# SECTION 7: THREE-DIMENSIONAL PLOTTING

ENGR 103 – Introduction to Engineering Computing

# 3-D plots



We'll consider two main categories of 3-D plots:

- ***3-D line plots***
  - A single independent variable – two of the variables are functions of the third
  - A line in 3-D space
- ***Surface plots***
  - A function of two variables – two independent variables, one dependent variable
  - *Height* of the function is dependent on position in the x,y plane

# 3-D Line Plots

# 3-D Line Plot – `plt.plot()`



- One independent variable, two dependent variables
  - E.g. $x = f(z), \;\; y = f(z)$
  - 3-D line plot – a curve in three-dimensional space
  - A ***parametric curve***
- First, create 3-D axes:
  - Create and clear a figure:

```
fig = plt.figure(1)
plt.clf()
```

  - Add 3-D axis subplot:

```
ax = fig.add_subplot(projection='3d')
```

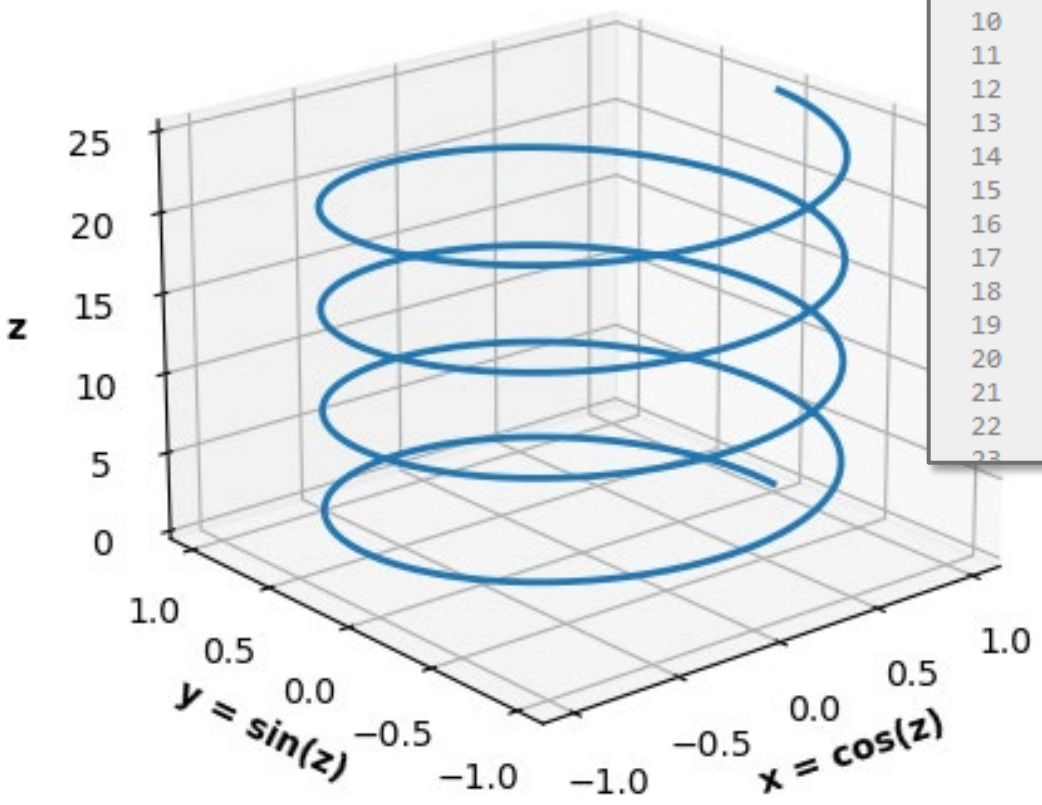
- Plot just like a 2-D plot, but with a vector of z-axis values:

```
ax.plot(x,y,z, linewidth=2)
```

# 3-D Line Plot – `plt.plot()`



$$x = \cos(z), \qquad y = \sin(z), \qquad 0 \le z \le 8\pi$$




```
10  z = np.linspace(0, 8*np.pi, 2000)
11  x = np.cos(z)
12  y = np.sin(z)
13
14  fig = plt.figure(1)
15  plt.clf()
16  ax = fig.add_subplot(projection='3d')
17  ax.view_init(elev=20, azim=-130)
18  ax.plot(x,y,z, linewidth=2)
19
20  ax.set_xlabel('x = cos(z)', fontweight='bold')
21  ax.set_ylabel('y = sin(z)', fontweight='bold')
22  ax.set_zlabel('z', fontweight='bold')
```

# 3-D Surface Plots

# 3-D Surface Plots



- Functions of two variables can be plotted as ***surfaces*** in 3-D space
  - $z = f(x, y)$
- Functions of one variable get evaluated at each point in the input variable vector
- Say we want to evaluate a function, $z = f(x, y)$, over a range of $x$ and $y$ values, e.g. $0 \leq x \leq 10$ and $0 \leq y \leq 5$
- Must evaluate $z$ not only at each point in $x$ and $y$, but ***at all possible combinations of $x$ and $y$***

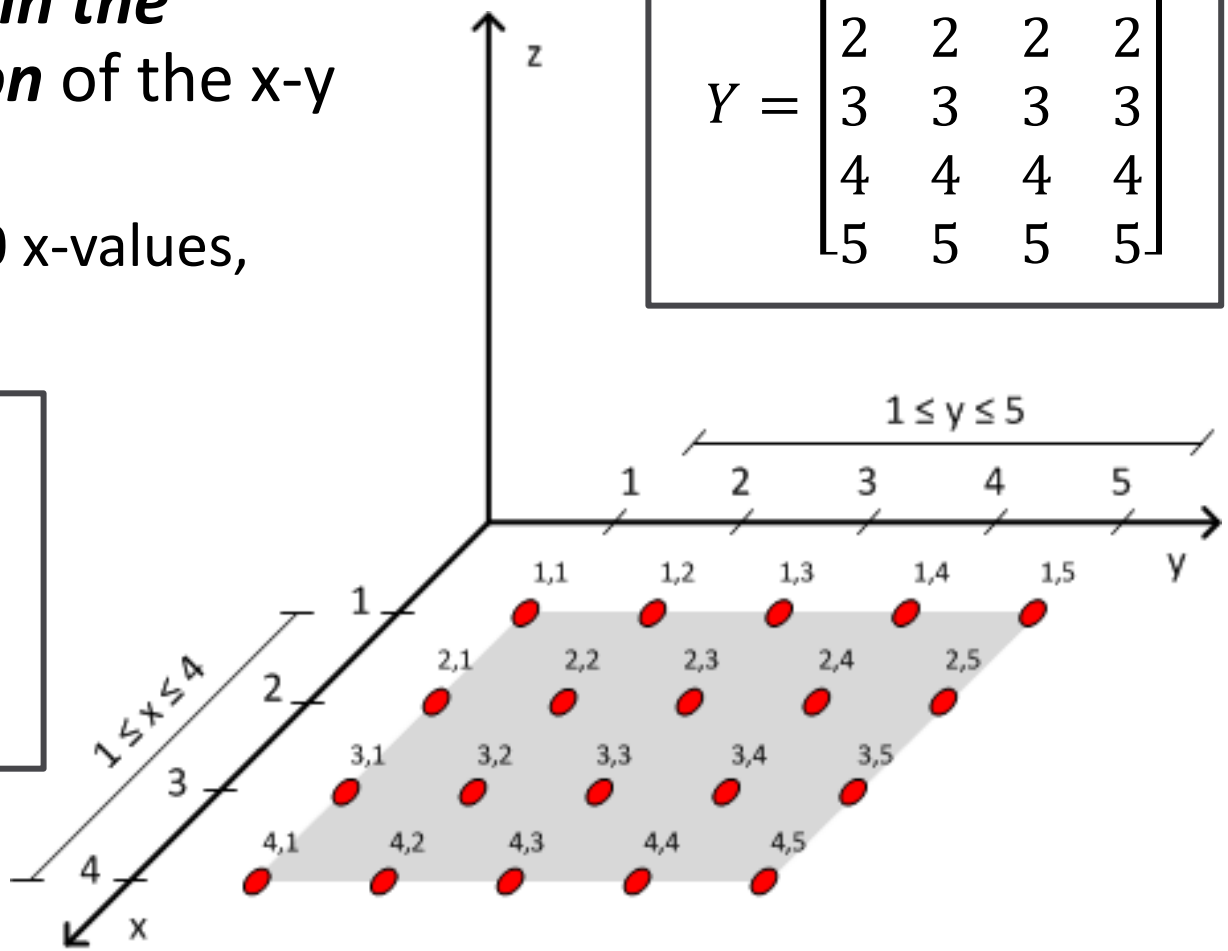
# 3-D Surface Plots - Input Matrices



- Must evaluate $z = f(x, y)$ at ***every point in the specified region*** of the x-y plane
  - ***20 points*** – 20 x-values, 20 y-values

$$Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$



# 3-D Surface Plots – np.meshgrid()



```
Xm, Ym = np.meshgrid(x,y)
```

- x and y are 1-D arrays (vectors)
  - Define ranges of x and y in the x-y plane
- Xm and Ym are 2-D arrays (matrices)
  - All coordinates in the region of the x-y plane specified by vectors x and y
  - np.shape(Xm) = np.shape(Ym) = (len(y), len(x))
  - Rows of Xm are x
  - Columns of Ym are y

# Function of Two Variables – Input Matrices



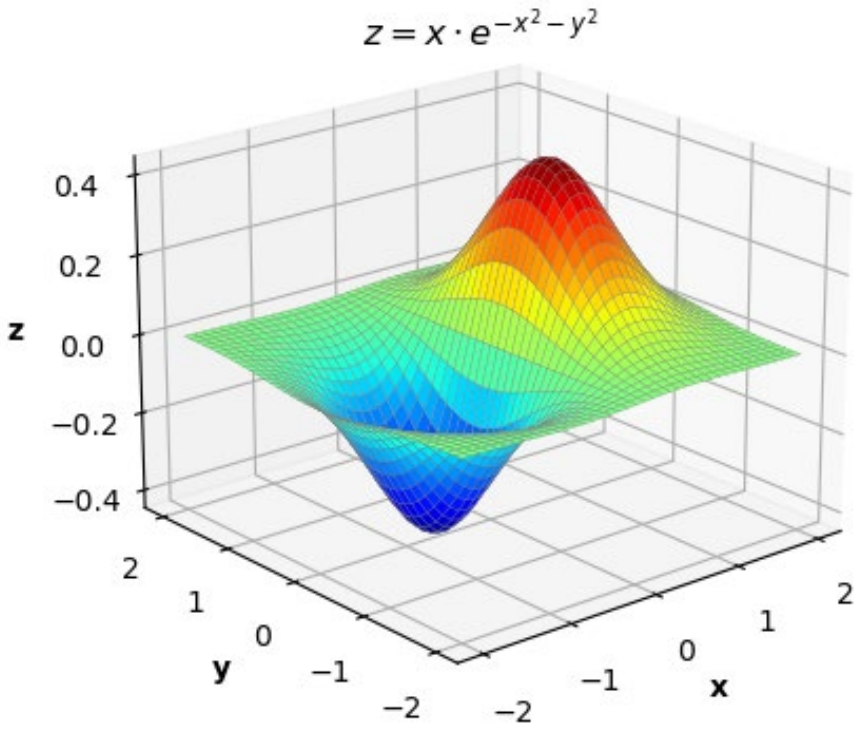
- The inputs to $z = f(x, y)$ are matrices
  - Create from x and y vectors using np.meshgrid()
- Example:
  - Evaluate $z = xe^{-x^2-y^2}$ for $-2 \leq x \leq 2, \quad -2 \leq y \leq 2$

```
26
27    x = np.arange(-2, 2.1, 0.1)
28    y = np.arange(-2, 2.1, 0.1)
29
30    Xm, Ym = np.meshgrid(x,y)
31
32    Z = Xm*np.exp(-Xm**2 - Ym**2)
33
```

# Surface Plot – plt.plot_surface()



```
ax.plot_surface(X, Y, Z)
```




```
3   import matplotlib.pyplot as plt
4   import numpy as np
5   from matplotlib import cm
```

```
35  plt.figure(2)
36  plt.clf()
37  ax = plt.axes(projection='3d')
38  # ax = plt.subplot(1,1,1, projection='3d')
39  ax.view_init(elev=20, azim=-130)
40  ax.plot_surface(Xm, Ym, Z,
41                          cmap=cm.jet,
42                          linewidth=0.2,
43                          edgecolor='grey')
44
45  ax.set_xlabel('x', fontweight='bold')
46  ax.set_ylabel('y', fontweight='bold')
47  ax.set_zlabel('z', fontweight='bold')
48  ax.set_title('$z = x\cdot e^{-x^2 - y^2}$',
49               fontweight='bold', y=0.95)
```

- Set viewing angle of 3-D plots:

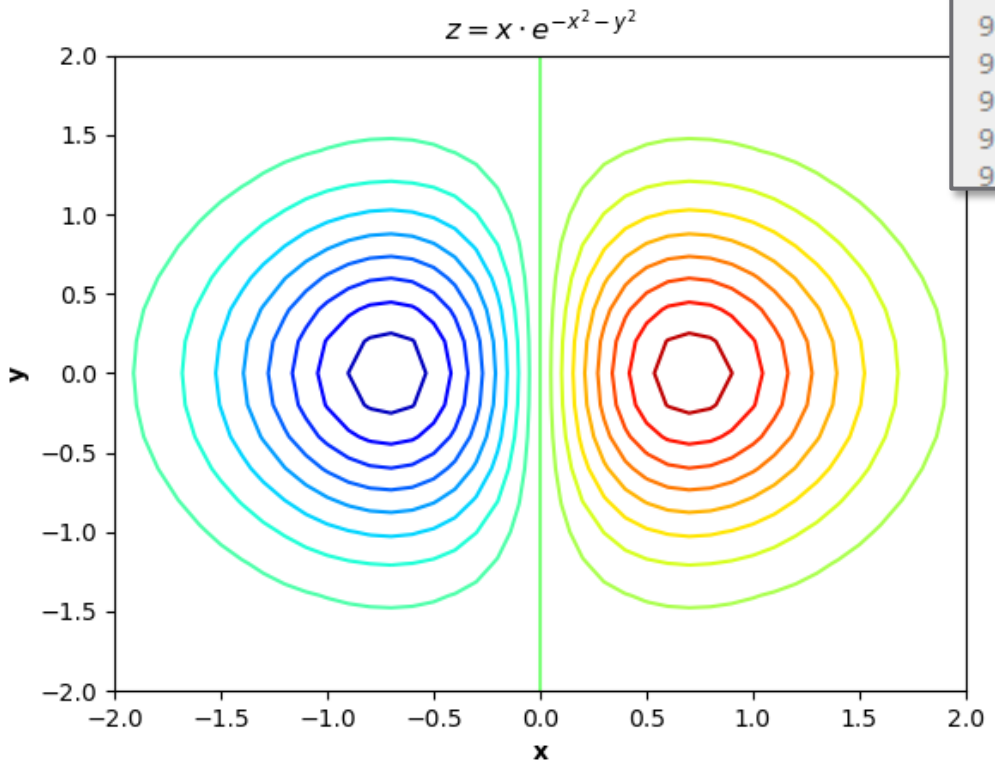
```
ax.view_init(
            azim=theta,
            elev=phi)
```

# 2-D Contour Plot – `plt.contour()`



```
plt.contour(X, Y, Z, N)
```

```
88
89    plt.figure(4)
90    plt.clf()
91
92    plt.contour(Xm, Ym, Z, 20, cmap=cm.jet)
93
94    plt.xlabel('x', fontweight='bold')
95    plt.ylabel('y', fontweight='bold')
96    plt.title('$z = x\cdot e^{-x^2 - y^2}$',
97              fontweight='bold')
98
```



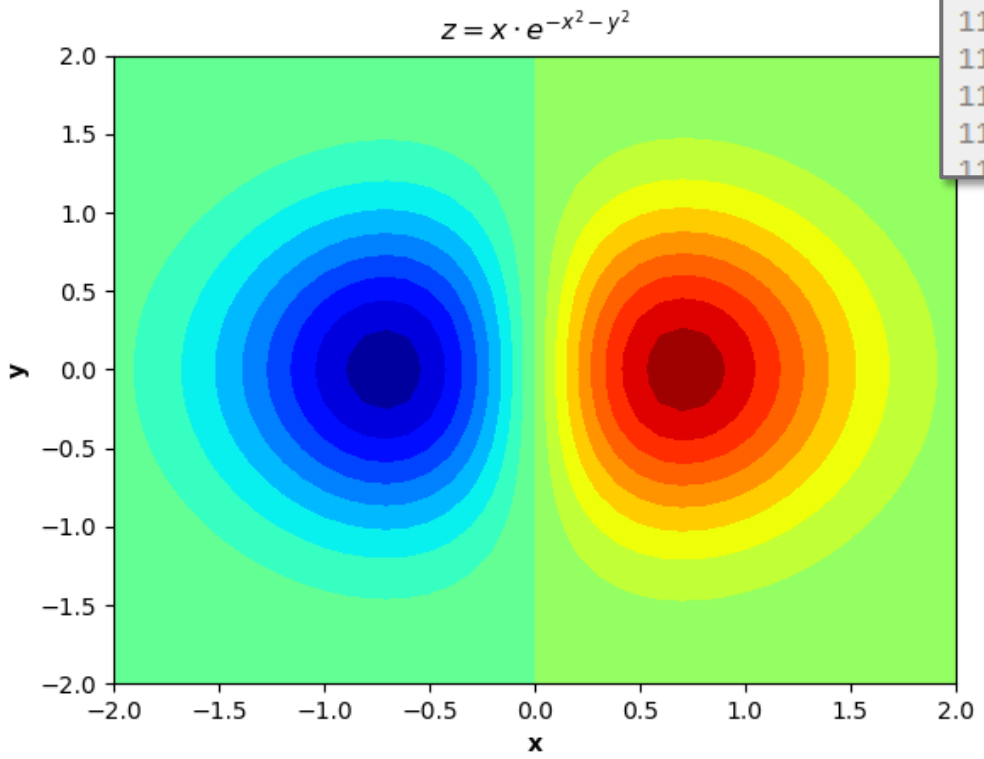

- A 2-D contour plot of the surface defined by Z
- N: # of contours

# 2-D Filled Contour Plot – `plt.contourf()`



```
plt.contourf(X, Y, Z, N)
```

```
106  plt.figure(5)
107  plt.clf()
108
109  plt.contourf(Xm, Ym, Z, 20, cmap=cm.jet)
110
111  plt.xlabel('x', fontweight='bold')
112  plt.ylabel('y', fontweight='bold')
113  plt.title('$z = x\cdot e^{-x^2 - y^2}$',
114            fontweight='bold')
```

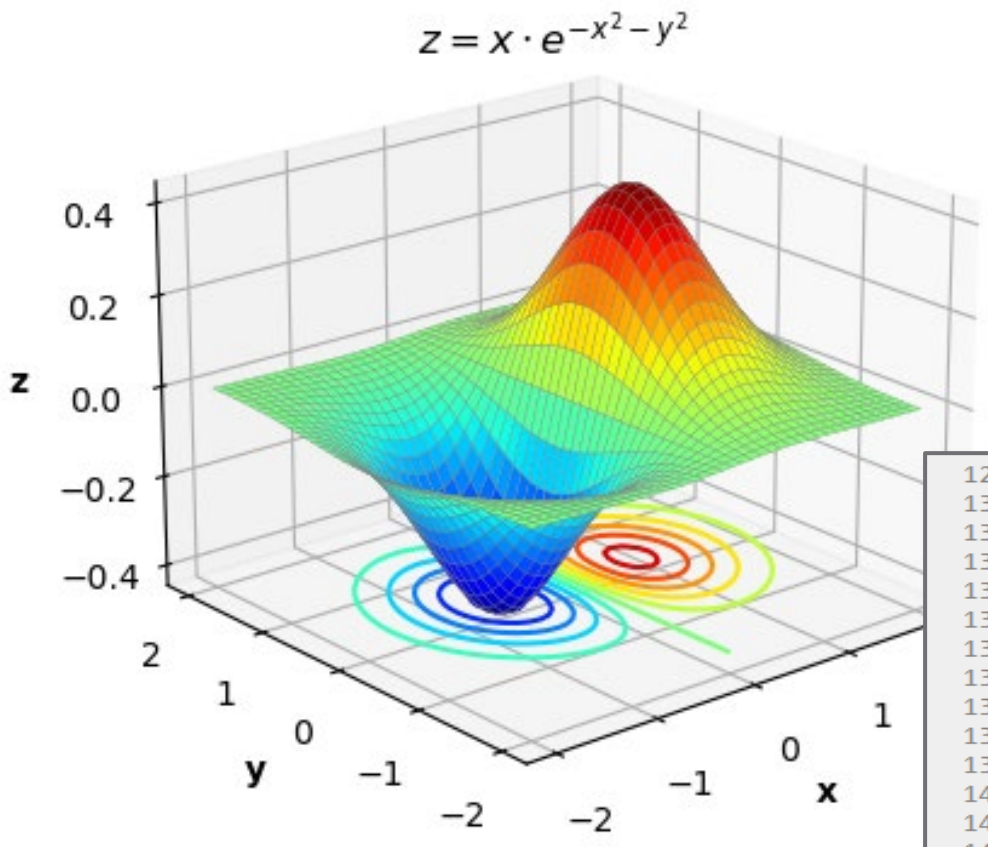



- Filled 2-D contour plot of the surface defined by Z
- N: # of contours

# Surface Plot with Contours – ax.contour()



```
ax.contour(X, Y, Z, N, zdir=<>, cmap=<>, offset=<>)
```



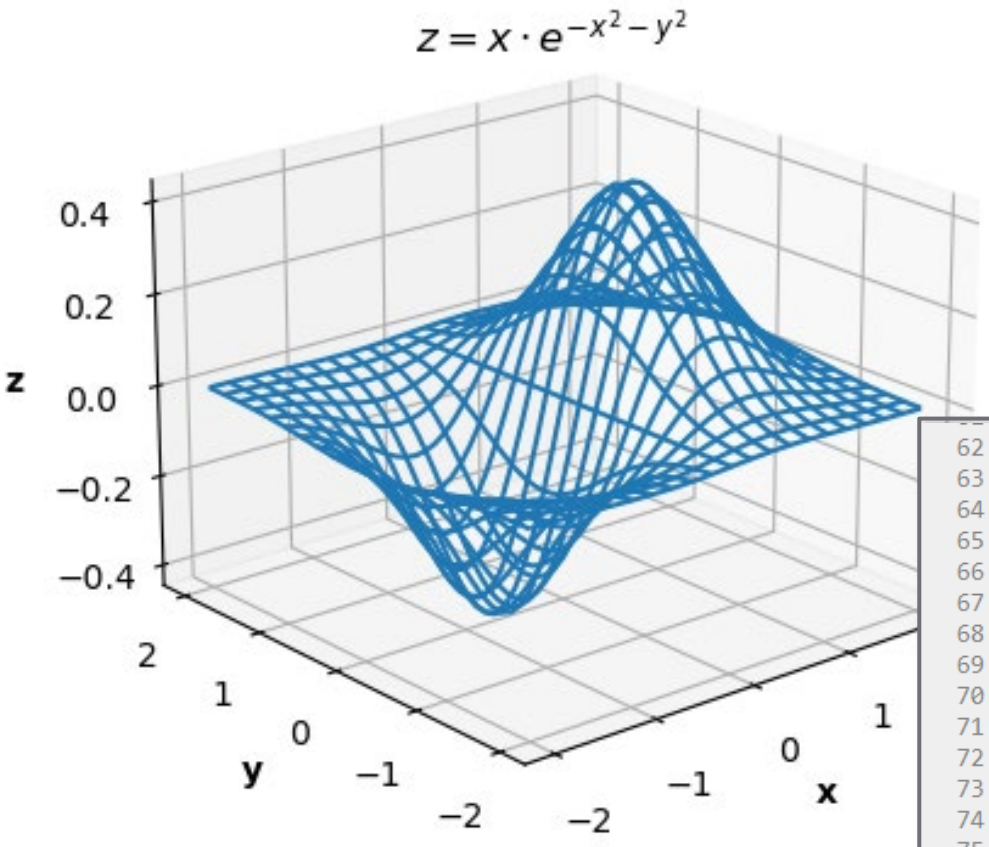

- Plot surface first
- N: # of contours
- `zdir`: axis normal to the contour-drawing plane
- `offset`: zdir-axis location for contours

```
129  fig = plt.figure(6)
130  plt.clf()
131  ax = plt.axes(projection='3d')
132  # ax = plt.subplot(1,1,1, projection='3d')
133  ax.view_init(elev=20, azim=-130)
134  ax.plot_surface(Xm, Ym, Z,
135                  cmap=cm.jet,
136                  linewidth=0.2,
137                  edgecolor='grey')
138
139  ax.contour(Xm, Ym, Z, 12, zdir='z',
140             cmap=cm.jet,
141             offset=-0.4)
```

# Wireframe Plot – ax.plot_wireframe()



```
ax.plot_wireframe(X, Y, Z, rstride=1, cstride=1)
```



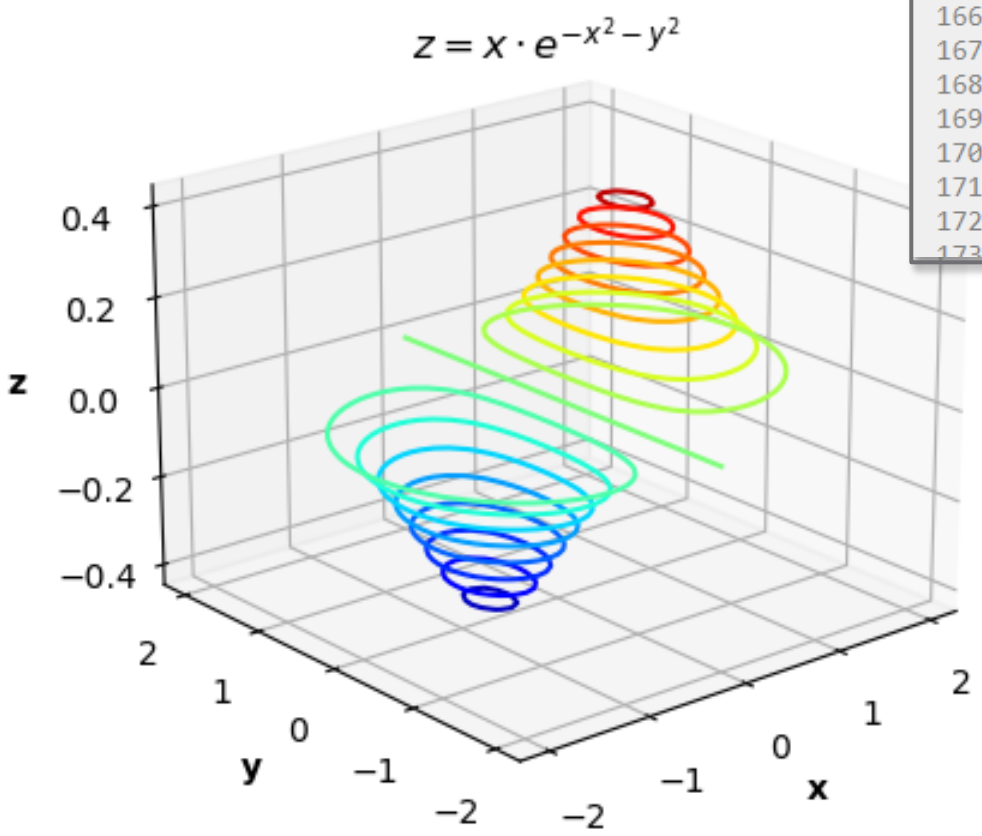

- `rstride`: downsampling in y (row) direction
  - *Optional* – default: 1
- `cstride`: downsampling in x (column) direction
  - *Optional* – default: 1

```
62  fig = plt.figure(3)
63  plt.clf()
64  ax = plt.axes(projection='3d')
65  # ax = plt.subplot(1,1,1, projection='3d')
66  ax.view_init(elev=20, azim=-130)
67
68  ax.plot_wireframe(Xm, Ym, Z, rstride=2, cstride=2)
69
70  ax.set_xlabel('x', fontweight='bold')
71  ax.set_ylabel('y', fontweight='bold')
72  ax.set_zlabel('z', fontweight='bold')
73  ax.set_title('$z = x\cdot e^{-x^2 - y^2}$',
74               fontweight='bold', y=0.95)
75
```

# 3-D Contour Plot – ax.contour()



```
ax.contour(x,y,z,N)
```




```
160  fig = plt.figure(7)
161  plt.clf()
162  ax = plt.axes(projection='3d')
163  # ax = plt.subplot(1,1,1, projection='3d')
164  ax.view_init(elev=20, azim=-130)
165
166  ax.contour(Xm, Ym, Z, 20, cmap=cm.jet)
167
168  ax.set_xlabel('x', fontweight='bold')
169  ax.set_ylabel('y', fontweight='bold')
170  ax.set_zlabel('z', fontweight='bold')
171  ax.set_title('$z = x\cdot e^{-x^2 - y^2}$',
172               fontweight='bold', y=0.95)
```

- A 3-D contour plot of the surface defined by Z
- N contours drawn at their corresponding z values

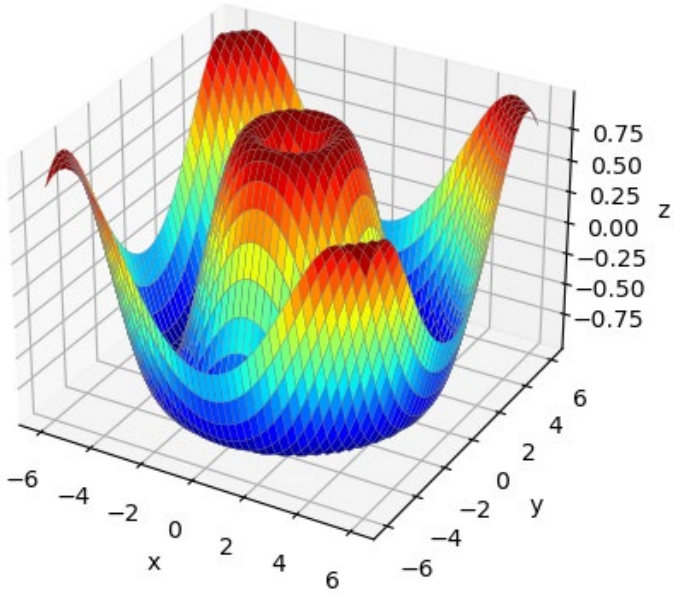
# Exercise – Surface Plot



Exercise

- Write a Python script to:
  - Define $x$ and $y$ vectors, each with 45 points over the range of [-6, 6]
  - Use `np.meshgrid()` to create $X$ and $Y$ coordinate matrices
  - Evaluate $Z = f(X, Y) = \sin\left(\sqrt{X^2 + Y^2}\right)$
  - Generate a surface plot of $Z$ (see p. 11)

- Experiment with any of the following:
  - Number of points in x and y
  - Color map
  - View angle
  - Type of plot



# Animation

# Animation – Creating Movies



```
FuncAnimation(fig, func, frames,
              init_func=<>, interval=200
              repeat=True, blit=False)
```

- `fig`: figure where frames are plotted
- `func`: the function that plots the frames
- `frames`: number of frames in the animation
- `init_func`: user-defined function to draw a clear frame - *optional*
- `interval`: frame period in msec – *optional* – default: `200`
- `repeat`: loop the animation – *optional* – default: `True`
- `blit`: apply drawing optimization – *optional* – default: `False`

- `FuncAnimation` is in the `matplotlib.animation` module:

```
from matplotlib.animation import FuncAnimation
```

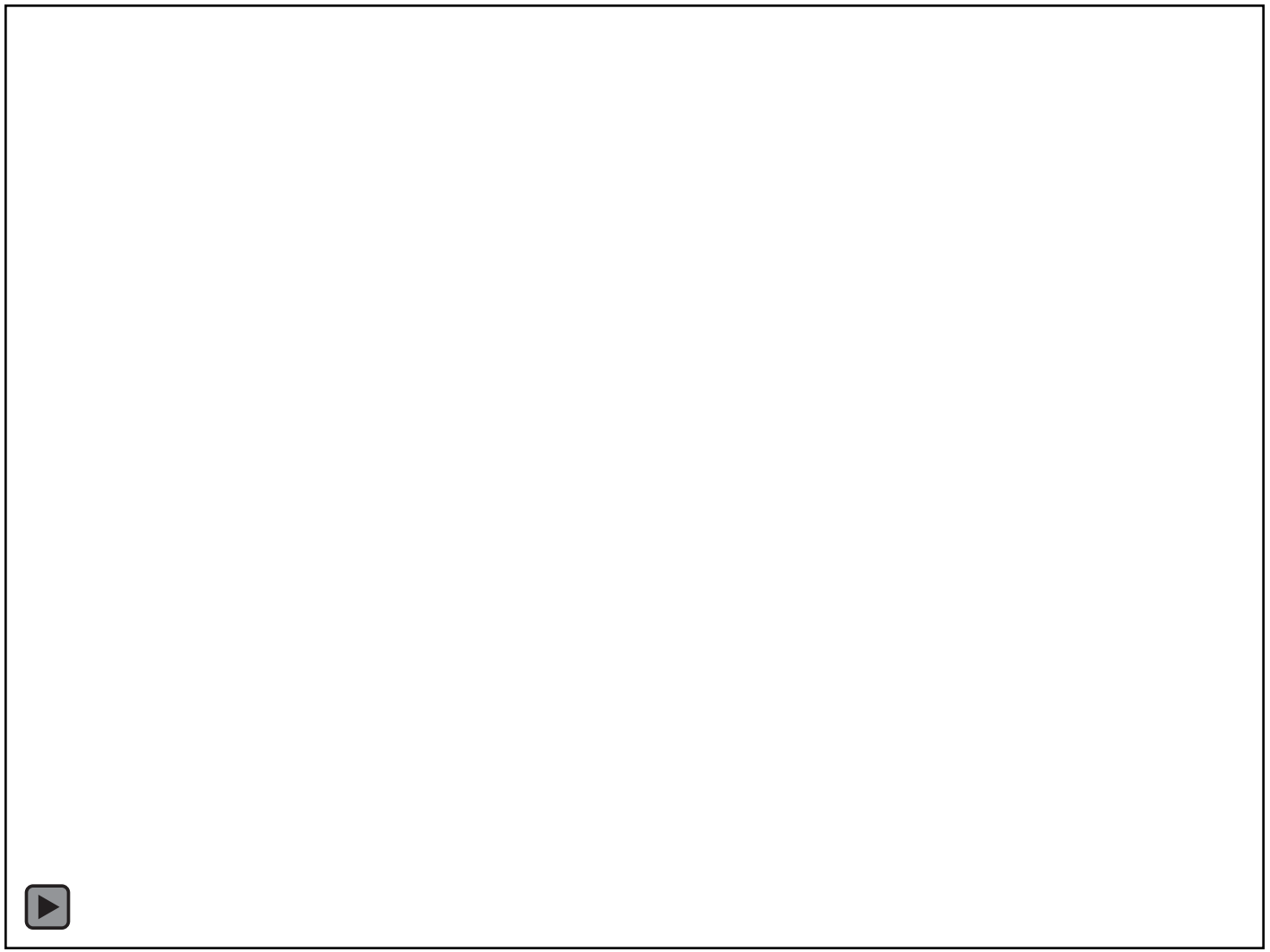
# Animation – Example



- Animate the motion of a projectile through the earth's gravitational field, neglecting drag
  - Initial velocity: $v_0$
  - Launch angle: $\theta_0$
  - Gravitational acceleration: $g = 9.81 \frac{m}{s^2}$
- Horizontal position: $x = v_0 \cos(\theta_0) \cdot t$
- Vertical position: $y = v_0 \sin(\theta_0) \cdot t - \frac{1}{2} g \cdot t^2$

# Animation – Example





# Animation – Generate the Data



- First, generate the data to be plotted
- Two options:
  - Generate all ahead of time (i.e. before plotting)
  - Generate point-by-point in your animation function
- Here, we'll create the data vectors first:

```
3   from numpy import sin, cos
4   import numpy as np
5   import matplotlib.pyplot as plt
6   from matplotlib.animation import FuncAnimation
7
8   g = 9.81
9   theta0 = np.radians(45)           # launch angle
10  v0 = 25                         # init. velocity
11
12  dt = 0.02                        # frame period
13
14  # calculate time for z = 0 (landing time)
15  tf = 2*v0/g*sin(theta0)
16  t = np.arange(0, tf, dt)        # time vector
17
18
19  x = v0*cos(theta0)*t
20  z = v0*sin(theta0)*t - 0.5*g*t**2
21
```

Horizontal and vertical positions of projectile

# Animation – Set Up the Axes



```
22  fig = plt.figure(1)
23  plt.clf()
24
25  # set up the axes to accomodate the full trajectory
26  # and do not allow them ot autoscale
27  ax = fig.add_subplot(autoscale_on=False,
28                       xlim=(0, 1.05*max(x)),
29                       ylim=(0, 1.1*max(z)))
30
31  # generate the lines.Line2D objects to be modified
32  # in the plotting function (makes saving work better)
33  loc, = ax.plot([], [], 'bo')
34  traj, = ax.plot([], [], 'r-', linewidth=2)
35  time_txt = ax.text(0.05*max(x), 1.0*max(z), '',
36                     fontsize=12, fontweight='bold')
37
38  ax.set_xlabel('x   [m]', fontweight='bold')
39  ax.set_ylabel('z    [m]', fontweight='bold')
40  ax.set_title(r'''Projectile Trajectory
41  $\theta_0 = {:.0f}\degree, V_0 = {}\/m/s$'''.
42               format(np.degrees(theta0), v0))
```

- Create/clear figure
- Create axes, set limits and turn off autoscaling
- Create empty line, marker, and text objects
- Set axis labels and title

# Animation – Initialization Function



- Define an ***initialization function*** to pass to `FuncAnimation()`
  - Called once, prior to first frame
  - Draws a clear frame
  - Updates line and text objects with empty lists and strings
  - Optional, but helps with indexing issues on first few frames

```
43
44  def frame_init():
45      traj.set_data([], [])
46      loc.set_data([], [])
47      time_txt.set_text('')
48
49      return traj, loc, time_txt
50
```

```
65
66  ani = FuncAnimation(fig, animate, len(t), init_func=frame_init,
67                      interval=dt*1000, repeat=False, blit=True)
68
```

# Animation – Plotting Function



- Define a ***plotting function*** to pass to `FuncAnimation()`
  - This is the function that is called repeatedly to create frames
  - Updates line and text objects with data for current frame

```
52  def animate(i):
53      x_i = x[i]
54      z_i = z[i]
55      xhist_i = x[:i+1]
56      zhist_i = z[:i+1]
57
58      traj.set_data(xhist_i, zhist_i)
59      loc.set_data(x_i, z_i)
60      time_txt.set_text('t = {:0.2f} sec'.format(t[i]))
61
62      return traj, loc, time_txt
```

```
65
66  ani = FuncAnimation(fig, animate, len(t), init_func=frame_init,
67                      interval=dt*1000, repeat=False, blit=True)
68
```

# Animation – Create the Animation



```
FuncAnimation(fig, func, frames,
              init_func=<>, interval=<>
              repeat=False, blit=True)
```

- `fig`: figure where frames are plotted
- `func`: the function that plots the frames
- `frames`: number of frames in the animation
- `init_func`: user-defined function to draw a clear frame
- `interval`: frame period in msec
- `repeat`: loop the animation or not – *optional* – default: `True`
- `blit`: apply drawing optimization or not – *optional* – default: `False`

```
65
66    ani = FuncAnimation(fig, animate, len(t), init_func=frame_init,
67                        interval=dt*1000, repeat=False, blit=True)
68
```

# Animation – Saving the Animation



- Save animation as .mp4 file
- May need to first install the `ffmpeg MovieWriter`
  - Open Spyder or an Anaconda console (Anaconda Prompt)
  - Enter in the console:

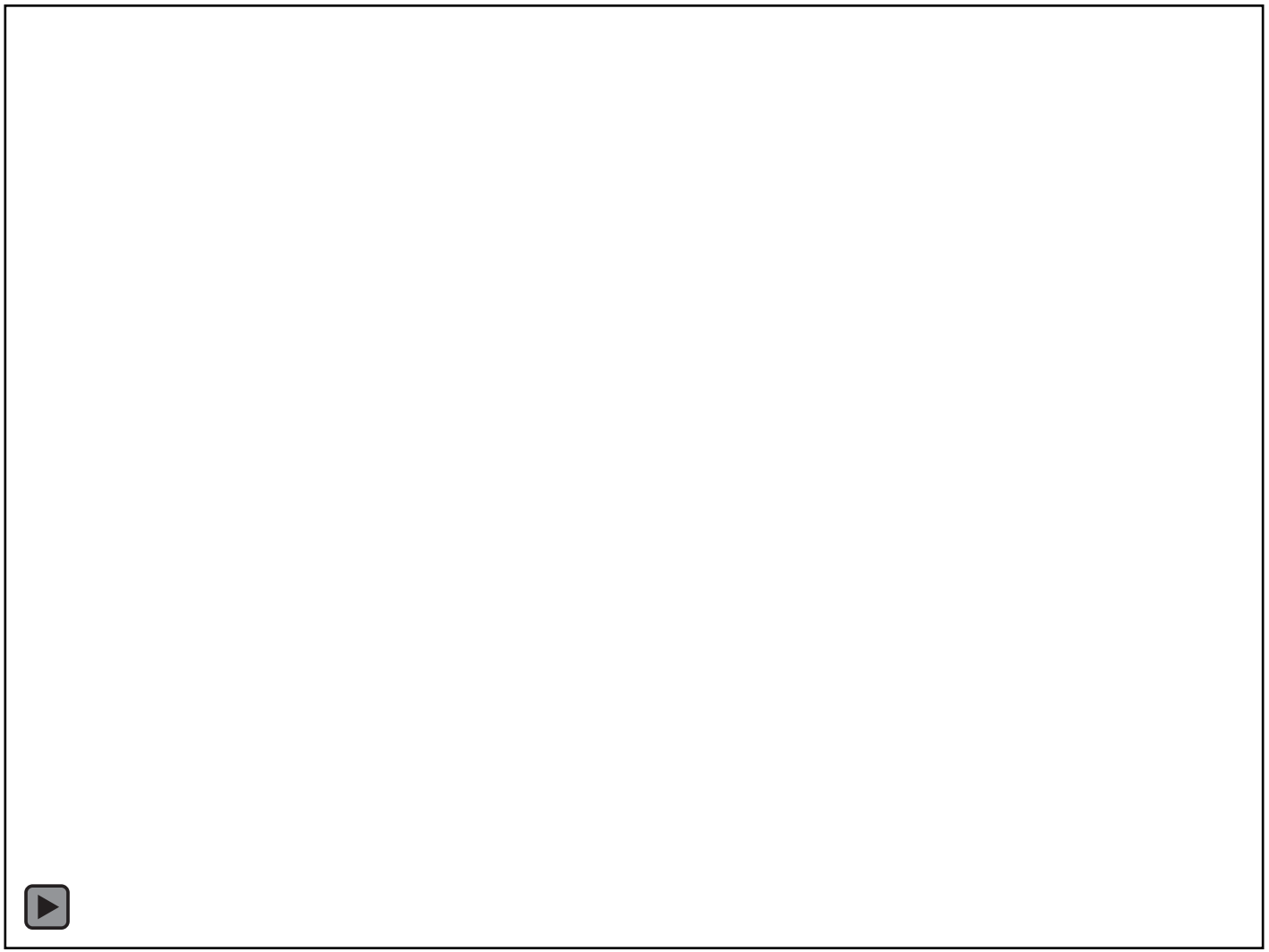
```
conda install -c conda-forge ffmpeg
```

- Run the `.save()` method on your animation object
  - Pass to it the desired file name

```
65
66  ani = FuncAnimation(fig, animate, len(t), init_func=frame_init,
67                      interval=dt*1000, repeat=False, blit=True)
68
69
70  ani.save('projectile.mp4')
71
```

# Animation – Example





# Exercise - Animation



Exercise

- Install `ffmpeg MovieWriter`, if you have not already done so
- Create a script to replicate the animation example presented in the notes
- Once you have it working, experiment with any of the following:
  - Adjust the frame period
  - Adjust projectile parameters
  - Try with and without specifying an `init_func`
  - Change line and marker characteristics
  - Play around with axis limits and autoscaling settings
  - Plot some other function (e.g., standing wave)