ME 789 CTPM : Course Project
Vibration in Circular membrane

Submitted by
Deepak Kumar Thakur

Under the Supervision of
Prof.Shyamprasad Karagadde

Department of Mechanical Engineering
Indian Institute of Technology Bombay,400076

Vibration in String

$$\alpha^2 \frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial t^2}$$

$$\frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{\Delta x^2} = \frac{1}{\alpha^2} \frac{u_i^{k-1} - 2u_i^k + u_i^{k+1}}{\Delta t^2}$$

Deriving dt (time marching) from stability condition

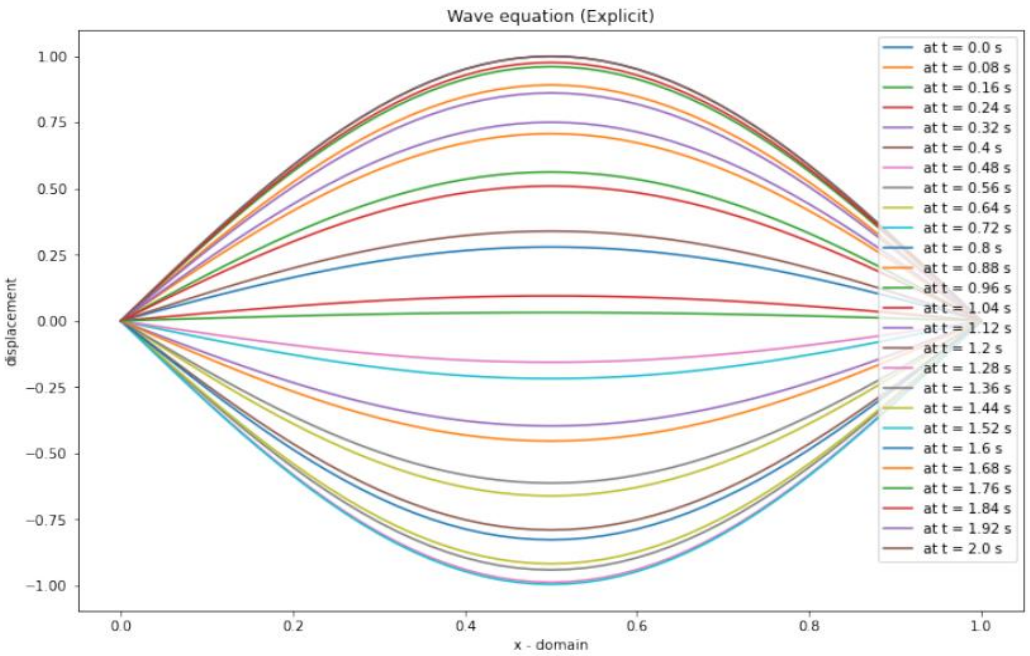
dt = 100 dx^2/α

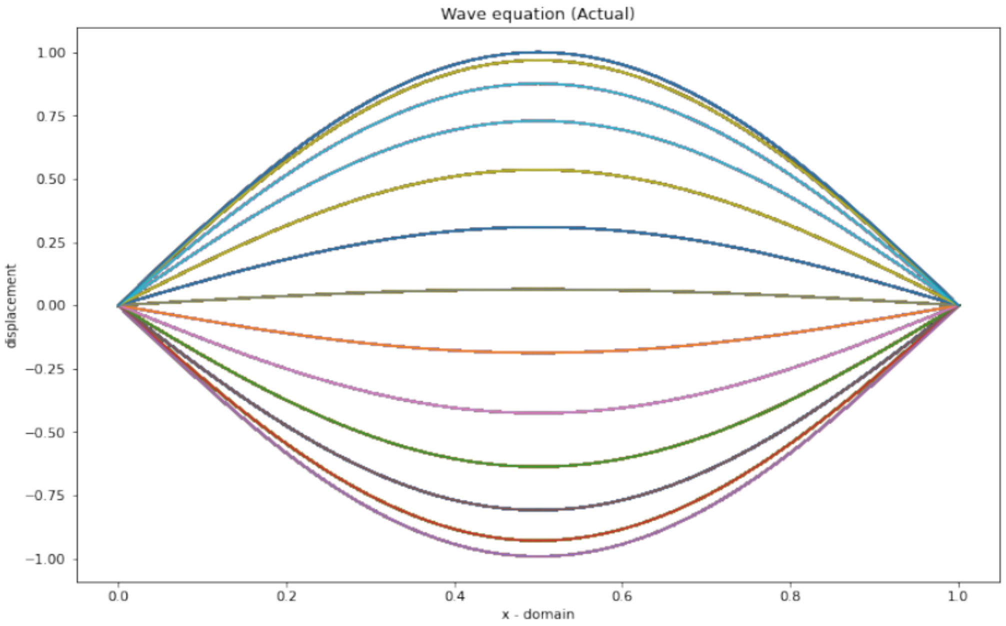
Boundary condition

u(0,t) = 0 and u(l,t) = 0

Initial Condition

u(x, 0) = sin(πx) and ut(x,0) = 0





Vibration in Rectangular Membrane(2D Cartesian Coordinate)

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \nabla^2 u = \frac{1}{a^2}\frac{\partial^2 u}{\partial t^2}$$

$$\frac{u_{i-1,j}^k - 2u_{i,j}^k + u_{i+1,j}^k}{\Delta x^2} + \frac{u_{i,j-1}^k - 2u_{i,j}^k + u_{i,j+1}^k}{\Delta y^2} = \frac{1}{a^2}\frac{u_{i,j}^{k-1} - 2u_{i,j}^k + u_{i,j}^{k-1}}{\Delta t^2}$$

Deriving dt (time marching) from stability condition

$C_x = C_y$

$C_x = 1/\sqrt{2}$

$dt = c_x\, dx$

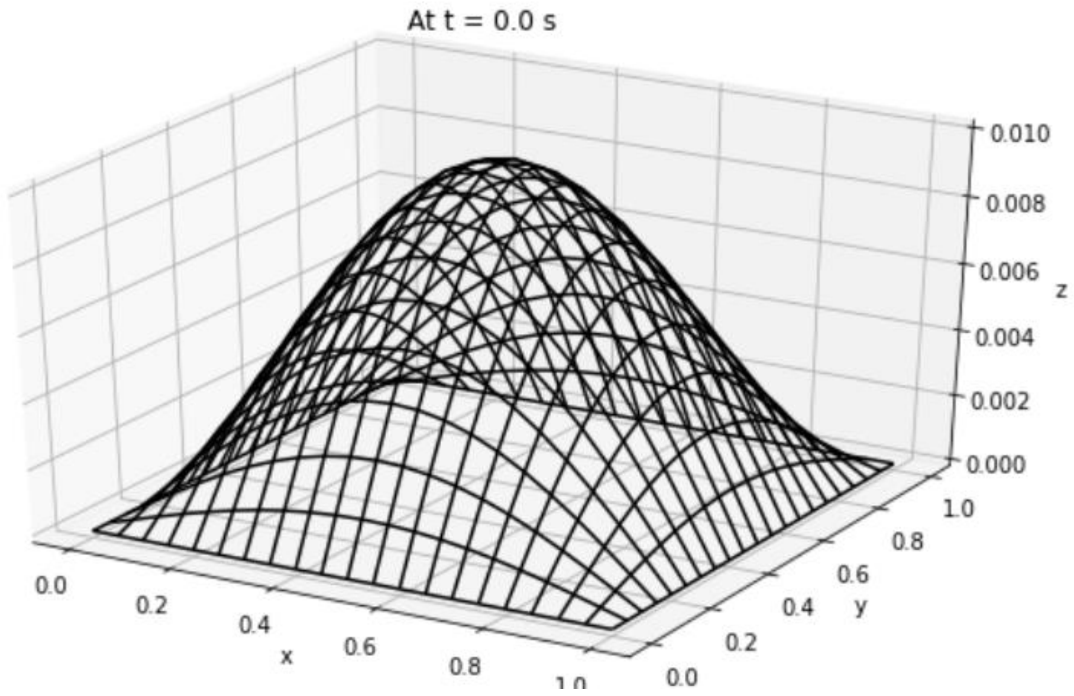
Boundary condition

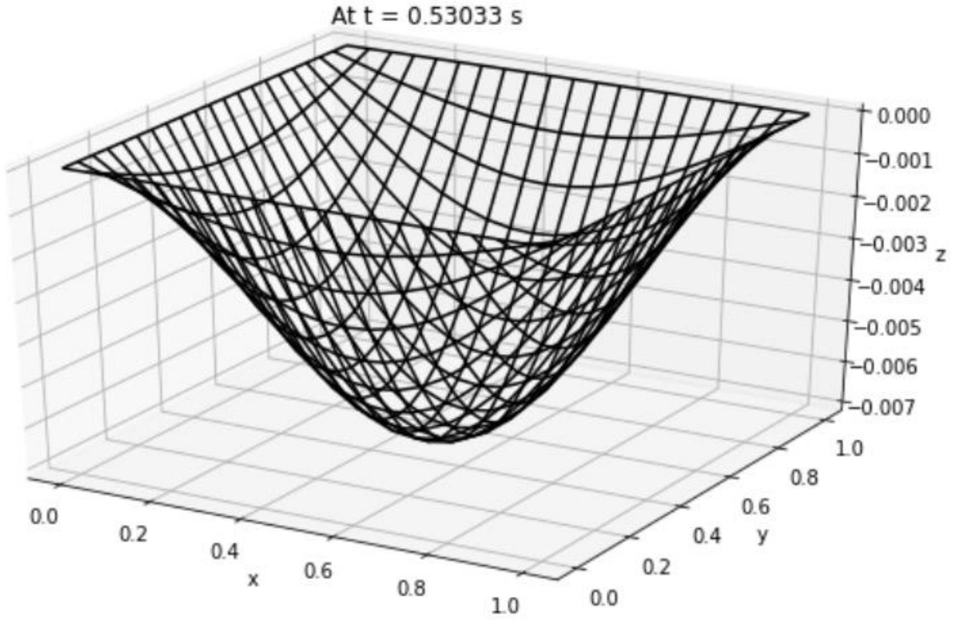
Zeros at all edges

Initial Condition

$u(x, y, 0) = f(x, y) = 0.01 \sin(\pi x) \sin(\pi y)$

with zero velocity everywhere





Vibration in Circular Membrane (2D Polar Coordinate)

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} = \frac{1}{a^2}\frac{\partial^2 u}{\partial t^2}$$

Axisymmetric

$$u_0^{k+1} = 4C^2 u_1^k + 2(1-2C^2)u_0^k - u_0^{k-1}$$

@At centre node

$$u_i^{k+1} = C^2\left(1-\frac{\Delta r}{2r_i}\right)u_{i-1}^k + 2(1-C^2)u_i^k + C^2\left(1+\frac{\Delta r}{2r_i}\right)u_{i+1}^k - u_i^k$$

Deriving dt (time marching) from stability condition

dt = c dr

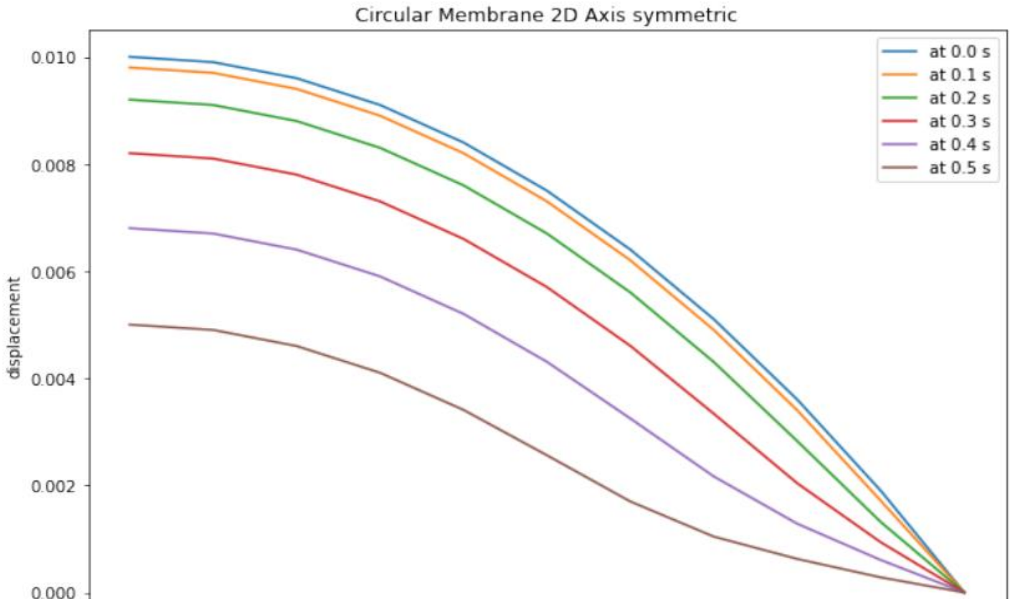
Boundary condition

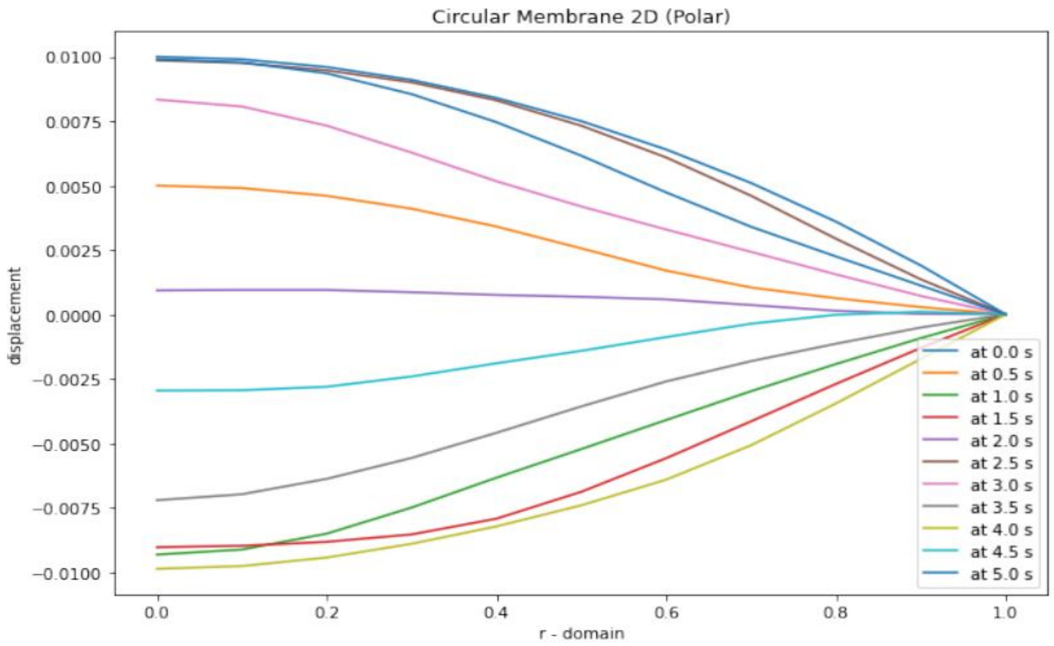
Zeros at r = R

Initial Condition

u(R, theta) = f (r) = 0.01(1− r ^{2})

with zero velocity everywhere





Vibration in Circular Membrane (2D Polar Coordinate)

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right)+\frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}=\frac{1}{a^2}\frac{\partial^2 u}{\partial t^2}$$

$$u_{i,j}^{k+1}=C_r^2\left(1-\frac{\Delta r}{r_i}\right)u_{i-1,j}^k+2\left\{1-C_r^2\left(1+\left[\frac{\Delta r}{r_i\Delta\theta}\right]^2\right)\right\}u_{i,j}^k+C_r^2\left(1+\frac{\Delta r}{r_i}\right)u_{i+1,j}^k$$
$$+\left(\frac{C_r\Delta r}{r_i\Delta\theta}\right)^2(u_{i,j-1}^k+u_{i,j+1}^k)-u_{i,j}^{k-1}$$

@At centre node

$$u^{k+1}(0,0)=C^2[u^k(-\Delta r,0)+u^k(\Delta r,0)+u^k(0,-\Delta r)+u^k(0,\Delta r)]$$
$$-2(1-C^2)u^k(0,0)-u^{k-1}(0,0)$$

$$\Delta t\le\frac{1}{\sqrt{\frac{1}{\Delta r^2}+\frac{1}{r_{min}^2\Delta\theta^2}}}$$

Boundary condition

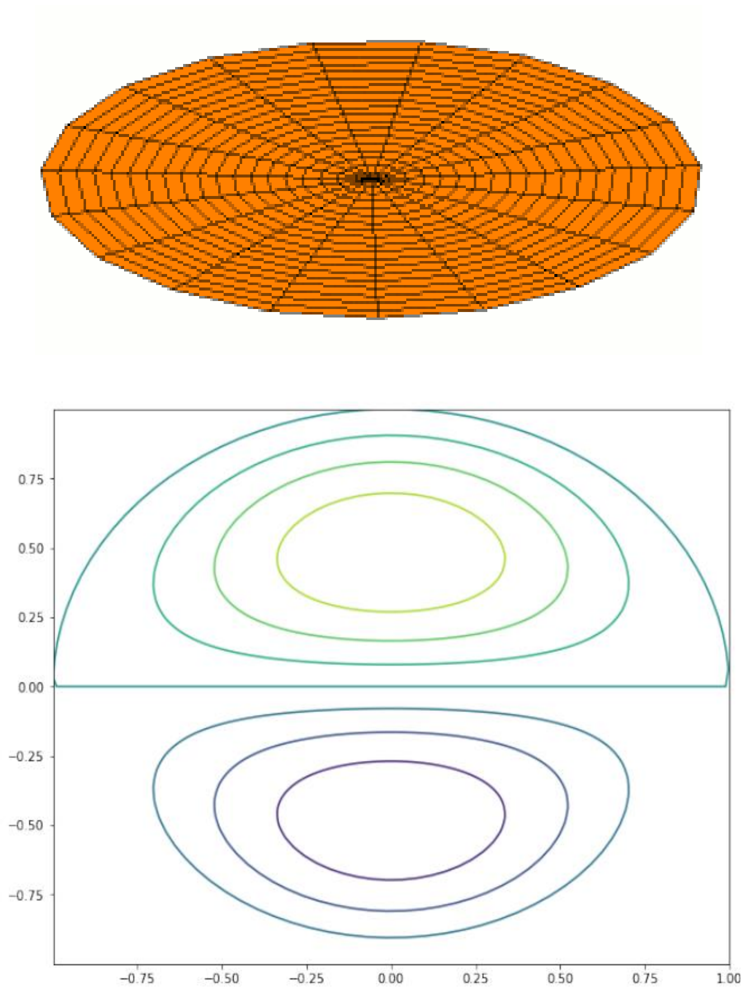
Zeros at r = R

Initial Condition

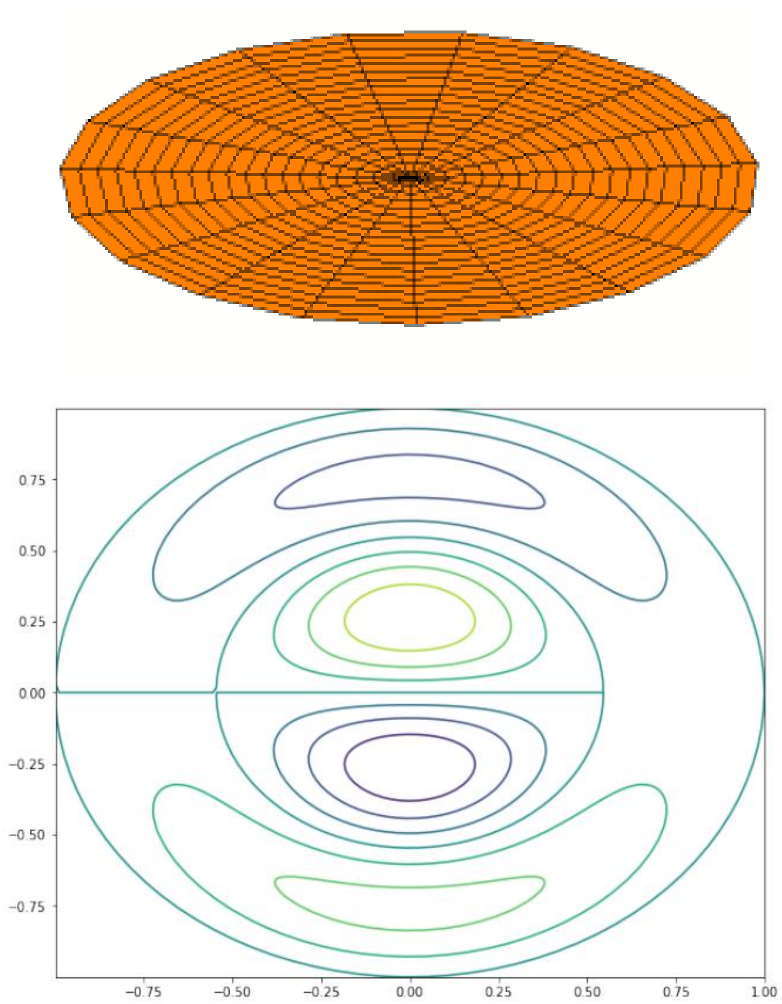
u = 0.01 sin(rπ) cosθ

with zero velocity everywhere

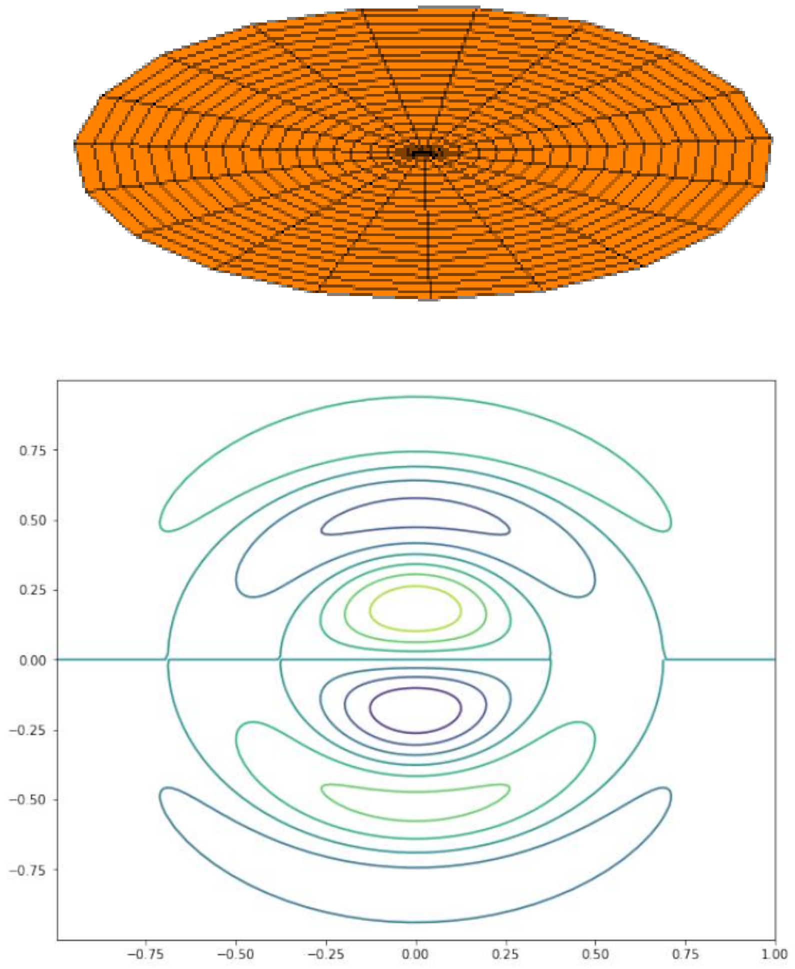
Vibration in Circular Membrane (Different modes)



U01

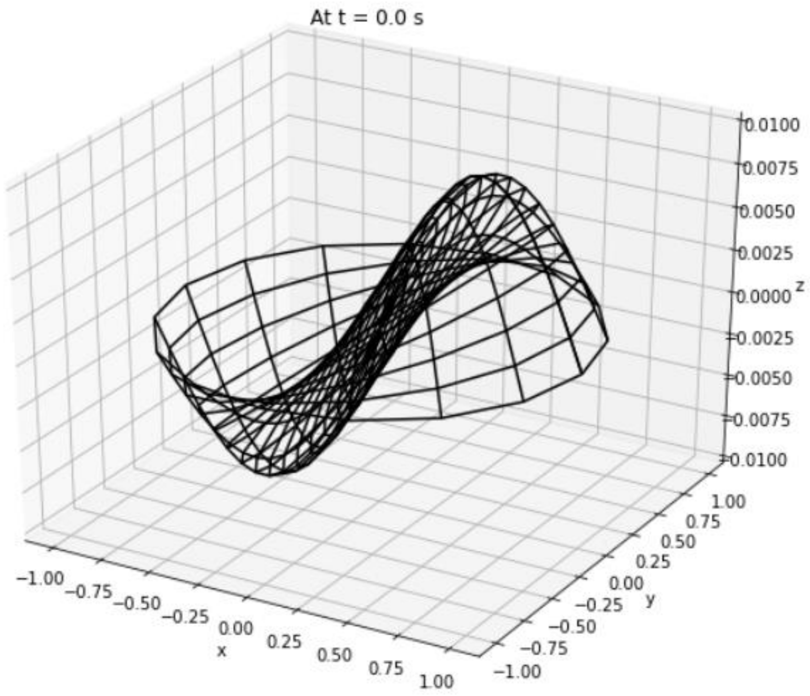


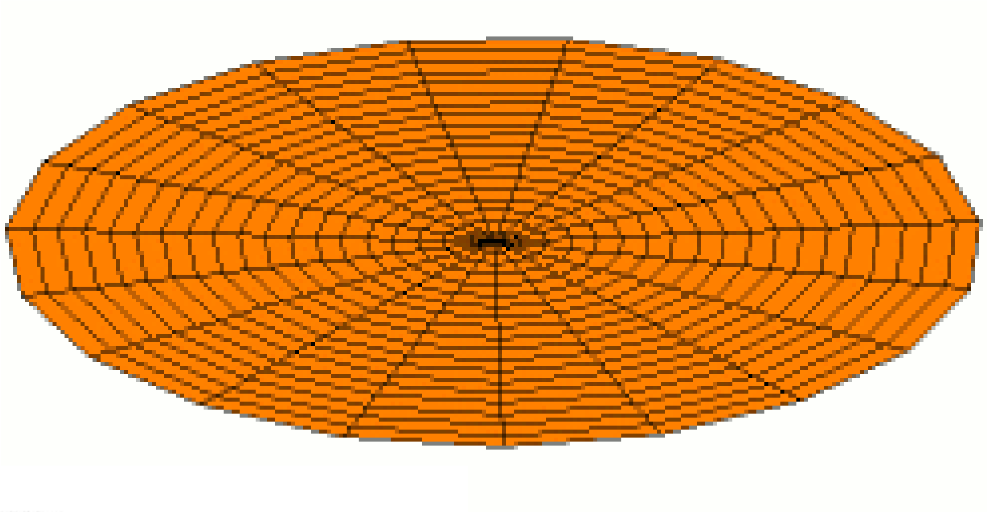
U02

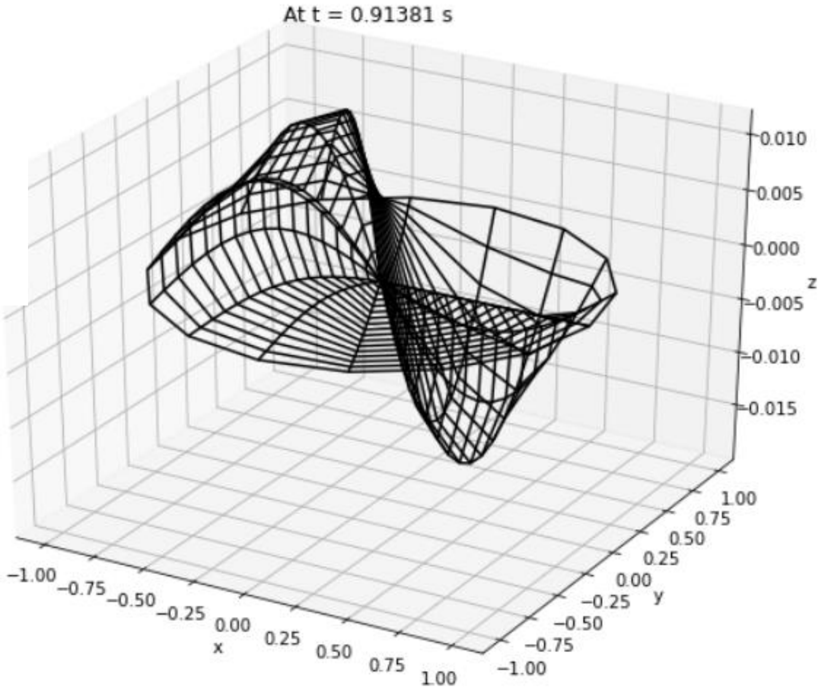


U03

Vibration in Circular Membrane (2D Polar Coordinate)







U11

Thanks!

Any question?

You can find me at:

213101001@iitb.ac.in

Reference:

1. Computational Methods in Engineering, S.P.Venkateshan
2. Vibrations of a circular membrane - Wikipedia

213101001_project

April 5, 2022

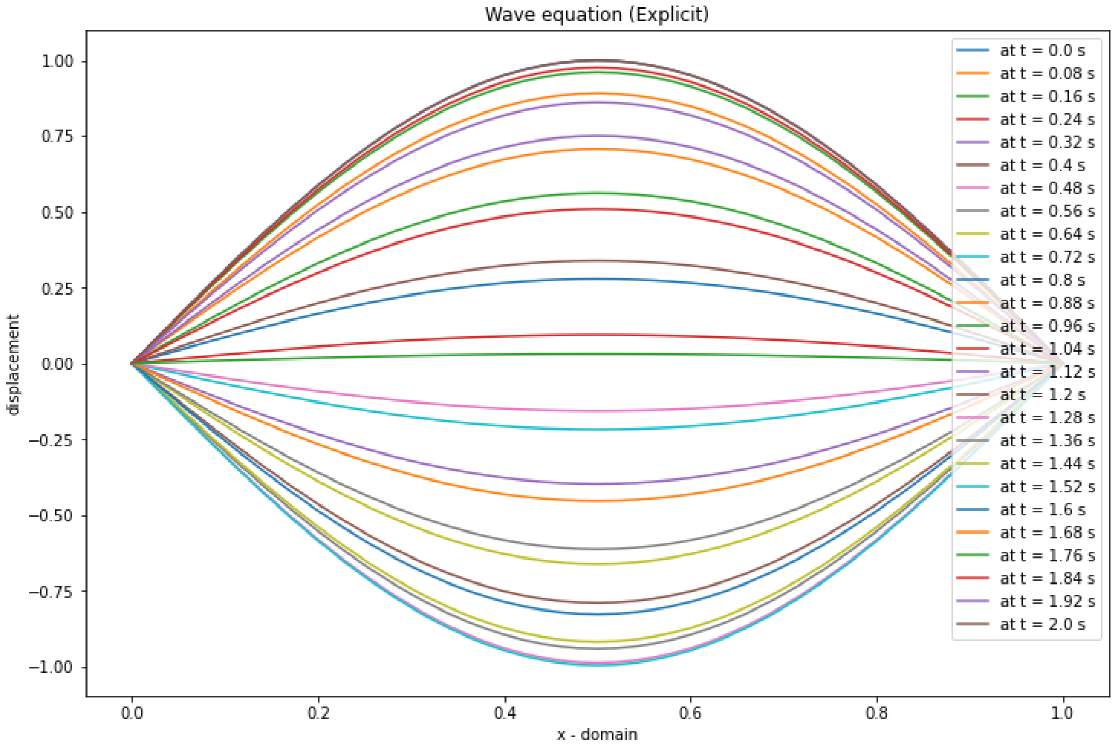
String Vibration (1D in Cartesian)

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



n = 50 #no. of elements
l = 1

dx  = l/n
x = np.linspace(0, l, n+1, endpoint = True)
print('x - domain : ', x)
print('dx : ', dx)

alpha  = 4
print('alpha : ', alpha)
dt = 100*dx**2/alpha
print('dt : ', dt)
c = alpha*dt**2/dx**2
# c = 1.5
print("c : ",c)

U = []

u_0 = np.zeros(n+1)
u_1 = np.zeros(n+1)
u_2 = np.zeros(n+1)

#Boundary conditions
#u(0,t) = 0 and u(l,t) = 0
u_0[0] = 0
u_0[-1] = 0

#Initial conditions
#u(x, 0) = sin(pi * x) and ut(x,0) = 0

for i in range(1, len(x)-1):
  u_0[i] = np.sin(np.pi * x[i])
```

```
U.append(u_0)

u_1 = u_0.copy() #applying first derivative
U.append(u_1)

t = 0
t_end = 200

#since at t = 0 and t = dt, u(x,t) is known, changes to dt

while t <= t_end:
  for i in range(1, len(x)-1):
    u_2[i] = c*u_1[i+1] + 2*(1-c)* u_1[i] + c*u_1[i-1] - u_0[i]
  u_2[0] = 0
  u_2[-1] = 0
  U.append(u_2)
  u_0 = u_1
  u_1 = u_2
  #print(u_2)
  u_2 = np.zeros(n+1)
  t = t + 1
fig, ax = plt.subplots(1,figsize = (12,8))
for i in range(0, len(U), 8):
  plt.plot(x, U[i],label = 'at t = '+ str(round(i*dt,2)) + ' s')
ax.set_title("Wave equation (Explicit)")
ax.set_xlabel('x - domain')
ax.set_ylabel('displacement')
ax.legend()
```

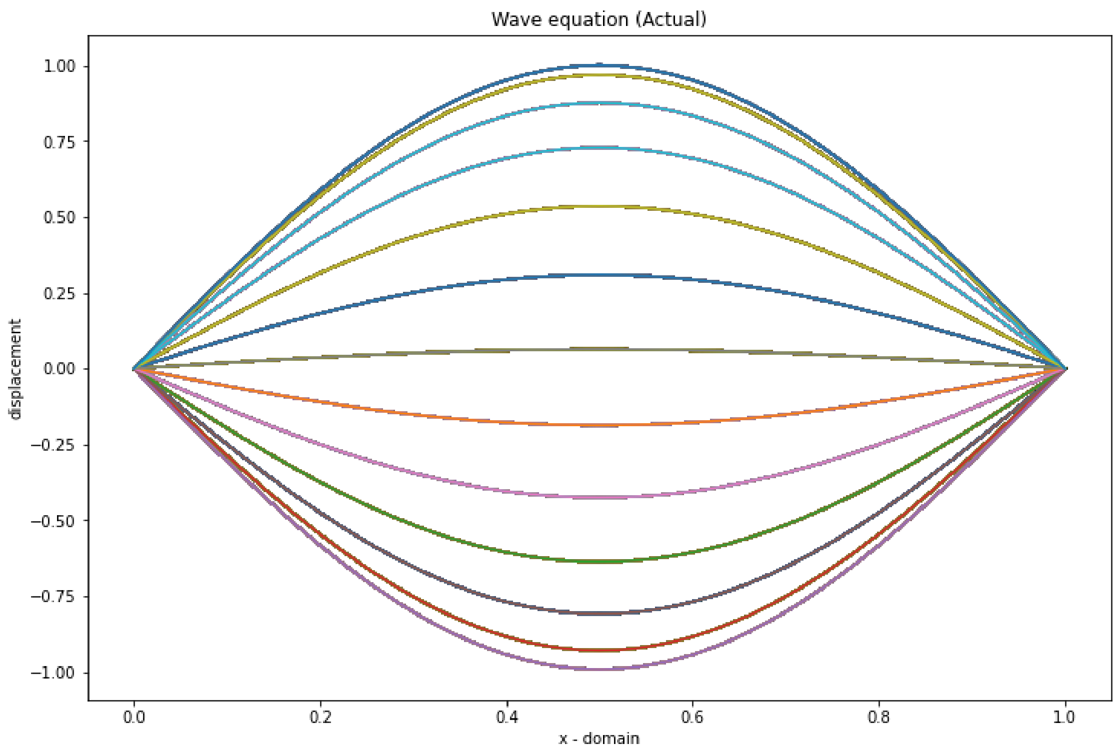
```
x - domain :  [0.   0.02 0.04 0.06 0.08 0.1  0.12 0.14 0.16 0.18 0.2  0.22 0.24
0.26
 0.28 0.3  0.32 0.34 0.36 0.38 0.4  0.42 0.44 0.46 0.48 0.5  0.52 0.54
 0.56 0.58 0.6  0.62 0.64 0.66 0.68 0.7  0.72 0.74 0.76 0.78 0.8  0.82
 0.84 0.86 0.88 0.9  0.92 0.94 0.96 0.98 1.  ]
dx :  0.02
alpha :  4
dt :  0.01
c :  1.0
```

```
[1]: <matplotlib.legend.Legend at 0x7fd7eac84810>
```



[2]:
```python
def actual_solution(x, t):
  u_actual = []
  for i in x:
    u_actual.append( np.sin(np.pi*i) * np.cos(2*np.pi * t))
  return u_actual

U_actual = []
t = 0
while t <= t_end:
  U_actual.append(actual_solution(x, t))
  #print(actual_solution(x, t))
  t = t + dt

fig, ax = plt.subplots(1,figsize = (12,8))
for i in range(0, len(U_actual), 8):
  plt.plot(x, U_actual[i])
ax.set_title("Wave equation (Actual)")
ax.set_xlabel('x - domain')
ax.set_ylabel('displacement')
```

[2]:
```
Text(0, 0.5, 'displacement')
```

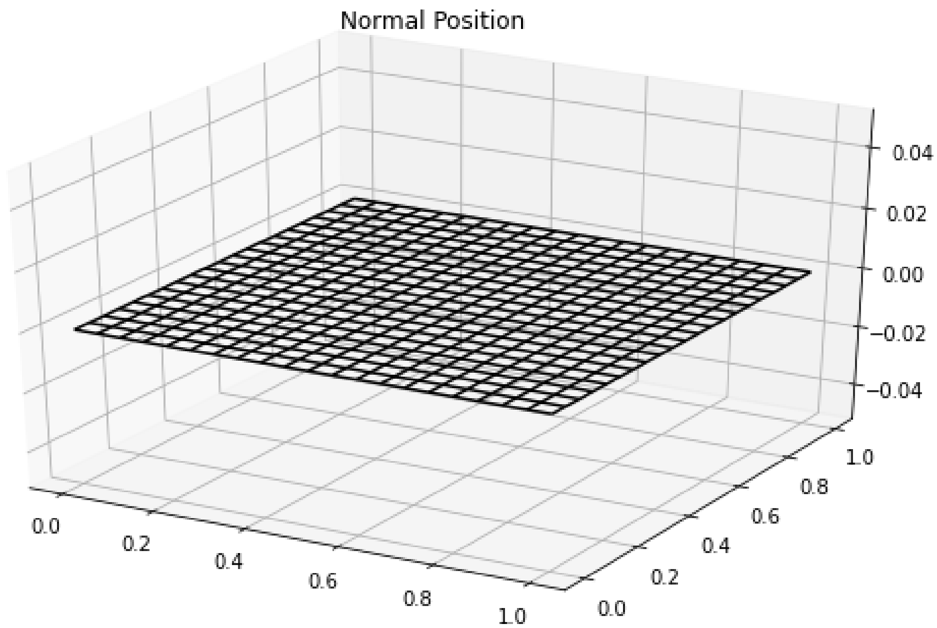


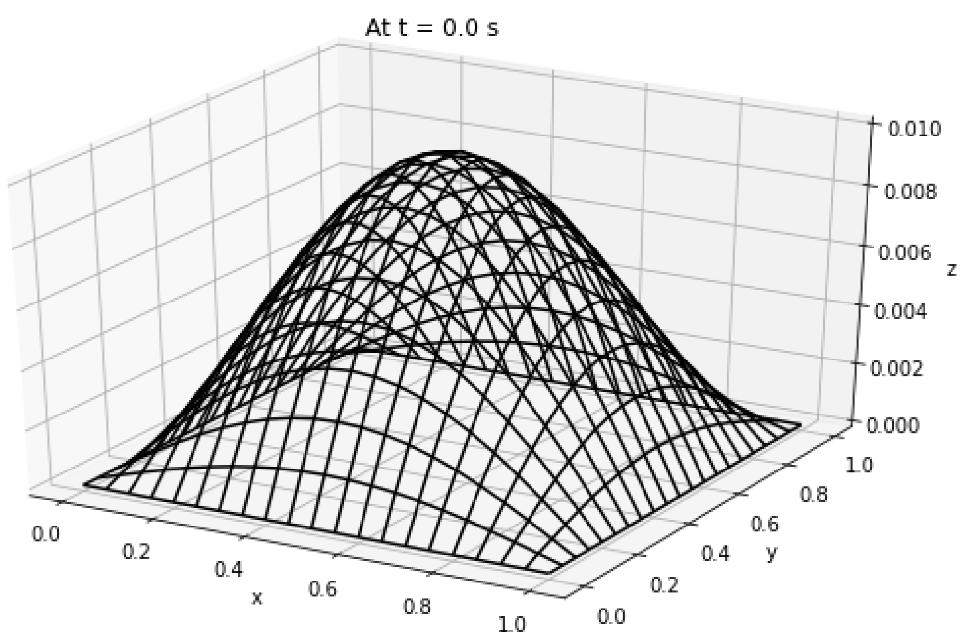
Rectangular Membrane

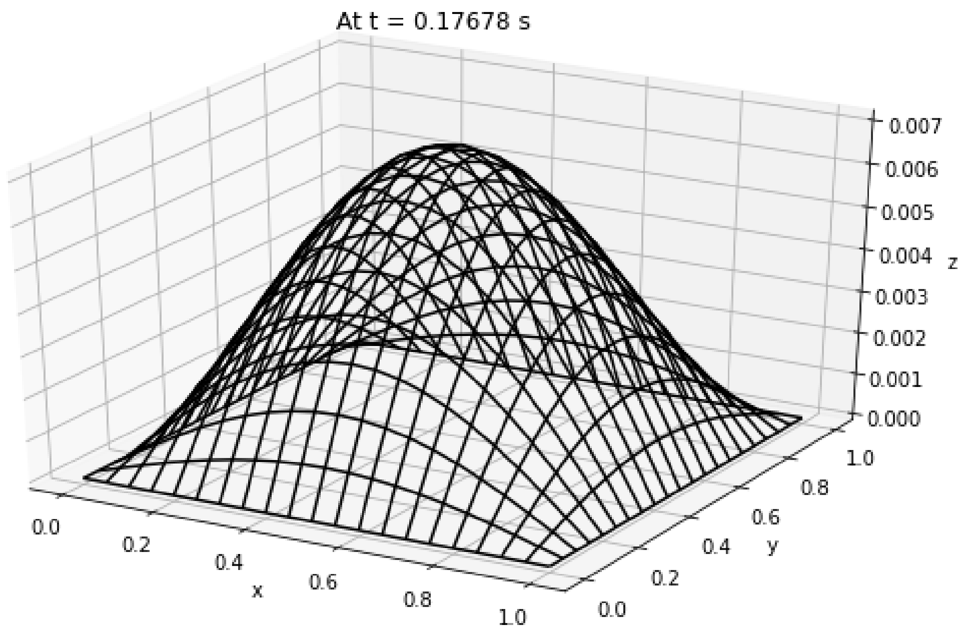
[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
# square 1 x 1
nx = 21
ny = 21
lx  = 1
ly = 1

U = []

# 1-2 C^2   0
x = np.linspace(0, lx, nx, endpoint = True)
y = np.linspace(0, ly, ny, endpoint = True)
print('x-domain : ', x)
print('y-domain : ', y)

dx = 1/( nx -1)
dy = 1/( ny -1)

[Y, X] = np.meshgrid(y, x) #creating mesh
```

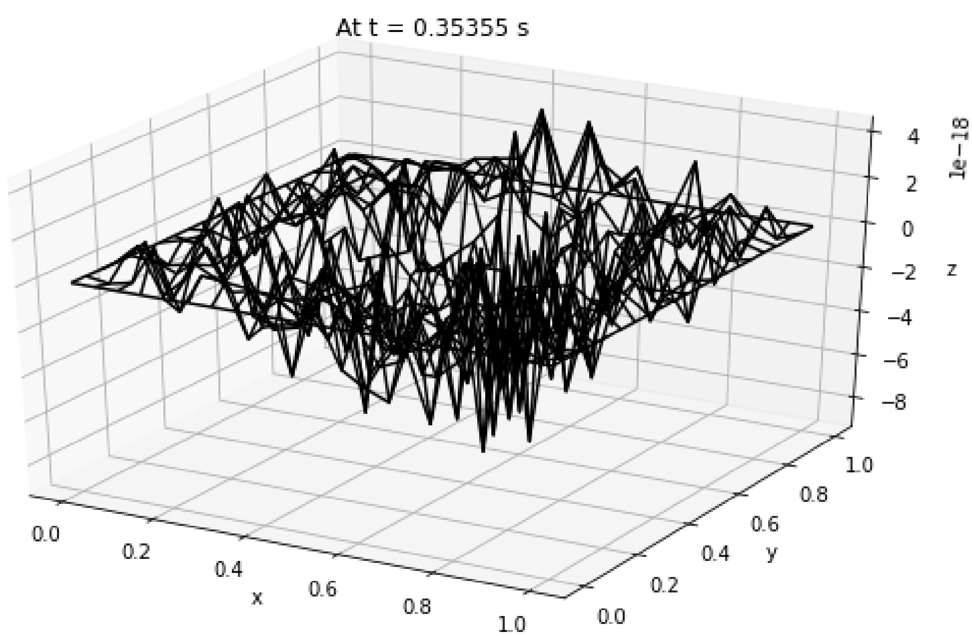
```
u0 = 0.01* np.sin(np.pi * X) * np.sin(np.pi * Y)
U.append(u0)
u1 = np.zeros((nx, ny))

cx  = 1/(2**(1/2))
dt = cx*dx
cy  = dt/dy #cx = cy

print('dt : ',dt)

# first time step when c^2  = 1/(2)^(1/2)
for i in range(1, nx-1):
  for j in range(1, ny-1):
    u1[i, j] = (u0[i-1,j] + u0[i+1, j] + u0[i, j-1] + u0[i, j+1])/4

U.append(u1)
u2 = np.zeros((nx, ny))

for t in range(2, 101):
  for i in range(1, nx-1):
    for j in range(1, ny-1):
      u2[i,j] =  0.5* (u1[i-1,j] + u1[i+1, j] + u1[i,j-1] + u1[i, j+1])-
 ↪u0[i,j] + 2 *(1 - 2 * 0.5) * u1[i, j]
  u0 = u1.copy()
  u1 = u2.copy()
  U.append(u2)
  u2 = np.zeros((nx,ny))


def plotting_2d(x,y,z,i):
  X, Y = np.meshgrid(x, y)
  fig = plt.figure(figsize = (10,6))
  ax = plt.axes(projection='3d')
  ax.plot_wireframe(X, Y, z, color='#000000')
  ax.set_xlabel('x')
  ax.set_ylabel('y')
  ax.set_zlabel('z')
  ax.set_title('At t = '+ str(round(i * dt, 5))+ ' s')

X,Y = np.meshgrid(x,y)
fig = plt.figure(figsize = (10,6))
ax  = plt.axes(projection='3d')
ax.plot_wireframe(X,Y,np.zeros((nx,ny)), color="black")
ax.set_title('Normal Position')

for i in range(0, 51,5):
  plotting_2d(x,y,U[i],i)
```

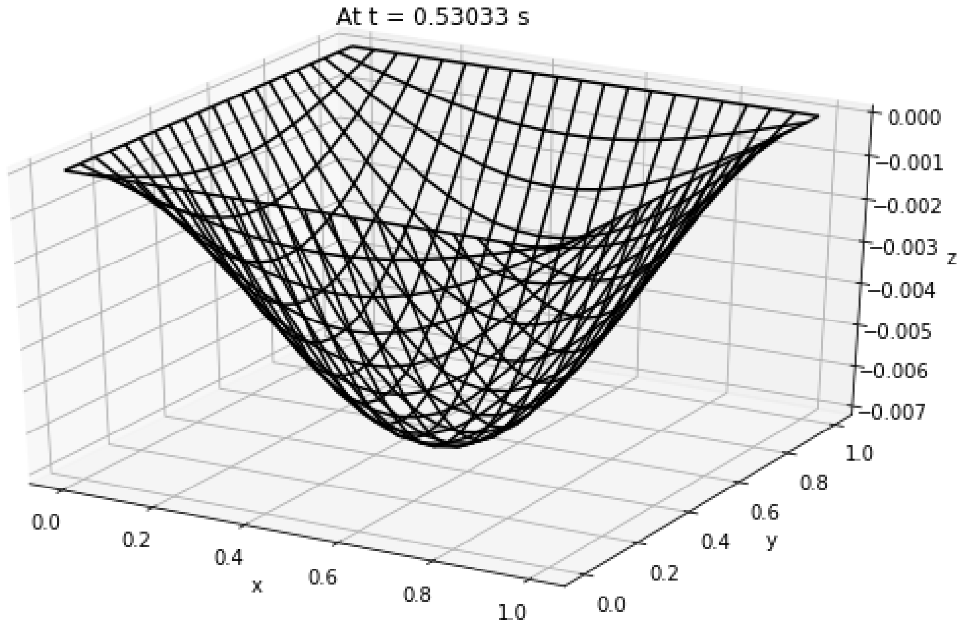
```
x-domain :  [0.   0.05 0.1  0.15 0.2  0.25 0.3  0.35 0.4  0.45 0.5  0.55 0.6
0.65
 0.7  0.75 0.8  0.85 0.9  0.95 1.  ]
y-domain :  [0.   0.05 0.1  0.15 0.2  0.25 0.3  0.35 0.4  0.45 0.5  0.55 0.6
0.65
 0.7  0.75 0.8  0.85 0.9  0.95 1.  ]
dt :  0.035355339059327376
```

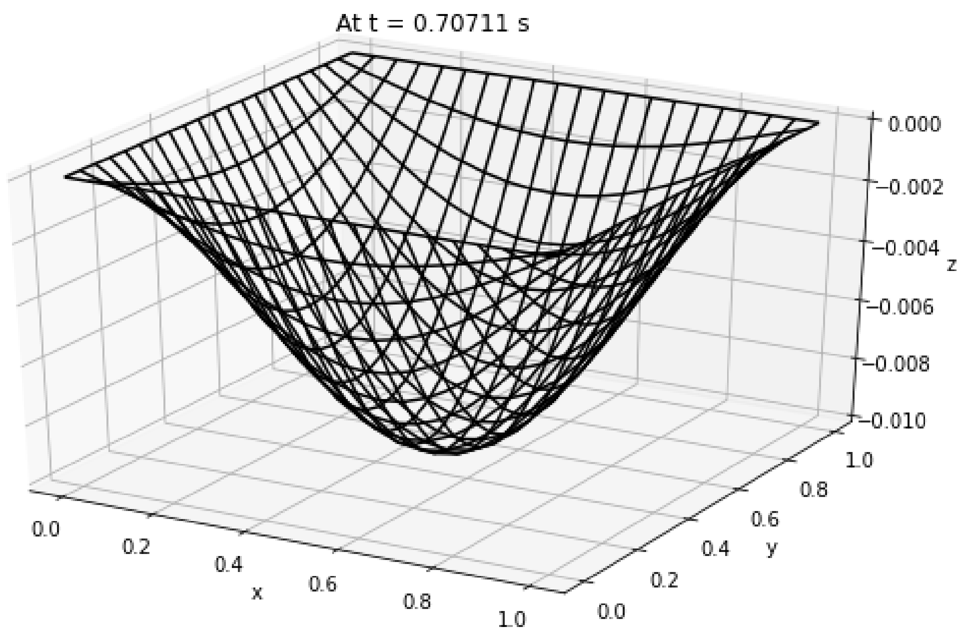


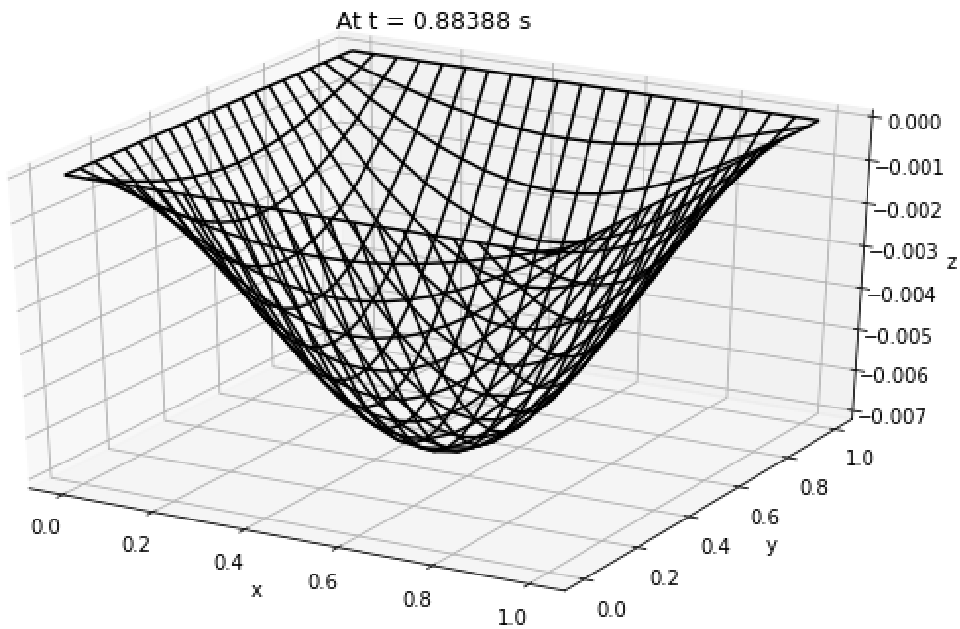


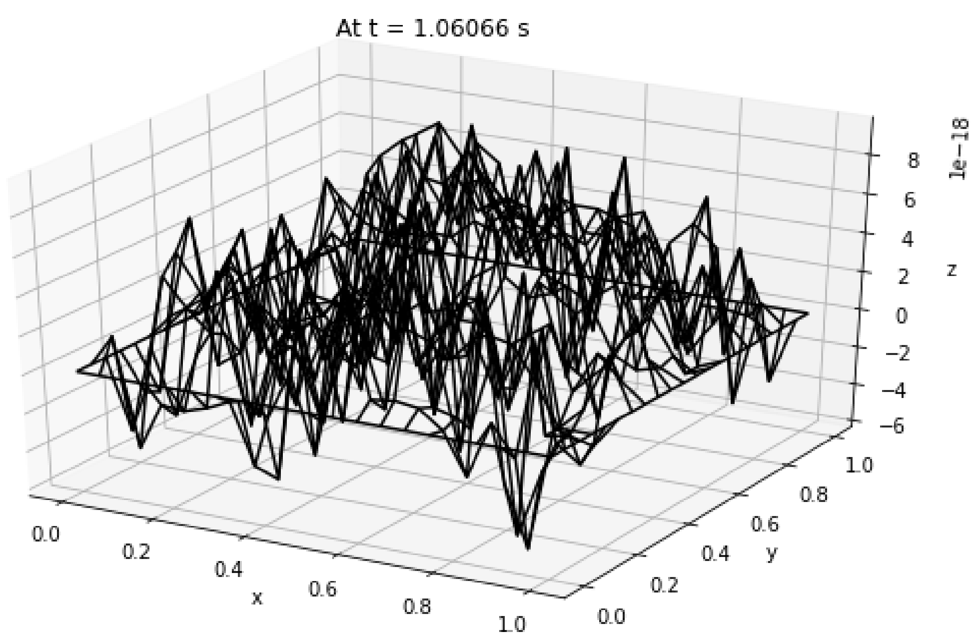

At t = 0.17678 s
x
y
z
0.0
0.2
0.4
0.6
0.8
1.0
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007

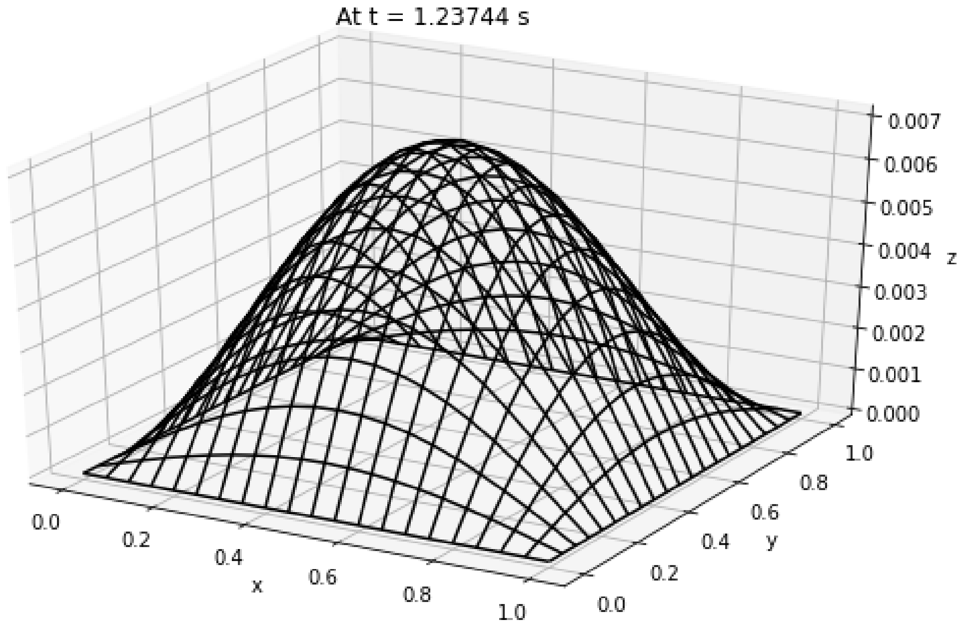

At t = 0.35355 s
x
y
z
1e−18
0.0
0.2
0.4
0.6
0.8
1.0
4
2
0
−2
−4
−6
−8

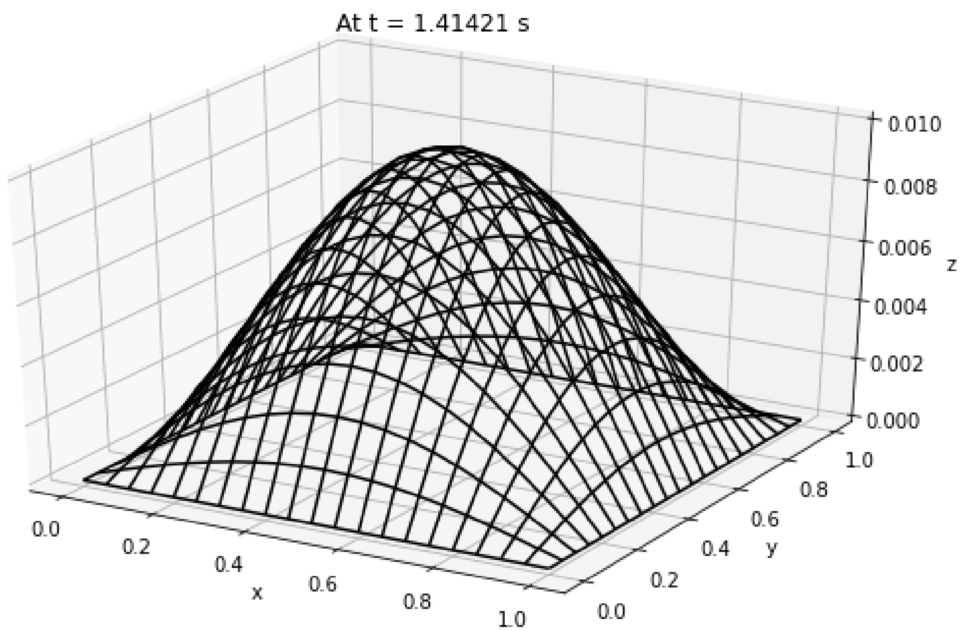

At t = 0.53033 s
0.000
−0.001
−0.002
−0.003
−0.004
−0.005
−0.006
−0.007
z
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y

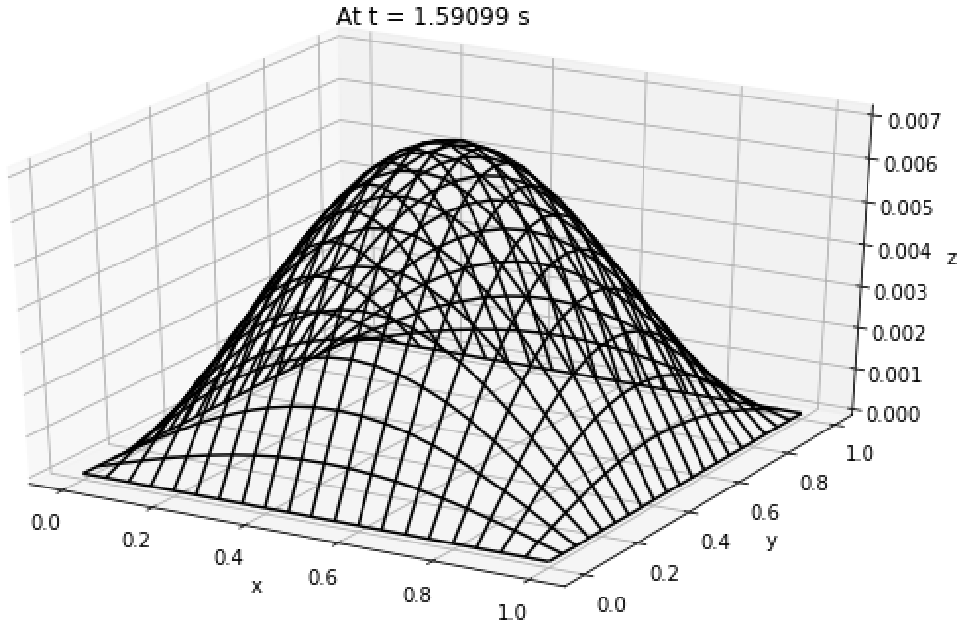

At t = 0.70711 s
0.000
−0.002
−0.004
−0.006
−0.008
−0.010
z
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y

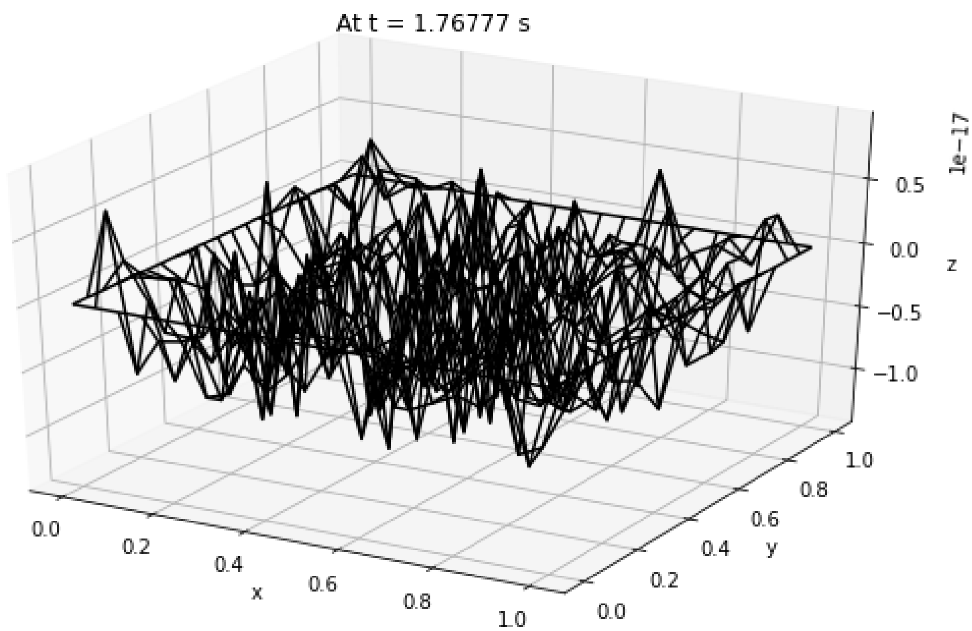

At t = 0.88388 s
0.000
−0.001
−0.002
−0.003
−0.004
−0.005
−0.006
−0.007
z
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y


At t = 1.06066 s
1e−18
8
6
4
2
0
−2
−4
−6
z
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y


At t = 1.23744 s
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
z
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y


At t = 1.41421 s
0.010
0.008
0.006
0.004
0.002
0.000
z
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y





Circular membrane (Axis symmetric)

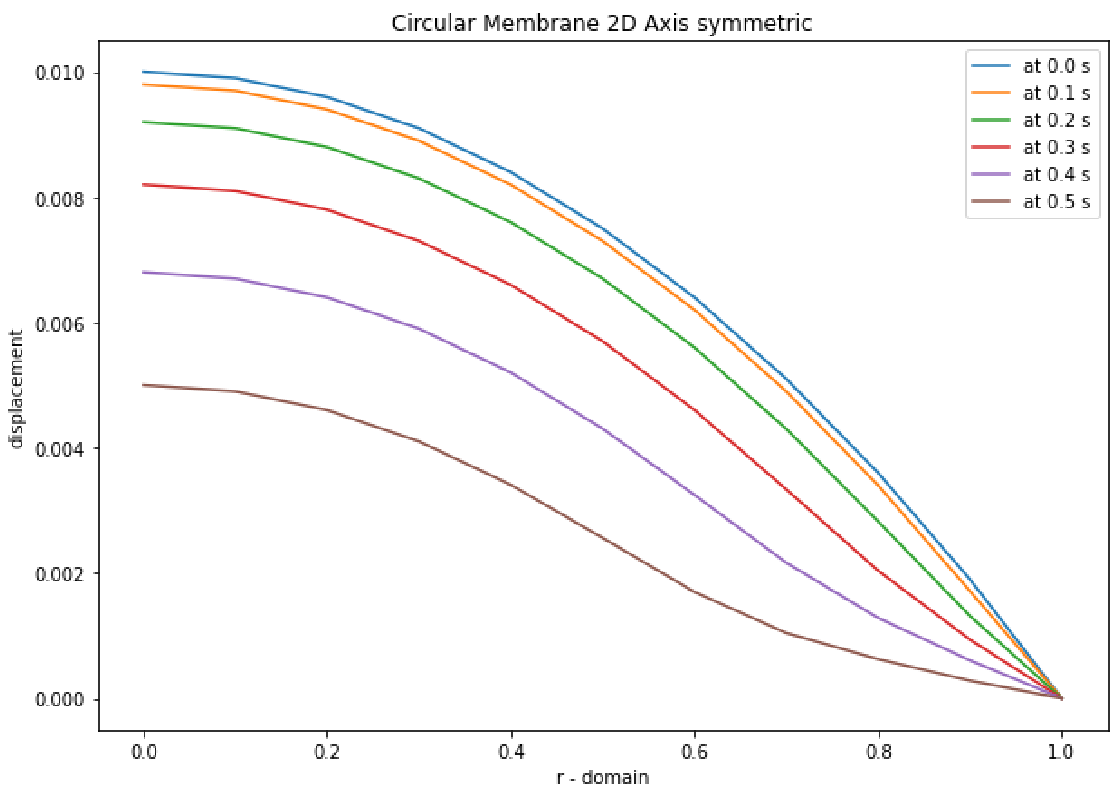
[4]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
#no of nodes
n = 11
r = np.zeros(n)
for i in range(n):
  r[i] = i*1/(n-1)
np.append(r, 1)
c = 0.5 #Courant Number
dr = 1/( n -1)
dt = c * dr
print('dt : ', dt)

U = []


#BCs
#u(r = R) = 0

#ICs
u0 = 0.01*(1 - r**2)
U.append(u0)

u1 = np.zeros(n)

u2 = np.zeros(n)

u1[0] = 2 * c**2 * u0[1] + (1-2*0.5**2) * u0[0]

for i in range(1, n-1):
  u1[i] = ((c**2)/2)*(1-dr/(2*r[i]))*u0[i-1] + (1-c**2)*u0[i] + (c**2/2) *(1 +␣
↪dr/(2*r[i])) * u0[i+1]
U.append(u1)

for k in range(2, 101):
  for i in range(1, n-1):
    u2[i] = c**2*(1-dr/(2* r[i]))*u1[i-1] + 2*(1-c**2)*u1[i] + c**2 * (1+ dr/(2␣
↪* r[i]))*u1[i+1] -u0[i]

  u2[0] = 4* c**2* u1[1] +2*(1 -2* c**2) * u1[0] - u0[0];
  u0 = u1.copy()
  u1 = u2.copy()
  U.append(u2)
  u2 = np.zeros(n)

#plotting
fig, ax = plt.subplots(1,figsize = (10,7))
for i in range(0, 12,2):
```

```
  plt.plot(r , U[i], label = 'at ' + str(round(i*dt, 4)) + ' s')
ax.set_title("Circular Membrane 2D Axis symmetric")
ax.set_xlabel('r - domain')
ax.set_ylabel('displacement')
ax.legend()
```

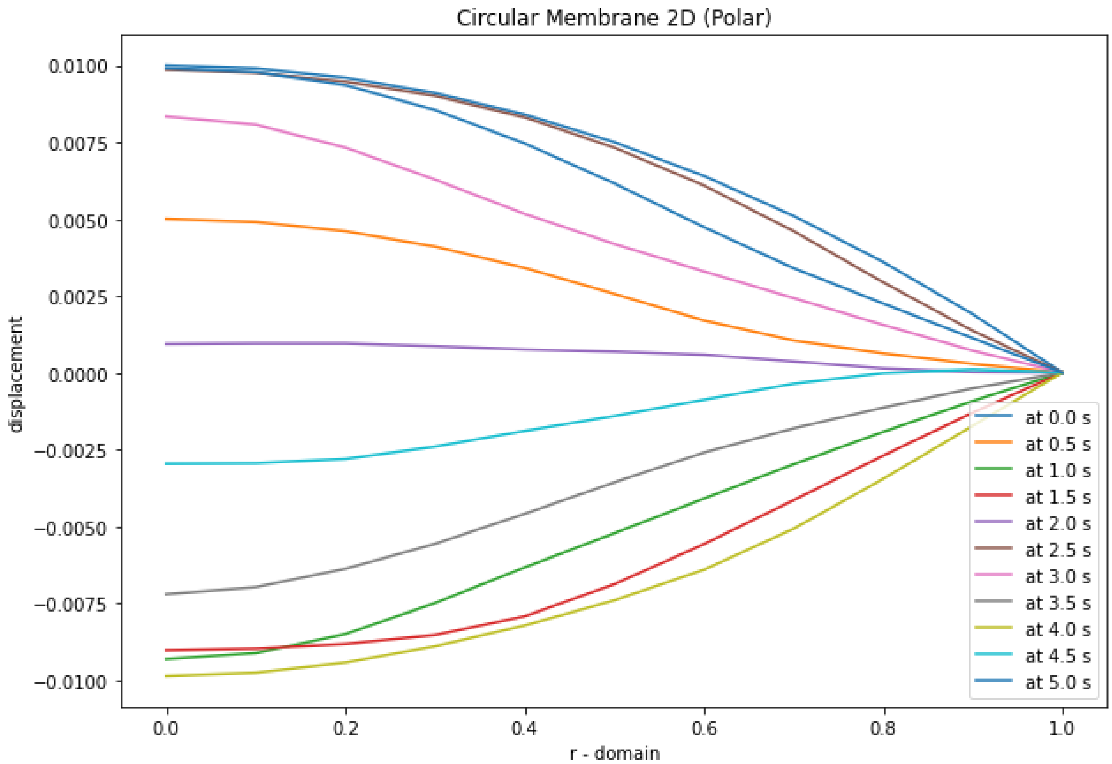
```
dt :  0.05
```

[4]: <matplotlib.legend.Legend at 0x7fd7e1c55610>



[5]:
```
fig, ax = plt.subplots(1,figsize = (10,7))
for i in range(0, 101,10):
  plt.plot(r , U[i], label = 'at ' + str(round(i*dt, 4)) + ' s')
ax.set_title("Circular Membrane 2D (Polar)")
ax.set_xlabel('r - domain')
ax.set_ylabel('displacement')
ax.legend()
```

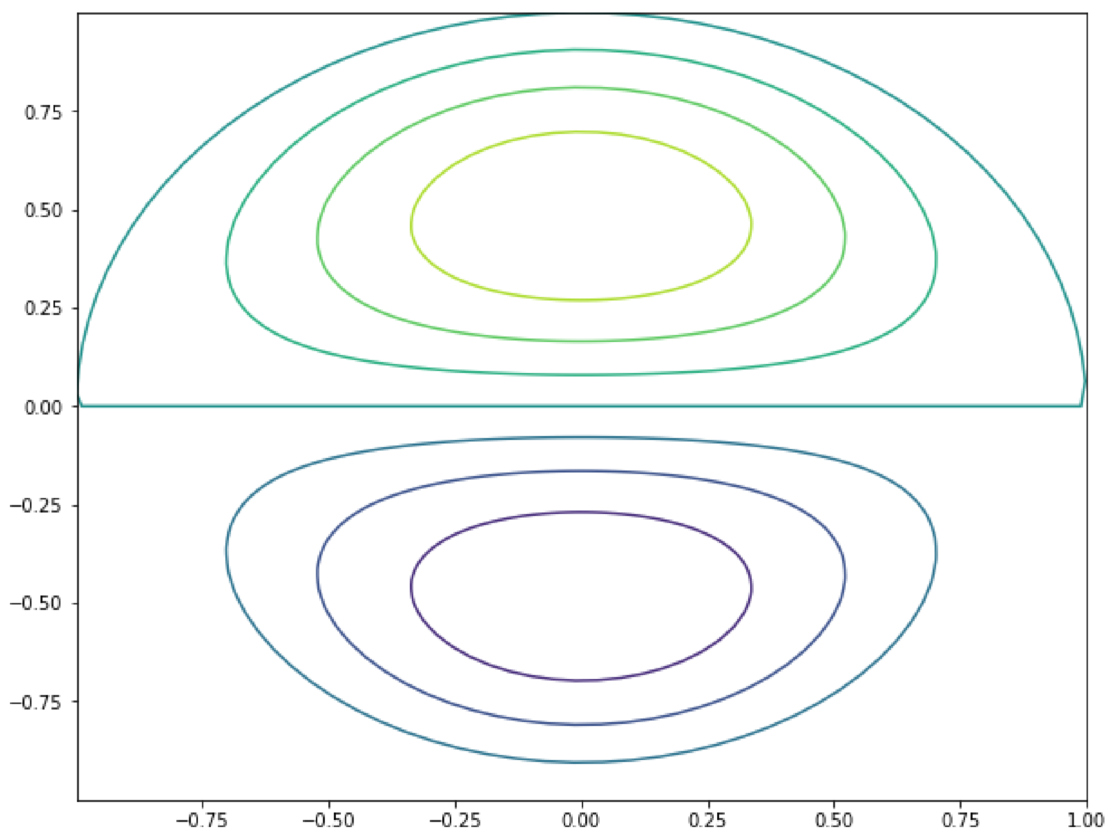
[5]: <matplotlib.legend.Legend at 0x7fd7e4a94210>

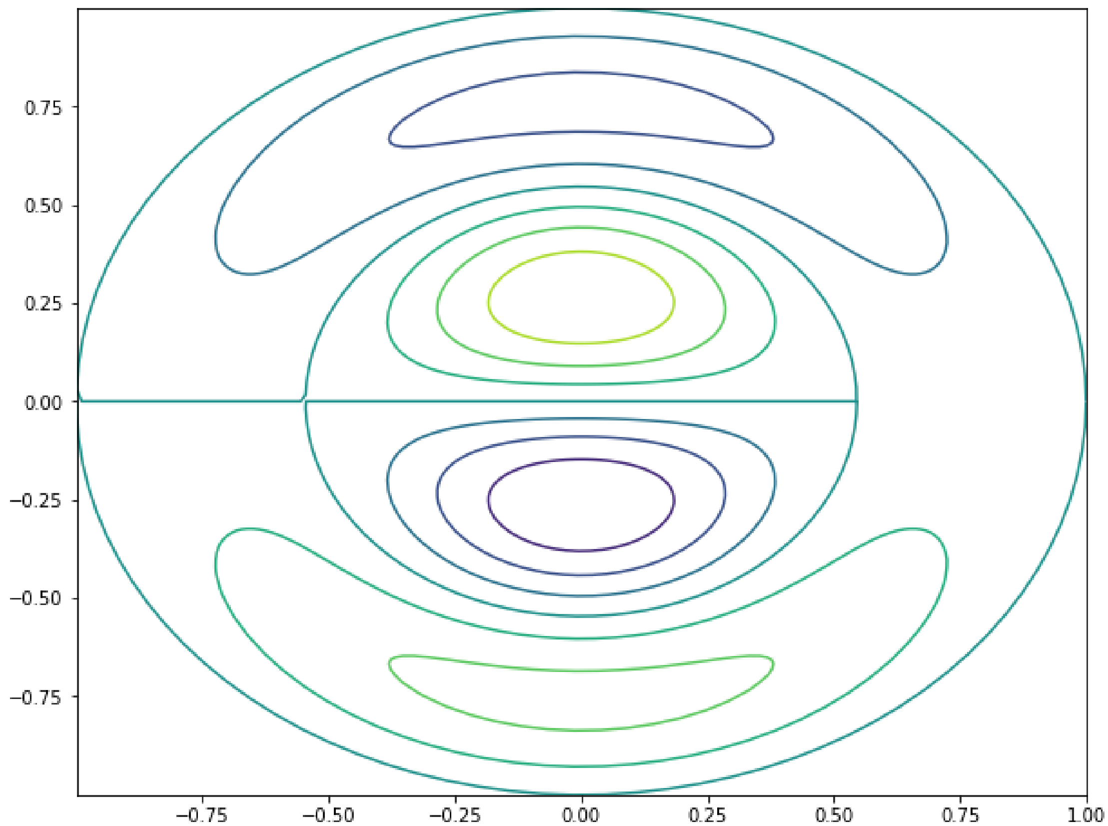


Modes

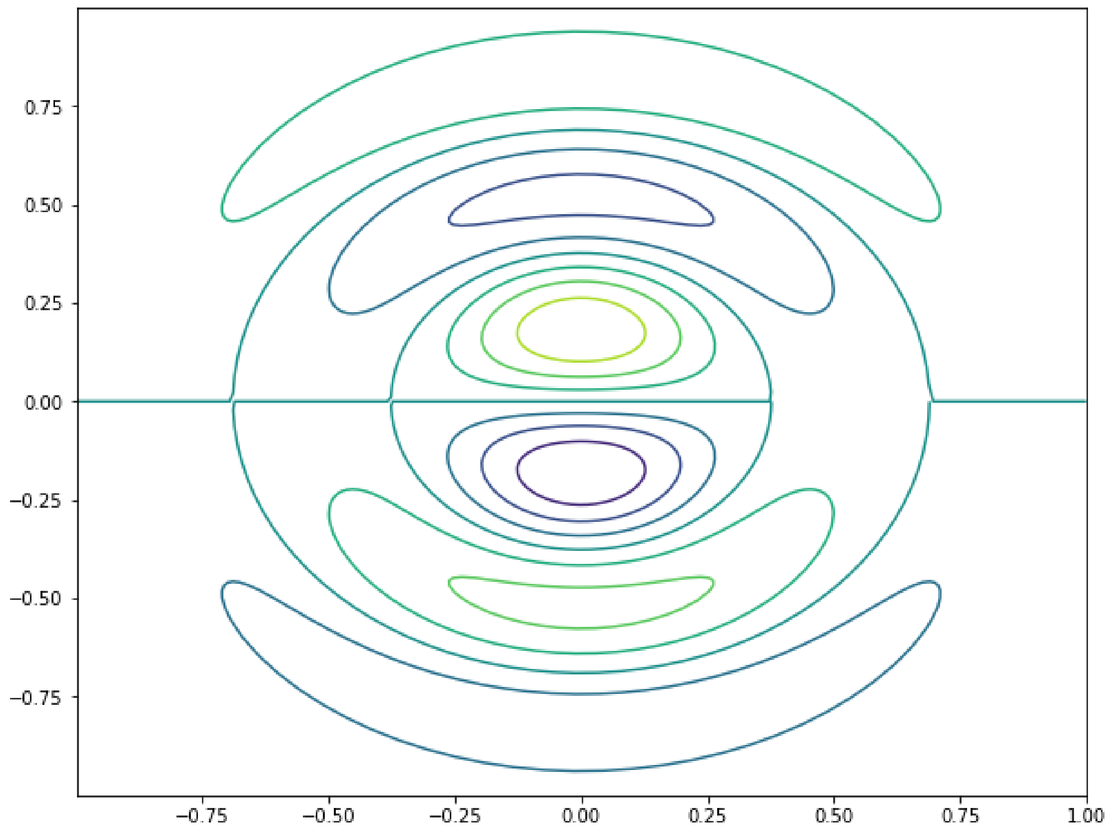
[6]:
```
import numpy as np
from scipy.special import jn, jn_zeros
import matplotlib.pyplot as plt

# Allow calculations up to m = mmax
mmax = 5

"""
Calculate the displacement of the drum membrane at (r, theta; t=0)
in the normal mode described by integers n >= 0, 0 < m <= mmax.
"""
def displacement(n, m, r, theta):

    # Pick off the mth zero of Bessel function Jn
    k = jn_zeros(n, mmax+1)[m]
    return np.sin(n*theta) * jn(n, r*k)

# Positions on the drum surface are specified in polar co-ordinates
r = np.linspace(0, 1, 100)
theta = np.linspace(0, 2 * np.pi, 100)
```

```
# Create arrays of cartesian co-ordinates (x, y) ...
x = np.array([rr*np.cos(theta) for rr in r])
y = np.array([rr*np.sin(theta) for rr in r])
# ... and vertical displacement (z) for the required normal mode at
# time, t = 0
n, m = 1,0
z = []
z1 = np.array([displacement(n, m, rr, theta) for rr in r])
z.append(z1)

n,m = 1,1
z2 = np.array([displacement(n, m, rr, theta) for rr in r])
z.append(z2)

n,m = 1,2
z3 = np.array([displacement(n, m, rr, theta) for rr in r])
z.append(z3)

for i in range(0, len(z)):
  fig = plt.figure(figsize = (10,8))
  plt.contour(x, y, z[i])
  plt.show()
```





0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



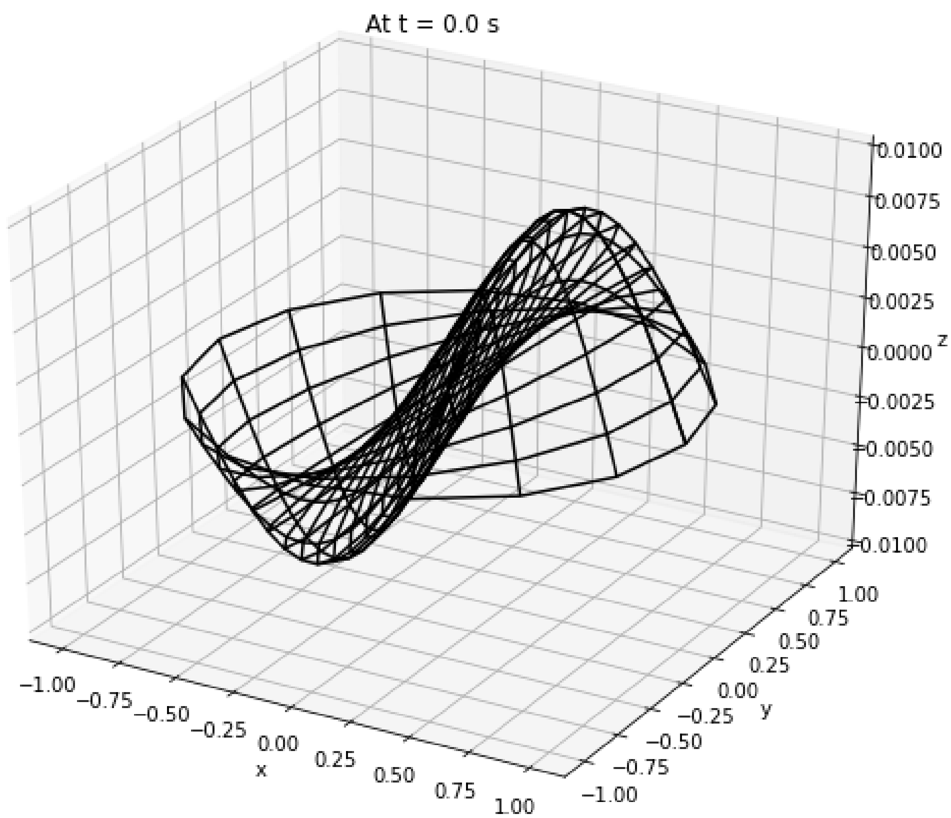
[15]:

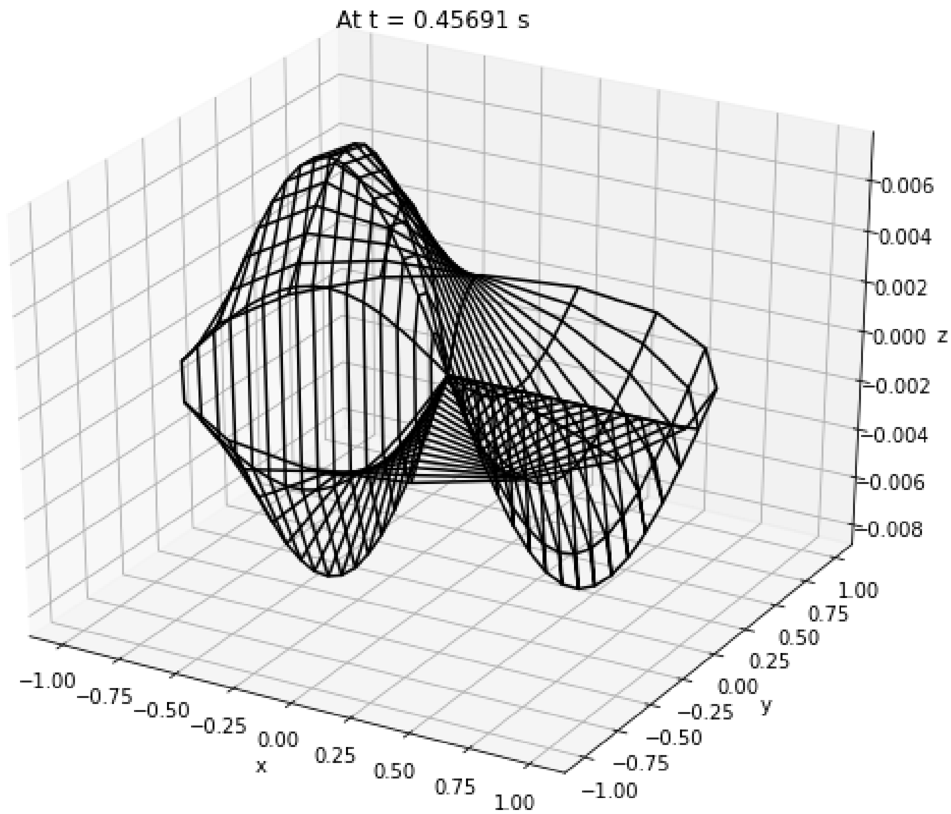
```
import numpy as np
import matplotlib.pyplot as plt
nt = 10 # each quarter 3 equal segments
nr = 41
ntheta = 4* nt +1

r  = np.linspace(0, 1, nr, endpoint = True)
theta = np.linspace(0,2*np.pi, ntheta, endpoint = True)
[R, Theta] = np.meshgrid(r, theta) #creating mesh


dr = 1/( nr -1)
dtheta = 2* np.pi /( ntheta -1)
cr = 1/ (1+1/ dtheta**2)**(1/2)
dt = cr * dr

print('r - domain : ', r, len(r))
print('theta - domain : ', theta, len(theta))
print('dt : ', dt)

U = []
```

```
u0 = 0.01*np.sin(R*np.pi) * np.cos(Theta)
U.append(u0)

u1 = np.zeros((nr, ntheta))

u2 = np.zeros((nr, ntheta))

u1[0,0] = 0.5*(cr**2 * (u0[1, 2] + u0[1,0] + u0[1, 3] + u0[1,1]) - 2*(1-cr**2)␣
↪* u0[0,0])

for i in range(1 , nr-1):
  for j in range(1, ntheta-1):
    u1[i,j] = 0.5*( cr**2 *( 1- dr/r[i])* u0[i-1,j] + 2 * (1-cr**2 * (1 + (dr/
↪(r[i] * dtheta))**2))*u0[i,j] + cr**2 * (1 + dr/r[i])*u0[i+1, j] + ((cr *␣
↪dr)/(r[i] *dtheta))**2 * (u0[i,j-1] + u0[i,j+1]))

U.append(u1)

# at centre r = 0, u = 0
# BC

for t in range(2, 101):
  u2[0,0] = cr**2 * (u1[1, 2] + u1[1,0] + u1[1, 3] + u1[1,1]) - 2*(1-cr**2) *␣
↪u1[0,0] - u0[0,0]
  for i in range(1, nr-1):
    for j in range(1, ntheta-1):
      u2[i,j] = cr**2 *( 1- dr/r[i])* u1[i-1,j] + 2 * (1-cr**2 * (1 + (dr/(r[i]␣
↪* dtheta))**2))*u1[i,j] + cr**2 * (1 + dr/r[i])*u1[i+1, j] + ((cr * dr)/
↪(r[i] *dtheta))**2 * (u1[i,j-1] + u1[i,j+1]) - u0[i,j]

  u0 = u1.copy()
  u1 = u2.copy()
  U.append(u2)
  u2 = np.zeros(( nr , ntheta ))

def plotting_2d(a,b,z,i):
  R, P = np.meshgrid(a,b)
  X, Y = R*np.cos(P), R*np.sin(P)
  fig = plt.figure(figsize=(10,8))
  ax = plt.axes(projection='3d')
  ax.plot_wireframe(X, Y, z, color='black')
  ax.set_xlabel('x')
  ax.set_ylabel('y')
  ax.set_zlabel('z')
  ax.set_title('At t = '+ str(round(i * dt, 5)) + ' s')

for i in range(0, 101,20):
```

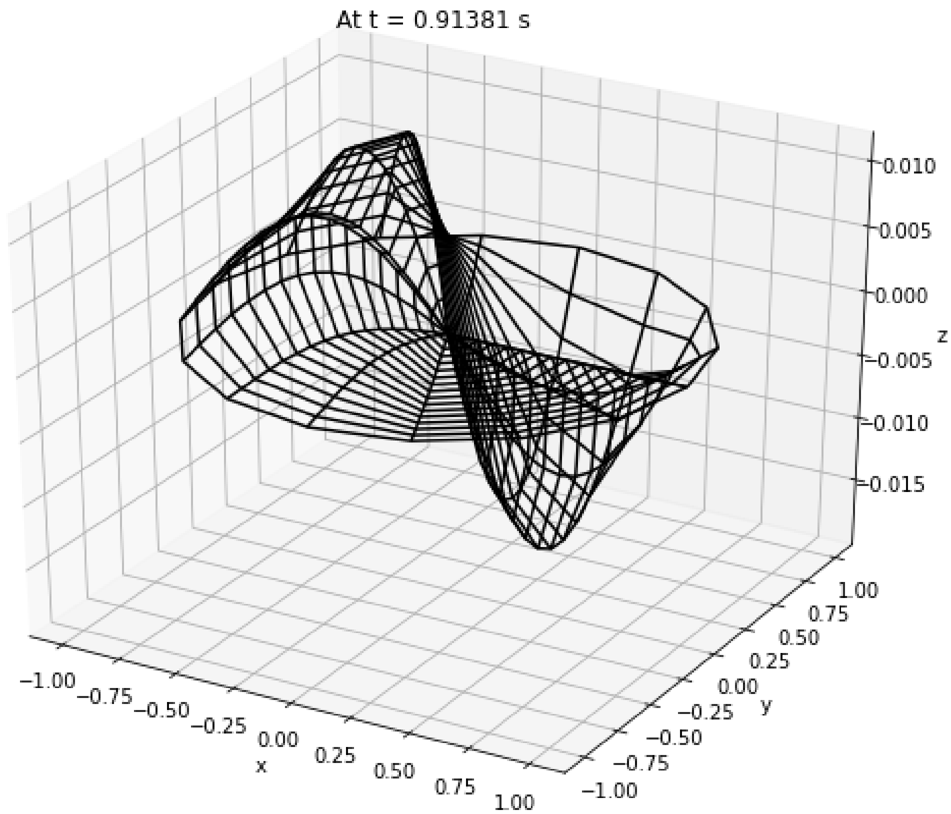
```
plotting_2d(r,theta,U[i],i)
```

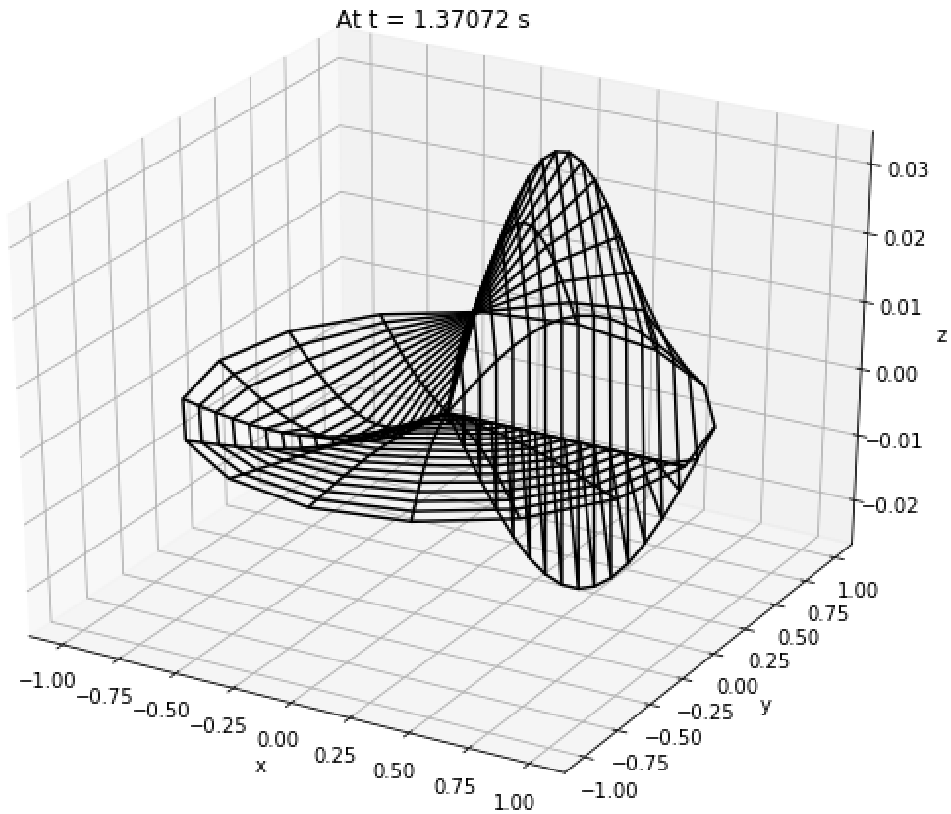
```
r - domain :  [0.     0.0625 0.125  0.1875 0.25   0.3125 0.375  0.4375 0.5
0.5625
 0.625  0.6875 0.75   0.8125 0.875  0.9375 1.    ] 17
theta - domain :  [0.         0.39269908 0.78539816 1.17809725 1.57079633
1.96349541
 2.35619449 2.74889357 3.14159265 3.53429174 3.92699082 4.3196899
 4.71238898 5.10508806 5.49778714 5.89048623 6.28318531] 17
dt :  0.022845307255505522
```

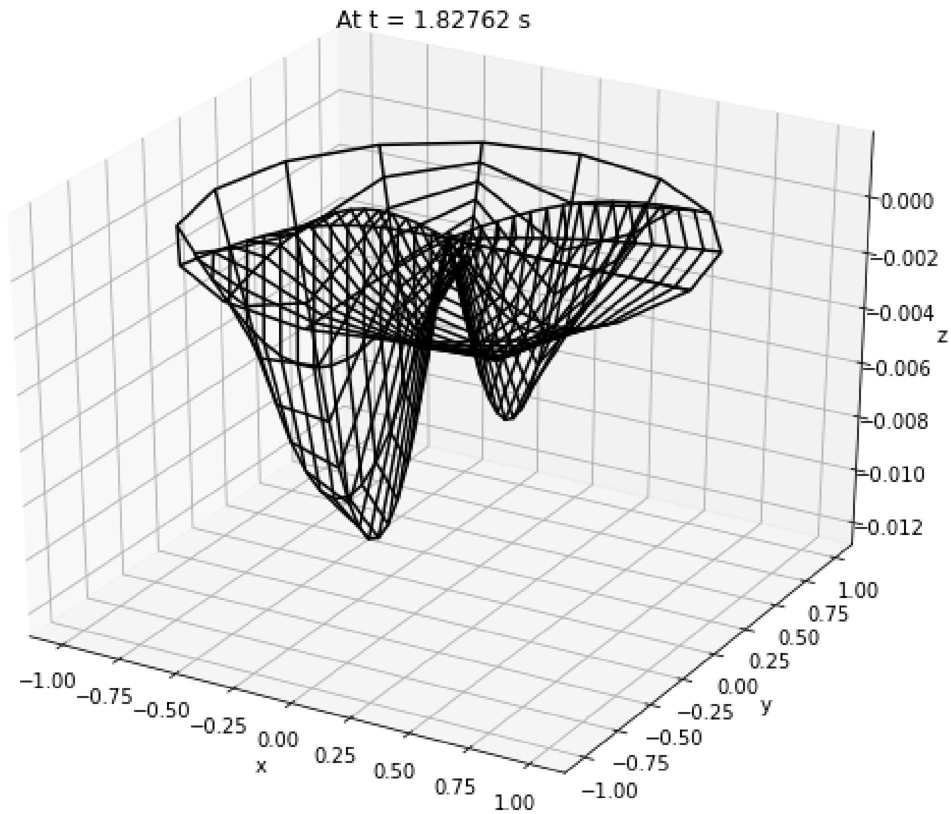


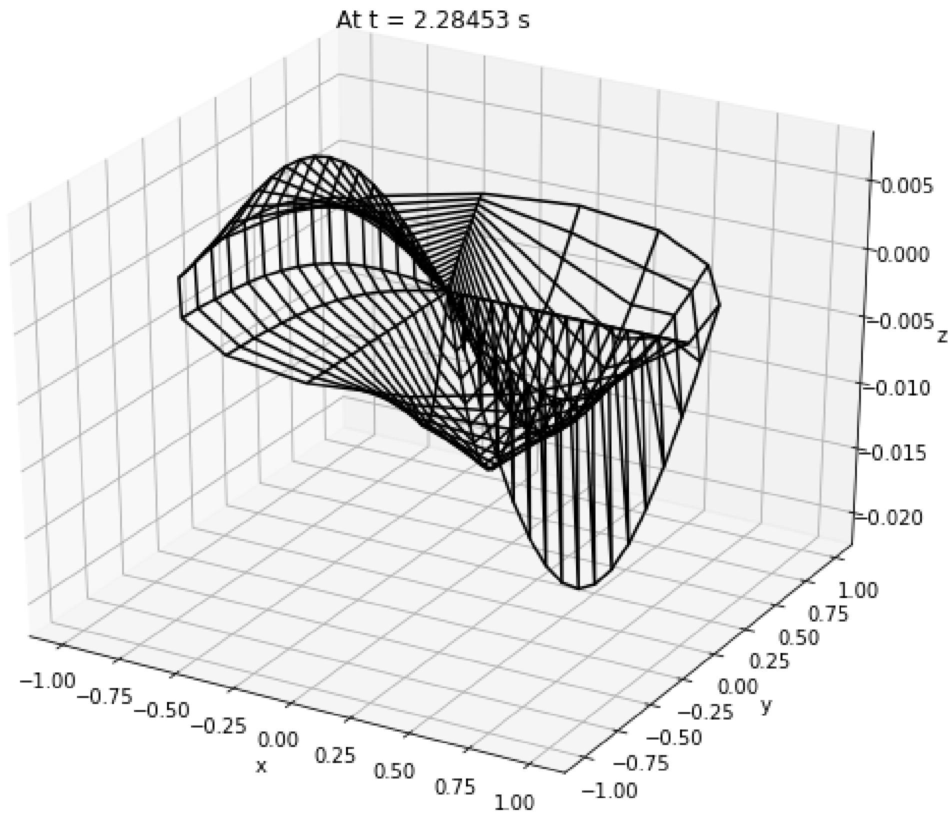


At t = 0.45691 s
0.006
0.004
0.002
0.000
z
−0.002
−0.004
−0.006
−0.008
−1.00
−0.75
−0.50
−0.25
0.00
x
0.25
0.50
0.75
1.00
1.00
0.75
0.50
0.25
0.00
y
−0.25
−0.50
−0.75
−1.00


At t = 0.91381 s
x
y
z
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0.010
0.005
0.000
−0.005
−0.010
−0.015


At t = 1.37072 s
0.03
0.02
0.01
0.00
−0.01
−0.02
z
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y


At t = 1.82762 s
0.000
−0.002
−0.004
−0.006
−0.008
−0.010
−0.012
z
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y



[ ]:
```
#Extra code for printing
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('213101001_project.ipynb')
```