Mat Plot Lib

# Part 1

* To show your plots inline in Jupyter notebook, use %matplotlib inline. In other IDEs, type plt.show() after every plotting to print the plot.
* Import pyplot from matplotlib with from matplotlib import pyplot as plt
* There are 2 ways to create a matplotlib plot: function and object-oriented methods respectively

# Functional method

* You can use plt.plot(x, y) to create a plot using functional method. x and y are the values on the x and y axis respectively.
* You can use plt.xlabel(a), , plt.ylabel(a) and plt.title(a) to add a label for the x axis, y axis and title respectively. ‘a’ is the string of the label or title as applicable.
* You can create multiple plots (also known as “multiplots’) using plt.subplot(a, b, c) as many times as necessary where a, b and c are the number of rows of plots, number of columns of plots and the position of the present plot respectively.

# Object oriented method

* The main idea is to create an empty figure and then call methods off it. The figure is a blank canvas
* You can create the figure object with fig\_object\_name = plt.figure()
* You can define the axis with axes = fig\_object\_name.add\_axes([a, b, c, d]) where a, b, c, and d are bottom, left, width and height respectively. Note: a, b, c and d must be between 0 and 1 because they indicate what percent of the figure the axes should be on.

a and b are starting position in the x and y axes while c and d are the width and height of the figure respectively.

* You can pass the data with axes.plot(x,y) where x and y are the data points
* You can add x axes label, y axes label and title with axes.set\_xlabel(‘ Xlabel’), axes.set\_ylabel(‘ Ylabel’) and axes.set\_title(‘ Title’) respectively.
* You can create multiple axes in the figure using multiple plots and call method on the respective axes.
* Create an empty figure, add axes, plot axes, customize.

Note: you can draw more than one axes on this figure by simply specifying different size and positioning.

# Part 2

* You can use the object oriented method to create subplots by using fig, axes = plt.subplots(nrows=a, ncolumns=b) where ‘fig, axes’ is a tuple, ‘nrows’ is number of rows and ‘ncols’ is number of columns.
* Note that axes is a list of matplotlib axes object that can be iterated. This is why a ‘for loop’ can be used to iterate and plot each one.
* You can automatically adjust overlapping layouts with plt.tight\_layout()
* You can iterate through the axes list by indexing e.g., axes[0].plot(x, y) or axes[a,b] for 1d and 2d (nrows > 1) shaped plots respectively.
* You can also call title and labels off each of the axes object.

Note: 1 row subplots are 1d arrays while higher rows are 2d arrays.

# Figure size, aspect ratio and dpi

* You can customize both the figure size and dpi while creating the figure and subplots respectively by passing ‘figsize’ and ‘dpi’ as arguments e.g., plt.subplots(nrows, ncols, figsize, dpi) or plt.figure(figsize, dpi) where nrows and ncols are integers, figsize is a tuple(in inches) and dpi is an integer.
* The figsize tuple is the width, height in a tuple.
* Create subplots, plot each, customize.

# Saving a figure as an image

* You can use matplotlib to save figures as png, img, jpg, svg etc.
* You save with fig.savefig(a, b) where a is the file name containing the desired image file type(e.g., jpg) and b is dpi. 'fig’ is the name of the figure you are saving.

# Legends

* To use legends, you must put labels in the plot instructions while plotting i.e. axes.plot(x, y, labels=’z’) where z is the legend label.
* You can call legends with axes.legend()
* You can position legends by passing a location in the legend call i.e. axes.legend(loc=t) where t is the preferred location which can be best, upper right etc. preferably, always use best with location code, 0.
* You can also specify loc with tuples indicating just like the locating of axes.

# Part 3 - customizations

* You can customize the line color, width etc. when plotting the graph e.g., axes.plot(x, y, color=a, linewidth = b, alpha = c) where a is color in string or RGB code, linewidth in number, and alpha which is the level of line transparency in number as well.

Notes: the lower the alpha value, the more transparent the line.

* You can also pass in line style (ls) with the linsestyle argument e.g., axes.plot(x, y, color=a, linewidth = b, alpha = c, linestyle = ‘d’) where d can be dash(--), default(-), (-.), (:) etc.
* You can add in a marker argument to mark out the specific data points used to create the line or curve e.g., axes.plot(x, y, color=a, linewidth = b, alpha = c, linestyle = ‘d’, marker=’o’). there are a variety of markers that can be used such as 0, +, \* etc.
* You can also customize the marker appearance such as marker face color, edge color, edge width, size etc. with makerfacecolor, markeredgecolor, markersize, markeredgewidth, etc. e.g., axes.plot(x, y, color=a, linewidth = b, alpha = c, linestyle = ‘d’, marker=’o’, makersize = ‘e’, makerfacecolor = ’f’, markeredgecolor = ‘g’, markeredgewidth = ‘h’).
* Marker face color and marker edge color (can be filled as strings, codes etc) colors the inside and outline of the marker respectively.
* Marker size (in number) describes the size of the marker
* Marker edge width describes the width of the marker’s edge in numbers.
* You can set the plot limit for the x and y axis to control the area to be plotted with ax.set\_xlim([a, b]) and ax.set\_ylim([c, d]) respectively. A and b are a tuple of the plot area in the x axis while c and are the tuple of the plot area in the y axis.
* In summary, the most common customizations are: color, lw, ls, alpha, marker, marker size, marker face color, marker edge color, marker edge width, plot limits for both x and y axes.