ASTR 121 – Spring 2016

Lab 1b – Introduction to MATLAB (Part 2)

Important dates:

- Prelab due: Monday, [Date TBA]
- Part One due: Friday, [Date TBA]
- Part Two due: Friday, [Date TBA]

Goals:

At the end of this lab, you should be able to…

- Extract the orbital period of an exoplanet from radial velocity data using MATALB
- Generate data using a mathematical function and random numbers
- Read in data from a file
- Explore possible underlying mathematical models representing numerical data

In this second part of the lab, you will expand on your MATLAB experience by generating synthetic data and testing it against mathematical models. Making synthetic (fake) data is often useful for testing analysis procedures, such as in a telescope data reduction "pipeline." By feeding your procedure fake data with a known signal, you can test how successful your algorithm is at pulling out this signal. You will also learn how to plot errorbars and read data from a file. Data from a telescope or a computer simulation is often represented as columns of text in a file that can be read in by MATLAB for analysis. All of these skills will be useful for the "real" labs that follow.

Background

If you recall from ASTR120 (and our brief look at spectroscopic binaries this semester), a planet or other body orbiting a star causes that star to orbit around the system barycenter, and this motion can be detected by observing the periodic shift of absorption lines in the star's spectrum. If the orbit is circular, a plot of relative speed vs. time is a sine curve. In this exercise, you will generate a sine curve of known amplitude and period, add "noise" to it to simulate measurement error, and then use a tool to try to extract the orbital period from the synthetic data. Real exoplanet radial velocity curves are analyzed in a similar manner, but with a lot more steps!

What follows is the suggested approach to this exercise, with a minimum of detail. If you get stuck at any point, ask your fellow students or an instructor for guidance. You should definitely look at what other groups are doing and take or provide suggestions! Remember, this is not a contest—take your time and be sure you understand each step before going on to the next one. Suggestion: alternate between your partner and yourself at the computer each step.

Procedure: Generate Synthetic Exoplanet Data

1. Getting Started. First, start MATLAB and create a new folder for the files you'll be writing and using today. Remember to suppress unnecessary output in your scripts and functions, and to comment where appropriate.

2. Generate an Exoplanet Signal. Write a function that accepts an array of time values and returns an array of sine curve values with amplitude and period that you specify. To make this as realistic as possible, consider specifying amplitude in meters per second and period in days, since typical exoplanets (particularly "hot Jupiters") have amplitudes of tens of meters per second and periods of several days.

3. Plot the Signal. Write a script to call this function and plot it, remembering to label the axes appropriately. You may need to experiment with the number of data points and the total time interval to get something nice to look at. What you see now is a "perfect" signal with no measurement noise, which is not very realistic

4. Add Some Noise. Use the random function1 to add noise to your data. For example, you could add a random value between -0.2 and $+0.2$ times (or $\pm$ 20%) the amplitude of your curve to every value returned by your function (it's easiest to do this in your script, but feel free to get fancy). Hint: `random('uniform', -1, 1, 1, 100)` generates 100 uniform random numbers between -1 and 1. Replot the result: you now have a noisy signal!

5. Show the Uncertainty. Real data is usually shown with errorbars to indicate the uncertainty in the measurements. Of course, we made up our noise, but we can analyze it to show the typical deviation. Use the `std` function to find the standard deviation of the noise you added, then use `errorbar` to plot corresponding errorbars in the $\pm y$ direction on each point (the errorbars will be the same for each point, but you do need to provide them in an array for plotting). To make this look better, do not connect the points in your plot. If you need to adjust the plot limits because of the errorbars, use `xlim` and `ylim`.

6. Analyze the Signal. You will use the MATLAB's curve fitting tool `cftool` to analyze your synthetic data and attempt to extract the period from the noise. Simply type "cftool" in the command window to get started. A new window will open up. Using the "X data," "Y data," and (optionally) "Weights" drop-down menus on the left, load your time, radial velocity, and uncertainty arrays into the tool.

    - Notice that the tool is attempting to fit a 1-degree polynomial (a straight line) to your data, which is obviously not a great representation. Play around to see what model(s) work better. At least one of these will provide an estimate of the period of the signal. How well does it match the value you used to generate the synthetic data in the first place? Regenerate the noise several times (at least twice more) and take note of the period in each case—does it change much? The standard deviation of the fitted values gives a sense of how well `cftool` is able to fit your data—that is the precision to which you would report your fit in the next step.
    - For handing in, print out a plot of one instance of your synthetic data with errorbars, write the names of all lab partners on it, and note the input amplitude and period of the synthetic data and the value of the period determined using `cftool` for that case. Also report the standard deviation found for the repeated noise generation in the previous step, to 1 sig fig (the precision of your reported period should be consistent with this value). Also submit your m files on ELMS.

7. Useful Fact. If you restart MATLAB and run your script, then restart and run again, you'll see that your random points are the same. This is because MATLAB uses the same random number generator seed each time it starts (the numbers are not truly random!). To ensure you get a (likely) different set of numbers each time, you can reseed the generator "randomly" by adding `rng('shuffle')` at the start of your script. There is a lot of literature out there about random number generation with computers...

Procedure: Analyzing Unknown Data in a File

For this short exercise, you will read in column data from a file, plot it, then use `cftool` to deduce a model that adequately describes the data. You will often have to read in data from a file for science research, and being able to plot it up quickly and do a brief analysis of it can be very helpful for exploring the data and perhaps deducing an underlying model.

1. Getting Started. First, start MATLAB and create a new folder for the files you'll be writing and using today. Remember to suppress unnecessary output in your scripts and functions, and to comment where appropriate.

1. Get the Data. Download `lab1_sample.txt` from the Lab 1 folder under Files on ELMS for this course. Put the file in your current MATLAB directory. Open the file inside MATLAB to see that it consists of 3 columns of numbers: these represent the relative frequency, line strength, and uncertainty (all in arbitrary units), respectively.

2. Read and Plot the Data. Write a script to read in the data using `dlmread` (there are a number of ways to read data into MATLAB depending on how the data are organized— see the documentation for more!). Plot the data using `errorbar`. Hint: if you read the data into matrix M, then the nth column of the matrix is `M(:,n)` —you may want to assign new vectors (e.g., `frq, str, err`) to these columns for easier reference.

3. Explore the Data. Use `cftool` to explore possible fits to the data. Which functional form works best? Notice that some functions (like Polynomial) have adjustable parameters that may be worth exploring. Also note the "Center and scale" checkbox.

4. Generate the Fitting Function. With `cftool` you can also output the fit in a number of ways. Let's create something reusable and customizable. In `cftool`, choose File → Generate Code. A new function named `createFit` will appear in an editor tab. Either save the function with that name, or rename it and save it with the new name.

5. Plot the Fit. Go back to your original script where you read in the data and either delete or comment out the plotting commands. Then call the function that `cftool` made for you, passing in the data arrays. Voila! You should edit the function to change the labels back to something more meaningful. Print out the plot for handing in, indicating all lab partner names on it, and submit your m files on ELMS.