Homework 0 - SVN and Python

Don't forget to `svn up` *before you issue any other commands in SVN—this is to guard against you changing a document that someone else is working on in the same directory*[1]*.*

Don't forget to `svn ci` *(with* `-m` *comments) frequently as you work. This allows others to see how your work progressed, and it automatically backs your work up as you produce it so that you're less likely to lose any of your work and/or so that you can revert to earlier versions of your work.*

Remember to comment your code carefully with your initials before every comment (as in `#ADM I just wrote a Python comment`*). Remember to provide an informative header for* **every** *function that you write.*

Homework

1. Write a Python function that is passed the values m and b in the expression for a straight line $y = mx + b$ and; **(i)** generates 10 floating point numbers in the range 0–10, at random, along the x-axis (*hint: look at the* `numpy.random.random` *function*); **(ii)** uses $y = mx + b$ to recover the appropriate y values; and **(iii)** scatters each of the 10 points in the $\pm y$ direction according to a random offset drawn from a Gaussian of standard deviation 0.5 centered on a given y value (*hint: look at the* `numpy.random.normal` *function*). Your function should return an array of the x values, an array of the y values, and an array of the error on y (this $y_{\rm error}$ is always 0.5).

2. Use `numpy.polyfit` to fit a straight line to your generated $x, y, y_{\rm error}$ data and recover the best fitting values of m_2 and b_2 (which may, of course differ from your original m and b).

3. Plot your data, the original line ($y = mx + b$) from which your data was drawn, and the best-fitting line ($y = m_2x + b_2$) that `numpy.polyfit` derived, to the screen.

4. Use `matplotlib` to create a hardcopy of your plot in PNG format (PNG is suitable for displaying on webpages). Try to make your plot look as professional as possible. Consider whether the x and y ranges displayed in the plot are appropriate. Use color to distinguish lines and points. Use an appropriate symbol to display your points, and don't forget to display the errorbar. Consider adding labels. Make sure your characters and lines are of appropriate thickness to display on a webpage (*hint: look at some of the examples from the* `matplotlib` *tutorial listed on the ASTR 5160 Week 1 links page*).

5. Finally, create a function that links the steps $1.-4.$ together to create a single Python module that is passed a value of m and a value of b *at the UNIX command line*, generates some mock $x, y, y_{\rm error}$ data, fits a line to the mock data, plots everything to the screen and makes a hardcopy in PNG format. Use the `time.time()` function to add a timer to your code, and print useful comments and times to the screen after each of the steps $(1.-4.)$ in this master procedure (*hint: consider the* `if__name__ == "__main__":` *command listed on the ASTR 5160 Week 1 links page*).

[1] this shouldn't be a big deal unless we're working collaboratively, but you should get into the habit *now*