Matplotlib Basics

1 What's Matplotlib?

Matplotlib is a popular data visualization library in Python. It allows you to create a wide range of plots, such as line plots, scatter plots, bar plots, histograms, and more.

1.1 How to import Matplotlib

You can import Matplotlib using the following command:

```python
import matplotlib.pyplot as plt
# or from matplotlib import pyplot as plt
```

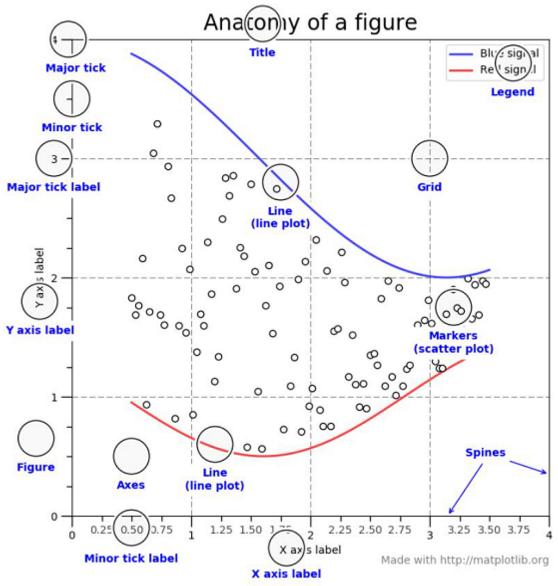
This is the standard way to import the library and gives you access to the pyplot submodule, which provides a convenient interface for creating and customizing plots.

1.2 Figure anatomy in matplotlib

The Matplotlib library, a Python library used for visualizing data, is built upon Numpy and consists of a tree-like hierarchical structure composed of objects that form each of the plots.

In Matplotlib, a "Figure" is the top-level container for a graph and can contain multiple "Axes" objects, which are not to be confused with the plural form of "Axis". An "Axes" object is a subplot that resides within a "Figure" and can be used to manipulate every aspect of the graph within it.

At the next level of the hierarchy beneath "Axes" are the tick marks, lines, legends, and text boxes. Every object within Matplotlib can be manipulated to create customized plots.



2 Basic Plotting

In this section, we'll cover the foundational concepts of plotting with Matplotlib.

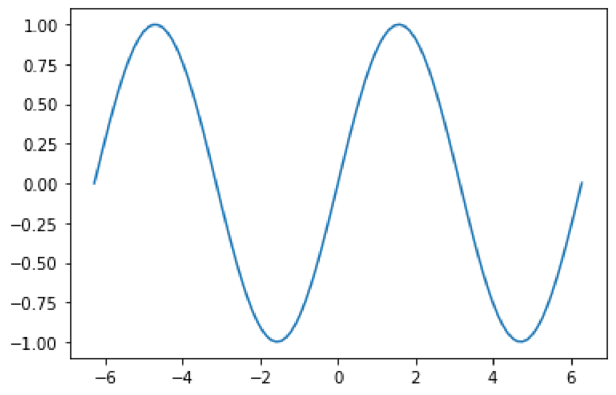
2.1 Line plots and Scatter Plots

Line plots To create a basic plot in Matplotlib, we can use the `plt.plot()` function. This function generates a two-dimensional plot for one or more iterable objects (lists of numbers or NumPy arrays) of points with known x and y coordinates.

It is important to note that the length of the x and y objects should be equal for the function to work correctly. For example, to plot the `sin` function we use the following code:

```
import matplotlib.pyplot as plt
import numpy as np
# Create an array of x values from 0 to 2*pi with a step size of 0.1
x = np.linspace(-2*np.pi, 2*np.pi, 100)

# Calculate y values for the sin function
y = np.sin(x)

# Plot the sine function
plt.plot(x, y)
# Diplay the result
plt.show()
```

`plt.plot` creates a Matplotlib object (here, a Line2D object) and `plt.show()` displays it on the screen.



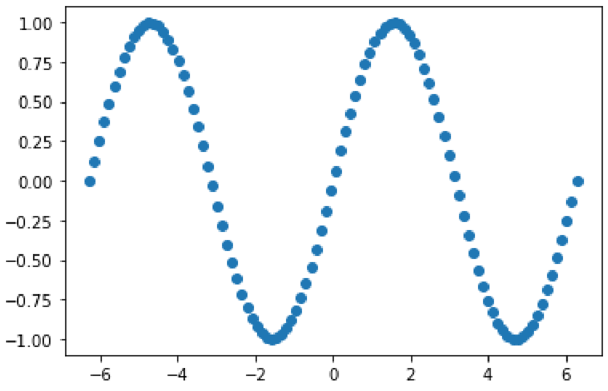
Scatter plots A scatter plot is a plot that shows the relationship between two variables. Unlike other plots, it represents each data point as a dot without any lines between them, with the position of the dot on the x and y axes indicating the values of the two variables for that data point. In Python, scatter plots can be created using `plt.scatter()` function. For example, we can plot the scatter plot of the previous function using the code: "'python import matplotlib.pyplot as plt import numpy as np # Create an array of x values from 0 to 2*pi with a step size of 0.1 x = np.linspace(*-2np.pi, 2*np.pi, 100)

3 Calculate y values for the sin function

y = np.sin(x)

4 Plot the sine function

plt.scatter(x, y) # Diplay the result plt.show() “‘

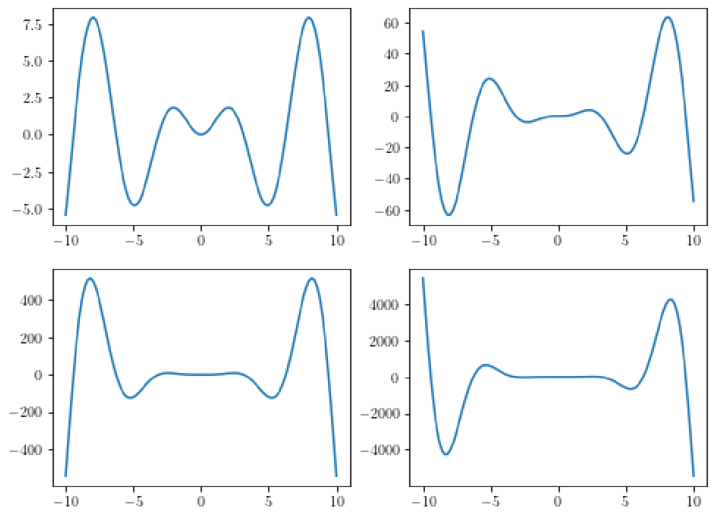


5 Use multiple subplots (axes)

You can create a new figure by calling the `plt.figure()` function. Once you have created a figure, you can add one or more axes objects to it using the `Figure.add_subplot()` or `plt.subplot()` methods which creates a grid of subplots within the figure.

For example, to create a figure with a size of 8x6 inches contains a 2x2 grid of 4 subplots, you can use the following code:

```
import matplotlib.pyplot as plt
import numpy as np
fig=plt.figure(figsize=(8,6))
x=np.linspace(-10,10,100)

ax1 = fig.add_subplot(3, 2, 1)
ax2 = fig.add_subplot(3, 2, 2)
ax3 = fig.add_subplot(3, 2, 3)
ax4 = fig.add_subplot(3, 2, 4)


ax1.plot(x,np.power(x,1.)*np.sin(x))
ax2.plot(x,np.power(x,2.)*np.sin(x))
ax3.plot(x,np.power(x,3.)*np.sin(x))
ax4.plot(x,np.power(x,4.)*np.sin(x))
plt.show()
```



6 Labels, Legends and Customization

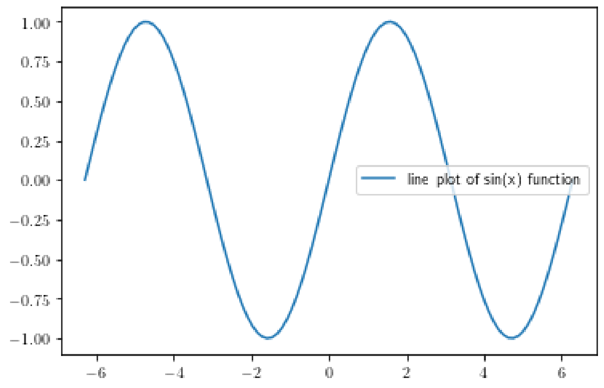
6.1 Labels and Legends

Labels and Legends are important components of a plot in Matplotlib that help in providing information about the data presented in the plot.

6.1.1 Plot legend

To add a label to each line in a plot, you can pass a string to the `label` argument of `plt.plot` function. However, to display the labels in the plot, you need to call `plt.legend()` function. For instance:

```
import matplotlib.pyplot as plt
import numpy as np
# Create an array of x values from 0 to 2*pi with a step size of 0.1
x = np.linspace(-2*np.pi, 2*np.pi, 100)

# Calculate y values for the sin function
y = np.sin(x)

# Plot the sine function
plt.plot(x, y,label="line plot of sin(x) function")

# Add the legend to the plot
plt.legend()
# plt.legend(loc=7)
# Diplay the result
plt.show()
```



By default, the legend will appear in the top-right corner of the plot. You can customize the location of the legend by using the `loc` argument of the `plt.legend()` function. The `loc` argument can be set to a string or integer value as given in the following table:

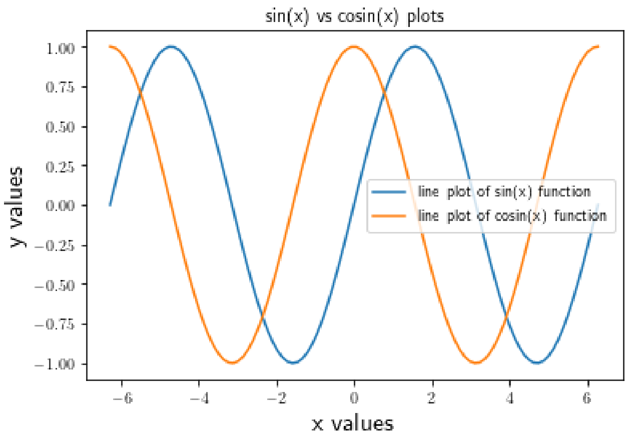
String	*Integer*
'best'	0
'upperright'	1
'upperleft'	2
'lowerleft'	3
'lowerright'	4
'right'	5
'centerleft'	6
'centerright'	7
'lowercenter'	8
'uppercenter'	9
'center'	10

The Plot Title Axis Labels To give a plot a title above its axes in Matplotlib, you can use the `plt.title` function by passing a string containing the title. Similarly, to label the **x-** and **y-axes**, you can use the `plt.xlabel` and `plt.ylabel` functions by passing the label text as string arguments.

You can also customize the font size of the title and labels using the `fontsize` parameter. For example,

```
import matplotlib.pyplot as plt
import numpy as np
# Create an array of x values from 0 to 2*pi with a step size of 0.1
x = np.linspace(-2*np.pi, 2*np.pi, 100)

# Calculate y values for the sin function
y = np.sin(x)

# Plot the sine function
plt.plot(x, y,label="line plot of sin(x) function")

# Calculate y values for the sin function
y = np.cos(x)

# Plot the cosine function
plt.plot(x, y,label="line plot of cosin(x) function")
```

```
# Add the legend to the plot
plt.legend()
#Add a title
plt.title("sin(x) vs cosin(x) plots")
# Add lable to x axis and y axis
plt.xlabel("x values", fontsize=16)
plt.ylabel("y values", fontsize=16)

# Diplay the result
plt.show()
```



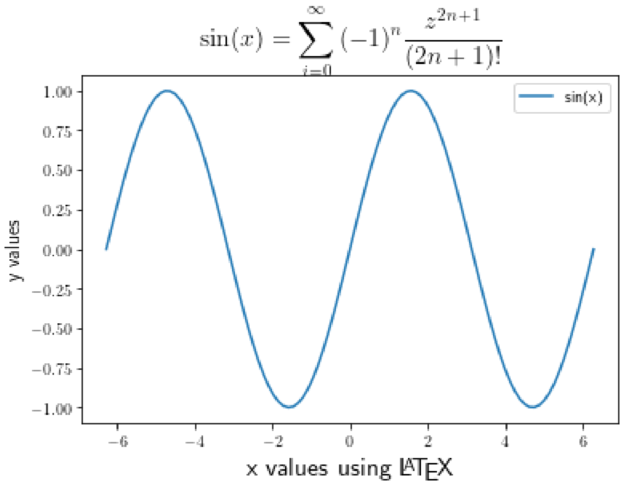
Using LATEX in pyplot You can use LaTeX syntax in your Matplotlib plot labels by enclosing the LaTeX code within dollar signs ($ $) in the label string.

However, you need to enable this option in Matplotlib's *"rc settings"* by calling `plt.rc('text', usetex=True)`.

Make sure you have LaTeX installed on your system to use LaTeX in your plot labels.

To prevent Python from escaping any characters, use raw strings (`r'xxx'`). For example,

```
import matplotlib.pyplot as plt
import numpy as np
# Create an array of x values from 0 to 2*pi with a step size of 0.1
x = np.linspace(-2*np.pi, 2*np.pi, 100)

# Calculate y values for the sin function
y = np.sin(x)

# Plot the sine function
plt.plot(x, y,label="sin(x)")
#Activate LaTeX
plt.rc('text', usetex=True)
# Add the legend to the plot
plt.legend()
#Add a title
plt.title(r"$\sin(x)=\displaystyle\sum_{i=0}^{\infty}{(-1)}^n\frac{z^{2n+1}}{{(2n+1)}!}$",fontsize=
# Add lable to x axis and y axis
plt.xlabel(r"x values using \LaTeX", fontsize=16)
plt.ylabel(r"y values ", fontsize=12)

# Diplay the result
plt.show()
```



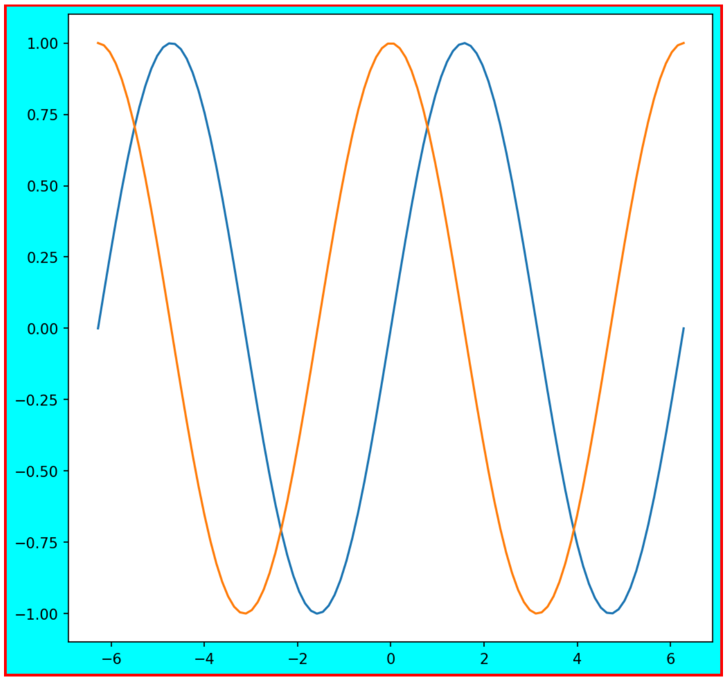
6.2 Customizing Plots

Customize figure `plt.figure()` can take several arguments to customize the created figure. Here are some of the common arguments:

Argument	*Description*
`num`	An identifier for the figure - if none is provided, an integer, starting at 1, is used and incremented with each figure created; alternatively, using a string will set the window title to that string when the figure is displayed with plt.show()
`figsize`	A tuple of figure (width, height), in inches
`dpi`	Figure resolution in dots per inch
`facecolor`	Figure background color
`edgecolor`	Figure border color

For example:

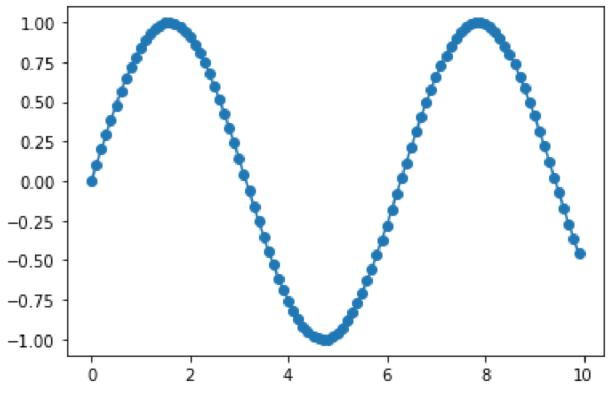
```
import matplotlib.pyplot as plt
import numpy as np
fig=plt.figure(figsize=(8,8),linewidth=3,dpi=150,edgecolor="red",facecolor='cyan')
x = np.linspace(-2*np.pi, 2*np.pi, 100)
plt.plot(x,np.sin(x))
plt.plot(x,np.cos(x))
plt.show()
```



Markers To customize markers in a plot using matplotlib library in Python, you can use the marker parameter of the `plot()` function. By default, the `plot()` function creates a line plot without any markers on the data points.

To add a marker on each point of the plotted data, you can set the marker parameter to the desired symbol. Here's an example code that plots a simple line graph with circle markers:

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

# Generate some sample data
x = np.arange(0, 10, 0.1)
y = np.sin(x)

# Create a plot with circle markers
plt.plot(x, y, marker='o')

# Show the plot
plt.show()
```



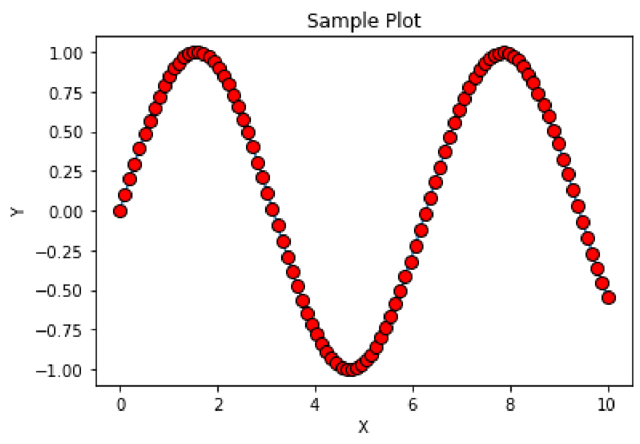
Other commonly used markers in the following table

Code	*Marker*	*Description*
.	$\cdot$	*Point*
o	$\circ$	*Circle*
$+$	$+$	*Plus*
x	$\times$	*Cross*
D	$\diamond$	*Diamond*
v	$\bigtriangledown$	Downward triangle
s	$\square$	*Square*
$*$	$\star$	*Star*

Here are some of the most commonly used marker properties:

- **markersize (abbreviated ms)**: Specifies the size of the marker, in points.
- **markevery**: Specifies how often a marker is shown on the plot. Set to a positive integer N to print a marker every N points; the default, None, prints a marker for every point.
- **markerfacecolor (abbreviated mfc)**: Specifies the fill color of the marker.
- **markeredgecolor (abbreviated mec)**: Specifies the edge color of the marker.
- **markeredgewidth (abbreviated mew)**: Specifies the width of the marker edge, in points. For example:

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

# Generate some sample data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Plot the data with markers
plt.plot(x, y, marker='o', markersize=8, markerfacecolor='red', markeredgecolor='black', markeredge

# Add some labels and a title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Sample Plot')

# Show the plot
plt.show()
```



Colors In Matplotlib, you can customize the color of plotted lines using the `color` parameter. There are several ways to specify a color:

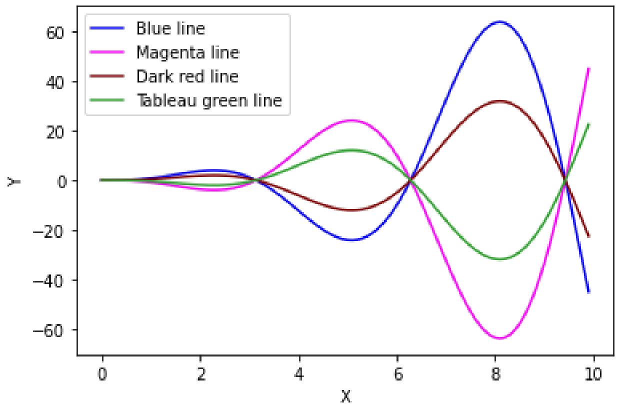
1. One-letter codes: You can use one-letter codes to represent some common colors, as shown in the table above. For example, color='r' specifies a red line and markers.
2. Tableau color sequence: Since Matplotlib 2.0, the default color sequence for a series of lines on the same plot is the more pleasing "Tableau" sequence. The string identifiers for Tableau colors are also given in the table above.
3. Shades of gray: You can specify shades of gray using a string representing a float in the range 0–1. Here, 0 represents black and 1 represents white.
4. HTML hex strings: You can use HTML hex strings to represent colors using their red, green, and blue (RGB) components in the range 00–ff. For example, color='#ff00ff' is magenta.
5. RGB tuples: You can also pass RGB components as a tuple of three values in the range 0–1. For example, color=(0.5, 0., 0.) represents a dark red color.

Basic color codes	Tableau colors
$b = blue$	$tab : blue$
$g = green$	$tab : orange$
$r = red$	$tab : green$
$c = cyan$	$tab : red$
$m = magenta$	$tab : purple$
$y = yellow$	$tab : brown$
$k = black$	$tab : pink$
$w = white$	$tab : gray$
	$tab : olive$
	$tab : cyan$

Here's an example code that shows how to use different color formats:

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

# Generate some sample data
x = np.arange(0, 10, 0.1)
y = np.sin(x)
```

```
# Create a plot with different colors
fig, ax = plt.subplots()
ax.plot(x, x**2*y, color='b', label='Blue line')
ax.plot(x, -x**2*y, color='#ff00ff', label='Magenta line')
ax.plot(x, 0.5*x**2 * y, color=(0.5, 0., 0.), label='Dark red line')
ax.plot(x, -0.5 *x**2 * y, color='tab:green', label='Tableau green line')

# Set axis labels and legend
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.legend()

# Show the plot
plt.show()
```



Line Styles and Widths Matplotlib provides several different line styles for plotting data. The line style can be specified using the linestyle argument of the `plot()` function. The default line style is a solid line, but there are several other styles available, such as dashed, dotted, and dash-dot.

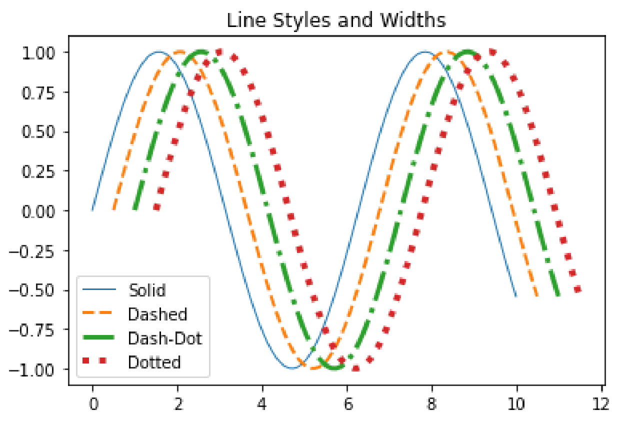
In addition to line styles, you can also specify the width of the line using the linewidth argument.

Here's an example of how to use line styles and widths in Matplotlib:

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

# Generate some sample data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Plot the data with different line styles and widths
plt.plot(x, y, linestyle='-', linewidth=1, label='Solid')
plt.plot(x + 0.5, y, linestyle='--', linewidth=2, label='Dashed')
plt.plot(x + 1, y, linestyle='-.', linewidth=3, label='Dash-Dot')
plt.plot(x + 1.5, y, linestyle=':', linewidth=4, label='Dotted')

# Add a legend and a title
```

```
plt.legend()
plt.title('Line Styles and Widths')

# Show the plot
plt.show()
```

This will produce a plot with four different line styles and widths, as shown below:



Here is a table of some of the most commonly used line styles in Matplotlib:

Code	Line style description
′—′	Solid line
′ — —′	Dashed line
′ — .′	Dash-dot line
′ :′	Dotted line

Plot Limits When you plot data in Matplotlib, the plot will automatically adjust the axis limits to fit the data. However, sometimes you may want to set the axis limits yourself to focus on a particular range of the data.

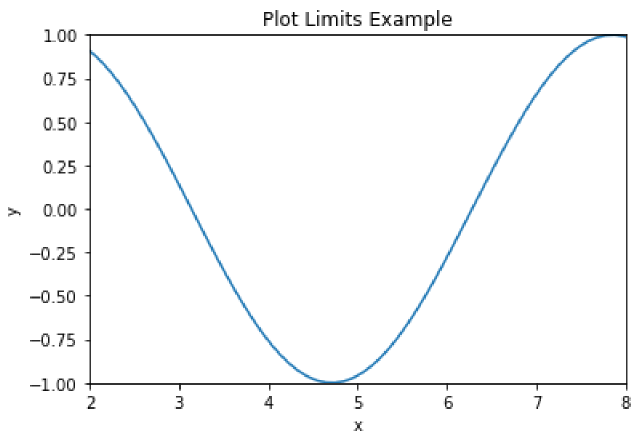
You can set the limits of the x-axis and y-axis using the `xlim()` and `ylim()` functions, respectively. Both functions take two arguments, the lower and upper bounds of the axis limits.

Here's an example of how to set plot limits in Matplotlib:

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

# Generate some sample data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Plot the data
plt.plot(x, y)

# Set the x-axis limits to [2, 8]
plt.xlim(2, 8)
```

```
# Set the y-axis limits to [-1, 1]
plt.ylim(-1, 1)

# Add a title and axis labels
plt.title('Plot Limits Example')
plt.xlabel('x')
plt.ylabel('y')

# Show the plot
plt.show()
```

This will produce a plot with x-axis limits of [2, 8] and y-axis limits of [-1, 1], as shown below:



Tick marks When you plot data in Matplotlib, the plot will automatically generate tick marks on the x-axis and y-axis to help label the plot. These tick marks are the small lines or marks on the axes that indicate the value at a particular location.

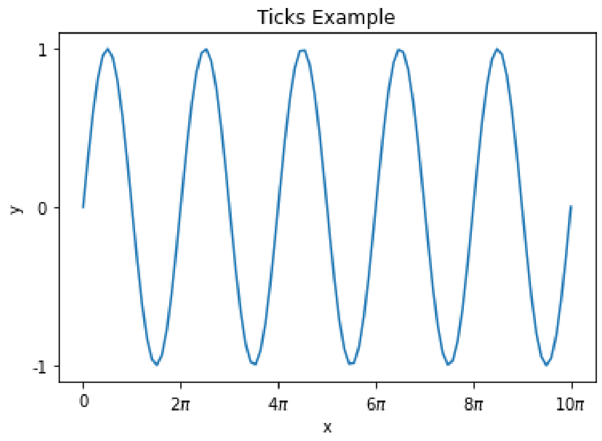
You can customize the tick marks using various functions in Matplotlib. The `xticks()` and `yticks()` functions allow you to set the locations and labels of the tick marks on the x-axis and y-axis, respectively.

Here's an example of how to customize tick marks in Matplotlib:

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

# Generate some sample data
x = np.linspace(0, 10*np.pi, 100)
y = np.sin(x)

# Plot the data
plt.plot(x, y)

# Set the x-axis tick marks to be [0, 2*pi, 4*pi, 6*pi, 8*pi, 10*pi]
plt.xticks([0, 2*np.pi, 4*np.pi, 6*np.pi, 8*np.pi, 10*np.pi], ['0', r'$2\pi$', r'$4\pi$', r'$6\pi$

# Set the y-axis tick marks to be [-1, 0, 1]
plt.yticks([-1, 0, 1], ['-1', '0', '1'])
```

```
# Add a title and axis labels
plt.title('Ticks Example')
plt.xlabel('x')
plt.ylabel('y')

# Show the plot
plt.show()
```

This will produce a plot with custom tick marks on both the x-axis and y-axis, as shown below:



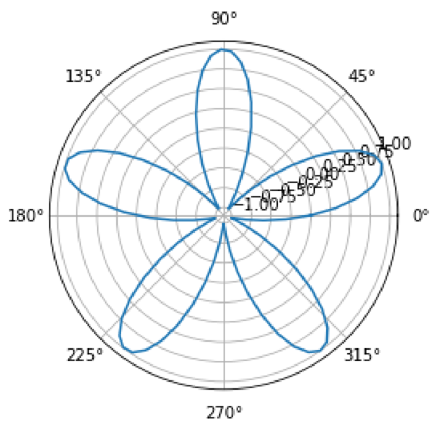
7 More Advanced Plotting

7.1 Polar Plots

Polar plots are a type of plot where data is plotted on a polar coordinate system instead of a Cartesian coordinate system. Polar plots are useful for visualizing data that is cyclical in nature. To create a polar plot, you can use the `plt.polar()` function. For example:

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

theta = np.linspace(0, 2*np.pi, 100)
r = np.sin(5*theta)

plt.polar(theta, r)
plt.show()
```

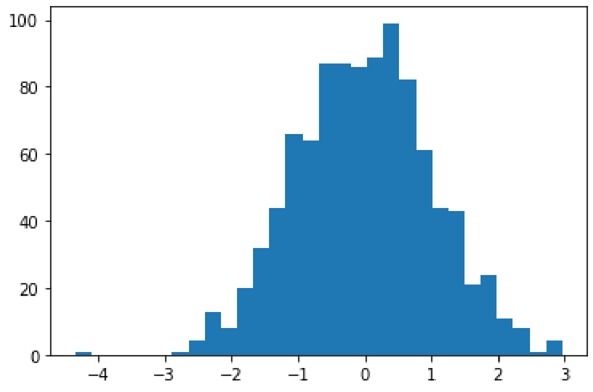
This will create a polar plot of a sine wave with five complete cycles:



7.2 Histograms

Histograms are a way to visualize the distribution of a set of data. They work by dividing the data into a set of intervals, or "bins," and then counting the number of data points that fall into each bin. To create a histogram in Matplotlib, you can use the `plt.hist()` function. For example:

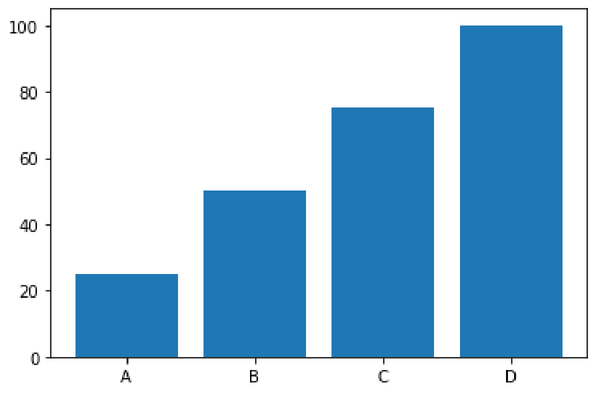
```
import matplotlib.pyplot as plt
import numpy as np

data = np.random.normal(0, 1, 1000)

plt.hist(data, bins=30)
plt.show()
```



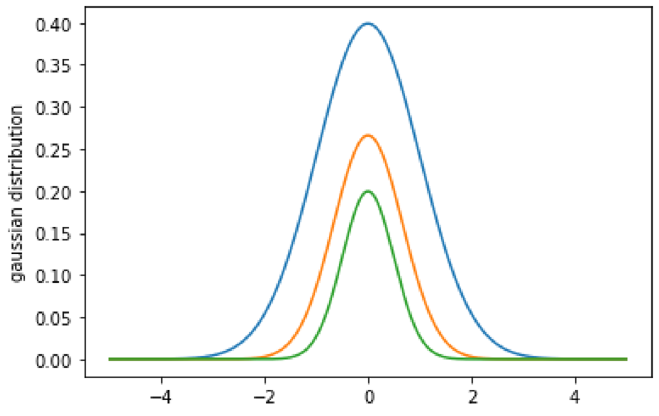
7.3 Bar Plot

Bar plots are used to compare different categories of data. They work by plotting bars of different heights to represent the values associated with each category. To create a bar plot in Matplotlib, you can use the plt.bar() function. For example:

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

data = [25, 50, 75, 100]

plt.bar(range(len(data)), data)
plt.xticks(range(len(data)), ['A', 'B', 'C', 'D'])
plt.show()
```

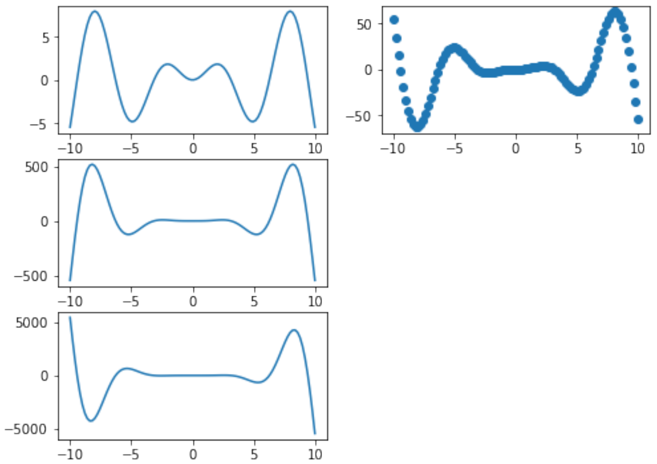


[12]:
```
#https://home.agh.edu.pl/~mariuszp/wfiis_stw/
 ↪Learning_Scientific_Programming_with_Python_2ed_Ch_Hill_Cambridge_2020.pdf
import numpy as np
from matplotlib import pyplot as plt
mean = 0; std = np.array([1,1.5,2]); variance = np.square(std[0])
x = np.arange(-5,5,.01)
f1 = np.exp(-np.square(x-mean)/2*variance)/(np.sqrt(2*np.pi*variance))
variance = np.square(std[1])
f2= np.exp(-np.square(x-mean)/2*variance)/(np.sqrt(2*np.pi*variance))
variance = np.square(std[2])
f3= np.exp(-np.square(x-mean)/2*variance)/(np.sqrt(2*np.pi*variance))
plt.plot(x,f1)
plt.plot(x,f2)
plt.plot(x,f3)
plt.ylabel('gaussian distribution')
plt.show()
```



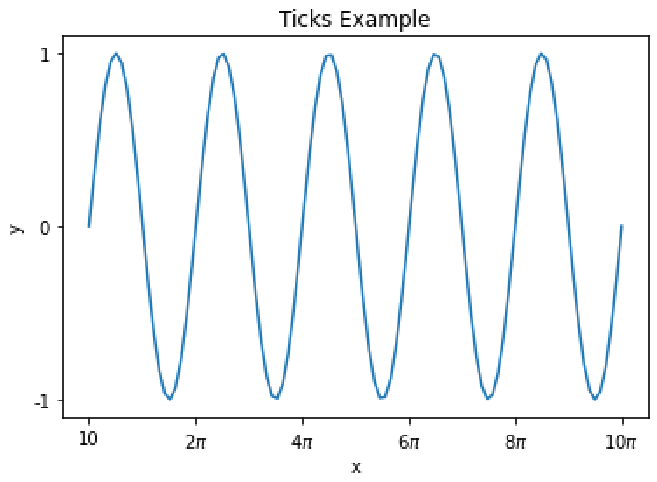
[3]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
fig=plt.figure(figsize=(8,6))
x=np.linspace(-10,10,100)

ax1 = fig.add_subplot(3, 2, 1)
ax2 = fig.add_subplot(3, 2, 2)
ax3 = fig.add_subplot(3, 2, 3)
ax4 = fig.add_subplot(3, 2, 5)


ax1.plot(x,np.power(x,1.)*np.sin(x))
ax2.scatter(x,np.power(x,2.)*np.sin(x))
ax3.plot(x,np.power(x,3.)*np.sin(x))
ax4.plot(x,np.power(x,4.)*np.sin(x))
plt.show()
```



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

      # Generate some sample data
      x = np.linspace(0, 10*np.pi, 100)
      y = np.sin(x)

      # Plot the data
      plt.plot(x, y)

      # Set the x-axis tick marks to be [0, 2*pi, 4*pi, 6*pi, 8*pi, 10*pi]
      plt.xticks([0, 2*np.pi, 4*np.pi, 6*np.pi, 8*np.pi, 10*np.pi], ['10', r'$2\pi$',␣
       ↪r'$4\pi$', r'$6\pi$', r'$8\pi$', r'$10\pi$'])

      # Set the y-axis tick marks to be [-1, 0, 1]
      plt.yticks([-1, 0, 1], ['-1', '0', '1'])

      # Add a title and axis labels
```

```
plt.title('Ticks Example')
plt.xlabel('x')
plt.ylabel('y')

# Show the plot
plt.show()
```



[1]:
```
import matplotlib.pyplot as plt
import numpy as np
fig=plt.figure(figsize=(8,6))
x=np.linspace(-10,10,100)

ax1 = fig.add_subplot(3, 2, 1)
ax2 = fig.add_subplot(3, 2, 2)
ax3 = fig.add_subplot(3, 2, 3)
ax4 = fig.add_subplot(3, 2, 4)


ax1.plot(x,np.power(x,1.)*np.sin(x))
ax2.plot(x,np.power(x,2.)*np.sin(x))
ax3.plot(x,np.power(x,3.)*np.sin(x))
ax4.plot(x,np.power(x,4.)*np.sin(x))
plt.show()
```