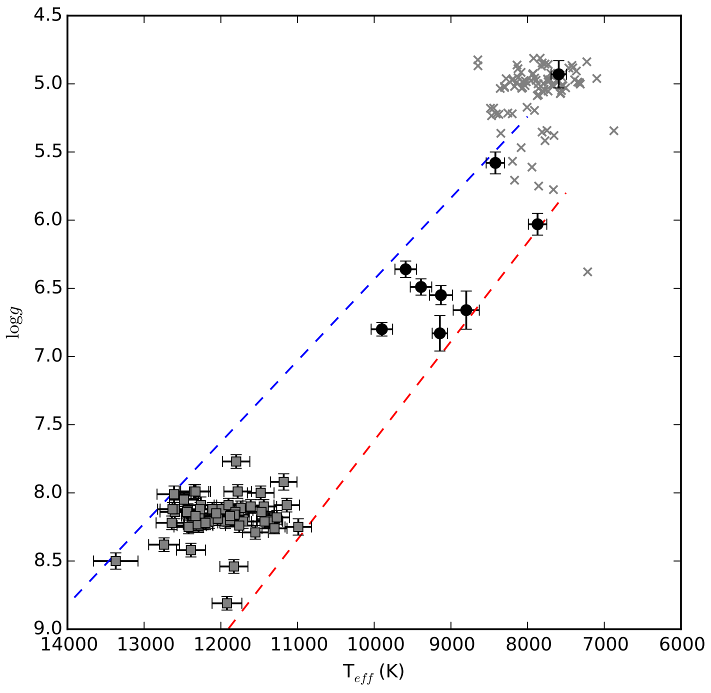
FRI Astronomy Lab #2

Goal: In this lab, we conclude our crash course in Python. This lab should leave you with a solid foundation from which you can continue to develop your coding prowess as necessitated by your research undertakings.

You will learn how to write conditional statements, code loops, and functions, as well as a more general way to deal with files. You will practice these skills in a small Python project that is part of actual ongoing research in the field of white dwarf asteroseismology.

Assignment: Employ functions, loops, and conditional statements in an effort to identify pulsating white dwarf stars from archival data.



Assignment Background: The figure above is directly from our recent McDonald Observatory observing proposal. It shows a number of white dwarfs positioned by the log of the gravity at their surface ($\log g$) and their surface temperatures (T_{eff}). The white dwarfs shown as gray squares near the bottom of the figure are "normal-mass" *pulsating* white dwarfs with masses near the average white dwarf mass of $\sim 0.6\,M_\odot$ (solar masses). The black dots at lower $\log g$ (higher on the plot) are the nine currently known pulsating extremely low-mass (ELM) white dwarfs.

The dashed lines on the figure are fits to the region of $\log g$–T_{eff} space where hydrogen-atmosphere white dwarfs are so-far found to pulsate.

Each gray "$\times$" symbolizes a potential new ELM pulsator, and we will be observing these candidates currently from McDonald. Your project here will be to see if you can identify signs of variability in existing data on some of these stars.

Procedures

1. We have created a Jupyter Notebook that has one potentially useful function in it (the function "get_vals"):

   https://github.com/FRI-whitedwarf/PythonTutorials/blob/master/Lab2_ex2.ipynb

   Feel free to copy and paste this in your Jupyter Notebook as needed. Note: this file also contains comments (lines that start with "#") that give a shortened outline of your possible work.

   You may also want to refer back to last week's Python tutorial:

   https://github.com/FRI-whitedwarf/PythonTutorials/blob/master/fri_python_workshop.ipynb

2. And now, a word of advice. You will be using several tools that you may not have used before. For example: the `numpy` functions `std` and `mean`, the `matplotlib` function `scatterplot`, and the `python` functions `glob` and `append`. Or, even if you have used these functions before, you probably don't remember *exactly* how to use them. The simple answer to this is to do "Google" searches for these functions and read the documentation and examples for them.

   For instance, in this lab you may wind up typing the following google searches:

Google Search:	`numpy std`
Google Search:	`numpy mean`
Google Search:	`matplotlib scatterplot`
Google Search:	`python glob`
Google Search:	`python append`

   Try doing this before you ask for help from a mentor, so that when they come to help, you will have something to point at and talk about (i.e., the documentation or other info you've pulled up).

3. Use your Terminal commands to create/make a folder/directory (the "`mk`" unix command could be helpful) on the computer you are working on, either one of the Macs in the lab or your own personal computer (Note on terminology: folders and directories are the same thing). Let's call the directory "Lab2" since this is the second lab. Change into this directory ("cd Lab2"). Now, using a browser, download all the files from this location on our course's Canvas page: `AST 210K > Files > Labs > lab2`. There will be one jupyter notebook called `Lab2_ex2.ipynb` and a bunch of other files with names such as `Teff7822logg5.350.csv`.

   Inspect the "csv" files (perhaps try the "`more`" command). Each is a collection of comma separated times, magnitudes (logarithmic brightness), and magnitude uncertainty measurements (among other things). These measurements were made by the Catalina Sky Survey whose primary objective was to discover new comets and asteroids, but was also useful in other areas of astronomy.

4. Open up a new Jupyter Notebook in the directory where you want to work on this project (i.e., the directory into which you placed the data files). **Read through the rest of these instructions before continuing on** so that you have a good mental image of how we are tackling the problem of identifying probable white dwarf pulsators from a list of candidates. To help you out, we've also included an outline of how a successful program might be written **at the end of this document.**

   The plan forward consists of four rather incremental tasks that we lay out in the next four procedure steps. It follows reasonably to perform each of these tasks in four sequential Jupyter Notebook cells.

5. First you want to import all the extra modules that you will be using to complete your project. Import `numpy` (typically imported `as np`), `import matplotlib.pyplot` (typically `as plt`), `import glob` (explained below), and enter the following "magic" command on its own line,

   ```
   %matplotlib inline
   ```

   to enable plotting in the Jupyter Notebook environment.

6. In the next cell, let's read in the list of data file names ending in ".csv". The `glob` library includes a function "glob" that acts much like the `ls` command in the Terminal, in that it returns a list of files matching a description. For example, `glob.glob('*.txt')` would return a list with string elements containing the filenames of all .txt files in your current working directory. Use the `glob.glob()` function to read in the list of '.csv' files in this directory. Call this list something like "files" or "fnames". Note: if we called this list "files", then "files[0]" is the name of the first file in the list, etc.

7. In the next cell, let's practice by reading in the data *in* the first file. You will need to use the `loadtxt` command with the appropriate values set for `skiprows` and `delimiter`. You will want to store the columns labelled "Mag" (for the magnitudes) and "Magerr" for the uncertainties/errors on the magnitudes. You can call these lists ("arrays", actually) something like "mags" and "mag_errs" (you are free to use your own names, but make them something related to the data, not "Steve" and "Kevin", for instance!).

8. In the next cell, we want to define a function that can quantify and return just how much an object's data make it appear variable. Each of the data files you've been provided contains a sequence of brightness (magnitude) measurements for a candidate extremely low-mass white dwarf with an associated uncertainty (magnitude error). The scatter in a set of measurements can be characterized by the standard deviation, σ, calculated as:

   $$\sigma = \sqrt{\sum (x_i - \langle x \rangle)^2} \qquad (1)$$

   where x_i are each of the measurements and $\langle x \rangle$ is the mean (average) measurement.

   If the scatter in the data is due only to the measurement error, the standard deviation of the measurements should have a value very close to the average measurement error. If the star is variable with an amplitude greater than the measurement error, we expect to see a standard deviation notably larger than the measurement error.

Write a function that takes as arguments both a list of magnitude measurements and a list of magnitude uncertainties, and returns the ratio of the standard deviation in the magnitudes to the mean value of the measurement uncertainties. Use the `np.std()` and `np.mean()` functions to simplify your code.

Test your function on the data you just read in (e.g., "Mag" and "Mag_err") and see what ratio it outputs. This step is mainly to make sure that it doesn't return an error. If it does, then there is a bug in your code that you need to find.

9. In the next cell we will deal with the full list of files. You will want to write a `for` loop that goes through each file in the list. First, just have it print out the file name of each element in the list. Then go back and add in the function "get_vals()" that we gave you in the `Lab2_ex.ipynb` file, so that you can now print out each file name, each $T_{\rm eff}$, and each $\log g$ value. Now, go back and add in the `loadtxt` command, so that you read in the list of magnitudes and the errors on the magnitudes for each star. Finally, call the function that you wrote in item 8 above to calculate the extra variability that each object possesses.

   This will likely take some debugging to get it working. Keep after it until it does.

10. We did great work in the previous step, but we didn't save any of the values we calculated. In particular, we'd like to save both the $T_{\rm eff}$ and the variability parameter returned by the function you wrote above for each star, so that we can make a plot later. Note: To do this, you will first need to define (set up) empty lists to store these values. Create one empty list with a name for the $T_{\rm eff}$ values and one for the variability metric. Again, use sensible names (so that you have a chance of figuring out what you did if you were to read this a year from now!).

    So... **loop through the files and** `append` **those values to your lists!**

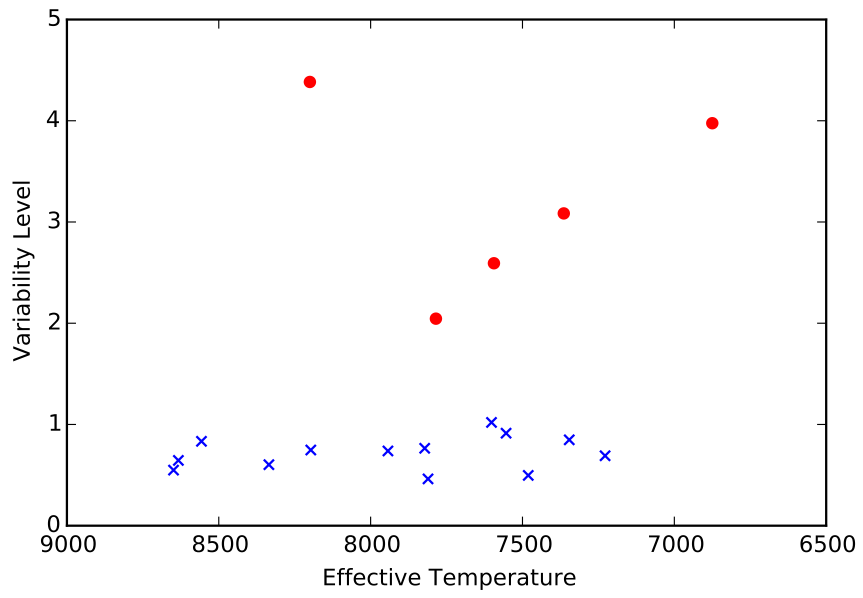
11. Finally, you'll want to make a pretty scatter plot of your results (`plt.scatter()`; use `shift+tab` in your Notebook to read the function documentation!). Since each of these candidate pulsators are near the same $\log g$ ($\sim 5-6$; bear in mind that $\log g$ on Earth is$\sim$3, so the gravitational acceleration experienced at the surface of these white dwarfs is extremely intense!), we'll see if we can localize the temperature range of the instability strip at this low-$\log g$ limit.

    Plot the level of variability (the value from your function) against $T_{\rm eff}$ for these stars. Oh, and since it's a convention to do so, plot the $T_{\rm eff}$ axis from high temperature on the left to low temperature on the right (e.g., see the Figure on page 1). You can set the x-axis limits with `plt.xlim()`.

    Let's consider (somewhat arbitrarily) that a value of 1.2 should be the limit between stars that do not show obvious variability in these data, and those that do appear to be variable. Plot the variable stars with a different symbol shape and color from the seemingly non-variable stars.

Label your axes with something sensible (`plt.xlabel()`, `plt.ylabel()`). Save a copy of your figure with the `plt.savefig()` command.

Your final plot should look something like this:



12. Submit your figure **and** a **PDF** copy of your Jupyter Notebook to Canvas. To create a PDF file of your notebook, go to the "File" tab/menu in the notebook and do "Print Preview" ("File > Print Preview"). When the preview shows up go to the Safari menu *in the bar at the top of the screen* and do "File > Export as PDF".

13. Great work this week! You are on your way to becoming a true Python master. Bask in the glory of a job well done...

Summary of Suggested Code Outline

Cell 1: Import all the modules that you use.

Cell 2: Define a function that takes a set of brightness measurements and their uncertainties as its two inputs and returns the the "variability level": the ratio of the measured standard deviation to the mean photometric uncertainty.

Cell 2b: (*Optional but smart*) Read in the data from one of the csv files that you transferred over and run it through your function, printing the result.

Cell 3: Set up empty lists for the values that you want to store: effective temperature and variability level.

Use the glob.glob function to get a list of the filenames of all of your data files.

Loop through the files in this list, reading in the data and appending the effective temperature (from the filename) and the variability level (from your function) to the lists that you already set up. (Make sure that you store the effective temperature as a numerical type and not as a string.)

Cell 4: Separate the stars into two groups: those with variability levels above the 1.2 threshold, and those below. Plot each group with a different symbol and color. Set the x axis to run "backwards" and label your axes. Save and display the figure.