PYPLOT TUTORIAL

http://matplotlib.org/users/pyplot_tutorial.html

Introducing Matplotlib

- Matplotlib is a plotting tool in Python
- matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB.
- Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.
- In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes
  - please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis.

Introducing Matplotlib

- **Importing matplotlib**
  - import matplotlib as mpl
  - import matplotlib.pyplot as plt
    - The plt interface is what we will use most often
- **Plotting from an Jupyter notebook**
  - %matplotlib inline
- **Plotting from a script (.py file)**
  - plt.show()

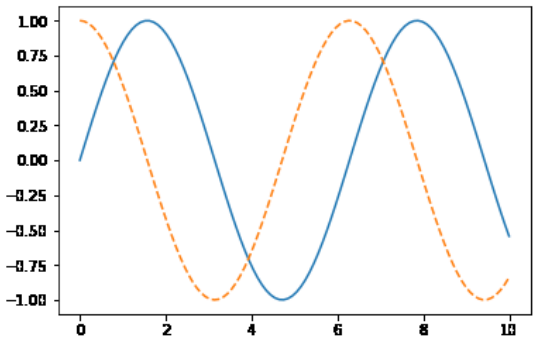
Figures and axes

- The ***figure*** (an instance of the class plt.Figure) can be thought of as a single container
  - contains all the objects representing axes, graphics, text, and labels.
- The ***axes*** (an instance of the class plt.Axes) is what we see above: a bounding box with ticks and labels

Figures and axes

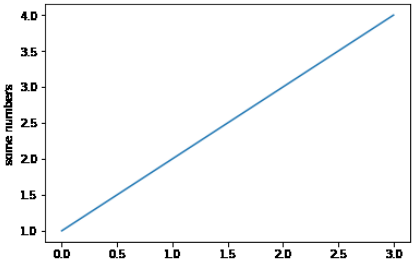
- Plots in matplotlib reside within a Figure object. You can create a new figure with **`plt.figure`**:

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
fig = plt.figure()
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--')
```



plot()


```
In [15]: plt.plot([1,2,3,4])
    ...: plt.ylabel('some numbers')
```

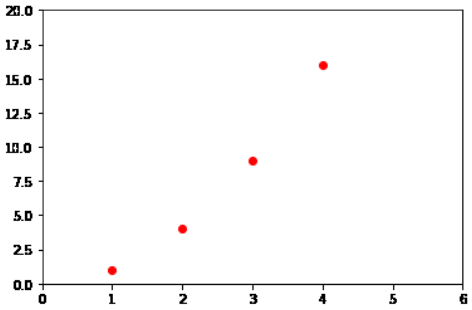


- If you provide a single list or array to the plot() command, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you.
- Since python ranges start with 0, the default x vector has the same length as y but starts with 0.
- Hence the x data are [0,1,2,3].
- `plt.plot` return value is a list of lines that were added.

plot()

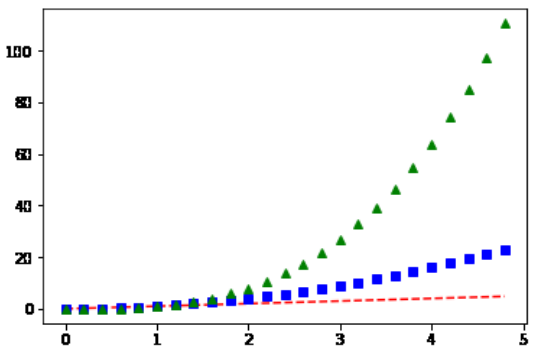

```
In [17]: plt.plot([1,2,3,4], [1,4,9,16], 'ro')
    ...: plt.axis([0, 6, 0, 20])
```

- For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot.
  - 'ro': red circle markers
- axis() command in the example above takes a list of [xmin, xmax, ymin, ymax] and specifies the viewport of the axes.



plot()

- you will use numpy arrays. In fact, all sequences are converted to numpy arrays internally.




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

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
```

Format string characters

Char	description	Char	description	Char	description
'-'	solid line style	'<'	triangle_left marker	'h'	hexagon1 marker
'--'	dashed line style	'>'	triangle_right marker	'H'	hexagon2 marker
'-.'	dash-dot line style	'1'	tri_down marker	'+'	plus marker
':'	dotted line style	'2'	tri_up marker	'x'	x marker
'.'	point marker	'3'	tri_left marker	'D'	diamond marker
','	pixel marker	'4'	tri_right marker	'd'	thin_diamond marker
'o'	circle marker	's'	square marker	'|'	vline marker
'v'	triangle_down marker	'p'	pentagon marker	'_'	hline marker
'^'	triangle_up marker	'*'	star marker		

Color abbreviations

character	color	character	color
‘b’	blue	‘m’	magenta
‘g’	green	‘y’	yellow
‘r’	red	‘k’	black
‘c’	cyan	‘w’	white

Controlling line properties

- Lines have many attributes that you can set: linewidth, dash style, antialiased, etc; see matplotlib.lines.Line2D.
- Use keyword args:
  - plt.plot(x, y, linewidth=2.0)
- Use the setter methods
  - line, = plt.plot(x, y, '-')
  - line.set_antialiased(False) # turn off antialising
- Use the setp() command.
  - lines = plt.plot(x1, y1, x2, y2)
  - # use keyword args
  - plt.setp(lines, color='r', linewidth=2.0)

Return value is a list of lines that were added.

Some Line2D properties

Property	Value Type
color or c	any matplotlib color
dash_capstyle	['butt' | 'round' | 'projecting']
dash_joinstyle	['miter' | 'round' | 'bevel']
dashes	sequence of on/off ink in points
label	any string
linestyle or ls	['-' | '--' | '-.' | ':' | 'steps' | ...]
linewidth or lw	float value in points
marker	['+' | ',' | '.' | '1' | '2' | '3' | '4']

Some Line2D properties

13

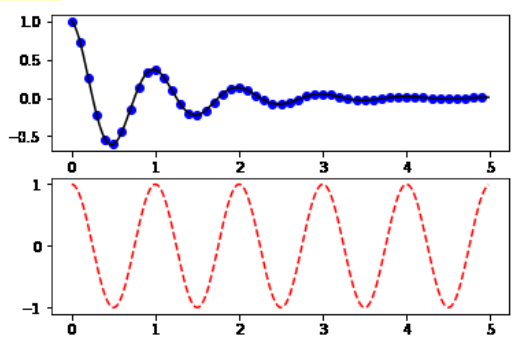
Property	Value Type
markersize or ms	float
solid_capstyle	['butt' | 'round' | 'projecting']
solid_joinstyle	['miter' | 'round' | 'bevel']
visible	[True | False]
xdata	np.array
ydata	np.array

http://matplotlib.org/api/lines_api.html#matplotlib.lines.Line2D

Multiple figures and axes

14

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

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)
plt.figure(1)
plt.subplot(211) #(rows,columns,panelNumber)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212) # identical to (2, 1, 2)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```

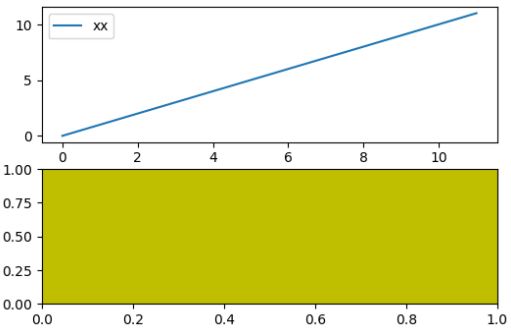


Multiple figures and axes

- matplotlib.pyplot.**subplot**(**args*, ***kwargs*)
  - Return a subplot axes positioned by the given grid definition.
  - subplot(nrows, ncols, plot_number)
    - *plot_number* starts at 1
    - In the case when *nrows*, *ncols* and *plot_number* are all less than 10, a convenience exists, such that the a 3 digit number can be given instead.
      - subplot(211)

Multiple figures and axes


```
In [18]: import matplotlib.pyplot as plt
   ...: %matplotlib inline
   ...: # plot a line, implicitly creating a subplot(111)
   ...: plt.plot([1,2,3])
   ...: # now create a subplot which represents the top plot of
          # a grid with 2 rows and 1 column. Since this subplot will
          # overlap the first, the plot (and its axes) previously
          # created, will be removed
   ...: ax = plt.subplot(211)
   ...: ax.plot(range(12), label='xx')
   ...: ax.legend()
   ...: plt.subplot(212, facecolor='y') #
```



Multiple figures and axes

- The figure() command here is optional because figure(1) will be created by default, just as a subplot(111) will be created by default if you don't manually specify any axes.
- Pyplot has the concept of the current figure and the current axes.
- All plotting commands apply to the current axes.
- plt.gcf()
  - Return the current figure
- plt.gca()
  - Return the current axes

Multiple figures and axes

- create multiple figures by using multiple figure() calls with an increasing figure number.

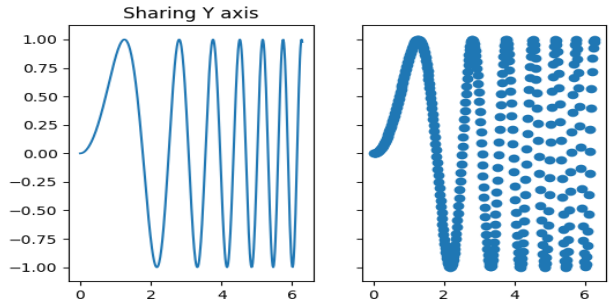
```
plt.figure(1)                # the first figure
plt.subplot(211)             # the first subplot in the first figure
plt.plot([1, 2, 3])
plt.subplot(212)             # the second subplot in the first figure
plt.plot([4, 5, 6])

plt.figure(2)                # a second figure
plt.plot([4, 5, 6])          # creates a subplot(111) by default

plt.figure(1)                # figure 1 current; subplot(212) still current
plt.subplot(211)             # make subplot(211) in figure1 current
plt.title('Easy as 1, 2, 3') # subplot 211 title
```

Multiple figures and axes

- matplotlib.pyplot.**subplots**(*nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None, **fig_kw*)
  - Create a figure and a set of subplots
  - This utility wrapper makes it convenient to create common layouts of subplots, including the enclosing figure object, in a single call.
- Returns
  - **fig** : matplotlib.figure.Figure object
  - **ax** : Axes object or array of Axes objects.
    - ax can be either a single matplotlib.axes.Axes object or an array of Axes objects if more than one subplot was created.

Multiple figures and axes


```
In [19]: x = np.linspace(0, 2*np.pi, 400)
In [20]: y = np.sin(x**2)
In [24]: f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
    ...: ax1.plot(x, y)
    ...: ax1.set_title('Sharing Y axis')
    ...: ax2.scatter(x, y)
```



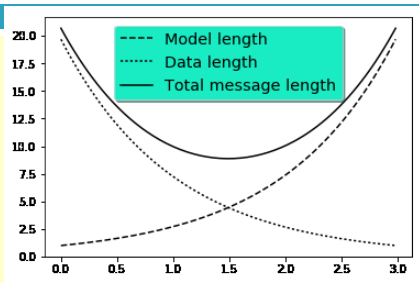
Multiple figures and axes

- matplotlib.pyplot.subplots_adjust(**args*, ***kwargs*)
  - subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None, hspace=None)

left = 0.125	the left side of the subplots of the figure
right = 0.9	the right side of the subplots of the figure
bottom = 0.1	the bottom of the subplots of the figure
top = 0.9	the top of the subplots of the figure
wspace = 0.2	the amount of width reserved for blank space between subplots, expressed as a fraction of the average axis width
hspace = 0.2	the amount of height reserved for white space between subplots, expressed as a fraction of the average axis height

Places a legend on the axes


```
a = b = np.arange(0, 3, .02)
c = np.exp(a)
d = c[::-1]
# Create plots with pre-defined labels.
fig, ax = plt.subplots()
ax.plot(a, c, 'k--', label='Model length')
ax.plot(a, d, 'k:', label='Data length')
ax.plot(a, c + d, 'k', label='Total message length')

legend = ax.legend(loc='upper center', shadow=True, fontsize='x-large')

# Put a nicer background color on the legend.
legend.get_frame().set_facecolor('#00FFCC')
```



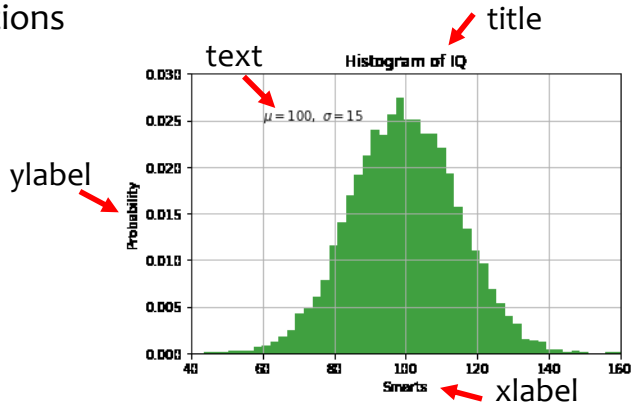
Places a legend on the axes

- Parameter loc: The location of the legend
  - int or string or pair of floats, default: 'upper right'

Location String	Location Code	Location String	Location Code
'best'	0	'center left'	6
'upper right'	1	'center right'	7
'upper left'	2	'lower center'	8
'lower left'	3	'upper center'	9
'lower right'	4	'center'	10
'right'	5		

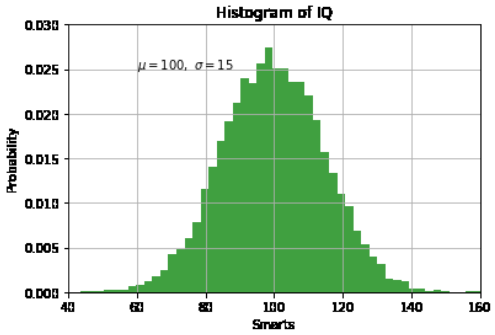
Working with text

- The text() command can be used to add text in an arbitrary location
  - The xlabel(), ylabel() and title() are used to add text in the indicated locations



Working with text


```
np.random.seed(19680801)
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
```



Text properties and layout

- you can customize the properties by passing keyword arguments into the text functions

```
t = plt.xlabel('my data', fontsize=14, color='red')
```

  - http://matplotlib.org/users/text_props.html#text-properties

Using mathematical expressions in text

- matplotlib accepts TeX equation expressions in any text expression.
- For example to write the expression $\sigma_i = 15$ in the title, you can write a TeX expression surrounded by dollar signs:

```
plt.title(r'$\sigma_i=15$')
```

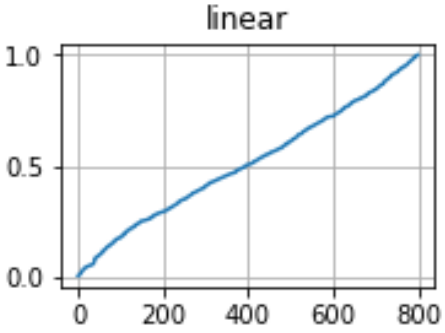
  - The r preceding the title string is important – it signifies that the string is a raw string and not to treat backslashes as python escapes.
- for details see Writing mathematical expressions.
  - http://matplotlib.org/users/mathtext.html#mathtext-tutorial

Logarithmic and other nonlinear axis

- matplotlib.pyplot supports not only linear axis scales, but also logarithmic and logit scales.
- This is commonly used if data spans many orders of magnitude. Changing the scale of an axis is easy:

```
plt.xscale('log')
```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter  # useful for `logit` scale

# Fixing random state for reproducibility
np.random.seed(19680801)
# make up some data in the interval ]0, 1[
y = np.random.normal(loc=0.5, scale=0.4, size=1000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))

# plot with various axes scales
plt.figure(1)
# linear
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)
```




```
# log
plt.plot(x, y)
plt.yscale('log')
plt.title('log')
plt.grid(True)

# symmetric log
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthreshy=0.01)
plt.title('symlog')
plt.grid(True)

# logit
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
```