Week 4

2D Arrays and Plotting

2D arrays

- So far, we have been working with one dimensional arrays (e.g. array([1,2,3,4,5,…])
- With “matching” 1D arrays for x and y we can plot 2D data- such as position vs time. Each “data point” contains two pieces of information: x, and y (or time and position).
- A 2D array allows us to plot 3D data points- x,y,z. For example, we may have two position variables and one value variable.

2D Arrays

- The common way to think about it is like a photograph. If you have a jpeg image, it is made up of a bunch of pixels (which relate back to the pixel detectors on the camera's CCD).
- You can look at an individual pixel (say, (512,512)), and you will find that that pixel has a number/value (which for jpeg relates to how bright/what color that pixel should be).
- The simpler case in astro imaging is usually that each pixel contains monochromatic information- it is just an intensity. ˜

Defining a 2D array

- We can define 2D arrays in several ways: manually, via hstack, and via vstack.
- Example

2D Arrays

- More often than not, we pull 2D arrays out of data files rather than constructing them ourselves
- Classic example is FITS image files (from telescopes). We will have a tutorial on them next week.
- Note: You can have even higher dimensional arrays- it all depends on how much information you need to store.

Matrices

- Numpy has functions for defining matrices. (np.matrix)
- In my experience, because arrays operate on matrix rules, it usually doesn't make a difference whether you use np.array to make a matrix structure or np.matrix.
- Other useful linear algebra commands: np.dot, np.cross, np.linalg.inv (take the inverse), np.transpose, np.diag, np.eye (for identity matrix)

Exercise

- Construct a 10x10 array of zeros (as efficiently as you can)

Solution 1

- arr = np.zeros(10)
- A = np.vstack((arr,arr,arr,arr,arr,arr,arr,arr,arr,arr))

Better Solution

- the numpy functions like np.ones, np.zeros let you specify 2 dimensionality
- A = np.zeros((10,10))
- B = np.ones((5,5))

Exercise

- construct a 2d array, 3x3, that looks like this:

[1,2,3]
[4,5,6]
[7,8,9]

- (Use np.arange)

Solution

- a1 = np.arange(1,4)
- a2 = np.arange(4,7)
- a3 = np.arange(8,10)
- A = np.vstack((a1,a2,a3))

Better Solution

- Numpy has a reshape command for Arrays- you can reshape a 1D matrix into a 2D like this:
- A = np.arange(1,10)
- A =A.reshape((3,3))
- in_one_line = np.arange(1,10).reshape((3,3))

Plotting

- Plotting is one of the most important parts of coding, because your results don't mean anything unless you can communicate them.
- Plotting can take on basically infinite customization- way too much to cover here. We will get into the basics, and a few of the bells and whistles of matplotlib. Beyond that, you basically look up what fancy thing you need when you need it.

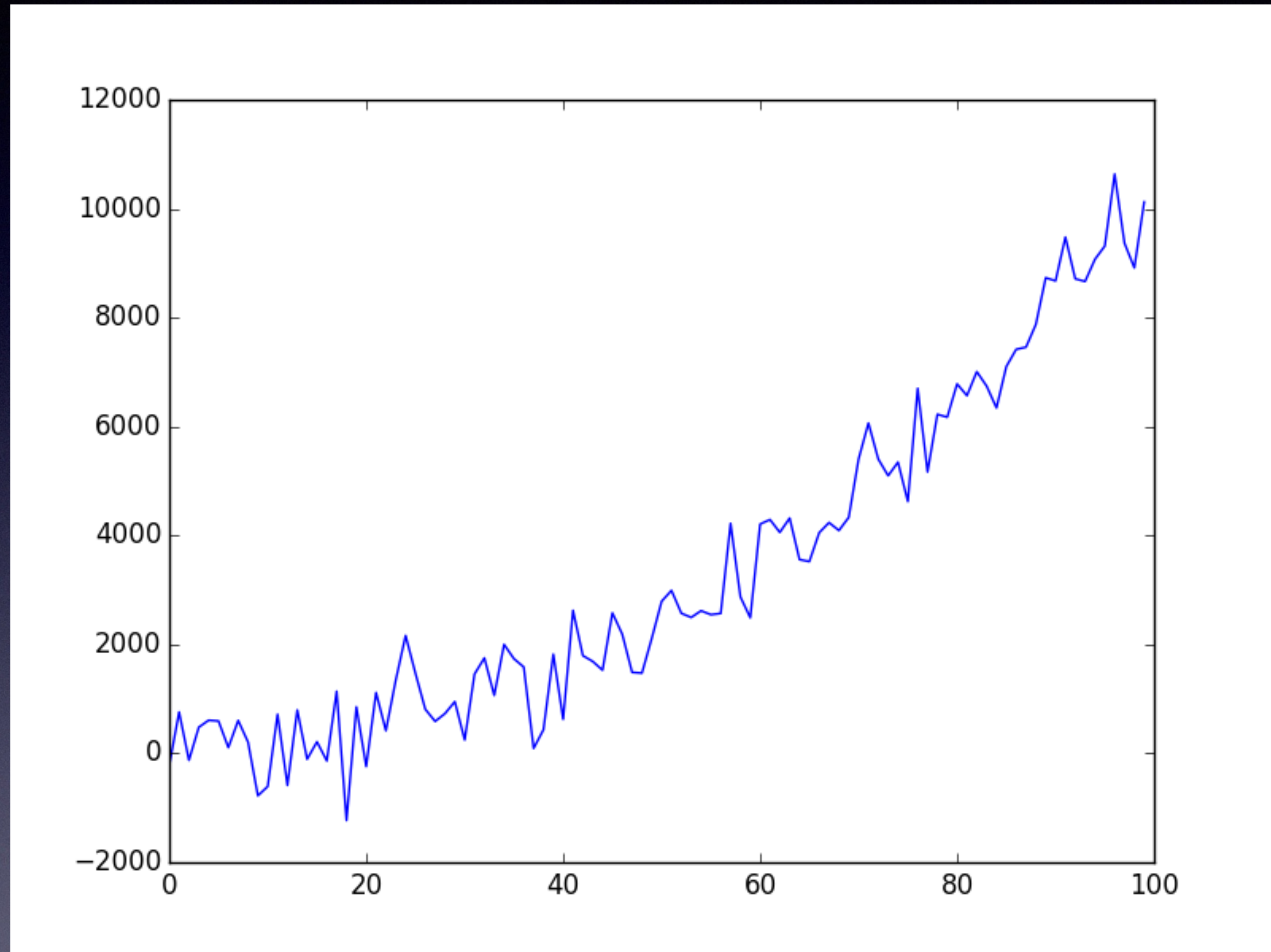
Basic Plotting

- We have already done this: absolute minimum- if you have 2 equal length arrays, one with x values and one with y values, you can use plt.plot(x,y) to plot a connected blue line (by default) of y vs x.
- The first change you can make to this is to plot individual data points rather than a continuous line (since data is never continuous right??)

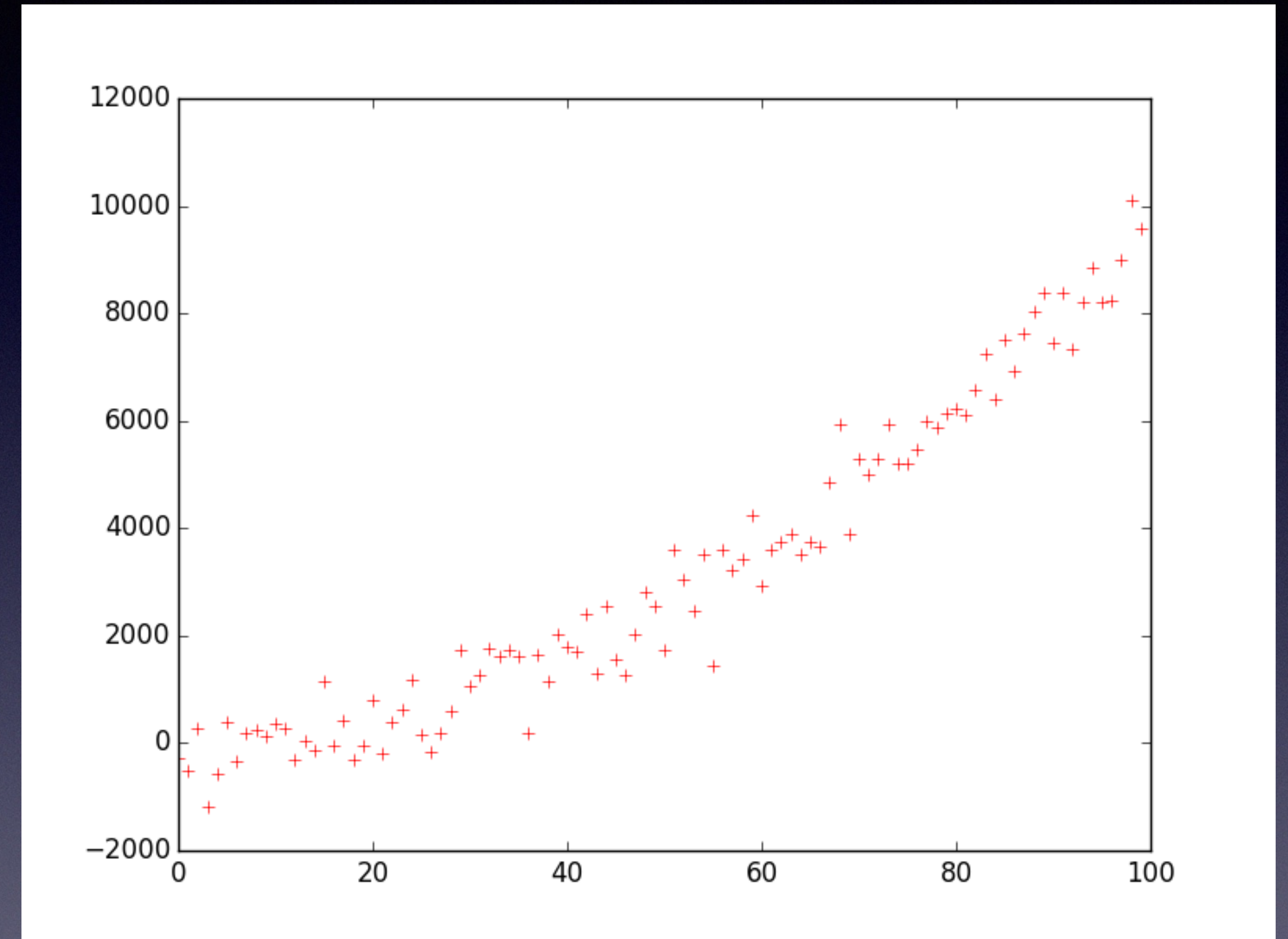
Plotting points individually

- The plt.plot command has a ton of specifiable arguments you can put in (use help(plt.plot) to pull up a lot of the options.
- The basic ones are color and line style
- plt.plot(x,y, ‘r+’) would plot the data points as red plusses (there are a lot of shortcuts)

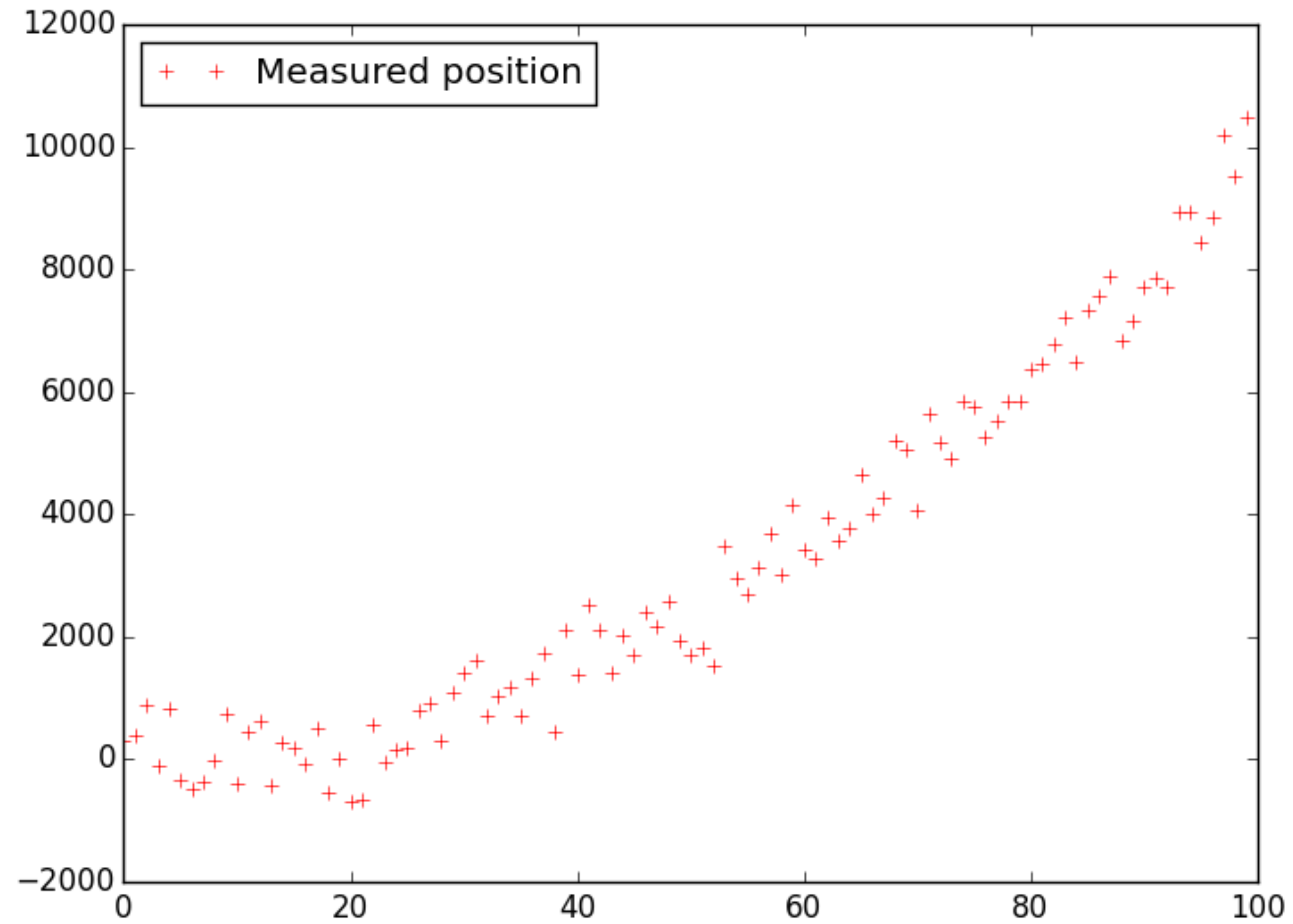
Fake data: x = np.arange(100)
y = x**2
y2 = y + 550 * np.random.normal(size=x.shape)



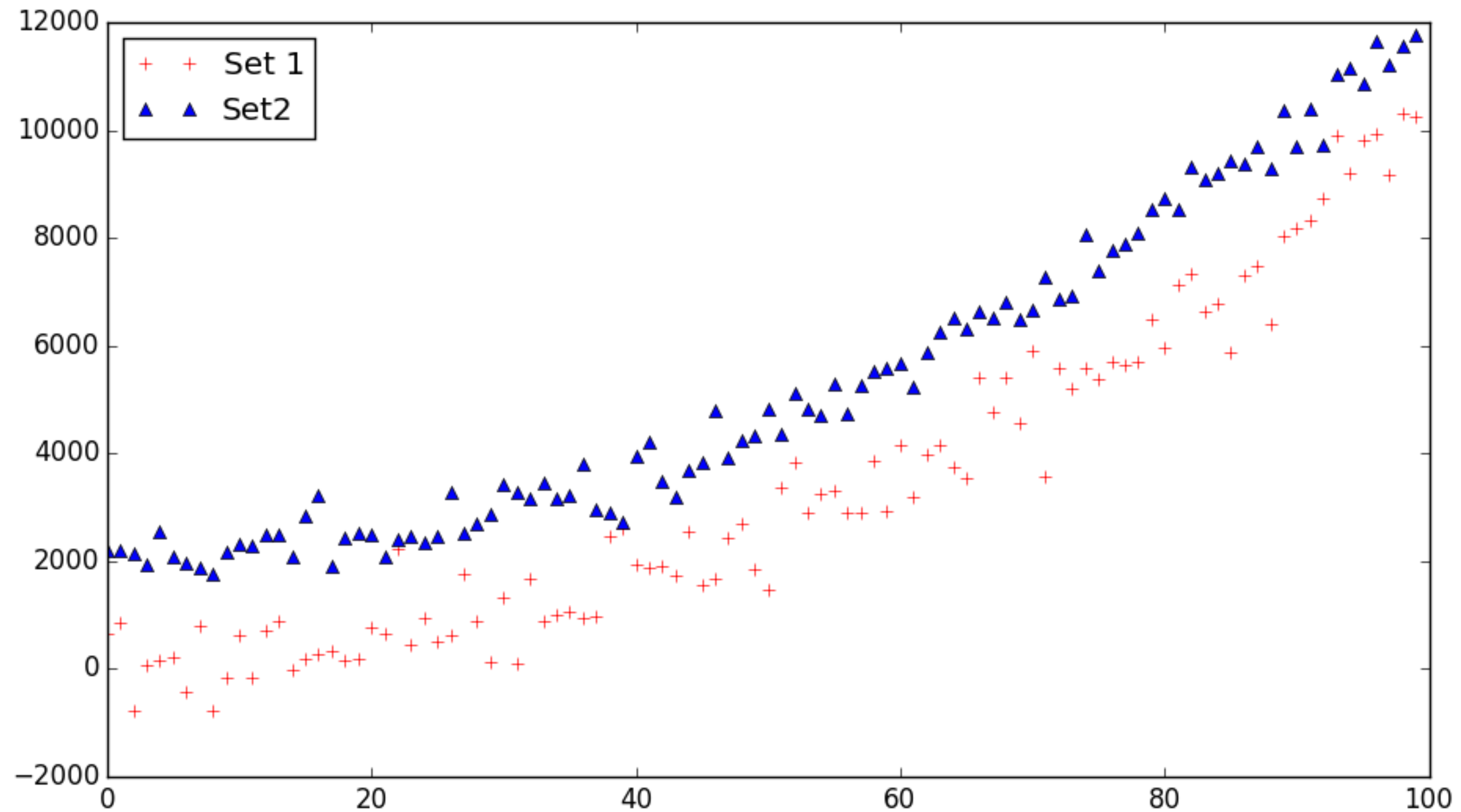
plt.plot(x,y2)



plt.plot(x,y2,’r+’)




```
plt.plot(x,y2,'r+',label='Measured position')
plt.legend(loc=2)
```

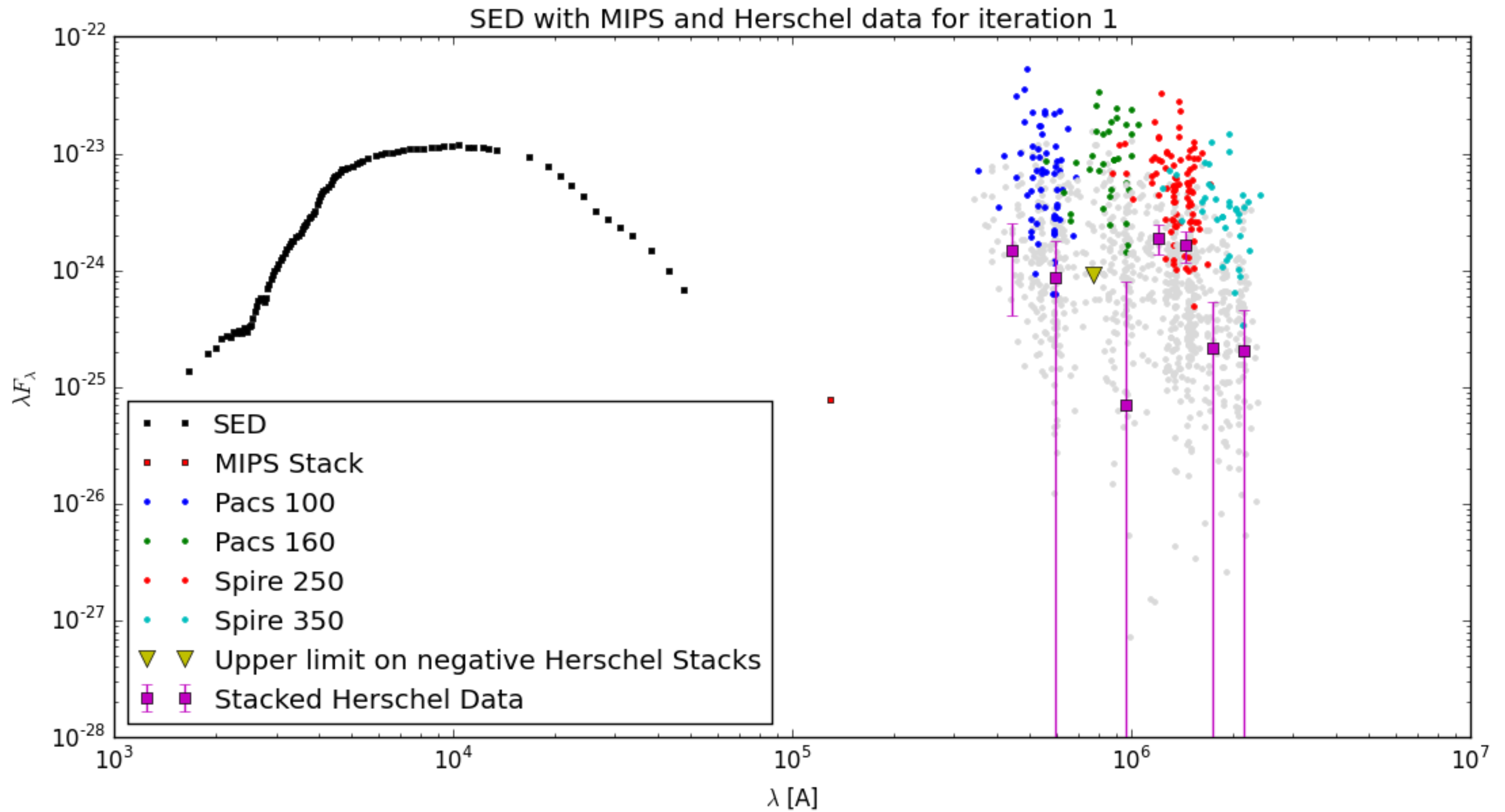


We can plot multiple data sets on the same graph just by plotting one after the other without creating a new figure (But it will only look good if they are in similar ranges)

```
================    =================================
character           description
================    =================================
``'-'``             solid line style
``'--'``            dashed line style
``'-.'``            dash-dot line style
``':'``             dotted line style
``'.'``             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
``'s'``             square marker
``'p'``             pentagon marker
``'*'``             star marker
``'h'``             hexagon1 marker
``'H'``             hexagon2 marker
``'+'``             plus marker
``'x'``             x marker
``'D'``             diamond marker
``'d'``             thin_diamond marker
``'|'``             vline marker
``'_'``             hline marker
================    =================================
```

```
==========  ========
character   color
==========  ========
'b'         blue
'g'         green
'r'         red
'c'         cyan
'm'         magenta
'y'         yellow
'k'         black
'w'         white
==========  ========
```

You can combine a color and a symbol in one string, e.g. ‘yD’ for yellow Diamond



Fun note: once you learn latex, you can use latex commands in your plot labels

On colors

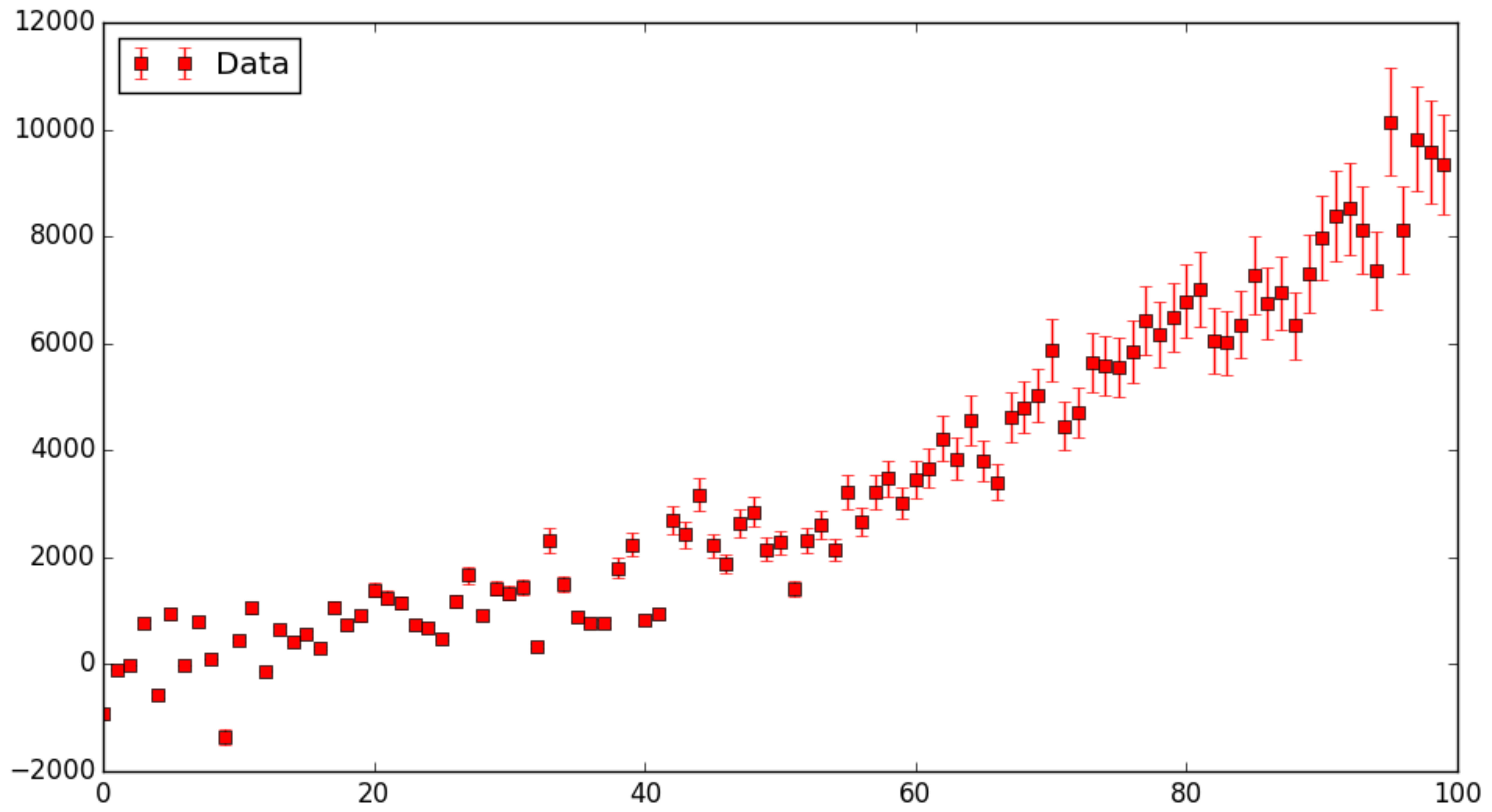
- If those aren't enough colors for you, matplotlib also allows you to select color by rgb value or hex…
- While this ‘r+’ shortcut works on plt.plot, it doesn’t on others (like plt.axvline, as we discovered).
- Experimentation and google are really the only way to be sure about those
- Other shortcuts include c=‘r’ for specifying a color, ls for line style, etc… Its a mess

Errorbars

- You can use the plt.errorbar function to plot data with error bars. Basically you can either specify a single error value for all data points, or have arrays (same length as x and y) with the errors for x and y

- plt.errorbar(x,y,xerr=err1, yerr=err2) #where err1, err2 are the error arrays. you can also specify a symbol with fmt=

- By default it assumes the same error above and below a point, but you CAN change that (rarely have to)




```
y_error = y2/np.random.randint(1,20)
plt.errorbar(x,y2,yerr=y_error, fmt='s', c='r', label='Data')
```

Advice

- Always title and label the axes of your graphs (you can see how in earlier tutorials).
- Use plt.tight_layout() always, to reduce the whitespace around the plots that get saved out.
- If you need some wacky plot type, go to http://matplotlib.org/gallery.html and look till you see something close enough to your needs that you can adapt it.

Final thoughts

- The example document for this week contains a bunch of different combinations of plotting data points. Try running them yourself, and see how the commands translate into things like legends and special symbols.
- From these examples you should be able to cobble together what you need in your own code.