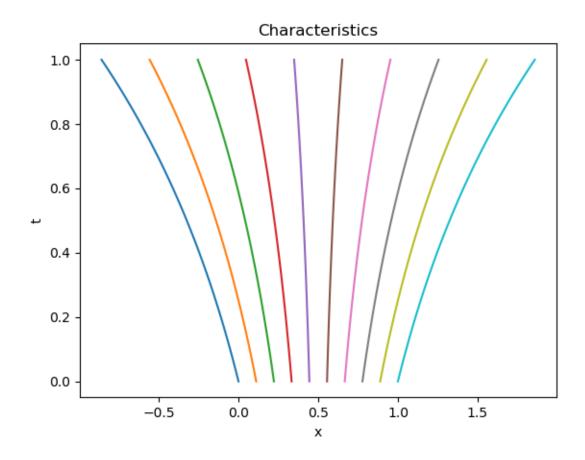
#### MATH 5620/6865: ASSIGNMENT\_6

#### JORDAN SAETHRE AND PAUL MUNDT

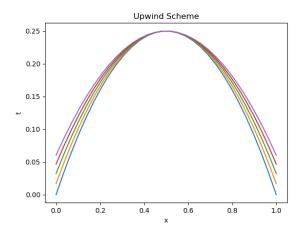
(1) Sketch the characteristics for the equation  $u_t + au_x = 0$  for  $0 \le x \le 1$  when  $a \equiv a(x) = x - \frac{1}{2}$ . Set up and implement the upwind scheme on a uniform mesh  $\{x_j = j\Delta x, j = 0, 1, \dots, J\}$  with an initial condition u(x,0) = x(1-x). Repeat the problem with  $a(x)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

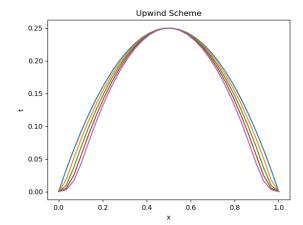
Characteristics: The characteristics of the equation  $u_t + au_x = 0$  for  $0 \le x \le 1$  when  $a \equiv a(x) = x - \frac{1}{2}$  are the solutions to the differential equation  $x' = x - \frac{1}{2}$ . The graphs of  $x(t) = Ce^t + 1/2$  when  $x(0) = \{1, 1/9, 2/9, 3/9, 4/9, 5/9, 6/9, 7/9, 8/9,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

```
1 | import numpy as np
2 | import matplotlib.pyplot as plt
3
4
5 | b = 1
6
   step = (b-a)/(n-1)
   iterates = 50
9
10
   # delta x = delta t
   xt_vector = np.linspace(a,b,n)
11
12
13 | # function a(x)
14 \mid a = lambda x: x - 1/2
15
   # initial condition function u(x,0)
16
17
   u = lambda x: x*(1-x)
18
19 | # initial condition
  | initial =np.array([u(xt_vector[i]) for i in range(len(xt_vector))])
20
21
22 | # vector for diagonal on matrix A
23 | a_vector = np.array([])
24
   for i in range(len(xt_vector)):
25
26
       if a(xt_vector[i])>0:
            a_vector = np.append(a_vector, 1 - a(xt_vector[i]))
27
28
       else:
            a_vector = np.append(a_vector, 1 + a(xt_vector[i]))
29
30
31 # diagonal matrix
```

```
A = np.diag(a_vector)
33
34
  # upwind scheme
35
   def upwind(initial, A, n):
       next_iterate = np.matmul(A,initial)
36
37
       for j in range(n):
38
          if a(xt_vector[j]) > 0:
39
              next_iterate[j] = next_iterate[j]
              + a(xt_vector[j])*(initial[j-1])
40
41
          else:
              next_iterate[j] = next_iterate[j]
42
              - a(xt_vector[j])*(initial[j+1])
43
       return (next_iterate)
44
45
46
   # plot solutions
   for i in range(iterates):
47
48
      if i%10 == 0:
49
          plt.plot(xt_vector, initial)
50
          initial = upwind(initial,A,n)
  plt.xlabel("x")
52 plt.ylabel("t")
  plt.title('Upwind Scheme')
54
  plt.show()
55
  56
   ### Repeat for Boundary Conditions and new a(x)#####################
57
   58
59
60 # function a(x)
  a = lambda x: 1/2 - x
61
62
63 # boundary condition
64 | # initial condition function u(x,0)
65 \mid u = lambda x: x*(1-x)
66
67 | # initial condition
68 | initial =np.array([u(xt_vector[i]) for i in range(len(xt_vector))])
69 \mid initial[0] = 0
70 \mid \text{initial} \lceil n-1 \rceil = 0
71
72 | # vector for diagonal on matrix A
  a_vector = np.array([])
73
74
  for i in range(len(xt_vector)):
75
       if a(xt_vector[i])>0:
76
77
           a_vector = np.append(a_vector, 1 - a(xt_vector[i]))
78
       else:
79
           a_vector = np.append(a_vector, 1 + a(xt_vector[i]))
80
81
  # diagonal matrix
```

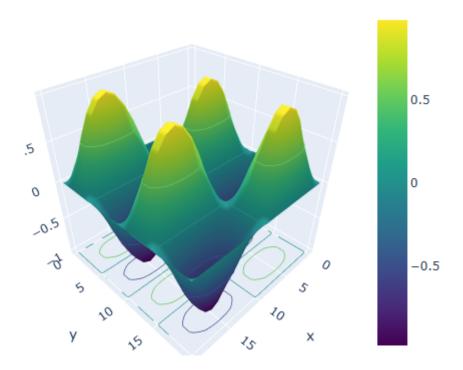
```
A = np.diag(a_vector)
82
83
   # plot solutions
84
85
   for i in range(iterates):
       if i%10 == 0:
86
            plt.plot(xt_vector, initial)
87
88
            initial = upwind(initial, A, n-1)
  plt.xlabel("x")
89
   plt.ylabel("t")
90
  plt.title('Upwind Scheme')
92 | plt.show()
```

(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  $\Delta$  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  $\Delta$  is considered a negative operator because the eigenvalue is negative.

## Code for FDM on Poisson Equation:

```
1 import numpy as np
2 | import pandas as pd
3 | import plotly.graph_objects as go
4
5 | x1 = -1
6 | xr = 1
7
   yb = -1
8
   yt = 1
9
   N = 20
10
   def Laplace(xl,xr,yb,yt,M,N):
11
12
        #equation
13
        f=lambda x,y: -5*np.pi**2*np.sin(np.pi*x)*np.sin(2*np.pi*y)
14
15
16
        # Dirichlet boundary conditions
17
        gb = lambda y: 0 # bottom
        gt = lambda y: 0 # top
18
19
        gl = lambda x: 0 # left
20
        gr = lambda x: 0 # right
21
22
        m = M + 1
23
        n = N + 1
24
        mn = m * n
25
26
       h = (xr - x1)/M
       h2=h**2
27
28
       k = (yt - yb)/N
29
        k2 = k * * 2
30
31
        x=np.linspace(x1,xr,m)
32
        y=np.linspace(yb,yt,n)
33
        A = np.zeros([mn,mn])
        b=np.zeros([mn,1])
34
35
36
        for i in range(1,m-1):
37
            for j in range(1,n-1):
                 A[i+(j)*m,i-1+(j)*m]=1/h2
38
39
                 A[i+(j)*m,i+1+(j)*m]=1/h2
                 A[i+(j)*m,i+(j)*m]=-(2/h2)-(2/k2)
40
41
                 A[i+(j)*m,i+(j-1)*m]=1/k2
                 A[i+(j)*m,i+(j+1)*m]=1/k2
42
43
                 b[i+(j)*m]=f(x[i],y[j])
44
        for i in range(0, m):
45
46
            j = 1
            A[i+(j-1)*m,i+(j-1)*m]=1
```

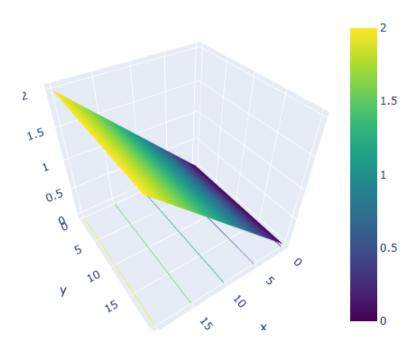
```
b[i+(j-1)*m]=gl(x[i])
48
49
            A[i+(j-1)*m,i+(j-1)*m]=1
50
           b[i+(j-1)*m]=gr(x[i])
51
52
53
       for j in range(2,n):
54
            A[i+(j-1)*m,i+(j-1)*m]=1
55
            b[i+(j-1)*m]=gb(y[j])
56
57
           A[i+(j-1)*m,i+(j-1)*m]=1
58
           b[i+(j-1)*m]=gt(y[j])
59
60
       #Find inverse of A
61
62
       Ainv = np.linalg.inv(A)
       # Find solution v = Ainv*b
63
64
       Solution = np.matmul(Ainv,b)
65
       # Reshape into a matrix
       Solution = np.reshape(Solution[0:mn],(n,m))
66
       return(Solution)
67
68
   def plot_surface(solution):
69
       surface_df = pd.DataFrame(solution)
70
71
       surface_df.to_csv('surface_data.csv', index = False)
       surface_data = pd.read_csv('surface_data.csv')
72
73
       fig = go.Figure(data=[go.Surface(z=surface_data.values,
74
            colorscale='Viridis')])
75
       fig.update_traces(contours_z=dict(show=True, usecolormap=True,
            highlightcolor="limegreen", project_z=True))
76
       fig.update_layout(title='Solution', autosize=False, width=500,
77
78
            height=500, margin=dict(1=65, r=50, b=65, t=90),
79
            xaxis = dict(visible = False))
       fig.show()
80
81
82 | 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:

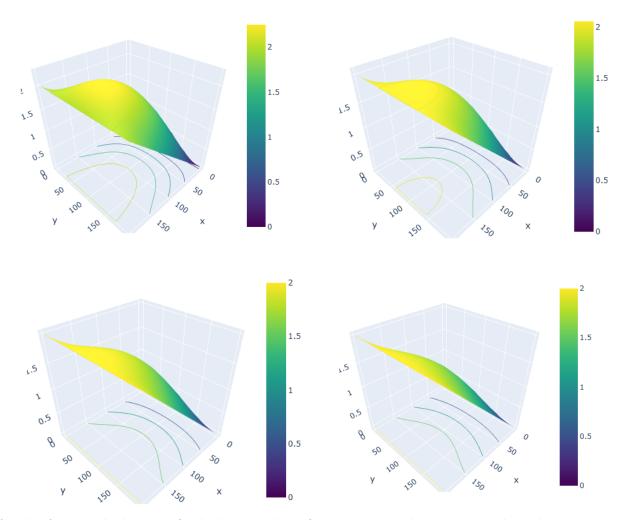
```
1 | import numpy as np
2 | import pandas as pd
3 | import plotly.graph_objects as go
4
5 | x1 = -1
6
   xr = 1
8
   yt = 1
   N = 20
9
10
   def Laplace(xl,xr,yb,yt,M,N):
11
12
13
        #equation
14
        f = lambda x, y: 0
15
16
        # Dirichlet boundary conditions
17
        gb = lambda y: 0 # bottom
18
        gt = lambda y: 2 # top
        gl = lambda x: x + 1 # left
19
        gr = lambda x: x + 1 # right
20
21
22
        m = M + 1
23
       n = N + 1
24
        mn = m * n
```

```
25
26
       h = (xr - x1)/M
27
       h2=h**2
       k = (yt - yb) / N
28
29
       k2 = k * * 2
30
31
       x=np.linspace(x1,xr,m)
       y=np.linspace(yb,yt,n)
32
33
       A = np.zeros([mn,mn])
34
       b=np.zeros([mn,1])
35
       for i in range (1, m-1):
36
37
            for j in range(1,n-1):
                A[i+(j)*m,i-1+(j)*m]=1/h2
38
39
                A[i+(j)*m,i+1+(j)*m]=1/h2
                A[i+(j)*m,i+(j)*m]=-(2/h2)-(2/k2)
40
41
                A[i+(j)*m,i+(j-1)*m]=1/k2
42
                A[i+(j)*m,i+(j+1)*m]=1/k2
                b[i+(j)*m]=f(x[i],y[j])
43
44
45
       for i in range(0, m):
46
            j=1
            A[i+(j-1)*m,i+(j-1)*m]=1
47
            b[i+(j-1)*m]=gl(x[i])
48
49
            j = n
            A[i+(j-1)*m,i+(j-1)*m]=1
50
            b[i+(j-1)*m]=gr(x[i])
51
52
       for j in range(2,n):
53
54
            i=0
            A[i+(j-1)*m,i+(j-1)*m]=1
55
            b[i+(j-1)*m]=gb(y[j])
56
57
            A[i+(j-1)*m,i+(j-1)*m]=1
58
59
            b[i+(j-1)*m]=gt(y[j])
60
61
       #Find inverse of A
62
       Ainv = np.linalg.inv(A)
63
       # Find solution v = Ainv*b
       Solution = np.matmul(Ainv,b)
64
       # Reshape into a matrix
65
66
       Solution = np.reshape(Solution[0:mn],(n,m))
67
       return(Solution)
68
   def plot_surface(solution):
69
70
       surface_df = pd.DataFrame(solution)
        surface_df.to_csv('surface_data.csv', index = False)
71
72
        surface_data = pd.read_csv('surface_data.csv')
73
       fig = go.Figure(data=[go.Surface(z=surface_data.values,
            colorscale='Viridis')])
74
```

```
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()
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:

```
1 import numpy as np
2 | import pandas as pd
3 | import plotly.graph_objects as go
4
5 # parameters
6 \mid deltaxy1 = 1/100 \# delta x = delta y
7
   step1 = 200
8 | deltat = 0.012
   iterates1 = 10
9
10 \mid \text{iterates2} = 20
11 \mid iterates3 = 30
12 \mid s1 = (deltat/(deltaxy1**2)) # mu_x = mu_y
13
14 | # initial condition
15 \mid u = lambda x, y: (x**2-1)*(y**2-1) + x + 1
16
17 def initial_mat(u,n):
18
       initial=np.zeros(shape=(n,n))
19
       inter=np.linspace(-1,1,n)
20
       for i in range(0,len(inter)):
21
            for j in range(0,len(inter)):
                if inter[i] == -1:
22
23
                     initial[i,j] = 0
24
                elif inter[i] == 1:
25
                     initial[i,j] = 2
26
                elif inter[j] == -1:
27
                     initial[i,j] = inter[i]+1
28
                elif inter[j] == 1:
                    initial[i,j] = inter[i]+1
29
30
                else:
31
                     initial[i,j] = u(inter[i],inter[j])
32
       initial = np.transpose(initial)
33
       return(initial)
34
   # tridiagonal matrix
35
   def tridag(L,M,U,k1=-1,k2=0,k3=1):
36
37
       return (np.diag(L,k1)+np.diag(M,k2)+np.diag(U,k3))
38
39 | # one adi xy sweep
   def adi_method(s, deltaxy, step, initial):
40
       # diagonals
41
42
       lower=[-s/2 for i in range(0,len(initial)-1)]
       main=[1+s for i in range(0,len(initial))]
43
       upper=[-s/2 for i in range(0,len(initial)-1)]
44
45
46
       # tridiagonal matrix
       M = tridag(lower, main, upper)
```

```
48
49
       # update edges of M for boundary conditions
50
       M[0,0] = 1
51
       M[0,1] = 0
52
       M[len(initial)-1, len(initial)-1] = 1
53
       M[len(initial)-1, len(initial)-2] = 0
54
       # calculate inverse
55
       Minv = np.linalg.inv(M)
56
57
       initial = np.transpose(initial)
58
       # x sweep
59
       for i in range(0, step):
60
            initial[i] = np.matmul(Minv, np.transpose(initial[i]))
61
62
           i = i + 1
63
       # y sweep
64
       initial = np.transpose(initial)
65
       for j in range(0, step):
66
            initial[j] = np.matmul(Minv, np.transpose(initial[j]))
            j = j + 1
67
68
       return initial
69
   def adi_loop(s, deltaxy, step, initial, iterates):
70
71
       solution = adi_method(s, deltaxy, step, initial)
72
       for n in range(0,iterates):
73
            adi_method(s, deltaxy, step, solution)
74
           n = n + 1
75
       return solution
76
   def plot_surface(solution):
77
78
       surface_df = pd.DataFrame(solution)
79
       surface_df.to_csv('surface_data.csv', index = False)
       surface_data = pd.read_csv('surface_data.csv')
80
       fig = go.Figure(data=[go.Surface(z=surface_data.values,
81
82
            colorscale='Viridis')])
       fig.update_traces(contours_z=dict(show=True, usecolormap=True,
83
84
           highlightcolor="limegreen", project_z=True))
       fig.update_layout(title='Solution', autosize=False, width=500,
85
86
        height=500, margin=dict(1=65, r=50, b=65, t=90),
        xaxis = dict(visible = False))
87
88
       fig.show()
89
   # plot solutions
   # initial
91
   plot_surface(initial_mat(u,step1))
93 | # after 10 iterations
94 | plot_surface(adi_loop(s1, deltaxy1, step1,
95
       initial_mat(u,step1), iterates1))
96 | # after 20 iterations
97 | plot_surface(adi_loop(s1, deltaxy1, step1,
```

```
98 | initial_mat(u,step1), iterates2))
99 | # # after 30 iterations
100 | plot_surface(adi_loop(s1, deltaxy1, step1,
101 | initial_mat(u,step1), iterates3))
```