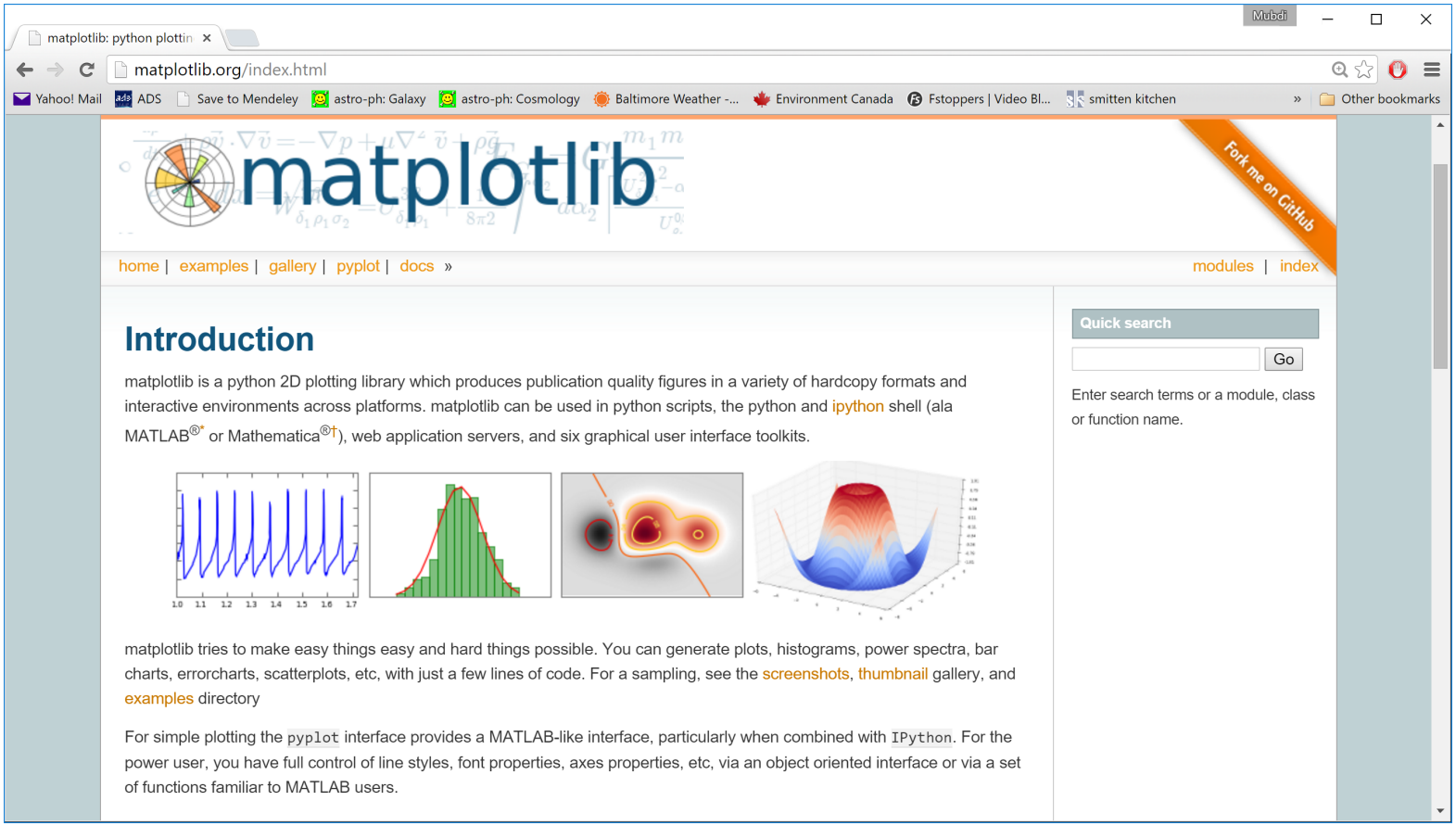
BASIC PLOTTING

DATA 601
Lecture 03
Part I

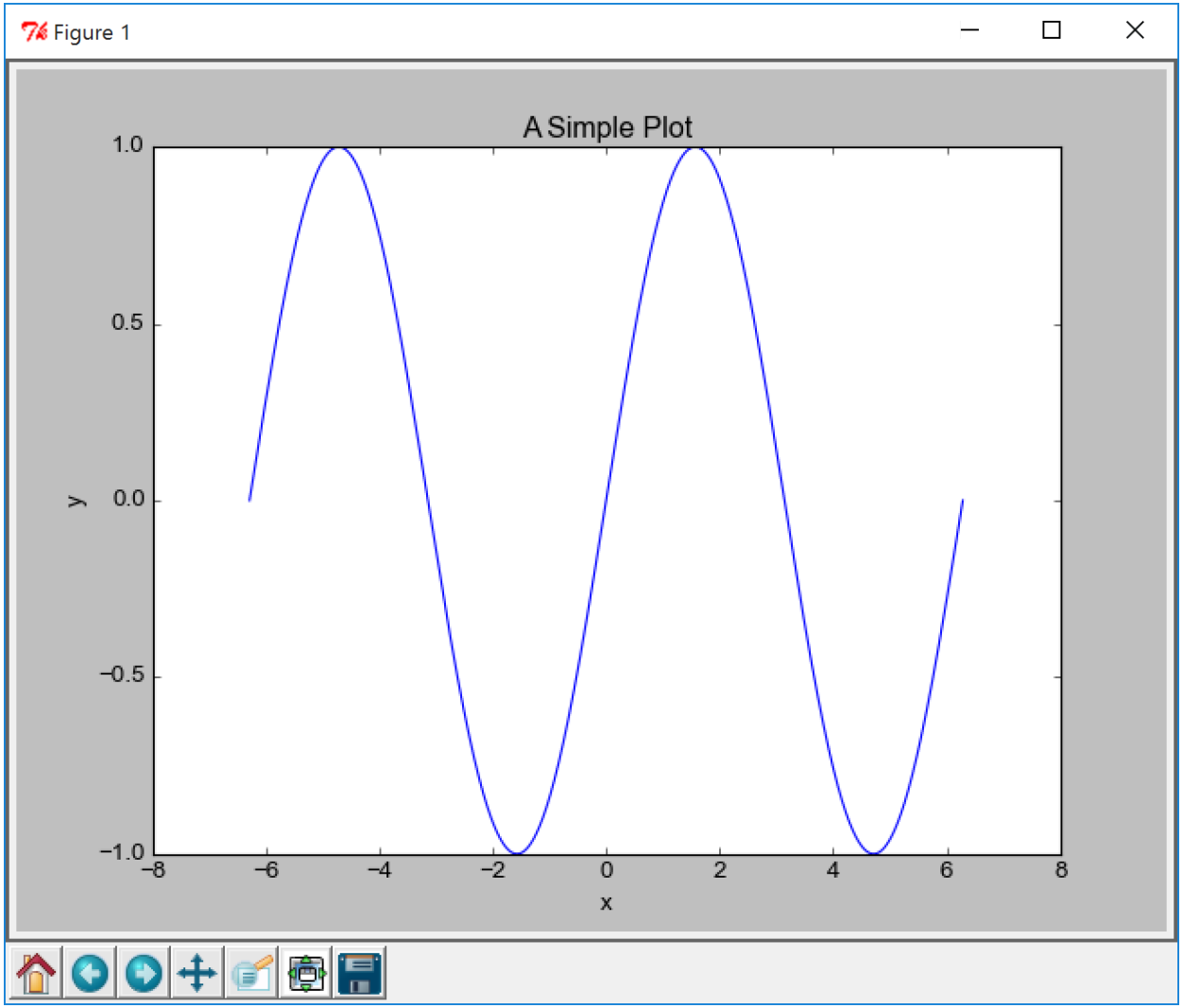
INTRODUCING MATPLOTLIB!



Very powerful plotting package.

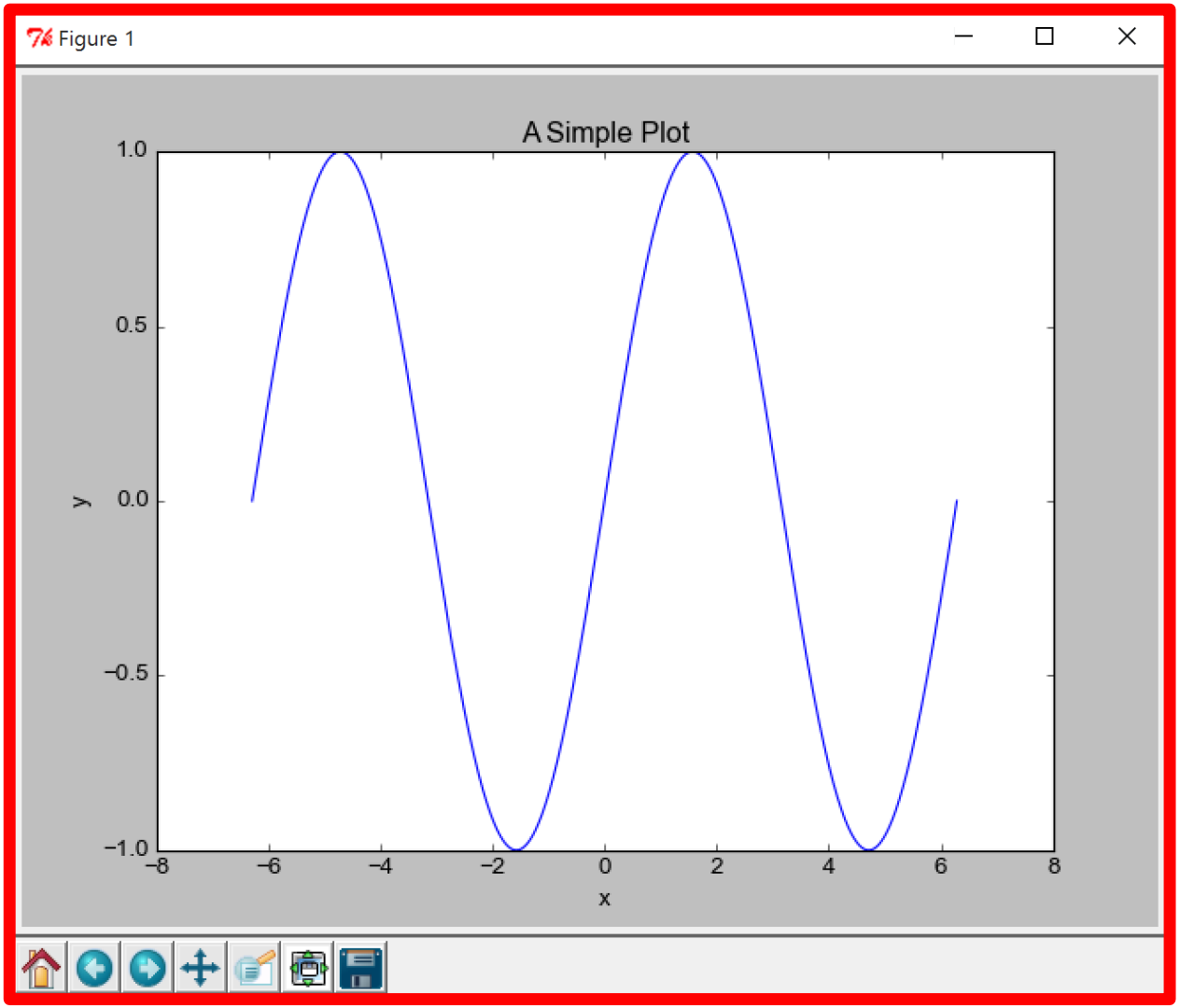
The Docs: http://matplotlib.org

ANATOMY OF A PLOT WINDOW



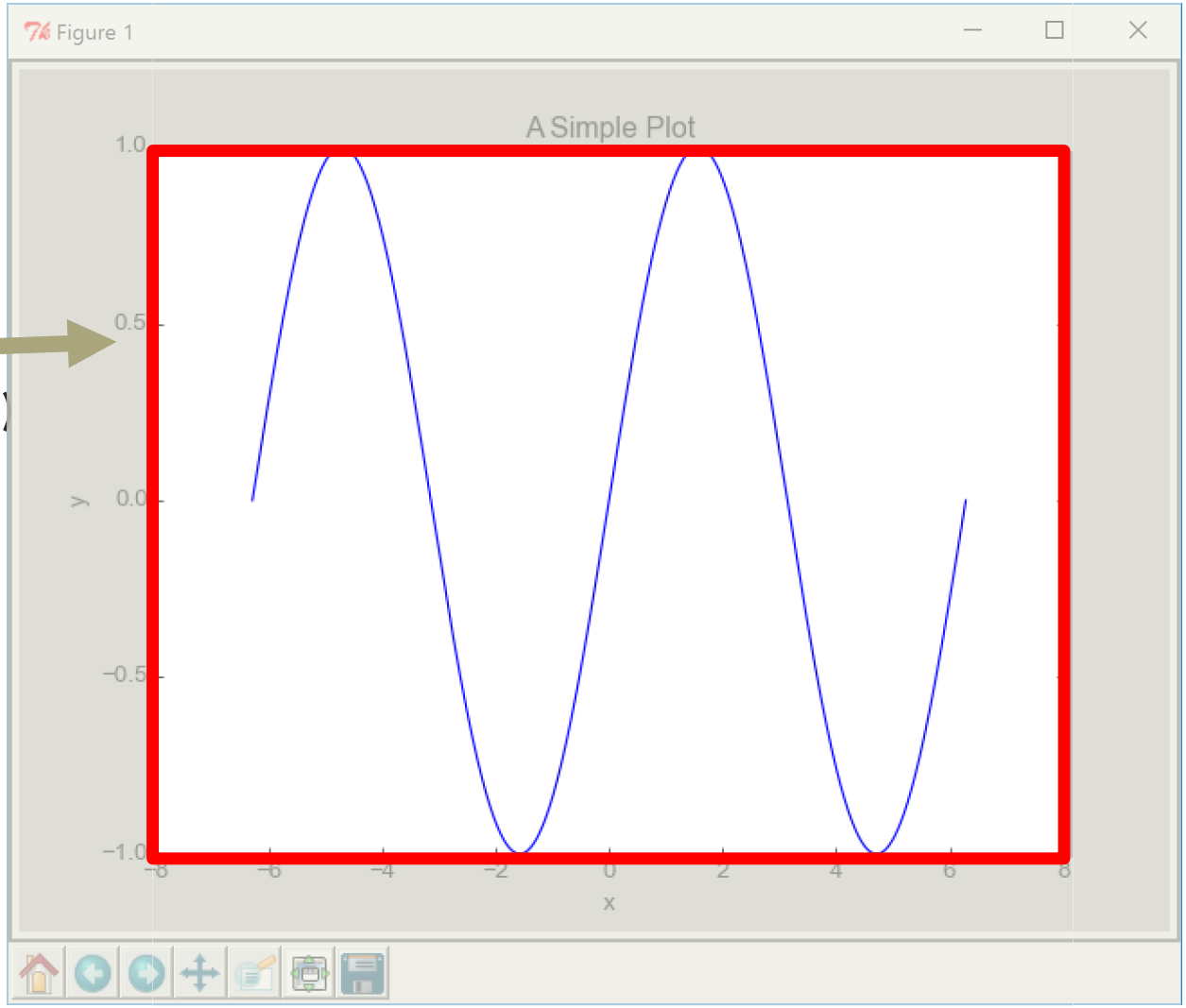
ANATOMY OF A PLOT WINDOW

Figure

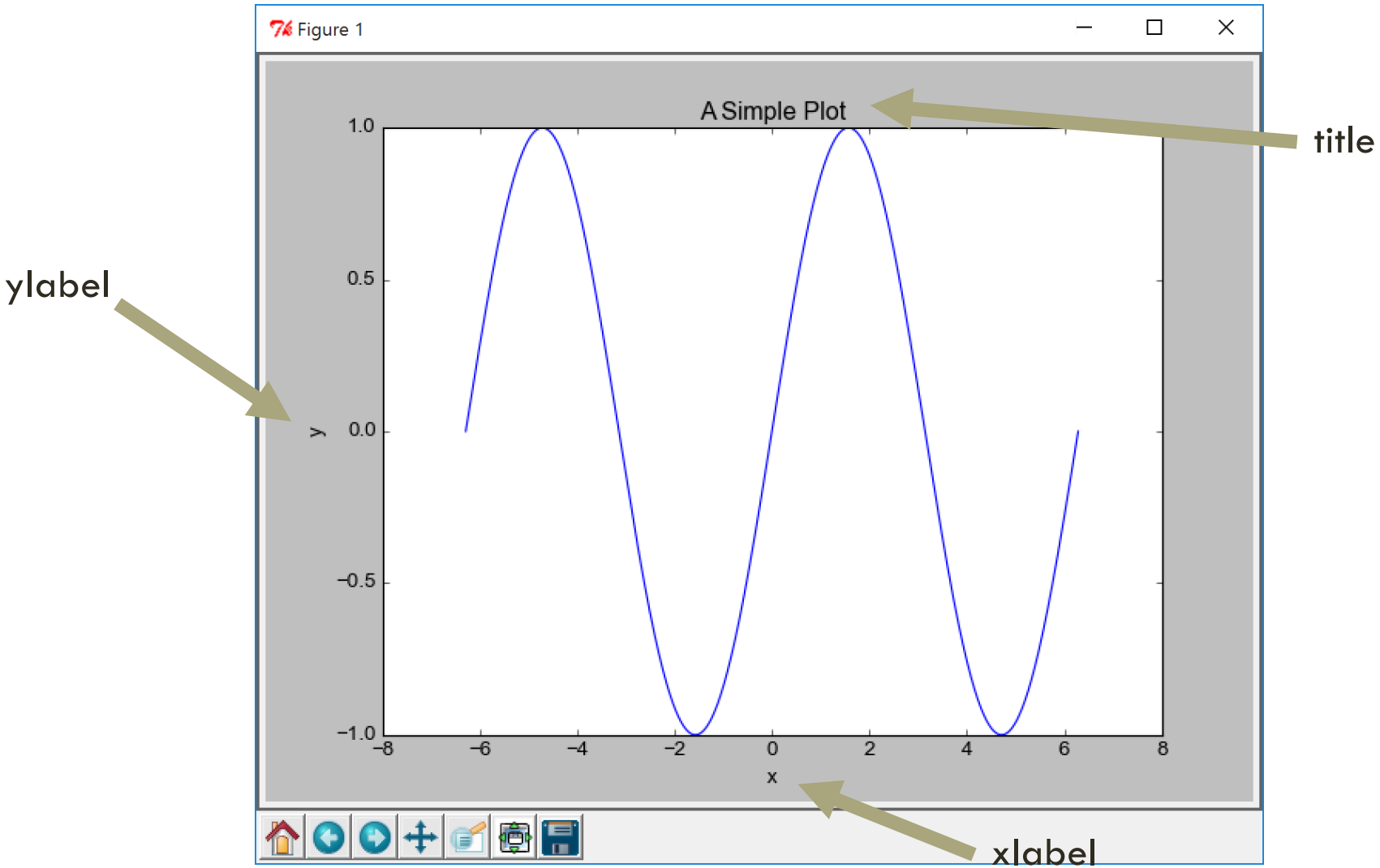


ANATOMY OF A PLOT WINDOW

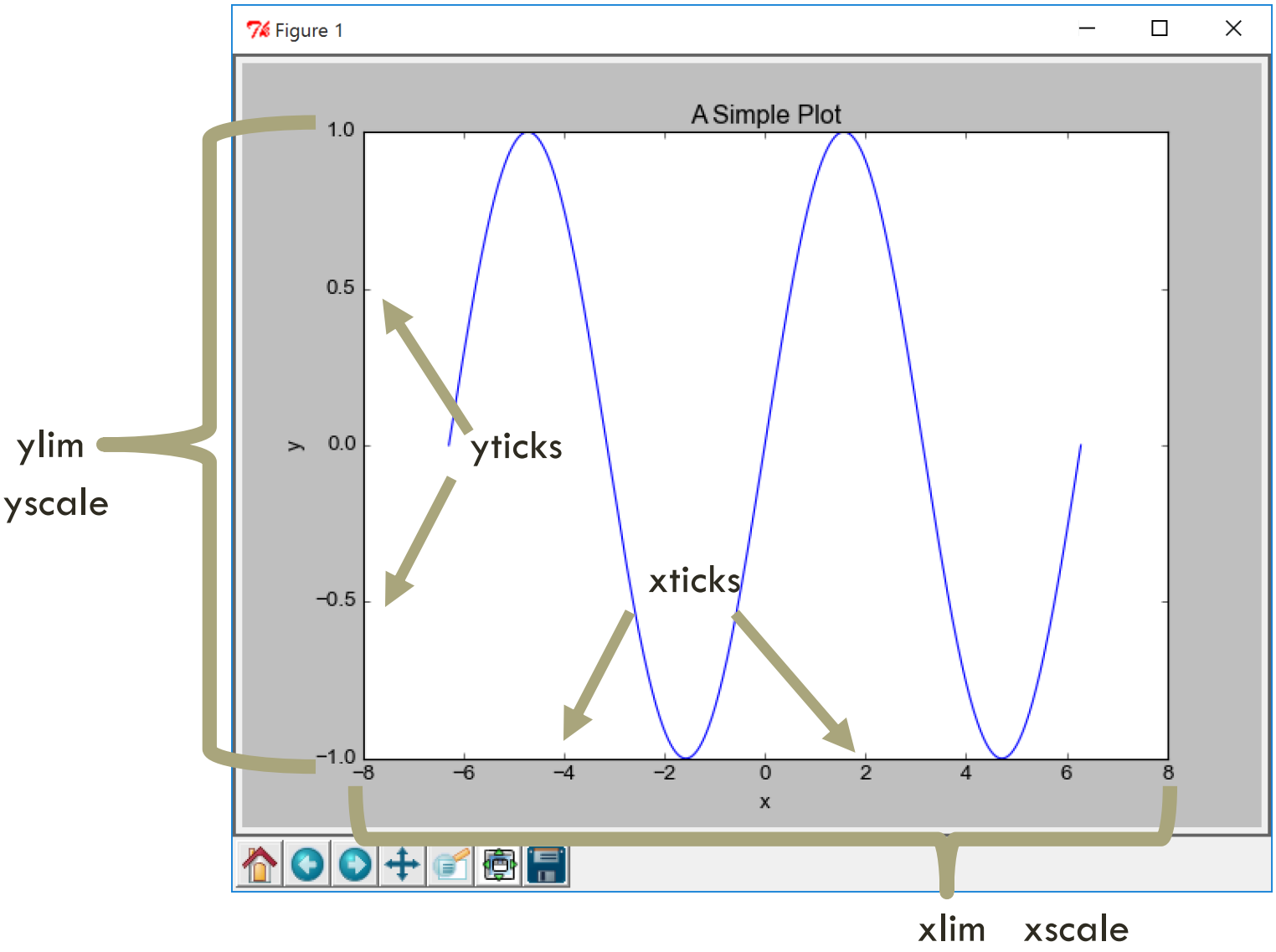
Axis
(confusingly...)

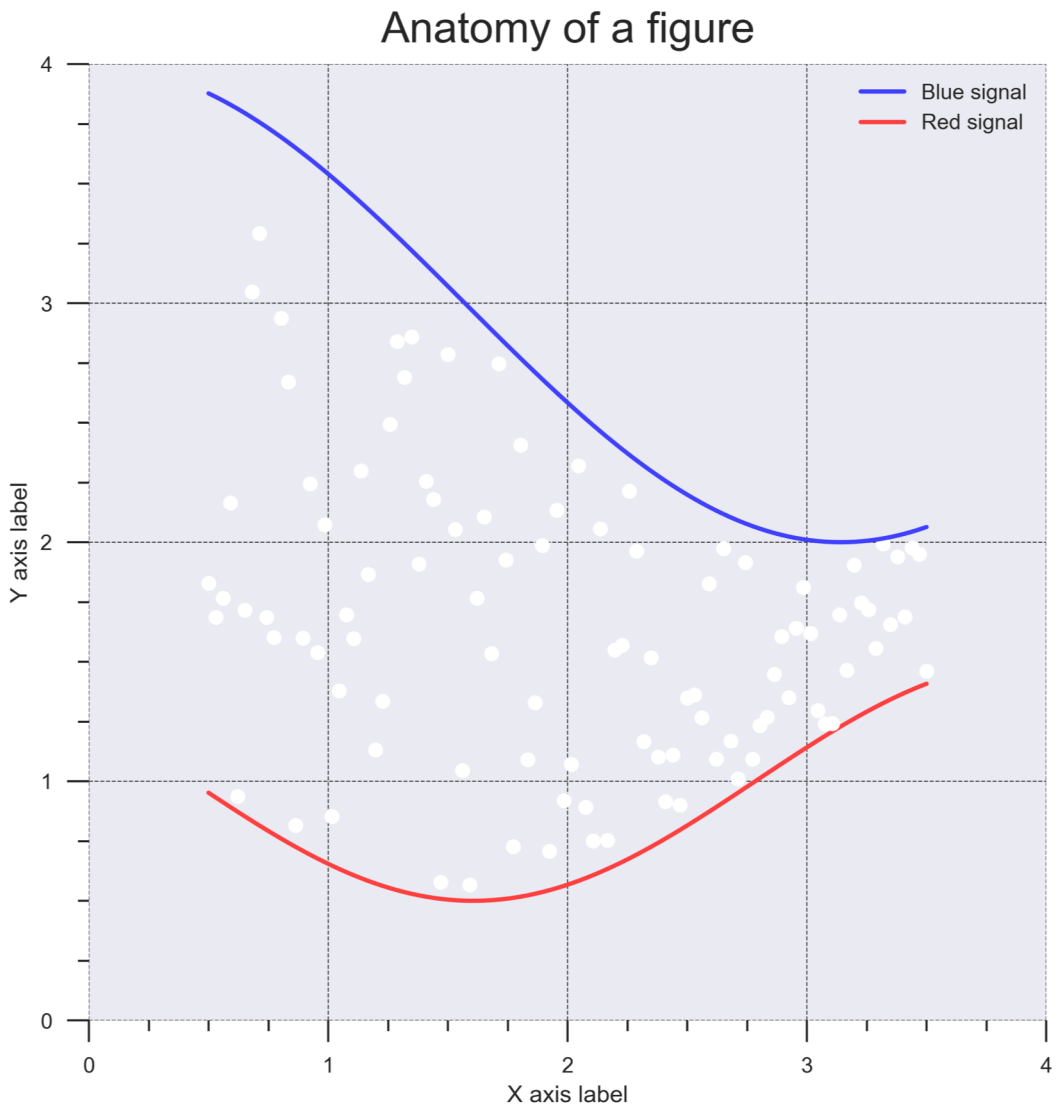


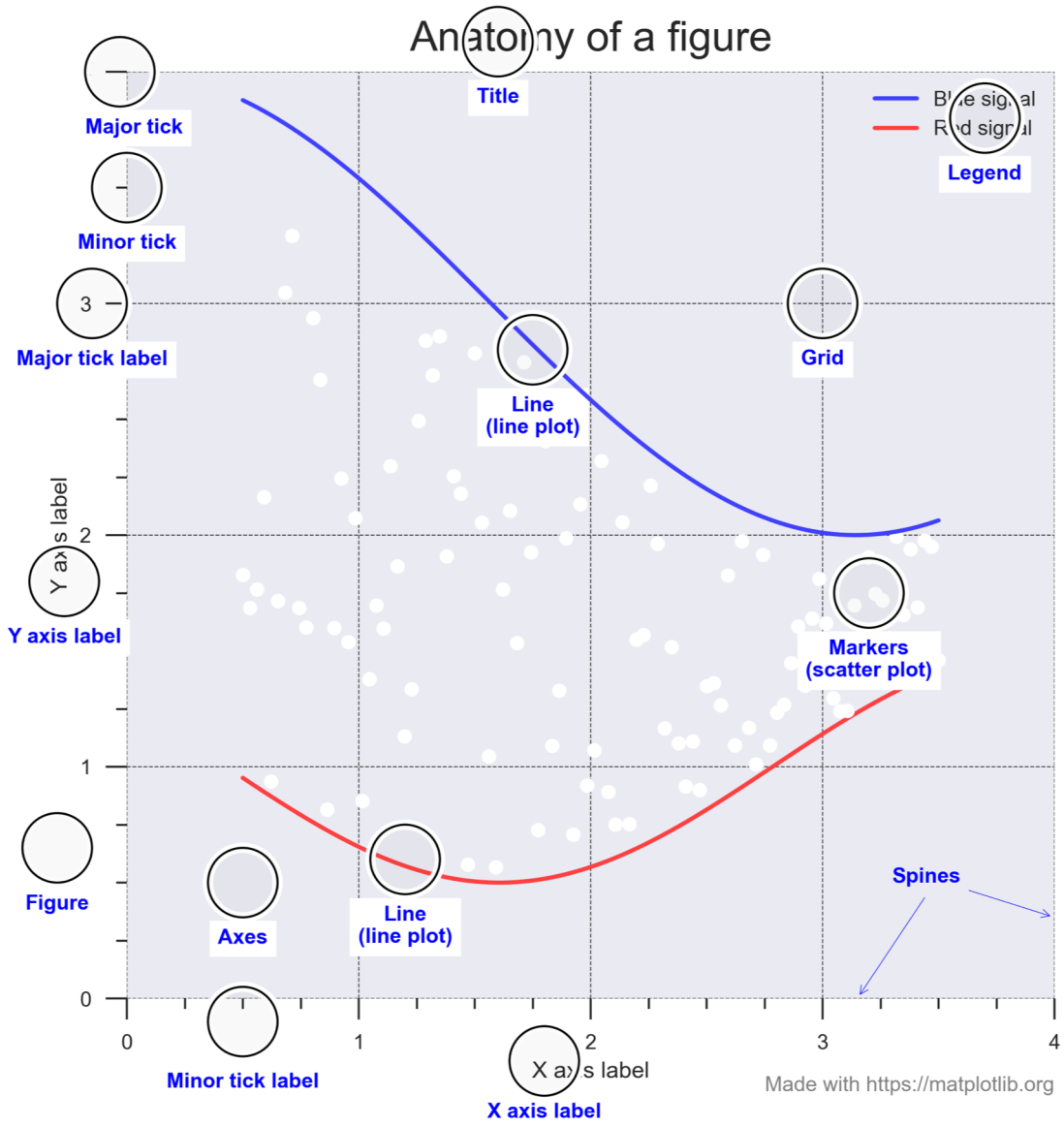
ANATOMY OF A PLOT WINDOW



ANATOMY OF A PLOT WINDOW




Anatomy of a figure
Blue signal
Red signal
Y axis label
X axis label
0
1
2
3
4


Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Legend
Grid
Line
(line plot)
Y axis label
Markers
(scatter plot)
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
Made with https://matplotlib.org

GETTING STARTED WITH MATPLOTLIB

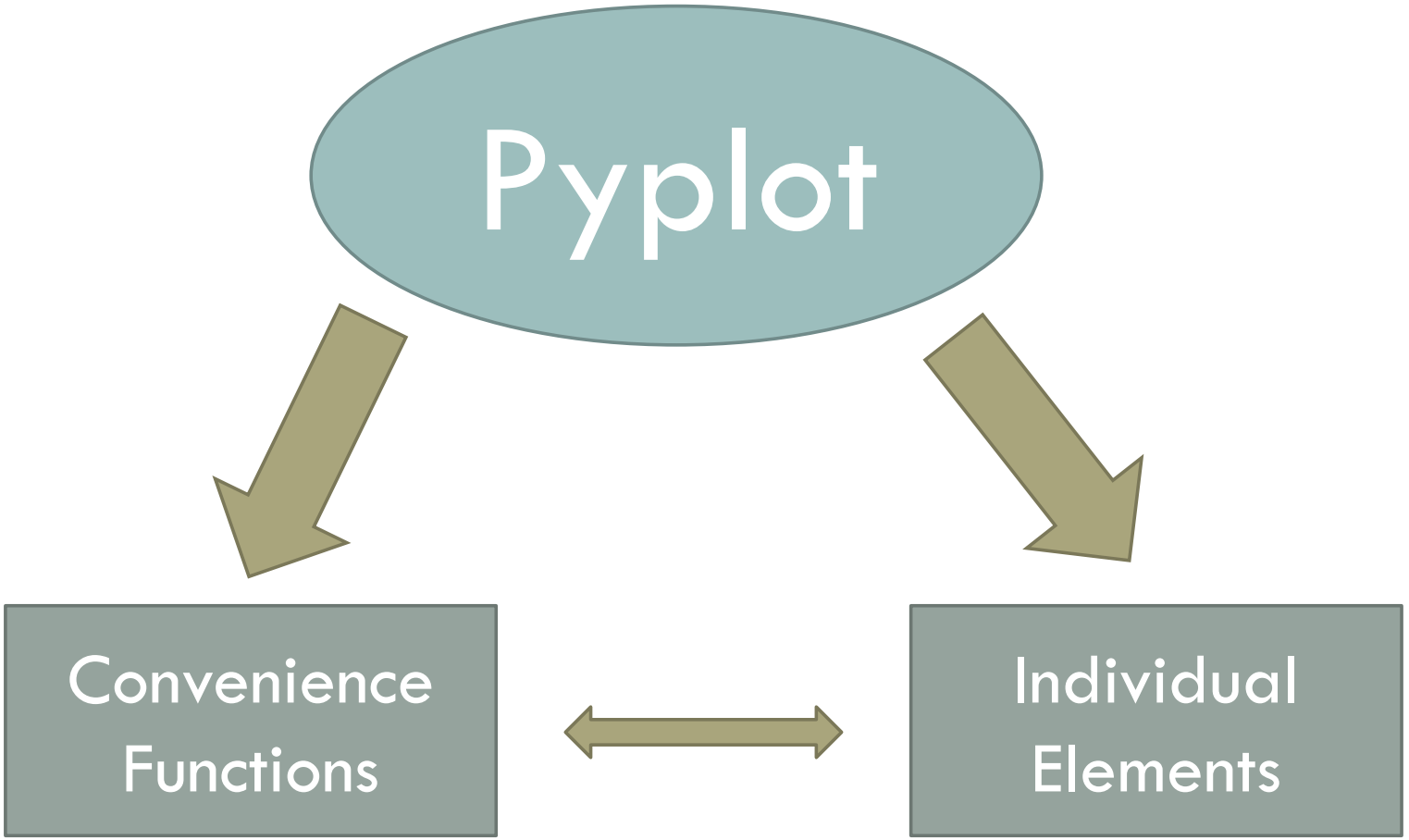
Matplotlib has multiple ways of interfacing with it, as well as a large number of additional modules and patches that extend its capabilities significantly. The main interface we'll be using for this work is the **pyplot** (MATLAB-like) interface:

```
import matplotlib.pyplot as plt
```

You can choose to run matplotlib either **interactively** or **non-interactively**. For the interactive mode, the plot gets updated as you go along. For non-interactive, the plot doesn't show up until you've finished everything. To switch between the two:

```
plt.ion() # Turn interactive mode on
plt.ioff() # Turn interactive mode off
plt.show() # Show the plot when interactive mode off
```

CHOOSE YOUR OWN ADVENTURE!



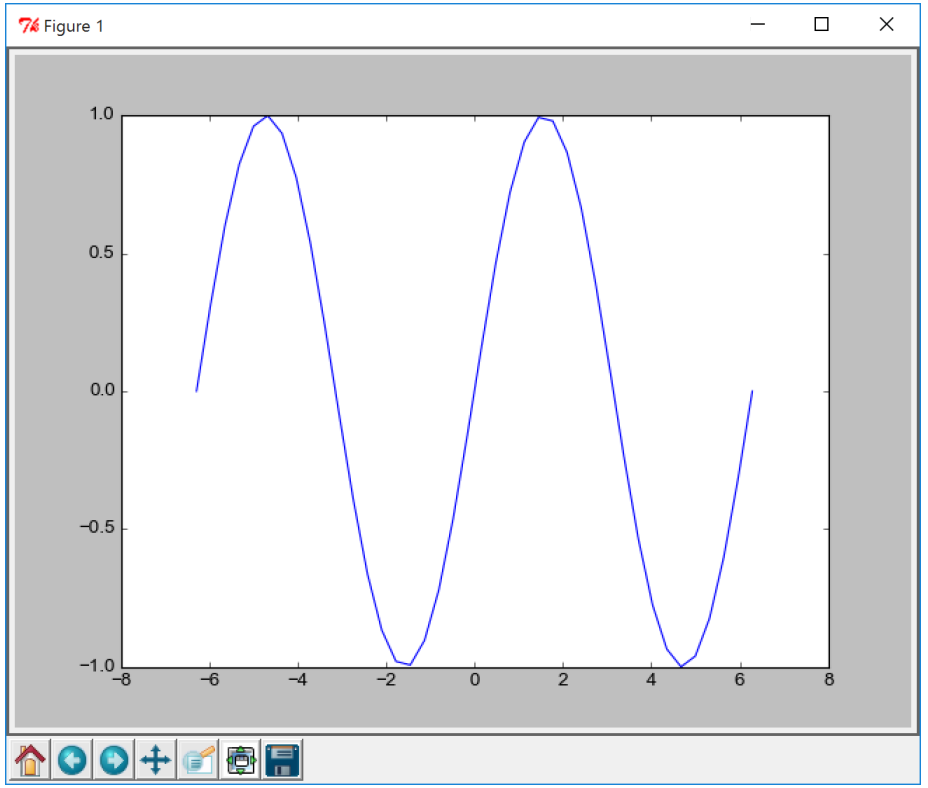
- Really simple to start
- Not as much flexibility

- Requires more coding
- Can plot anything!

SIMPLE PLOTTING BASICS

Much of your power is in the plot command:

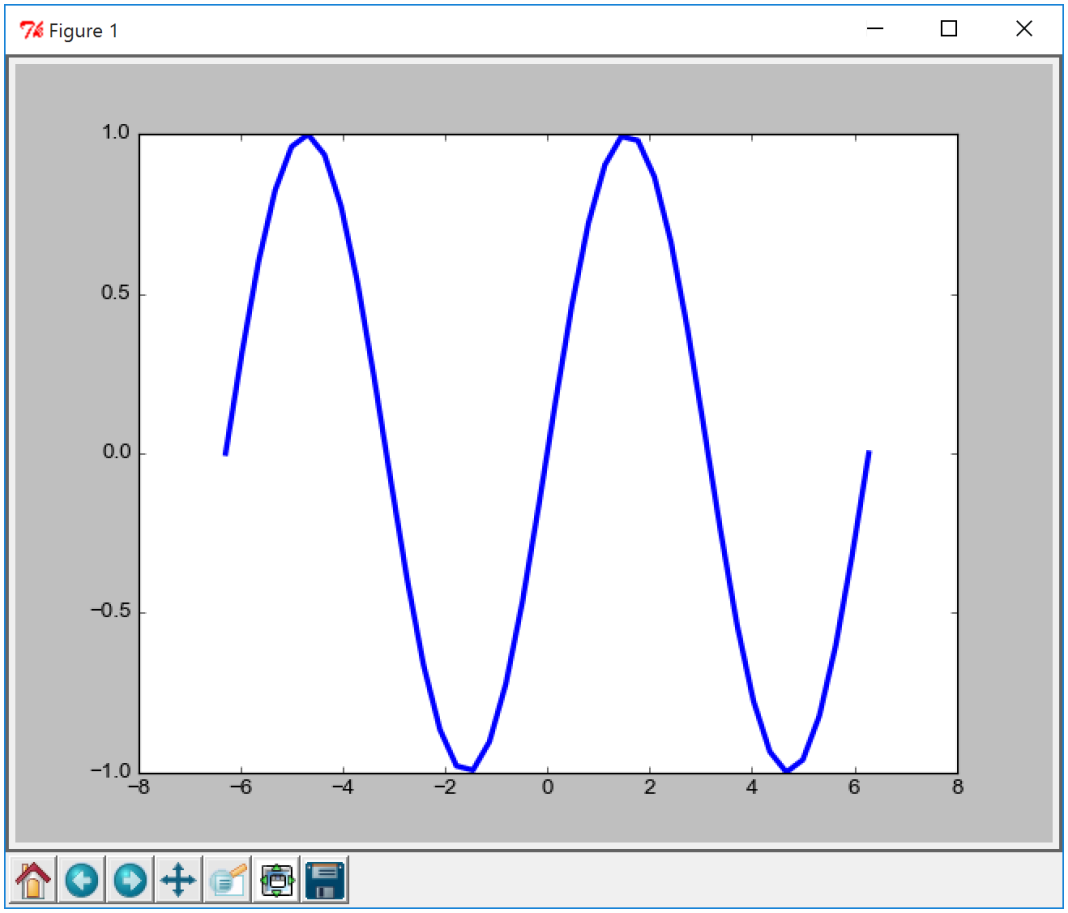
```
# The simplest of plots
import numpy as np
from numpy import pi
x = np.arange(-2*pi,2*pi,pi/8)
y = np.sin(x)
plt.ion()
plt.plot(x, y)
```



SIMPLE PLOTTING BASICS

Change the line width:

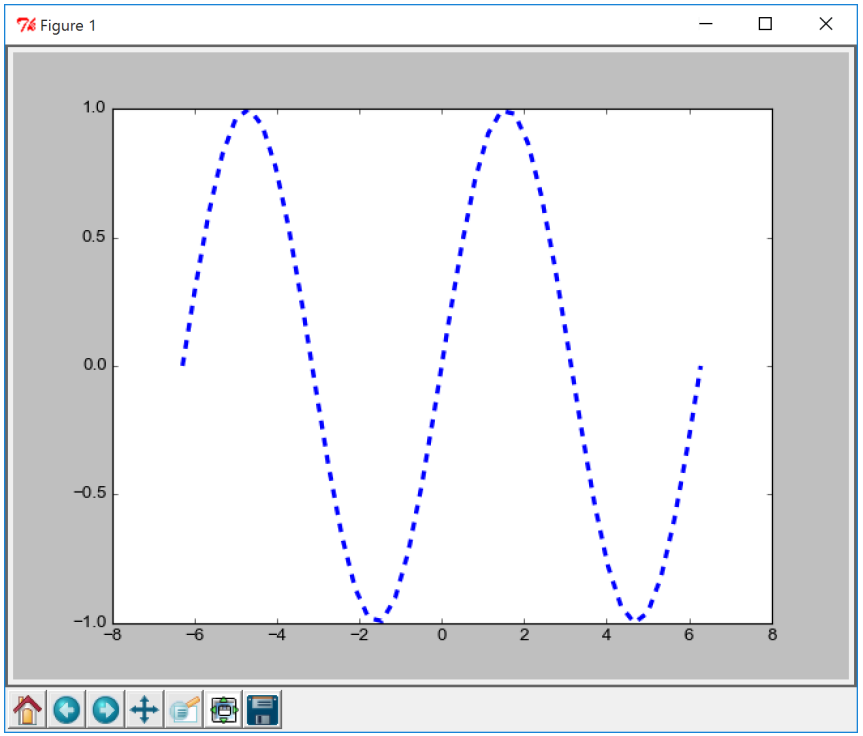
```
plt.plot(x, y, linewidth=3)
```



SIMPLE PLOTTING BASICS

Change the line style:

```
plt.plot(x, y, linewidth=3, linestyle='dashed')
```

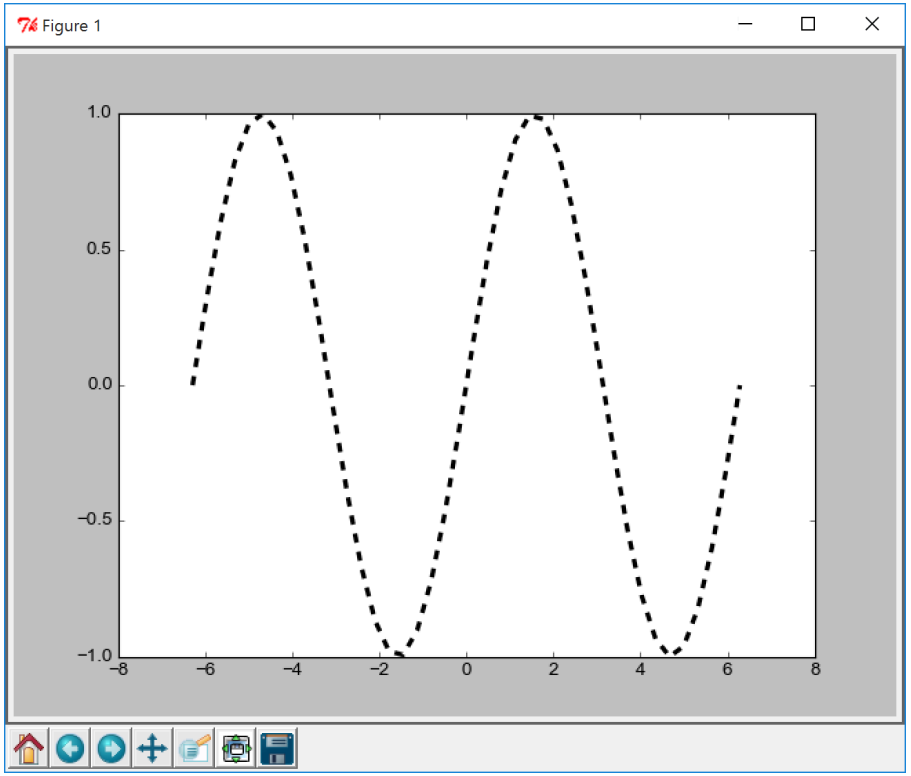


character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

SIMPLE PLOTTING BASICS

Change the line color:

```
plt.plot(x, y, linewidth=3, linestyle='dashed', color='k')
```



COLO(U)RS IN MATPLOTLIB

In matplotlib, colours can be specified in a number of ways

Basic Colours

Most basic (primary and secondary) colours can be quoted by their first letter:

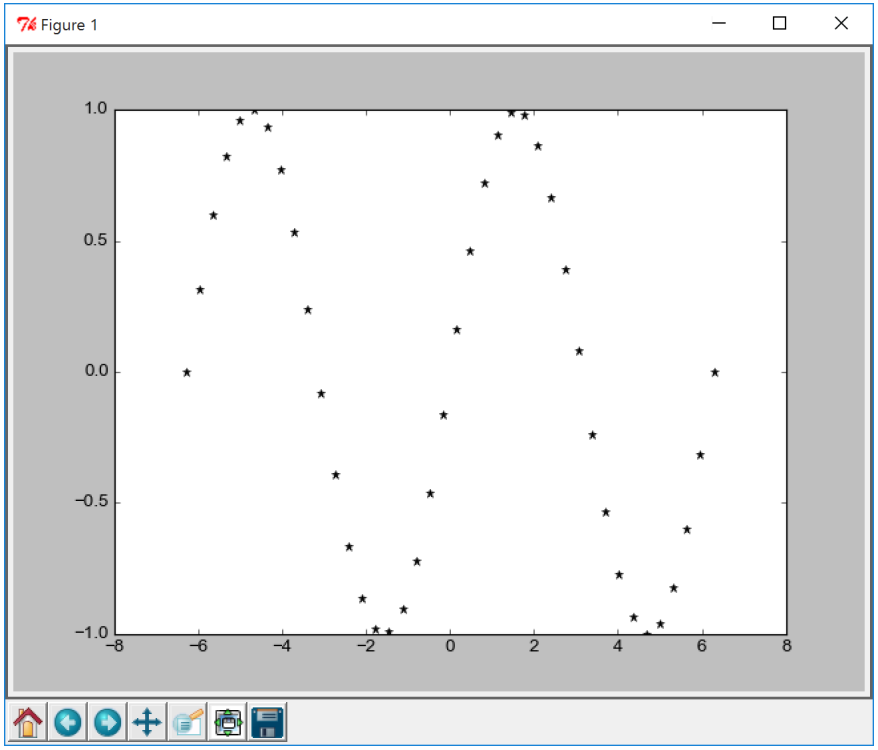
'b' – blue
'r' – red
'g' – green
'y' – yellow
'w' – white
'm' – magenta
'c'– cyan
'k' – black

For other methods, e.g. RGB or html, please visit https://matplotlib.org/stable/tutorials/colors/colors.html

SIMPLE PLOTTING BASICS

Add point markers:

```
plt.plot(x, y, linestyle='none', color='k', marker='*')
```



MARKER STYLES

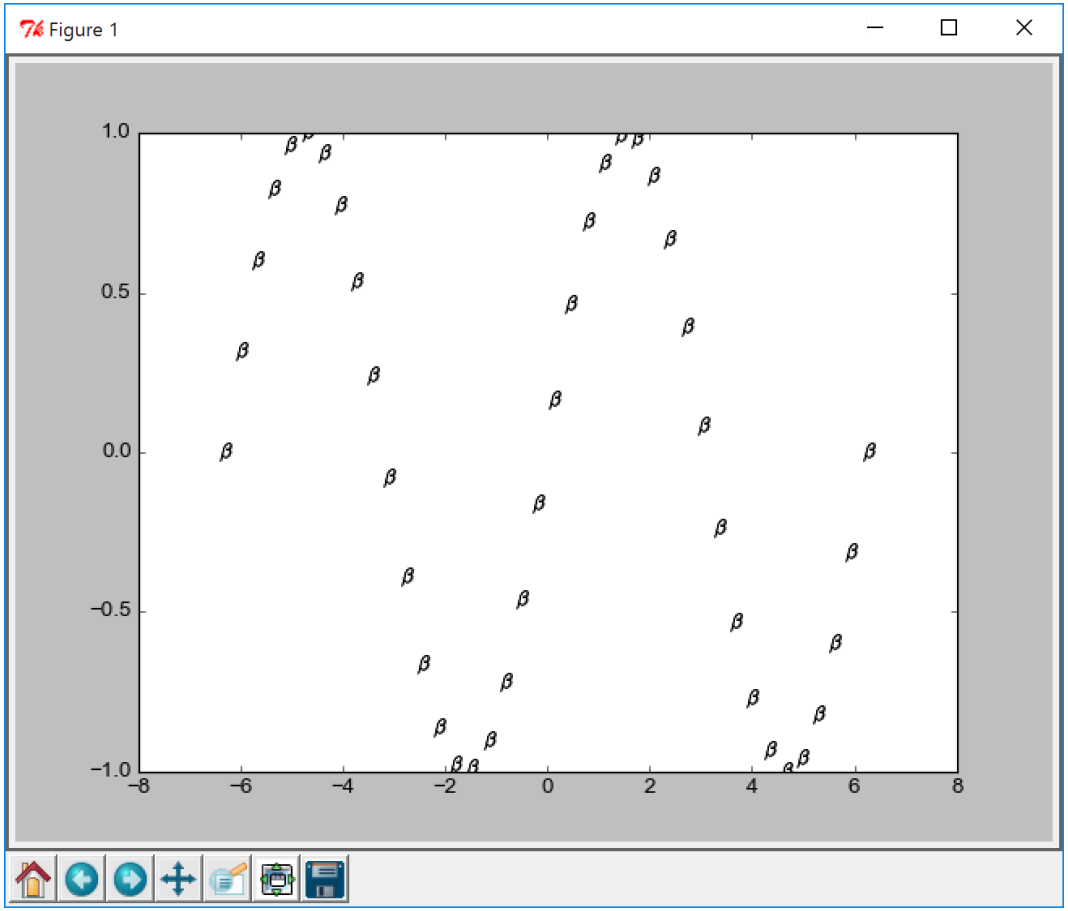
character	description
`'.'`	point marker
`','`	pixel marker
`'o'`	circle marker
`'v'`	triangle_down marker
`'^'`	triangle_up marker
`'<'`	triangle_left marker
`'>'`	triangle_right marker
`'1'`	tri_down marker
`'2'`	tri_up marker
`'3'`	tri_left marker
`'4'`	tri_right marker
`'8'`	octagon marker

`'s'`	square marker
`'p'`	pentagon marker
`'P'`	plus (filled) marker
`'*'`	star marker
`'h'`	hexagon1 marker
`'H'`	hexagon2 marker
`'+'`	plus marker
`'x'`	x marker
`'X'`	x (filled) marker
`'D'`	diamond marker
`'d'`	thin_diamond marker
`'|'`	vline marker
`'_'`	hline marker

SIMPLE PLOTTING BASICS

Can use LaTeX symbols:

```
plt.plot(x, y,
linestyle='none',
color='k',
marker='$\\beta$',
markersize=10)
```



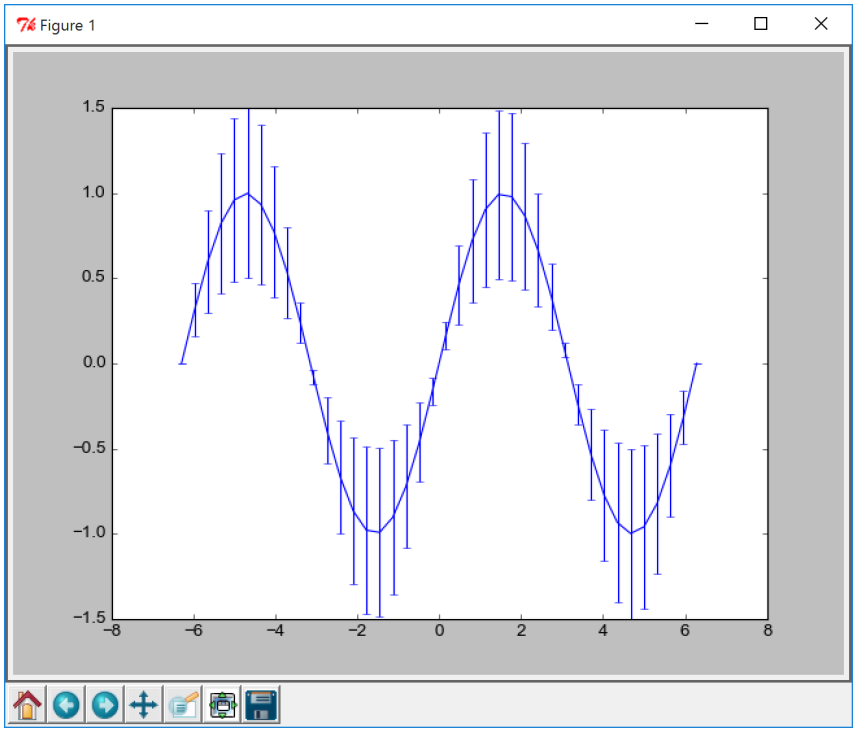
PRO TIP:

For a scatter plot, use plt.scatter() instead

SIMPLE PLOTTING BASICS

Creating error bars:

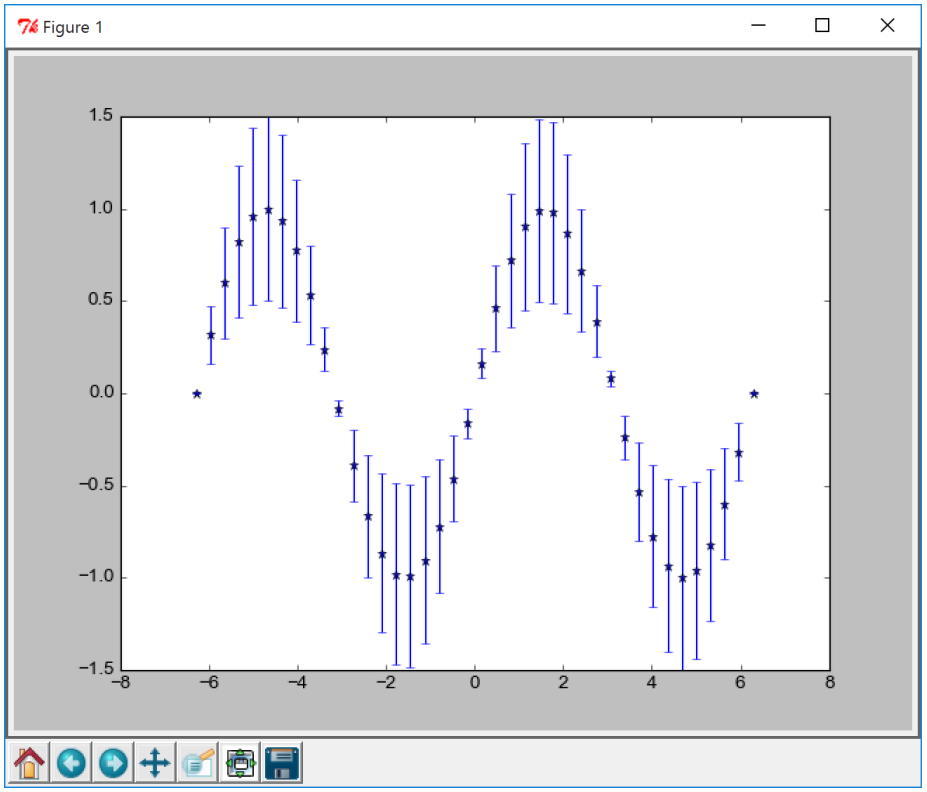
```
n = len(x)
yerr=np.random.uniform(0,.5,n)
plt.errorbar(x, y, yerr=yerr)
```



SIMPLE PLOTTING BASICS

Creating error bars without lines:

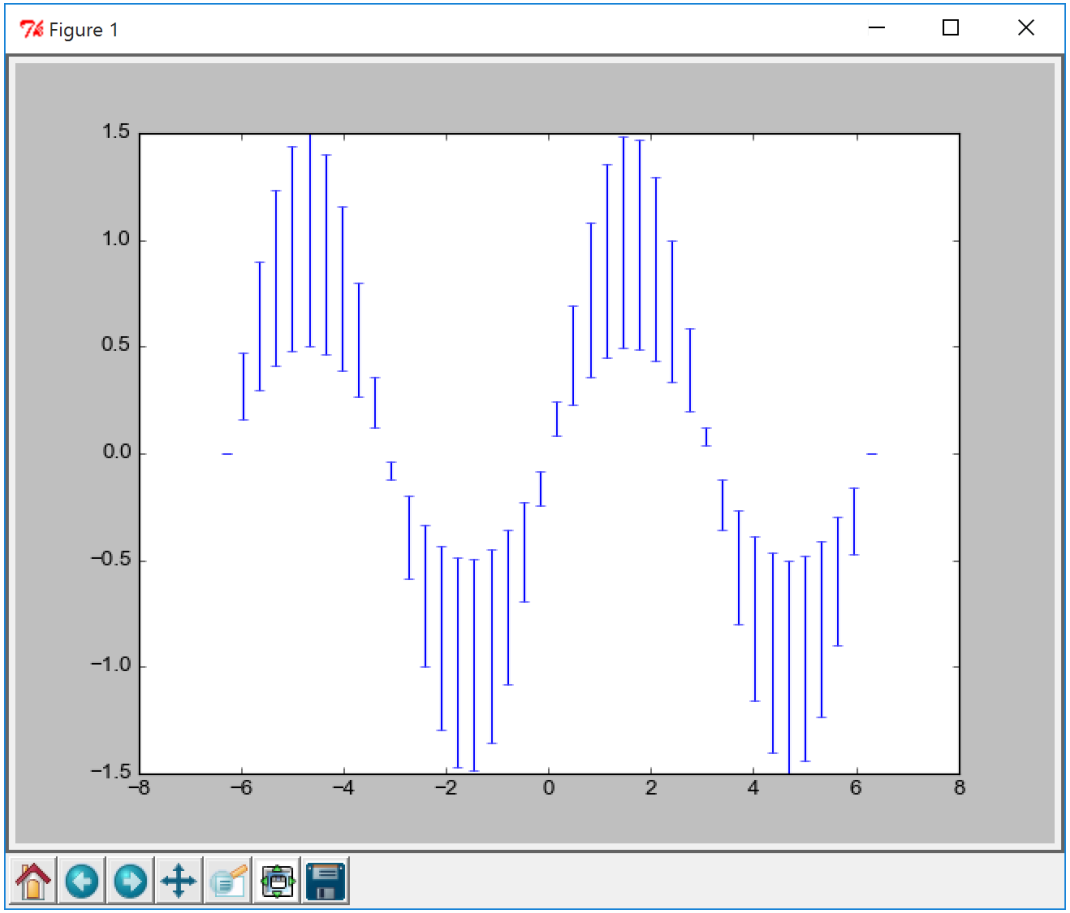
```
plt.errorbar(x, y, yerr=yerr, fmt='*')
```



SIMPLE PLOTTING BASICS

Just error bars:

```
plt.errorbar(x, y,
yerr=yerr,
fmt='none')
```



PRO TIP:

All of these functions have **many** more options. Check the docs.

HOUSEKEEPING FUNCTIONS

To deal with the various figures and axes that there can be, you have the following housekeeping functions:

```
# Clearing Plots
plt.cla() # Clear Current Axis
plt.clf() # Clear Current Figure

# Getting active objects
ax1 = plt.gca() # Get Current Axis
fig1 = plt.gcf() # Get Current Figure

# Make new figure
plt.figure() # Make new figure (with defaults)
plt.figure(figsize=(6,8)) # Make new figure (6”x8”)
```

SETTING AXIS PROPERTIES

You can (at any time in the plotting) change the range (lim), scale (log or linear), labels or ticks on a plot. Replace x with y (or vice versa) when necessary:

```
# Limits and Scale
plt.xlim([0, 5]) # Set x-limits to 0 -> 5
plt.yscale('log') # Set y-axis to logarithmic

# Setting Labels
plt.xlabel('X-axis') # Label the X-axis
plt.title("Title") # Set the Axis title

# Setting Ticks
plt.xticks([0, 0.5, 3, 4.9]) # Location of x-ticks
```

LABELS AND LEGENDS

You can use "labels" on any plot object to automatically populate a legend:

```
plt.figure(figsize=(5,3))
plt.plot(x,y,color='r',label='Test Data')
plt.legend(loc=3,frameon=False)
```

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

makes legend background transparent

SAVING A FIGURE

Saving a figure is a one-line operation. Matplotlib will figure out what format you want by the extension of the filename:

```
plt.savefig("myplot.pdf") # Saving as a PDF
plt.savefig("myplot.png") # Saving as a PNG
plt.savefig("myplot.eps") # Saving as an EPS

# Can also determine what output DPI:
plt.savefig("myplot.jpg", dpi=300)
```

PRO TIP:

EPS files do not support transparency natively

BUILDING FROM THE GROUND UP

This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```

BUILDING FROM THE GROUND UP

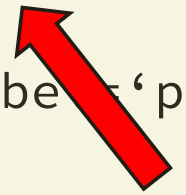
This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```



This uses matplotlib's location format, which takes the format of:

[left, bottom, width, height]

where each of the numbers are from 0 to 1 (in units of a fraction of the figure)

BUILDING FROM THE GROUND UP

This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```

All of those major plotting functions (i.e, plot, scatter, legend, *et cetera*) are now just methods on the axis.

BUILDING FROM THE GROUND UP

This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```

All axis properties (i.e., x/ylim, x/yscale) can be set by the methods axis.set_*property*. Also, you can get the current values for these by axis.get_*property*.

BUILDING FROM THE GROUND UP

This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```

Saving the figure is a method of the figure itself

This is particularly useful if you have multiple figures and axes.

CUSTOMIZING DEFAULTS

There's a lot of different parameters that matplotlib chooses by default, but you can set your own using a **matplotlibrc** file. This file will not exist by default, but you can download a sample one here:

http://matplotlib.org/_static/matplotlibrc

The place to put this file depends on your platform:

Windows: *UserDirectory*/.matplotlib/matplotlibrc
(i.e., C:/Users/*username*/.matplotlib/matplotlibrc)

MacOS: *UserDirectory*/.matplotlib/matplotlibrc
(i.e., Users/*username*/.matplotlib/matplotlibrc)

Linux: *UserDirectory*/.config/matplotlib/matplotlibrc
(i.e., /home/*username*/.config/matplotlib/matplotlibrc)

EXERCISE TIME!

EXERCISE-1

plot f(x) = x, f(x) = x^2, f(x) = x^3, all in the same figure, for x changing from 0 to 2 linearly and using

- "linear", "quadratic", and "cubic" as legends
- cont. line for f(x) = x, dashed and dotted-dashed lines for the others
- "x" as x-label
- "f(x)" as y-label
- "Simple Plot" as the title

Set figure size to 5x3, replot, and save your figure as a png file.