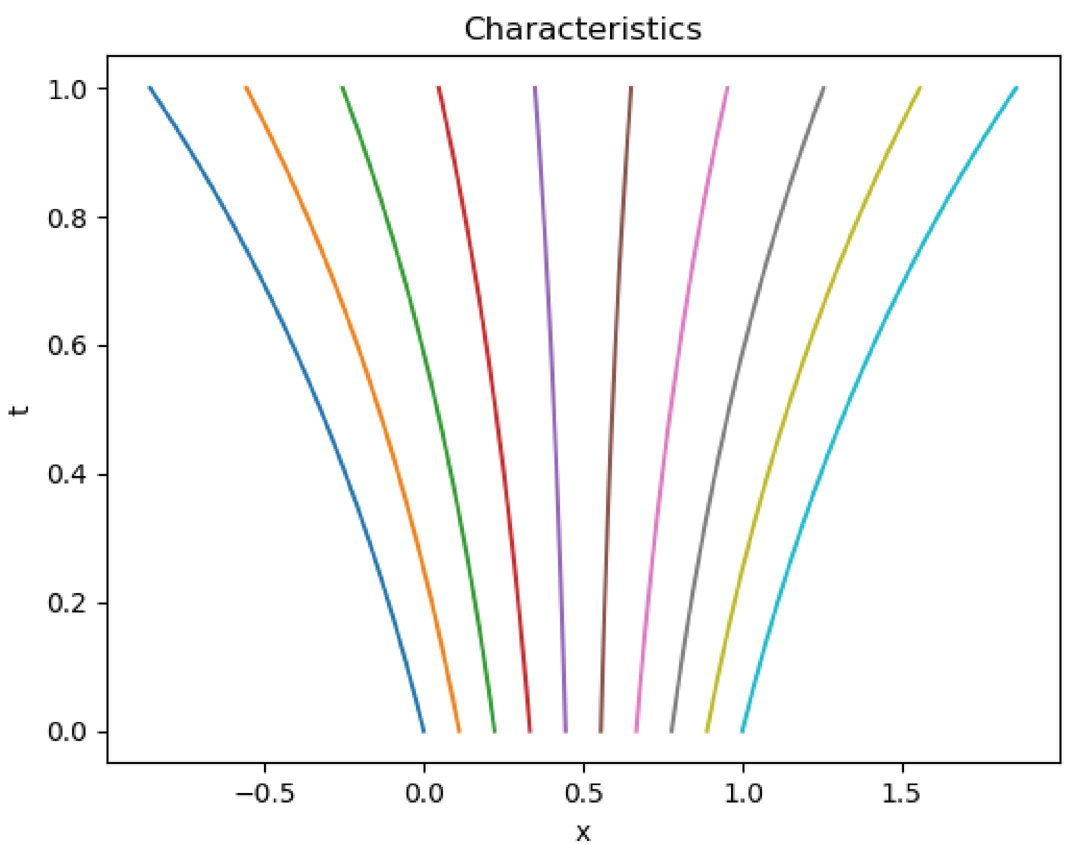
MATH 5620/6865: ASSIGNMENT_6

JORDAN SAETHRE AND PAUL MUNDT

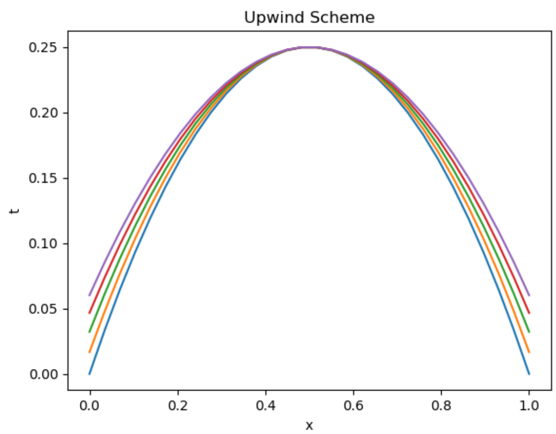
(1) Sketch the characteristics for the equation $u_t + au_x = 0$ for $0 \leq x \leq 1$ when $a \equiv a(x) = x - \frac{1}{2}$. Set up and implement the upwind scheme on a uniform mesh $\{x_j = j\Delta x, j = 0, 1, \cdots, J\}$ with an initial condition $u(x, 0) = x(1 - x)$. Repeat the problem with $a(x) = \frac{1}{2} - x$ with boundary conditions $u(0, t) = u(1, t) = 0$. (Think a bit on the differences between these two problems)

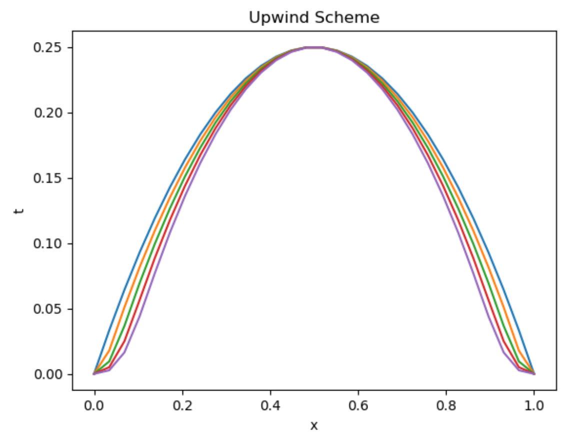
Characteristics: The characteristics of the equation $u_t + au_x = 0$ for $0 \leq x \leq 1$ when $a \equiv a(x) = x - \frac{1}{2}$ are the solutions to the differential equation $x' = x - \frac{1}{2}$. The graphs of $x(t) = Ce^t + 1/2$ when $x(0) = \{1, 1/9, 2/9, 3/9, 4/9, 5/9, 6/9, 7/9, 8/9, 1\}$ are shown below:



Date: April 9, 2020.

Output Upwind Scheme: For $a(x) = x - \frac{1}{2}$ and $a(x) = \frac{1}{2} - x$, respectively:





Code for Upwind Scheme:

```
import numpy as np
import matplotlib.pyplot as plt

a = 0
b = 1
n = 30
step = (b-a)/(n-1)
iterates = 50

# delta x = delta t
xt_vector = np.linspace(a,b,n)

# function a(x)
a = lambda x: x - 1/2

# initial condition function u(x,0)
u = lambda x: x*(1-x)

# initial condition
initial =np.array([u(xt_vector[i]) for i in range(len(xt_vector))])

# vector for diagonal on matrix A
a_vector = np.array([])

for i in range(len(xt_vector)):
    if a(xt_vector[i])>0:
        a_vector = np.append(a_vector, 1 - a(xt_vector[i]))
    else:
        a_vector = np.append(a_vector, 1 + a(xt_vector[i]))

# diagonal matrix
```

```
A = np.diag(a_vector)

# upwind scheme
def upwind(initial, A, n):
    next_iterate = np.matmul(A,initial)
    for j in range(n):
        if a(xt_vector[j]) > 0:
            next_iterate[j] = next_iterate[j]
            + a(xt_vector[j])*(initial[j-1])
        else:
            next_iterate[j] = next_iterate[j]
            - a(xt_vector[j])*(initial[j+1])
    return (next_iterate)

# plot solutions
for i in range(iterates):
    if i%10 == 0:
        plt.plot(xt_vector, initial)
        initial = upwind(initial,A,n)
plt.xlabel("x")
plt.ylabel("t")
plt.title('Upwind Scheme')
plt.show()

####################################################################
### Repeat for Boundary Conditions and new a(x)####################
####################################################################

# function a(x)
a = lambda x: 1/2 - x

# boundary condition
# initial condition function u(x,0)
u = lambda x: x*(1-x)

# initial condition
initial =np.array([u(xt_vector[i]) for i in range(len(xt_vector))])
initial[0] = 0
initial[n-1] = 0

# vector for diagonal on matrix A
a_vector = np.array([])

for i in range(len(xt_vector)):
    if a(xt_vector[i])>0:
        a_vector = np.append(a_vector, 1 - a(xt_vector[i]))
    else:
        a_vector = np.append(a_vector, 1 + a(xt_vector[i]))

# diagonal matrix
```

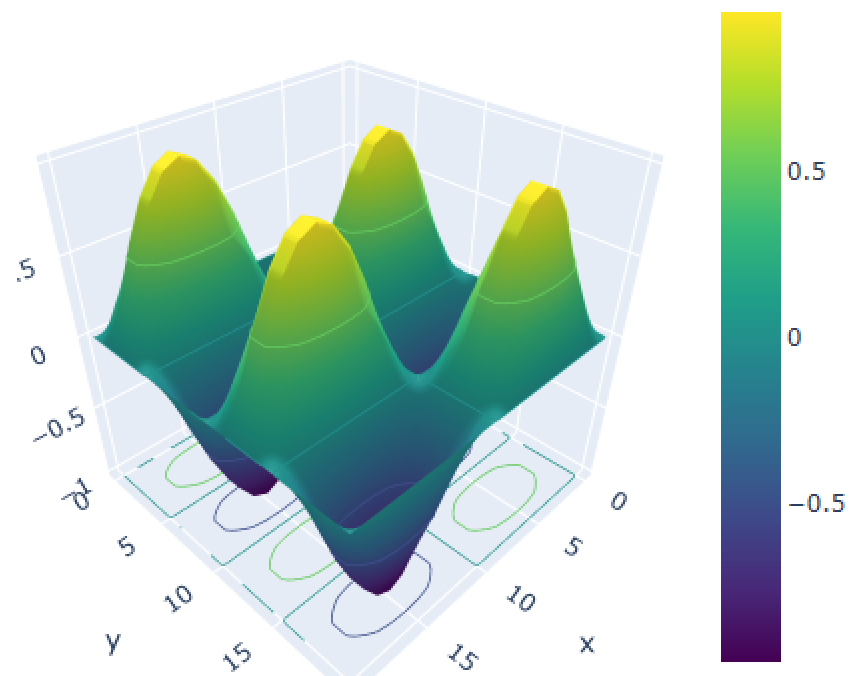
```
A = np.diag(a_vector)

# plot solutions
for i in range(iterates):
    if i%10 == 0:
        plt.plot(xt_vector, initial)
        initial = upwind(initial,A,n-1)
plt.xlabel("x")
plt.ylabel("t")
plt.title('Upwind Scheme')
plt.show()
```

(2) Implement the finite difference method to solve the Poisson equation

$$u_{xx} + u_{yy} + 5\pi^2 \sin(\pi x)\sin(2\pi y) = 0$$

in $[-1,1] \times [-1,1]$ with 0 Dirichlet boundary condition on the boundary. Check your solution with the exact solution $u(x,y) = \sin(\pi x)\sin(2\pi y)$. What can you say about this solution from the aspect of eigenvalue and eigenfunction? Can you understand better why we say Δ is a negative operator?(Hint: Yes, you can!)

Output FDM on Poisson Equation:



The Poisson equation is an eigenvalue eigenfunction problem. Notice that it is of the form $U' = \lambda U$ where $\lambda = -5\pi^2$ and $U = \sin(\pi x)\sin(2\pi y)$. The Laplacian operator Δ is considered a negative operator because the eigenvalue is negative.

Code for FDM on Poisson Equation:

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go

xl = -1
xr = 1
yb = -1
yt = 1
N = 20
M = 20
def Laplace(xl,xr,yb,yt,M,N):

    #equation
    f=lambda x,y: -5*np.pi**2*np.sin(np.pi*x)*np.sin(2*np.pi*y)

    # Dirichlet boundary conditions
    gb = lambda y: 0 # bottom
    gt = lambda y: 0 # top
    gl = lambda x: 0 # left
    gr = lambda x: 0 # right

    m=M+1
    n=N+1
    mn=m*n

    h=(xr-xl)/M
    h2=h**2
    k=(yt-yb)/N
    k2=k**2

    x=np.linspace(xl,xr,m)
    y=np.linspace(yb,yt,n)
    A=np.zeros([mn,mn])
    b=np.zeros([mn,1])

    for i in range(1,m-1):
        for j in range(1,n-1):
            A[i+(j)*m,i-1+(j)*m]=1/h2
            A[i+(j)*m,i+1+(j)*m]=1/h2
            A[i+(j)*m,i+(j)*m]=-(2/h2)-(2/k2)
            A[i+(j)*m,i+(j-1)*m]=1/k2
            A[i+(j)*m,i+(j+1)*m]=1/k2
            b[i+(j)*m]=f(x[i],y[j])

    for i in range(0,m):
        j=1
        A[i+(j-1)*m,i+(j-1)*m]=1
```

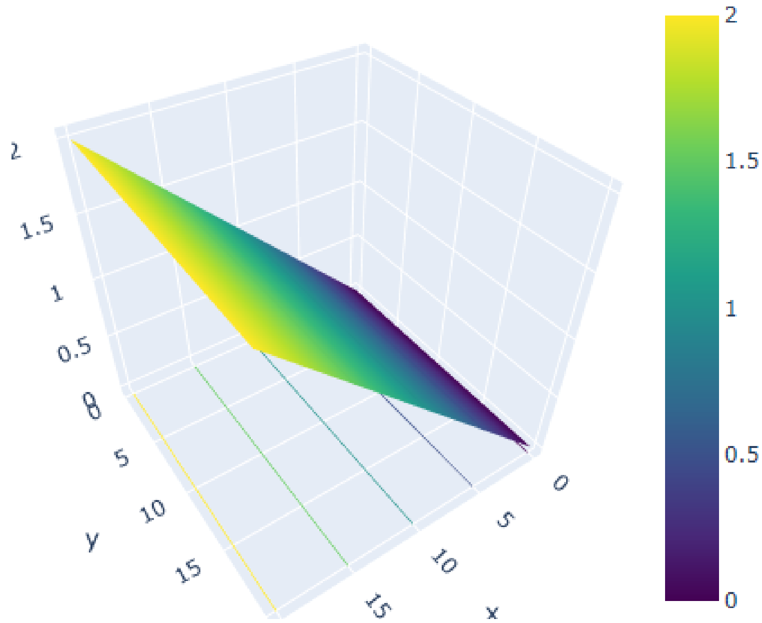
```
        b[i+(j-1)*m]=gl(x[i])
        j=n
        A[i+(j-1)*m,i+(j-1)*m]=1
        b[i+(j-1)*m]=gr(x[i])

    for j in range(2,n):
        i=0
        A[i+(j-1)*m,i+(j-1)*m]=1
        b[i+(j-1)*m]=gb(y[j])
        i=M
        A[i+(j-1)*m,i+(j-1)*m]=1
        b[i+(j-1)*m]=gt(y[j])

    #Find inverse of A
    Ainv = np.linalg.inv(A)
    # Find solution v = Ainv*b
    Solution = np.matmul(Ainv,b)
    # Reshape into a matrix
    Solution = np.reshape(Solution[0:mn],(n,m))
    return(Solution)

def plot_surface(solution):
    surface_df = pd.DataFrame(solution)
    surface_df.to_csv('surface_data.csv', index = False)
    surface_data = pd.read_csv('surface_data.csv')
    fig = go.Figure(data=[go.Surface(z=surface_data.values,
        colorscale='Viridis')])
    fig.update_traces(contours_z=dict(show=True, usecolormap=True,
        highlightcolor="limegreen", project_z=True))
    fig.update_layout(title='Solution', autosize=False, width=500,
        height=500,margin=dict(l=65, r=50, b=65, t=90),
        xaxis = dict(visible = False))
    fig.show()

plot_surface(Laplace(xl,xr,yb,yt,M,N))
```

(3) Implement the finite difference method to solve the Laplace equation $u_{xx} + u_{yy} = 0$ in $[-1,1] \times [-1,1]$ with boundary condition

$$u(x,y) = \begin{cases} 0, & x = -1 \\ 2, & x = 1 \\ x+1, & y = -1 \\ x+1, & y = 1. \end{cases}$$

Use the previous ADI method for heat diffusion equation to solve $u_t = u_{xx} + u_{yy}$ with initial condition $u(x,y,0) = (x^2-1)(y^2-1) + x + 1$ and the same boundary condition above to $t = 10, 20, 30$. Compare the solutions with the solution from the Laplace equation.

Output FDM on Laplace Equation:



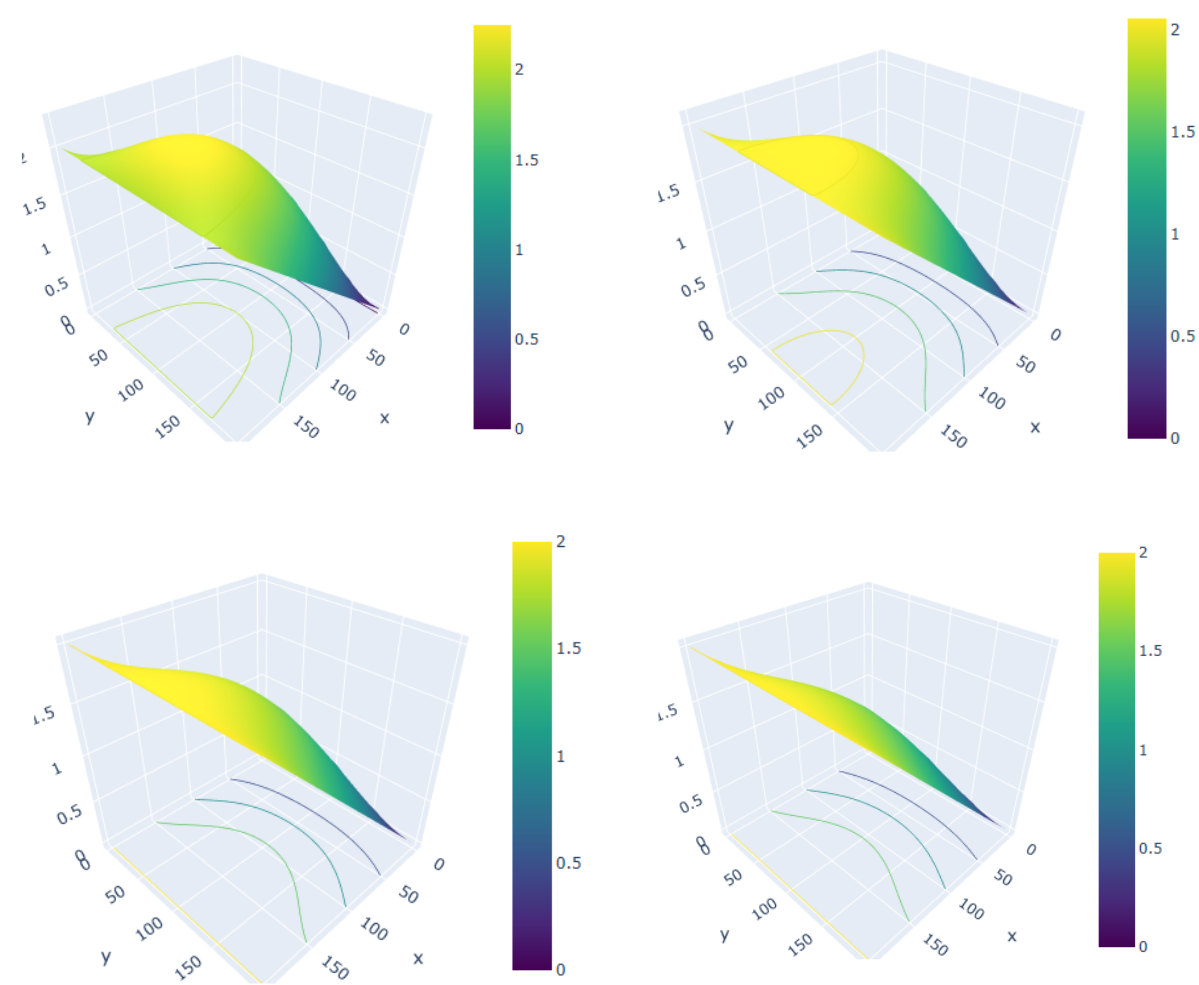
Code for FDM on Laplace Equation:

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go

xl = -1
xr = 1
yb = -1
yt = 1
N = 20
M = 20
def Laplace(xl,xr,yb,yt,M,N):

    #equation
    f=lambda x,y: 0

    # Dirichlet boundary conditions
    gb = lambda y: 0 # bottom
    gt = lambda y: 2 # top
    gl = lambda x: x + 1 # left
    gr = lambda x: x + 1 # right

    m=M+1
    n=N+1
    mn=m*n
```

```

    h=(xr-xl)/M
    h2=h**2
    k=(yt-yb)/N
    k2=k**2

    x=np.linspace(xl,xr,m)
    y=np.linspace(yb,yt,n)
    A=np.zeros([mn,mn])
    b=np.zeros([mn,1])

    for i in range(1,m-1):
        for j in range(1,n-1):
            A[i+(j)*m,i-1+(j)*m]=1/h2
            A[i+(j)*m,i+1+(j)*m]=1/h2
            A[i+(j)*m,i+(j)*m]=-(2/h2)-(2/k2)
            A[i+(j)*m,i+(j-1)*m]=1/k2
            A[i+(j)*m,i+(j+1)*m]=1/k2
            b[i+(j)*m]=f(x[i],y[j])

    for i in range(0,m):
        j=1
        A[i+(j-1)*m,i+(j-1)*m]=1
        b[i+(j-1)*m]=gl(x[i])
        j=n
        A[i+(j-1)*m,i+(j-1)*m]=1
        b[i+(j-1)*m]=gr(x[i])

    for j in range(2,n):
        i=0
        A[i+(j-1)*m,i+(j-1)*m]=1
        b[i+(j-1)*m]=gb(y[j])
        i=M
        A[i+(j-1)*m,i+(j-1)*m]=1
        b[i+(j-1)*m]=gt(y[j])

    #Find inverse of A
    Ainv = np.linalg.inv(A)
    # Find solution v = Ainv*b
    Solution = np.matmul(Ainv,b)
    # Reshape into a matrix
    Solution = np.reshape(Solution[0:mn],(n,m))
    return(Solution)

def plot_surface(solution):
    surface_df = pd.DataFrame(solution)
    surface_df.to_csv('surface_data.csv', index = False)
    surface_data = pd.read_csv('surface_data.csv')
    fig = go.Figure(data=[go.Surface(z=surface_data.values,
        colorscale='Viridis')])
```

```
    fig.update_traces(contours_z=dict(show=True, usecolormap=True,
        highlightcolor="limegreen", project_z=True))
    fig.update_layout(title='Solution', autosize=False, width=500,
        height=500,margin=dict(l=65, r=50, b=65, t=90),
        xaxis = dict(visible = False))
    fig.show()

plot_surface(Laplace(xl,xr,yb,yt,M,N))
```

Output ADI: initial, 10, 20, and 30 iterations, respectively:



As the ADI method is run for higher number of iterations on the given initial condition $u(x, y, 0) = (x^2 - 1)(y^2 - 1) + x + 1$ with the above boundary conditions we see that it is diffusing into a plane which happens to be the same as the solution to the above Laplace equation.

Code for ADI:

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go

# parameters
deltaxy1 = 1/100 # delta x = delta y
step1 = 200
deltat = 0.012
iterates1 = 10
iterates2 = 20
iterates3 = 30
s1 = (deltat/(deltaxy1**2)) # mu_x = mu_y

# initial condition
u = lambda x,y: (x**2-1)*(y**2-1) + x + 1

def initial_mat(u,n):
    initial=np.zeros(shape=(n,n))
    inter=np.linspace(-1,1,n)
    for i in range(0,len(inter)):
        for j in range(0,len(inter)):
            if inter[i] == -1:
                initial[i,j] = 0
            elif inter[i] == 1:
                initial[i,j] = 2
            elif inter[j] == -1:
                initial[i,j] = inter[i]+1
            elif inter[j] == 1:
                initial[i,j] = inter[i]+1
            else:
                initial[i,j] = u(inter[i],inter[j])
    initial = np.transpose(initial)
    return(initial)

# tridiagonal matrix
def tridag(L,M,U,k1=-1,k2=0,k3=1):
    return(np.diag(L,k1)+np.diag(M,k2)+np.diag(U,k3))

# one adi xy sweep
def adi_method(s, deltaxy, step, initial):
    # diagonals
    lower=[-s/2 for i in range(0,len(initial)-1)]
    main=[1+s for i in range(0,len(initial))]
    upper=[-s/2 for i in range(0,len(initial)-1)]

    # tridiagonal matrix
    M = tridag(lower,main,upper)
```

```

    # update edges of M for boundary conditions
    M[0,0] = 1
    M[0,1] = 0
    M[len(initial)-1, len(initial)-1] = 1
    M[len(initial)-1, len(initial)-2] = 0

    # calculate inverse
    Minv = np.linalg.inv(M)

    initial = np.transpose(initial)
    # x sweep
    for i in range(0, step):
        initial[i] = np.matmul(Minv,np.transpose(initial[i]))
        i = i + 1
    # y sweep
    initial = np.transpose(initial)
    for j in range(0, step):
        initial[j] = np.matmul(Minv,np.transpose(initial[j]))
        j = j + 1
    return initial

def adi_loop(s, deltaxy, step, initial, iterates):
    solution = adi_method(s, deltaxy, step, initial)
    for n in range(0,iterates):
        adi_method(s, deltaxy, step, solution)
        n = n + 1
    return solution

def plot_surface(solution):
    surface_df = pd.DataFrame(solution)
    surface_df.to_csv('surface_data.csv', index = False)
    surface_data = pd.read_csv('surface_data.csv')
    fig = go.Figure(data=[go.Surface(z=surface_data.values,
        colorscale='Viridis')])
    fig.update_traces(contours_z=dict(show=True, usecolormap=True,
        highlightcolor="limegreen", project_z=True))
    fig.update_layout(title='Solution', autosize=False, width=500,
     height=500,margin=dict(l=65, r=50, b=65, t=90),
     xaxis = dict(visible = False))
    fig.show()

# plot solutions
# initial
plot_surface(initial_mat(u,step1))
# after 10 iterations
plot_surface(adi_loop(s1, deltaxy1, step1,
    initial_mat(u,step1), iterates1))
# after 20 iterations
plot_surface(adi_loop(s1, deltaxy1, step1,
```

```
    initial_mat(u,step1), iterates2))
# # after 30 iterations
plot_surface(adi_loop(s1, deltaxy1, step1,
    initial_mat(u,step1), iterates3))
```