Lecture 6

Learning outcomes:

- *Data Visualization for Engineering:*
  - *Plotting 2D and 3D graphs relevant to engineering problems*

Understanding Engineering Graphs

- Engineering Graphs are fundamental to interpreting and analysing data in the field of engineering.
- These graphs provide a visual representation of data and can assist in identifying underlying patterns, trends, and relationships.
- Understanding this form of data representation can significantly aid in designing effective solutions to engineering problems.

Cartesian Plots in Engineering

- A Cartesian plot is a type of graph that depicts relationships between two variables where values for each are plotted along X and Y axes.

- Cartesian plots, sometimes referred to as Cartesian coordinates or a Cartesian grid, are incredibly valuable graph types in Engineering Mathematics. On these graphs, you plot data points according to their x- and y- coordinates. The coordinates represent the intersection of the lines drawn from the point vertically (Y-axis) and horizontally (X-axis) to the axes.

- One example of a Cartesian plot is a straight line graph represented by the formula:

$y = mx + c$

where m is the gradient of the line and c is the y-intercept. You can characterise these plots into:

- ✓1D (One Dimensional)
- ✓2D (Two Dimensional)
- ✓3D (Three Dimensional)

❑ Interpreting Engineering Graphs

➢Engineering graphs provide you with a powerful tool for data interpretation. However, it's important not only to read them correctly but to also avoid common errors that could potentially distort the data analysis.

❑ Reading and Analysing Engineering Graphs

➢A systematised approach to reading any kind of graph includes recognising the type of graph used, understanding the scales used on the axes, identifying key data points, and interpreting the overall trend. For instance, when analysing a logarithmic plot, if the x-axis (horizontal) operates on a logarithmic scale, each increment can represent a tenfold increase. On the other hand, if the y-axis (vertical) is logarithmic, it requires a relative comparison rather than an absolute one.

Tools and Techniques for Engineering Graphs

- The effective creation and interpretation of Engineering Graphs often demand the use of various tools and technologies. From conventional methods like graph paper and compasses to sophisticated software applications, these tools are crucial in inscribing and deciphering intricate graphical data.

- Examples of Popular Engineering Graph Tools

➢Introducing digital tools into your Engineering Graphs' development can aid in creating more precise, complex, and visually appealing graphical data. Here are a few examples of popular digital tools:

- Microsoft Excel: An excellent tool for constructing and analysing various types of graphs. Its wide array of pre-constructed graph types and customisable options make it a go-to choice for many engineers.
- Matplotlib: A popular Python Library extensively used in producing high quality 2D and 3D graphs. With its customisable feature, it becomes an excellent tool for engineering mathematics.
- Tableau: An advanced data visualisation tool that excels in creating interactive charts, making data analysis and interpretation intuitive. It's especially beneficial for larger datasets.
- AutoCAD: A software application extensively used for creating 2D and 3D Engineering Graphs in fields like architectural and mechanical engineering.
- Graphing Calculator: An electronic calculator capable of plotting graphs, solving simultaneous equations, and performing numerous other tasks.

Plotting in Engineering Mathematics using Tools

- When constructing graphs for engineering mathematics, physical tools have limited use due to the complexity and precision demanded by these graphs.

- Modern digital tools mentioned above offer the required advanced features to conveniently plot these graphs.

- These tools notably have a wide range of built-in functionalities that include, but are not limited to, customising data series, recalculating automatically when data is modified, applying a trend line to your data, among others.

Key Tips and Tricks for Effective Plotting in Engineering Mathematics

- While these digital tools ease plotting graphs, certain tips can make the process more efficacious:

- Proper Labelling: Always label the axes accurately and provide units.
- Scale Selection: Select your scale carefully to ensure the data is neither squashed nor stretched.
- Data Segregation: When working with multiple data sets, colour code for easy distinction.
- Accuracy: Ensure precise plotting of data points and drawing of lines between them.
- Legend Usage: A valuable tool for helping readers understand the plotted data better.

Avoiding Common Pitfalls in Plotting Engineering Graphs

- While plotting graphs, it's also important to be aware of some of the common errors that can lead to misinterpretations.
- Neglecting Negative Values: Don't forget to include negative numbers if your data range includes them.
- Non-uniform Scale: Always have a uniform scale for accurate representation of the data.
- Missing Outliers: Ensure to include all data points, particularly the outliers, for a complete picture.
- Starting From Zero: Starting the y-axis from zero can often help give an accurate representation of the data.

✓These potential pitfalls can distort the picture the data presents and hence, they should be avoided for accurate and meaningful interpretation.

2D and 3D-Plotting In Matplotlib

❖Matplotlib library from Python is quite a potent tool in plotting engineering mathematics graphs.

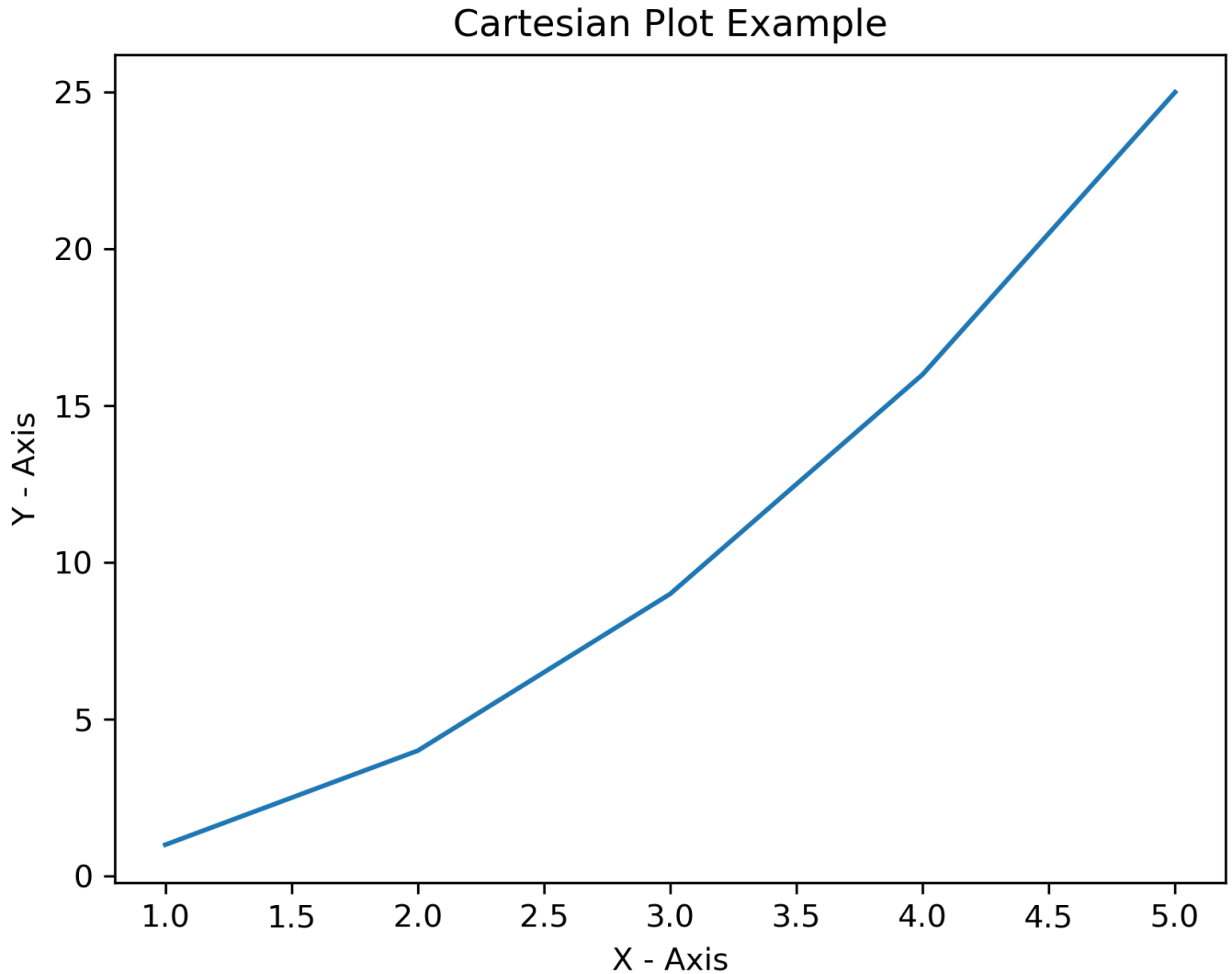
❖Matplotlib is the workhorse of visualization in Python and therefore, it should always be your first choice, before trying anything else.

❑ 2D-plotting in matplotlib

- To see how plotting with matplotlib works, let’s start with a simple example of 2D curve plotting,

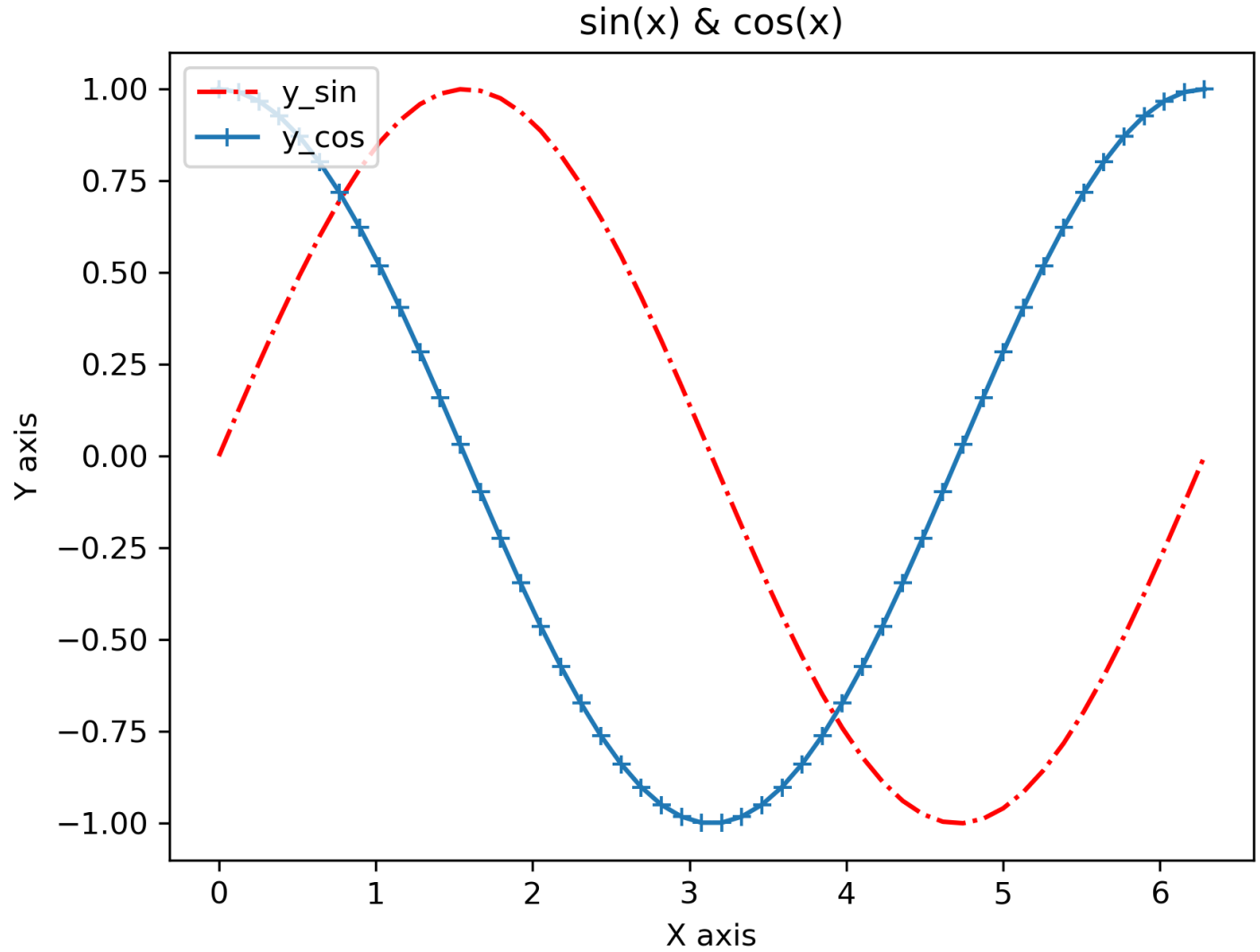
```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
plt.plot(x, y)
plt.xlabel('X - Axis')
plt.ylabel('Y - Axis')
plt.title('Cartesian Plot Example')
plt.show()
```

✓Here, 'x' and 'y' are the respective coordinates. This code will generate a Cartesian graph with 'x' and 'y' as the axes.


Cartesian Plot Example
25
20
15
10
5
0
Y - Axis
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
X - Axis

Draw multiple graphics in one drawing

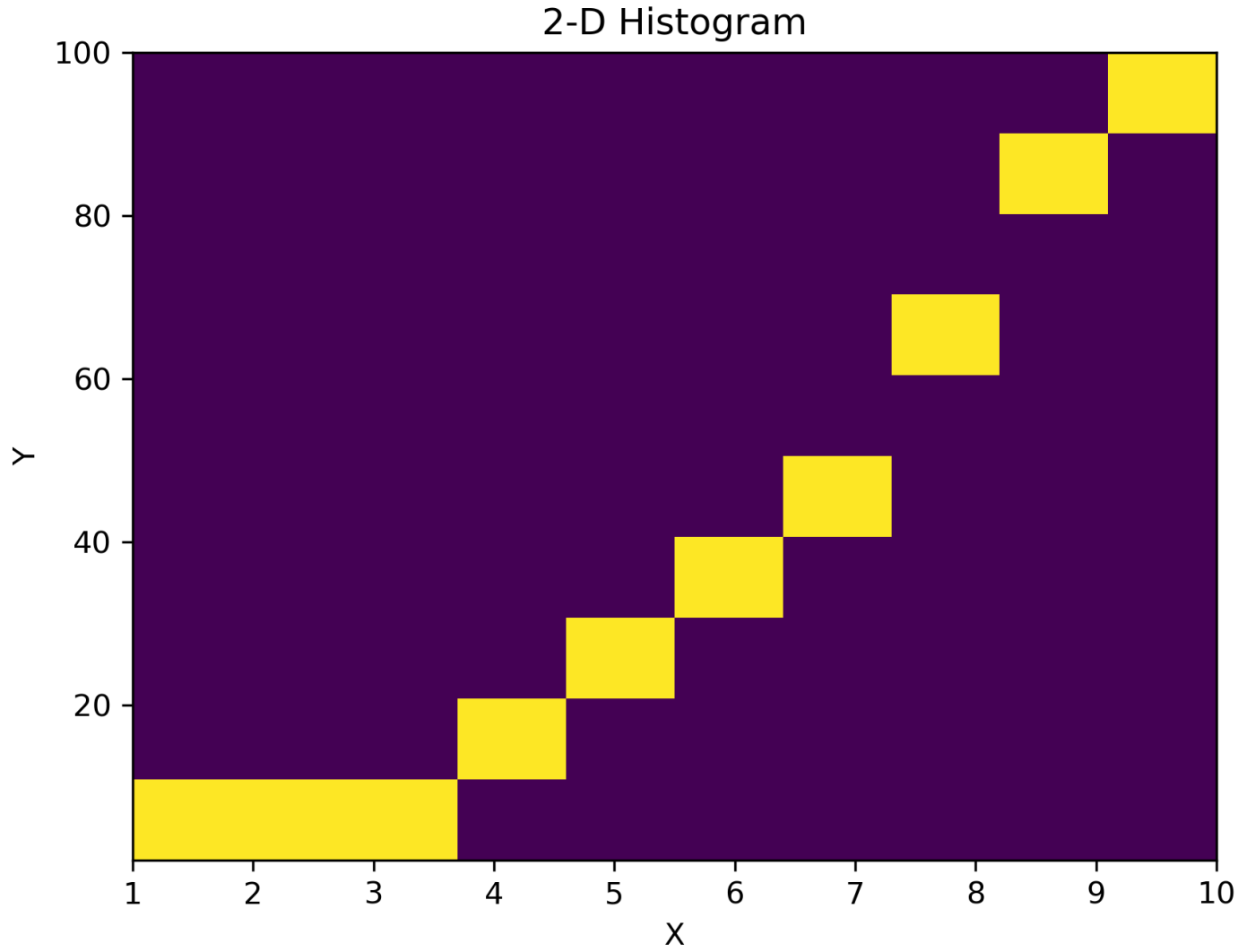
```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
# Generates 50 evenly spaced values between 0 and 2π (approx. 6.28), which serves as the X-axis values
x = np.linspace(0, 2 * np.pi, 50)
y_sin = np.sin(x) # Computes the sine value for each x.
y_cos = np.cos(x) # Computes the cosine value for each x.
plt.title('sin(x) & cos(x)')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.plot(x, y_sin, color = "red", linewidth = 1.5, linestyle = "-.", label = "y_sin")
plt.plot(x, y_cos, marker = '+', linestyle = '-', label = 'y_cos')
plt.legend(loc = "upper left") # Adds a legend in the upper left corner, displaying "y_sin" and "y_cos".
plt.show()
```


sin(x) & cos(x)
y_sin
y_cos
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y axis
0
1
2
3
4
5
6
X axis

Create a simple 2d histogram using matplotlib

- Here, we are creating a simple 2d histogram using matplotlib in Python.
- Firstly we have to import matplotlib module. Create a simple dataset to be plotted on the x and y-axis using a list of some integers and then used hist2d() function to plot the graph. we have also added labels on the x and y-axis using xlabel() and ylabel() functions and defined the title using title() function. At last, plotting the 2d histogram using the plt.show() function.

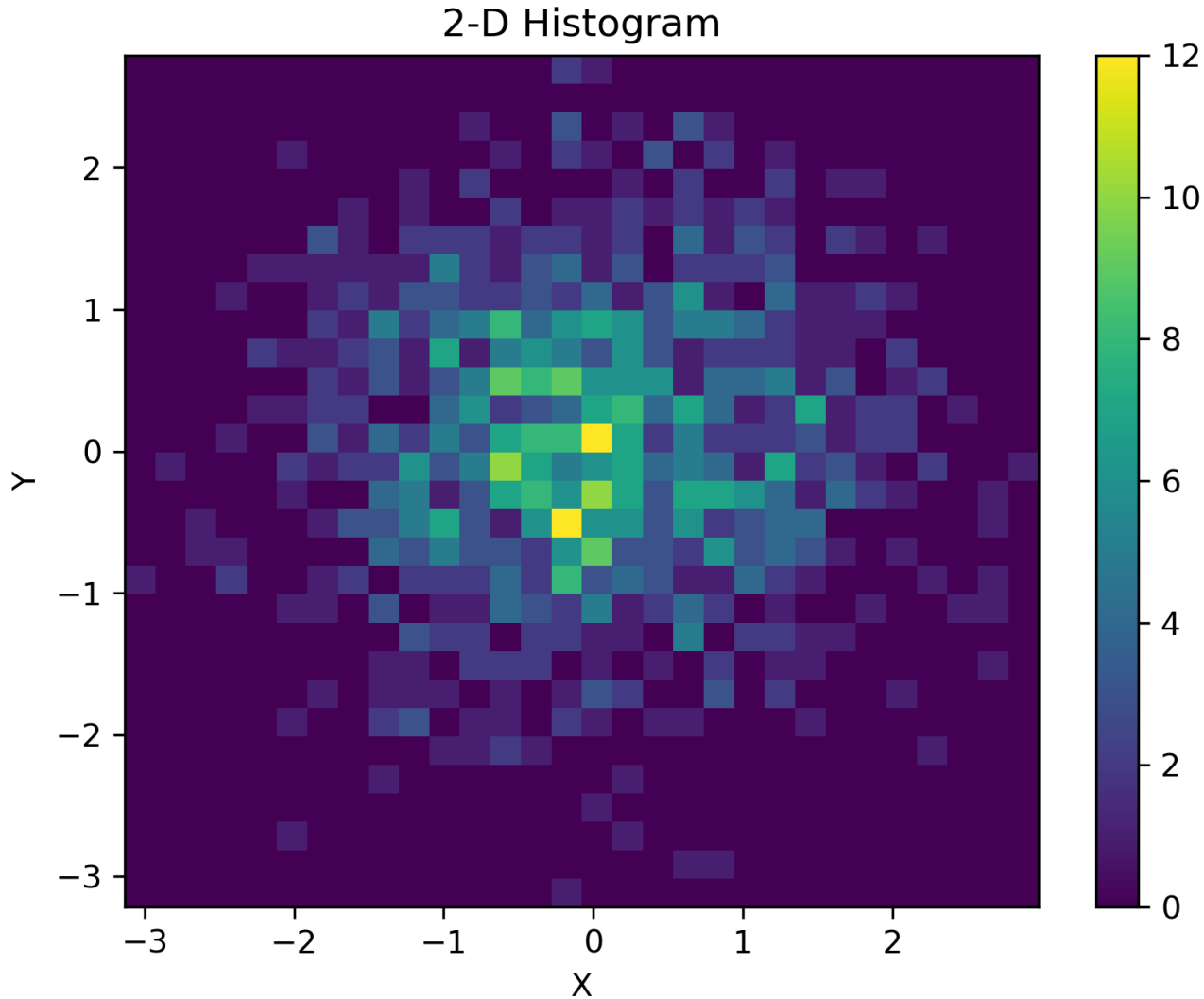
```
# Import required modules
import matplotlib.pyplot as plt
# create data for x and y-axis
x = [1,2,3,4,5,6,7,8,9,10]
y = [1,4,9,16,25,36,49,64,81,100]
# Create a 2-D histogram
plt.hist2d(x, y)
# Add labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('2-D Histogram')
# Display the plot
plt.show()
```


2-D Histogram
100
80
60
40
20
Y
1
2
3
4
5
6
7
8
9
10
X

Create a 2d histogram using Matplotlib and NumPy in Python

- Here, we are going to create a 2d histogram using NumPy which is used to create a random dataset.
- This time we have used bins(30, 30) as parameter in our hist2d() function, and cmap which is a color map used to define the colors of the plot we set the cmap value as ‘viridis’ which is its default value. Use colorbar() function to draw a color bar at the right side of the plot indicating which color denotes what set values. Adding labels and then plotting the 2d histogram using the plt.show() function.

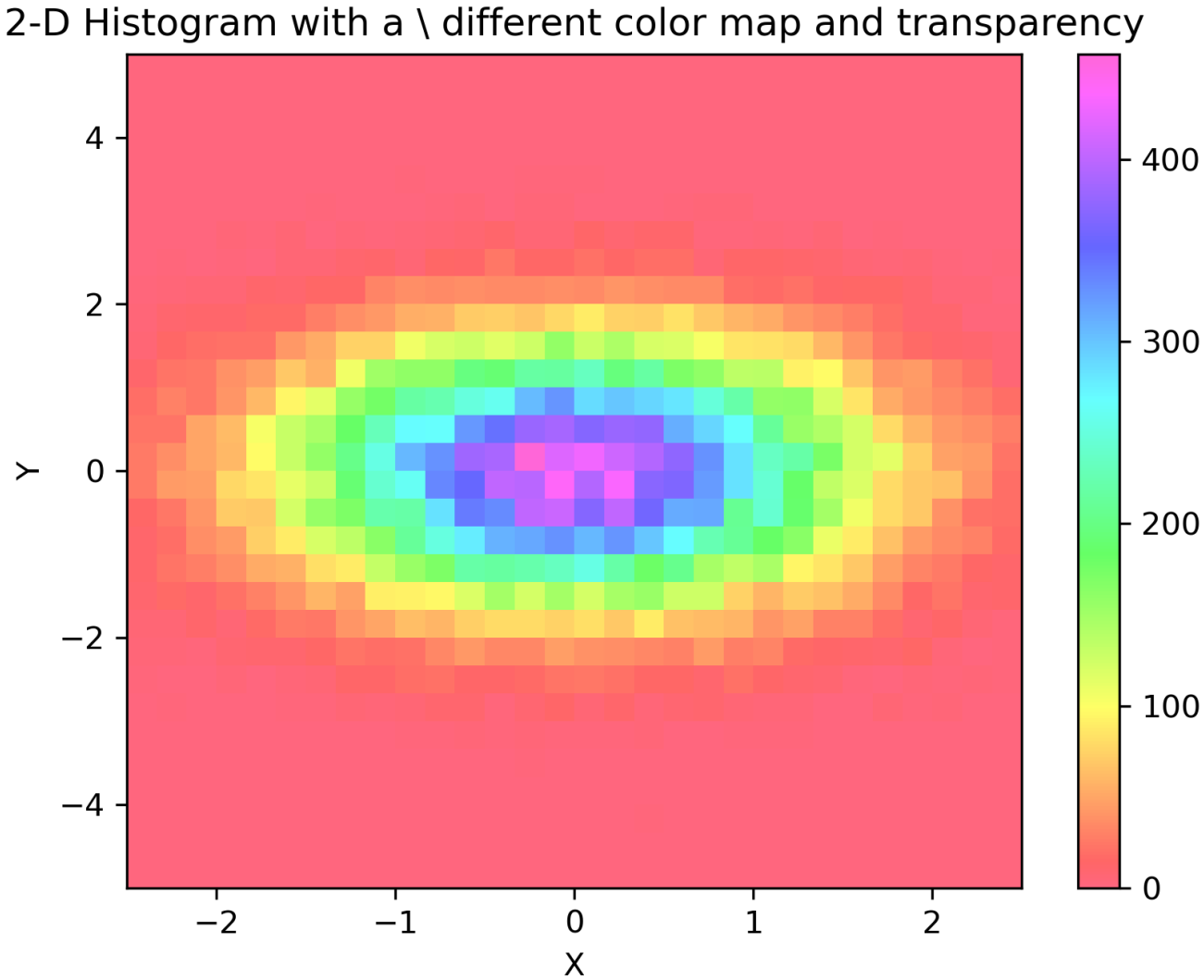
```
# Importing required modules
import matplotlib.pyplot as plt
import numpy as np
# Generate random data
x = np.random.randn(1000)
y = np.random.randn(1000)
# Create a 2-D histogram
plt.hist2d(x, y, bins=(30, 30), cmap='viridis')
plt.colorbar()
# Add labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('2-D Histogram')
# Display the plot
plt.show()
```


2-D Histogram
Y
X
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
12
10
8
6
4
2
0

Customize the 2d histogram using Matplotlib in Python

- Here, we have plotted the 2d histogram using Matplotlib with some customization.
- The procedure is the same as above with some changes to plot the customized 2d histogram. Here we have imported cm which is a color map from matplotlib to change the color of the plot. In hist2() function we have changed the color to a rainbow using cmap and made the plot transparent setting alpha=0.6. Its value ranges from 0 to 1. We have also set the range of the x and y-axis to (-2.5,2.5) and (-5,5) respectively as shown in the output.

```
# Import required libraries
import matplotlib.pyplot as plt
from matplotlib import cm     # For using colormaps.
import numpy as np
# Generate random data
x = np.random.randn(50000)
y = np.random.randn(50000)
# Create a 2-D histogram with a
# different color map and transparency
# bins=(30, 30) → Divides the X & Y axes into 30 bins each (making a 30x30 grid).
# cmap=cm.gist_rainbow → Uses the "gist_rainbow" colormap for vibrant colors.
# range=[(-2.5,2.5), (-5,5)] → Sets the X-axis range from -2.5 to 2.5 and Y-axis range from -5 to 5.
plt.hist2d(x, y, bins=(30, 30), cmap=cm.gist_rainbow, alpha=0.6, range=[(-2.5,2.5),(-5,5)])
plt.colorbar() #  Adds a color bar to indicate density levels (higher frequency = brighter color).
# Add labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('2-D Histogram with a \ different color map and transparency')
# Display the plot
plt.show()
```


2-D Histogram with a \ different color map and transparency
4
2
0
−2
−4
Y
−2
−1
0
1
2
X
400
300
200
100
0

❑ 3D-plotting in matplotlib

- 3D plots are very important tools for visualizing data that have three dimensions such as data that have two dependent and one independent variable.

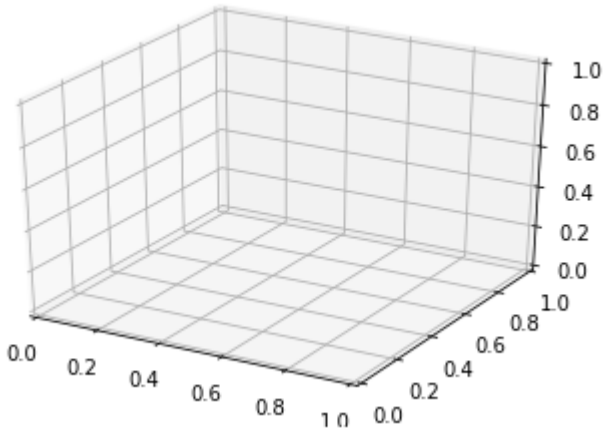
- By plotting data in 3d plots we can get a deeper understanding of data that have three variables.

- We can use various matplotlib library functions to plot 3D plots.

Examples to plot 3D

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

fig = plt.figure()
ax = plt.axes(projection='3d')
```



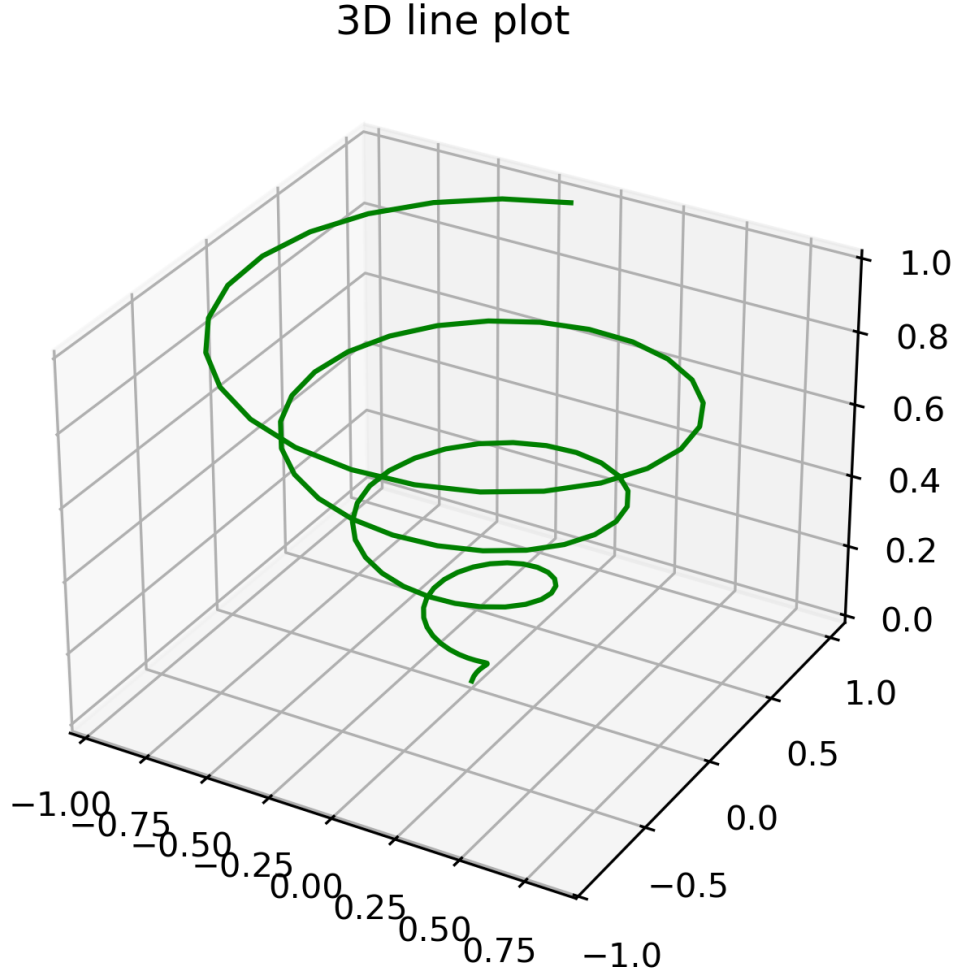
- With the above syntax three -dimensional axes are enabled and data can be plotted in 3 dimensions. 3 dimension graph gives a dynamic approach and makes data more interactive. Like 2-D graphs, we can use different ways to represent to plot 3-D graphs. We can make a scatter plot, contour plot, surface plot, etc. Let’s have a look at different 3-D plots.

- Graphs with lines and points are the simplest 3-dimensional graph. We will use ax.plot3d and ax.scatter functions to plot line and point graph respectively.

3-Dimensional Line Graph Using Matplotlib

For plotting the 3-Dimensional line graph we will use the mplot3d function from the mpl_toolkits library. For plotting lines in 3D we will have to initialize three variable points for the line equation. In our case, we will define three variables as x, y, and z.

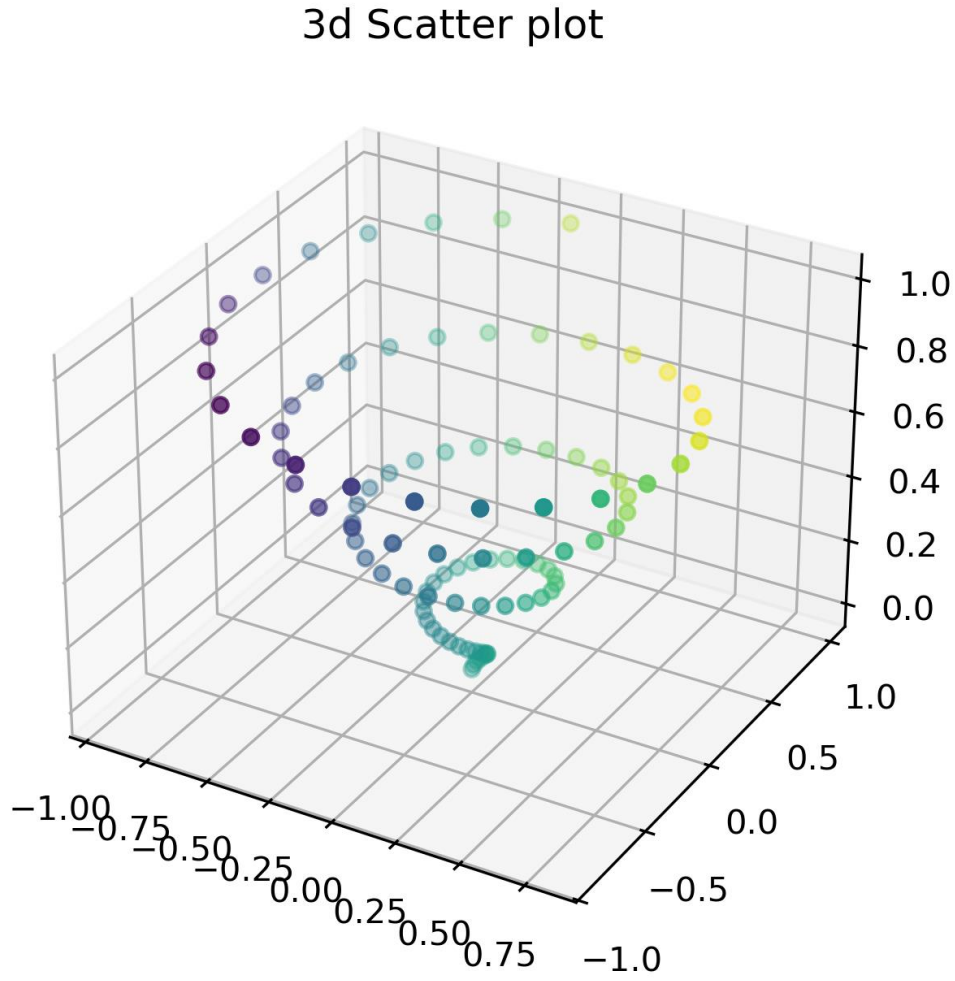
```
# importing mplot3d toolkits, numpy and matplotlib
from mpl_toolkits import mplot3d  # mpl_toolkits.mplot3d → Enables 3D plotting in Matplotlib.
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
# syntax for 3-D projection
ax = plt.axes(projection ='3d') # Creates a 3D figure and initializes a 3D axis using projection='3d'.
# defining all 3 axis
z = np.linspace(0, 1, 100) # Generates 100 evenly spaced values from 0 to 1 (Z-axis).
x = z * np.sin(25 * z) # Defines x as a spiral function.
y = z * np.cos(25 * z) # Defines y as a spiral function.
# plotting
ax.plot3D(x, y, z, 'green')
ax.set_title('3D line plot')
plt.show()
```


3D line plot
1.0
0.8
0.6
0.4
0.2
0.0
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75

3-Dimensional Scattered Graph Using Matplotlib

To plot the same graph using scatter points we will use the scatter() function from matplotlib. It will plot the same line equation using distinct points.

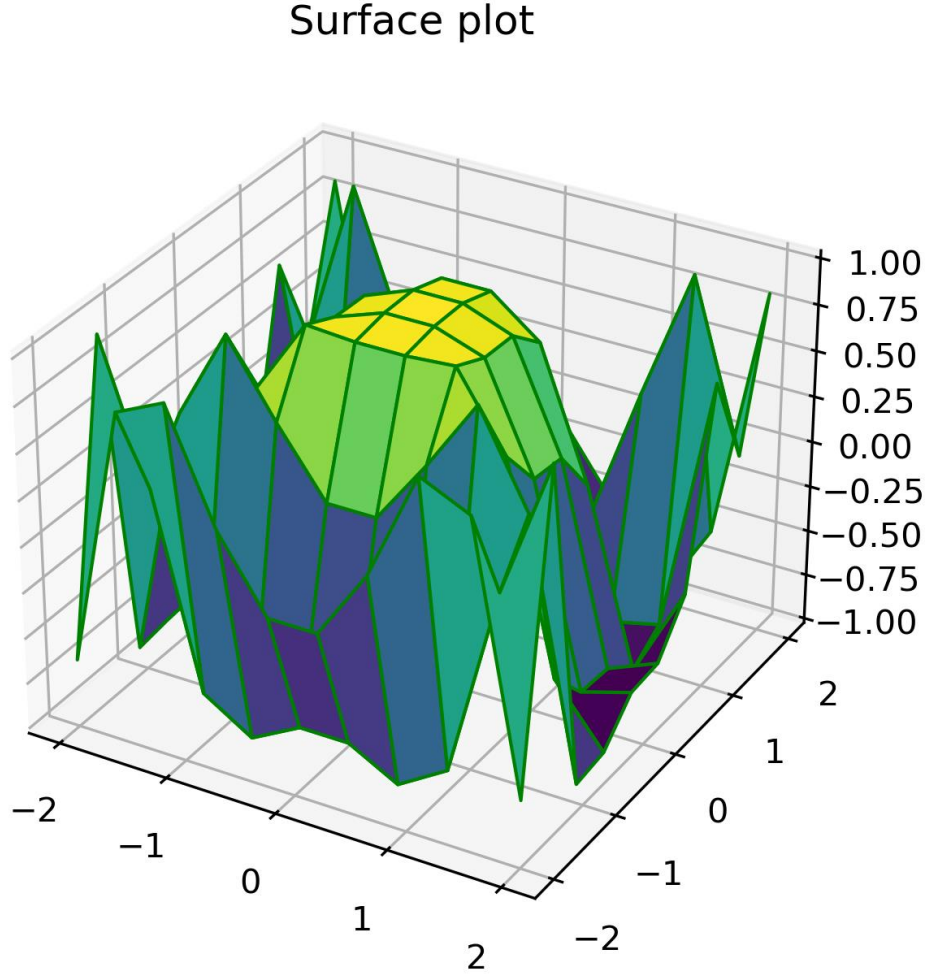
```
# importing mplot3d toolkits
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
# syntax for 3-D projection
ax = plt.axes(projection ='3d')
# defining axes
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)
c = x + y
ax.scatter(x, y, z, c = c) # creates a 3D scatter plot where points are plotted in (x, y, z) space with colors determined by c.
# syntax for plotting
ax.set_title('3d Scatter plot')
plt.show()
```


3d Scatter plot
1.0
0.8
0.6
0.4
0.2
0.0
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75

- Surface Graphs using Matplotlib library

Surface graphs and Wireframes graph work on gridded data. They take the grid value and plot it on a three-dimensional surface. We will use the plot_surface() function to plot the surface plot.

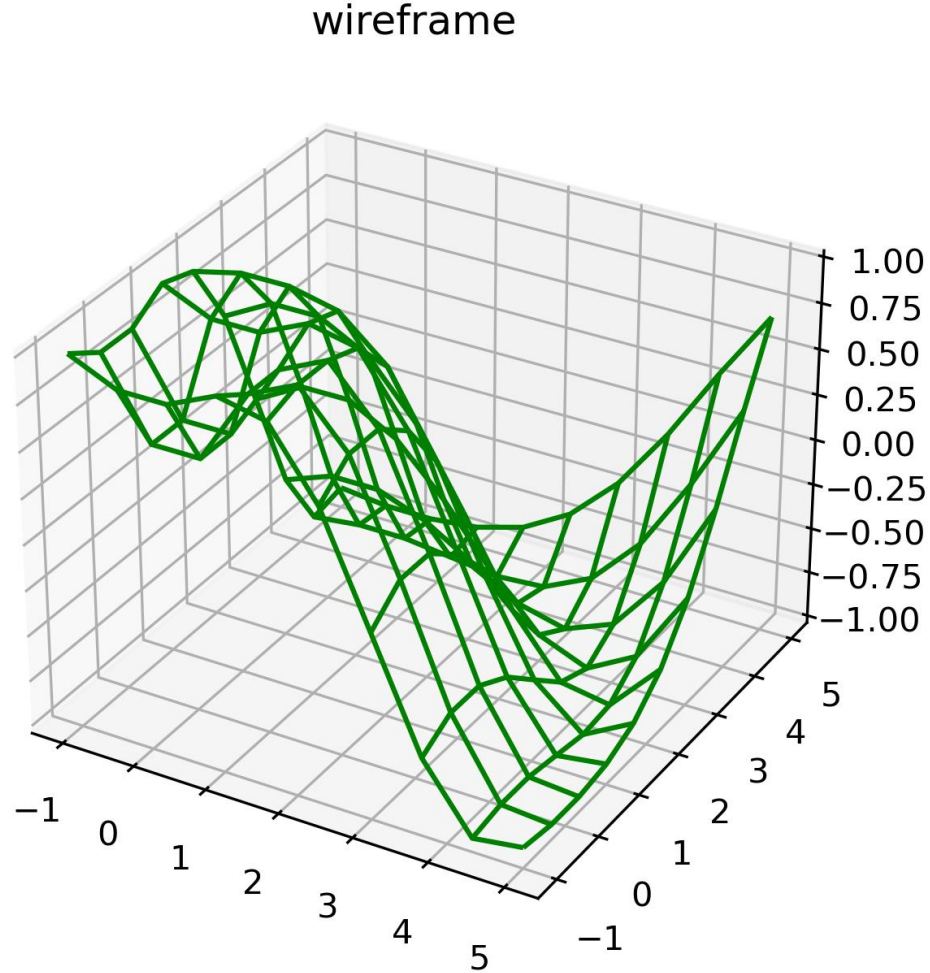
```
# Importing libraries
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
# Defining surface and axes
x = np.outer(np.linspace(-2, 2, 10), np.ones(10)) # Creates a 10x10 grid for X-axis values.
y = x.copy().T  # Copies X values and transposes them to get Y values.
z = np.cos(x ** 2 + y ** 3) # Computes Z values using a mathematical function.
# Creating a figure
fig = plt.figure()
# Syntax for 3D plotting
ax = plt.axes(projection='3d')
# Plotting the surface
ax.plot_surface(x, y, z, cmap='viridis', edgecolor='green') # Uses the "viridis" colormap / Outlines the grid with green edges.
# Adding title
ax.set_title('Surface Plot')
# Displaying the plot
plt.show()
```


Surface plot
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−2
−1
0
1
2
2
1
0
−1
−2

- Wireframes graph using Matplotlib library

For plotting the wireframes graph we will use the plot_wireframe() function from the matplotlib library.

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
# function for z axis
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))
# x and y axis
x = np.linspace(-1, 5, 10)
y = np.linspace(-1, 5, 10)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig = plt.figure()
ax = plt.axes(projection ='3d')
ax.plot_wireframe(X, Y, Z, color ='green')
ax.set_title('wireframe');
```

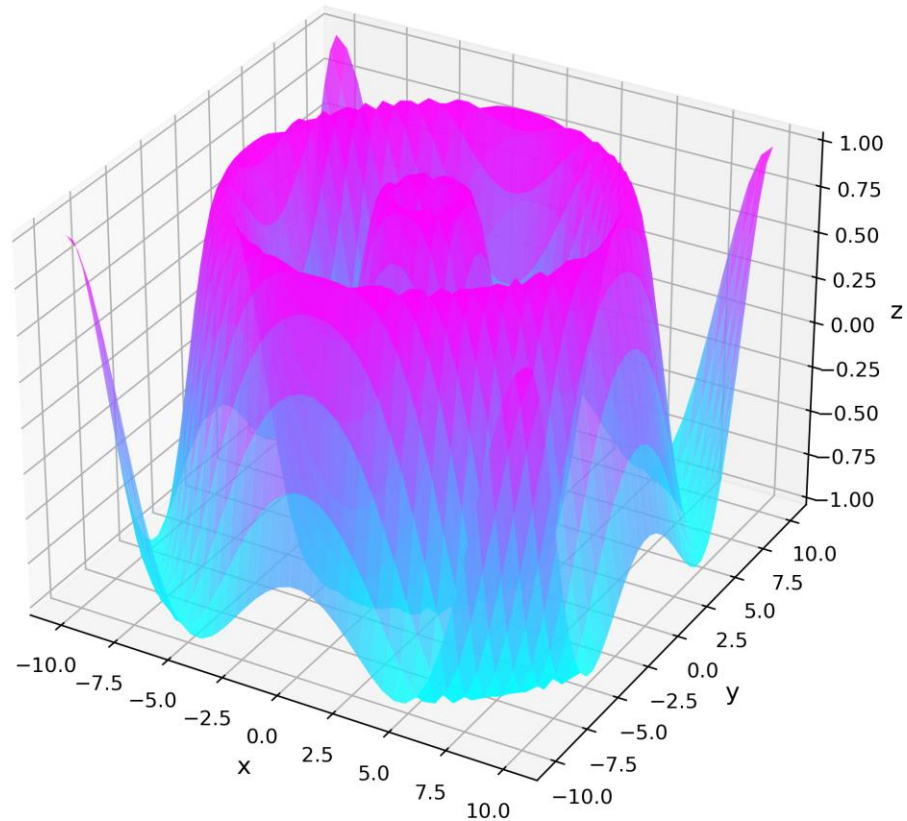

wireframe
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1
0
1
2
3
4
5
−1
0
1
2
3
4
5

- Contour Graphs using Matplotlib library

The contour graph takes all the input data in two-dimensional regular grids, and the Z data is evaluated at every point. We use the ax.contour3D function to plot a contour graph. Contour plots are an excellent way to visualize optimization plots.

```
def function(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))
# Creates 40 evenly spaced points from -10 to 10.
x = np.linspace(-10, 10, 40)
y = np.linspace(-10, 10, 40)
X, Y = np.meshgrid(x, y) # Forms a grid of coordinates for plotting.
Z = function(X, Y) # Computes Z values using the function f(x,y), applying the sine function to each coordinate pair.
fig = plt.figure(figsize=(10, 8))
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, cmap='cool', alpha=0.8) # Uses the "cool" colormap (blue to pink).
ax.set_title('3D Contour Plot of function(x, y) =\ sin(sqrt(x^2 + y^2))', fontsize=14)
ax.set_xlabel('x', fontsize=12)
ax.set_ylabel('y', fontsize=12)
ax.set_zlabel('z', fontsize=12)
plt.show()
```

3D Contour Plot of function(x, y) =\ sin(sqrt(x^2 + y^2))



- Plotting Surface Triangulations In Python

The above graph is sometimes overly restricted and inconvenient. So by this method, we use a set of random draws. The function ax.plot_trisurf is used to draw this graph. It is not that clear but more flexible.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.tri import Triangulation
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))
```

```
# Creates 30 points between -6 and 6.
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = f(X, Y) # Computes the Z values.
# Creates a triangular mesh using Triangulation().
# Flattens X and Y using .ravel(), converting them into 1D arrays.
tri = Triangulation(X.ravel(), Y.ravel())
fig = plt.figure(figsize=(10, 8))
# Creates a 3D figure and initializes 3D axes
ax = plt.axes(projection='3d')
ax.plot_trisurf(tri, Z.ravel(), cmap='cool', edgecolor='none', alpha=0.8) # Plots a triangulated 3D surface.
ax.set_title('Surface Triangulation Plot of f(x, y) =\ sin(sqrt(x^2 + y^2))', fontsize=14)
ax.set_xlabel('x', fontsize=12)
ax.set_ylabel('y', fontsize=12)
ax.set_zlabel('z', fontsize=12)
plt.show()
```

Surface Triangulation Plot of f(x, y) =\sin(sqrt(x^2 + y^2))