**An Introduction to Making Plots with Python**

The `matplotlib` plotting library for Python (part of `pylab`) makes publication quality figures that are easy to modify and save. The example below plots $\cos(2\pi t)$ vs. *t*.

```
from pylab import *

t = linspace(0.0, 2.0, 100)  # makes a list
y = cos(2*pi*t)  # makes a list with same length as t

figure()  # opens a new figure
plot(t, y, ls='-', color='r', linewidth=1.0)

xlabel('time (s)')
ylabel('voltage (mV)')
title('A Simple Plot')
grid()

show()
```

The first line loads `pylab` library. The `linspace` command returns a list of evenly spaced numbers from the first argument to the second argument, where the number of elements is given by the third argument. You can add the argument `endpoint=False` if you don't want the list to include the second endpoint. Many of the predefined functions in Python are "vectorized" which means that they can accept a list as input. For example, when the `cos` function has an argument `t` that is a list, it will return a list. That means that `y` will be a list containing the cosines of the elements in the list `t` (you can print those variables to check this).

The `figure` command opens a new figure window. If you want a second line to appear in a different figure, you should put a second `figure` command before the second `plot` command. If you want multiple plots to appear in a single figure, all of the `plot` commands should be below a single `figure` command.

The first argument of the `plot` command contains the horizontal coordinates and the second contains the vertical coordinates. In other words, the example above makes a plot of `y` vs. `t`. Some options for the `linestyle` (or `ls`) argument are:

`-` = solid  `--` = dashed  `:` = dotted  `-.` = dash-dot

The `color` (or `c`) argument set the color of the line. Some of the options are:

| | | | |
|---|---|---|---|
| `r` = red | `g` = green | `b` = blue | `k` = black |
| `c` = cyan | `m` = magenta | `y` = yellow | `w` = white |

The `linewidth` argument takes an integer.

The `grid` command can be used to add a grid to the figure. The `color` argument can be used with this command.

The `show` command tells Python to draw the figures. It should appear *after* the last plotting command. The "toolbar" shown below will appear below each figure.

1. The fourth button is the "Pan/Zoom" button, which has two modes. After clicking this button, do either of the following:
   (a) Press the left mouse button and hold it, dragging it to a new position. When you release it, the data under the point where you pressed will be moved to the point where you released.
   (b) Press the right mouse button, dragging it to a new position. The x axis will be zoomed in proportionate to the rightward movement and zoomed out proportionate to the leftward movement.
2. The sixth button is the "Zoom to rectangle" button. Click this toolbar button to activate this mode. Put your mouse somewhere over an axis and press the left mouse button. Drag the mouse while holding the button to a new location and release. The axis view limits will be zoomed to the rectangle you have defined.
3. The first button is the "Home" button, which will restore the original view.
4. The last button is the "Save" button. Click this button to save the figure as one of the following image types: PNG, PS, EPS, SVG.

You could also manually set the limits on the axes with the `xlim` and `ylim` commands, which take two arguments for the lower and upper limits.

With the `plot` command, it is optional to add markers for each point on the list. Some of the options for `marker` argument are:

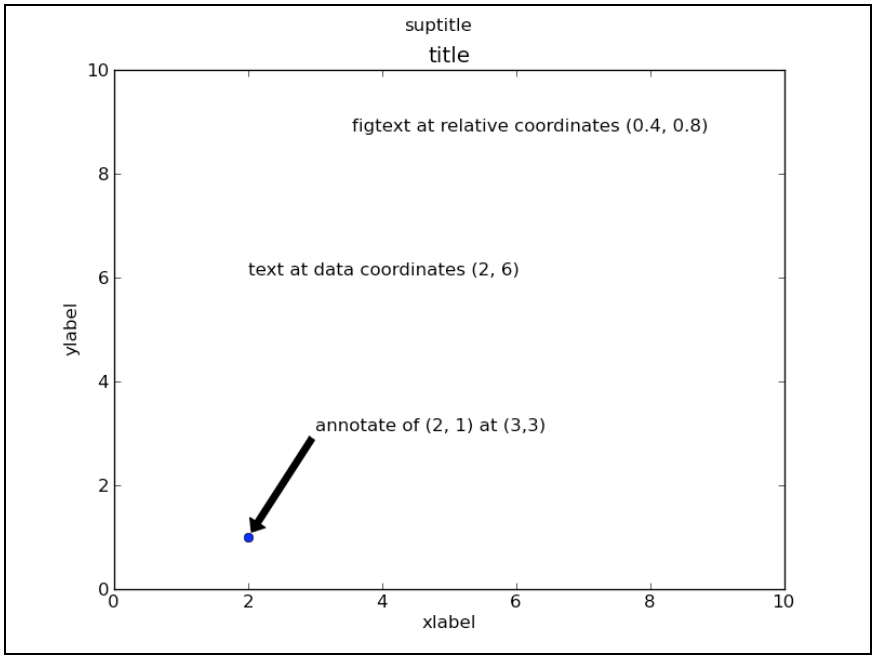
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| `.` | = points | `o` | = circles | `s` | = squares | `D` | = diamonds |
| `h` | = hexagons | `8` | = octagons | `^` | = up triangles | `v` | = down triangles |

The `markersize` (or `ms`) argument is an integer used to set the size of the markers The `linestyle` can also be set to `None` to show only the markers.

The `matplotlib` commands listed below place text on a figure. (The main difference between `text` and `figtext` is how the location is specified.) For all of these text commands, the size can be adjusted with the `fontsize` argument. The `color` argument can be used with these command.

| | |
|---|---|
| `xlabel` | label the horizontal axis |
| `ylabel` | label the vertical axis |
| `title` | add a title just above the axes |
| `suptitle` | add a title farther above the axes |
| `text` | add text at an arbitrary location in data coordinates |
| `figtext` | add text at an arbitrary location in relative coordinates |
| `annotate` | add an annotation, with optional arrow |

The figure below shows examples of the various kinds of text.




There are a few other plotting functions that are likely to be useful. The `scatter` function makes a scatter plot, which is useful for plotting data. The `size` (or `s`) argument is used to set the size, instead of `markersize`.

```
scatter(x, y, c='b', s=3, marker='o')
```

The `errorbar` function makes a scatter plot with error bars. Putting error bars in the horizontal direction is not required, so the `xerr` is optional

```
errorbar(x, y, yerr, xerr, marker='s', markersize=2)
```

One of the axes can be made logarithmic with the `semilogx` or `semilogy` function. Both axes can be made logarithmic using the `loglog` function. These functions can be used to plot lines, markers, or both.

```
semilogx(x, y, ls=':')
semilogy(x, y, marker='.')
loglog(x, y, ls='-'
```

For logarithmic scales, including grid lines makes it much easier to estimate values on a graph. The following command will put grid lines at both the major tick marks and at the minor tick marks in between.

```
grid(which='majorminor')
```

If you want an error bar plot on a semilog scale, placing the command `semilogy()` with no arguments immediately after the `errorbar` command will make the vertical axis logarithmic.

Multiple plots can appear in the same figure by placing them below a single `figure()` command. In that case, it is useful to add a legend by adding the `label` argument with text to each plotting command and using the `legend()` command.

Exercise:
Make plots of $\exp(-t/10)\cos(2\pi t)$ and $\exp(-t/10)$ vs. $t$ in the same figure. Let $t$ range from 0 to 20. Use a solid, green line for the first function and red circles for the second. Label the horizontal axis "Time (s)" and the vertical axis "Position (cm)" in blue.

Additional documentation is available at:

http://matplotlib.sourceforge.net/users/pyplot_tutorial.html (a good place to start)
http://matplotlib.sourceforge.net/ (links to documentation for all commands)