Math 104C Homework 6 Code

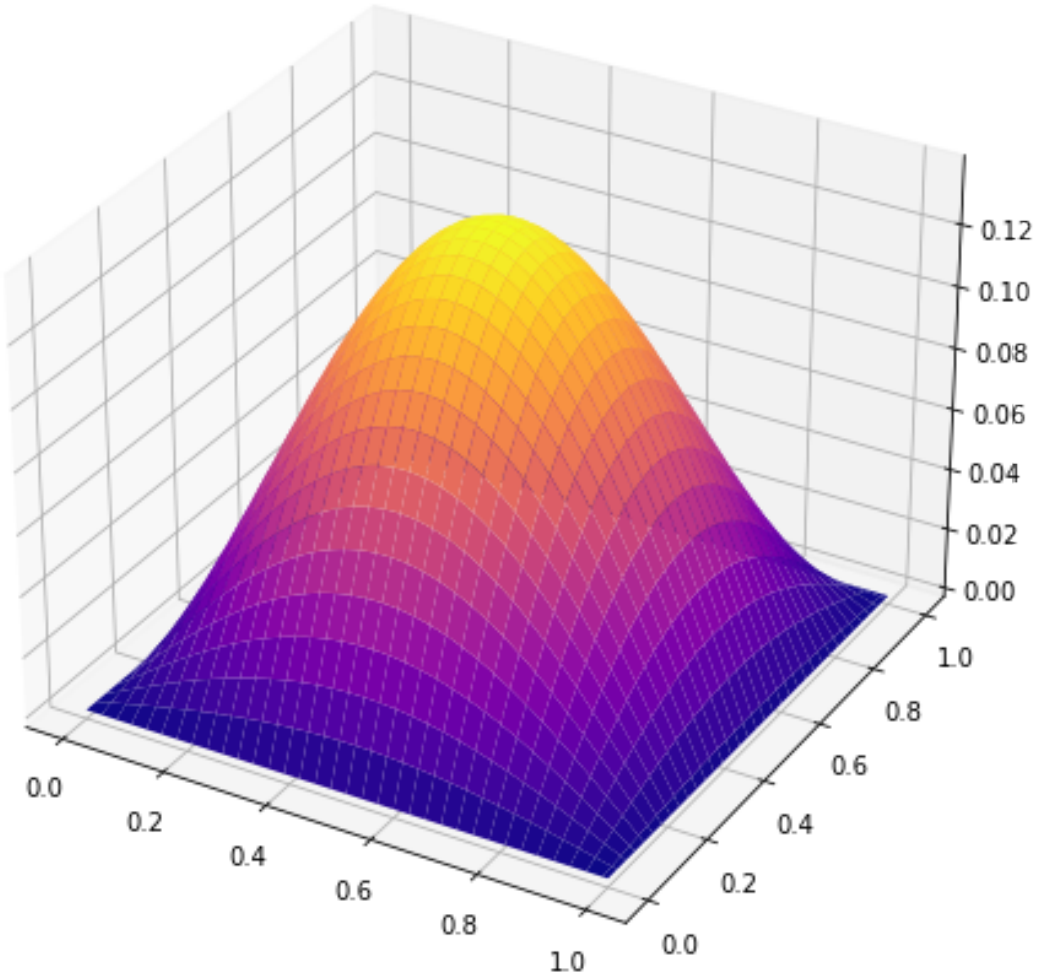
May 20, 2022

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[13]:
```
def construct_tridiag_mat(n,alpha):
    matrix = -2 * np.eye(n)

    for i in range(n-1):
        matrix[i+1][i] = 1
        matrix[i][i+1] = 1

    return alpha * matrix

def ADI(u_init,dx,dt,N):
    alpha = dt / (2*(dx**2))

    M = len(u_init)
    A = construct_tridiag_mat(M-2,alpha)
    I = np.eye(M-2)

    P = I - A
    Q = I + A

    current_approx = u_init[1:-1,1:-1]

    for i in range(1,N+1):
        star_approx = np.linalg.solve(P, np.matmul(Q,current_approx))
        current_approx = np.linalg.solve(P, np.matmul(Q,star_approx))

    return np.pad(current_approx,1,'constant')

dx = 1e-2
dt = 1e-3
M = 101
N = 100

x = np.linspace(0,1,M)
y = np.linspace(0,1,M)
```

```
x_values, y_values = np.meshgrid(x,y)
u_init = np.multiply(np.sin(np.pi * x_values), np.sin(np.pi * y_values))
u_approx = ADI(u_init,dx,dt,N)
z_values = u_approx

fig = plt.figure(figsize=(8,8))
ax = plt.axes(projection = '3d')
ax.plot_surface(x_values,y_values,z_values, cmap = 'plasma')
ax.set_title('Surface Mesh')

'''numerical stability doesn't play a role in choice of delta t and delta x␣
 ↪because the ADI method is unconditionally stable'''
```

[13]: "numerical stability doesn't play a role in choice of delta t and delta x because the ADI method is unconditionally stable"



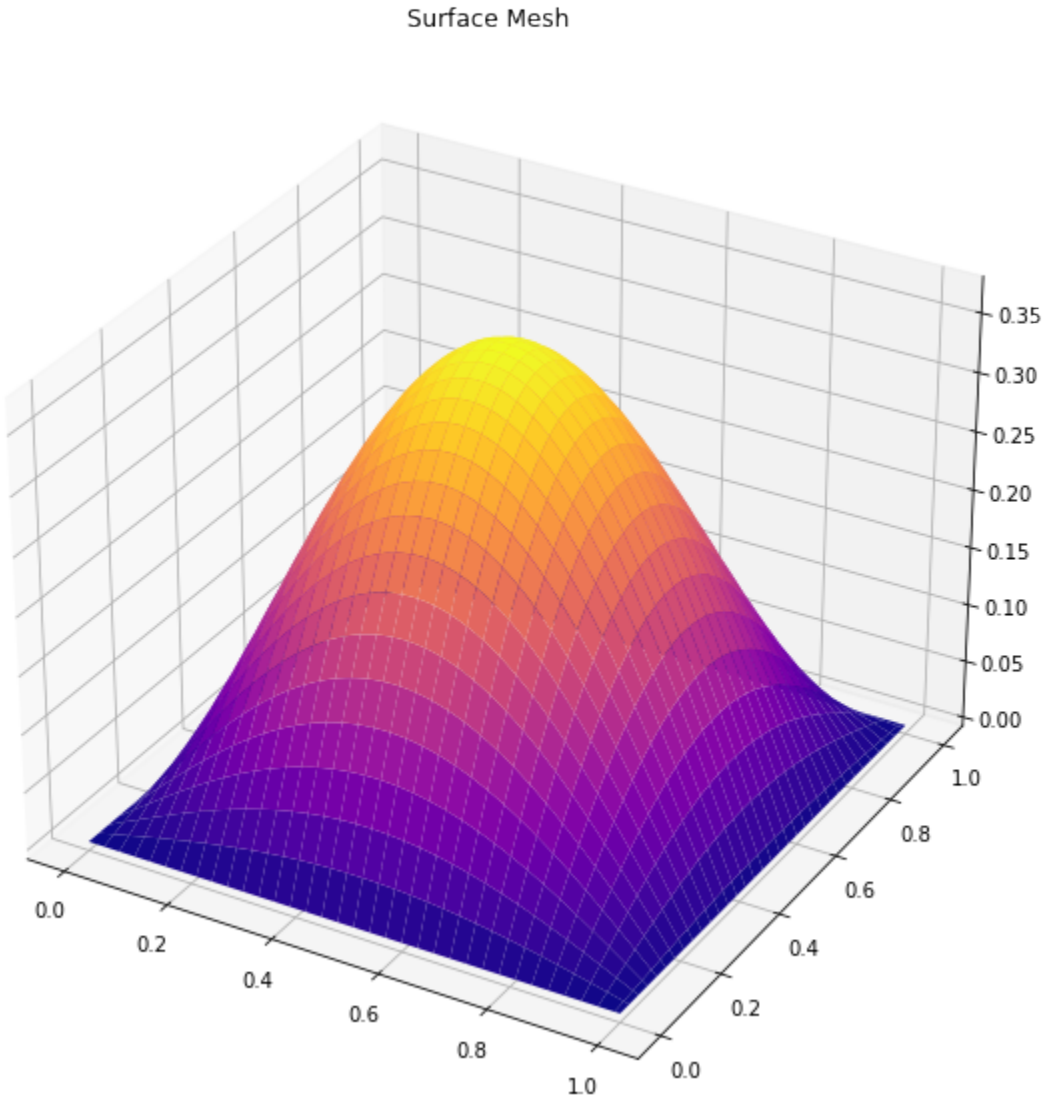
[12]:
```
M = N = 64
h = 1/M
x = np.linspace(0,1,M+1)
y = np.linspace(0,1,M+1)
x_values, y_values = np.meshgrid(x,y)
u_init = np.multiply(np.sin(np.pi*x_values), np.sin(np.pi*y_values))
u_h = ADI(u_init,h,h,N)

M2 = N2 = 128
h2 = 1/M2
x2 = np.linspace(0,1,M2+1)
y2 = np.linspace(0,1,M2+1)
```

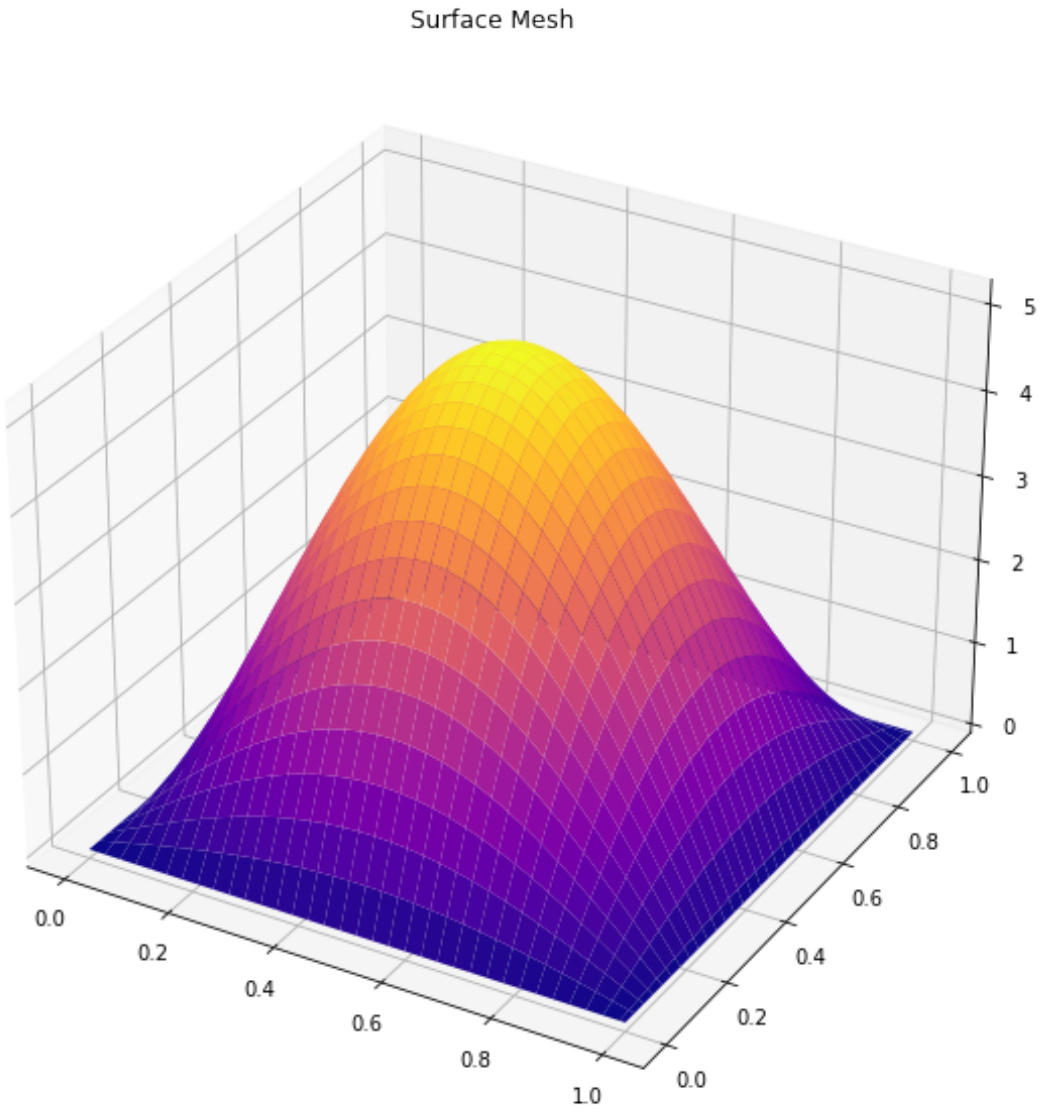
```
x_values2, y_values2 = np.meshgrid(x2,y2)
u_init2 = np.multiply(np.sin(np.pi*x_values2), np.sin(np.pi*y_values2))
u_h2 = ADI(u_init2,h2,h2,N2)


M4 = N4 = 256
h4 = 1/M4
x4 = np.linspace(0,1,M4+1)
y4 = np.linspace(0,1,M4+1)
x_values4, y_values4 = np.meshgrid(x4,y4)
u_init4 = np.multiply(np.sin(np.pi*x_values4), np.sin(np.pi*y_values4))
u_h4 = ADI(u_init4,h4,h4,N4)

order = (u_h[32,32] - u_h2[64,64]) / (u_h2[64,64] - u_h4[128,128])

print(order, 'Here we see that we have a second order rate of convergence')
```

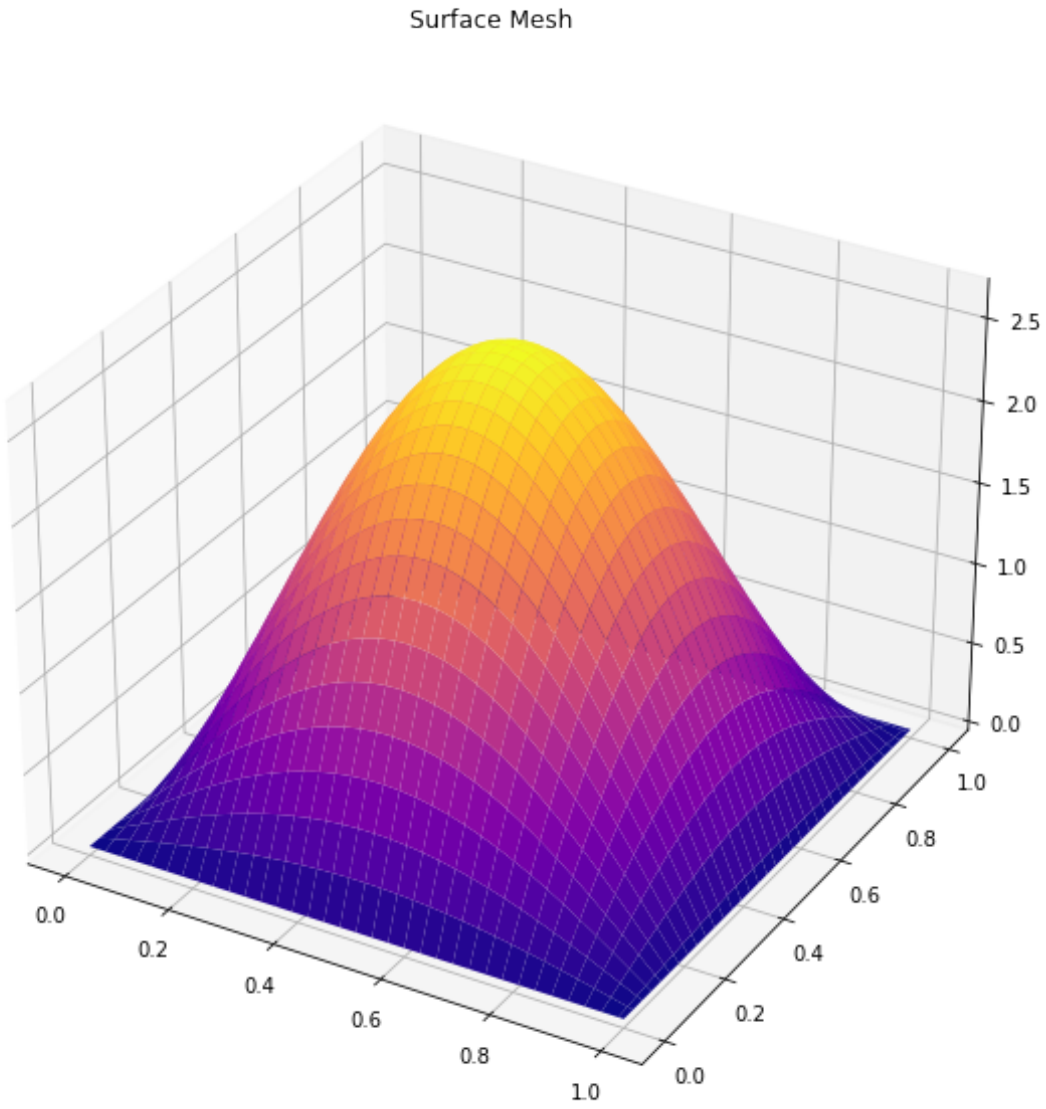
```
3.9467556748521724 Here we see that we have a second order rate of convergence
```

[14]:
```
## part 2C

u_approx = ADI(u_init, dx,dt,50)
z_values = u_approx
fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection = '3d')
ax.plot_surface(x_values,y_values,z_values,cmap = 'plasma')
ax.set_title('Surface Mesh')

u_approx = ADI(u_init, dx,dt,500)
z_values = u_approx
fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection = '3d')
ax.plot_surface(x_values,y_values,z_values,cmap = 'plasma')
ax.set_title('Surface Mesh')

u_approx = ADI(u_init, dx,dt,1000)
z_values = u_approx
fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection = '3d')
ax.plot_surface(x_values,y_values,z_values,cmap = 'plasma')
ax.set_title('Surface Mesh')
```

[14]:
```
Text(0.5, 0.92, 'Surface Mesh')
```


Surface Mesh
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


Surface Mesh
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



2D.

Here we should use the ADI method, simply because it is computationally cheaper while producing the same results.