Gravitational-Wave Data Analysis
Exercises – Day 1

Peter S. Shawhan
CGWA Summer School, May 28, 2012

Web site with lecture slides, exercise data files, and solutions (posted after class):
tinyurl.com/GWAdata

0. General advice

These exercises are written with Matlab in mind. However, you should be able to do all of these things with other software such as Octave (very similar to Matlab), Mathematica, etc. Feel free to use one of those if you prefer.

I recommend writing a script to do each part rather than typing commands directly into Matlab. That way it's easier to go back and correct mistakes or incrementally improve your solution.

1. Matlab warm-ups

For today's exercises, assume that all time series have a sampling rate of 16384 Hz.
Remember that you can use the `helpdesk` (or type '`help <functionName>`' on the Matlab command line) for assistance, but also feel free to experiment!

a. Let's construct 1 second of data starting at t=0. Create a Matlab array (called '`t`', perhaps) containing the times of the data samples. It should have 16384 elements, the first being 0.0000, and the last being *slightly less than* 1.0000 .

b. Generate a time series which is the sum of a 74.7 Hz sine wave with amplitude 5 and an 820 Hz sine wave with amplitude 2 .

c. Use Matlab's `randn` function to generate one second of stationary, Gaussian, white noise with a standard deviation of 1.4 .

d. Add together your time series from parts b and c and plot the resulting time series (with the time in seconds on the *x* axis), with a title and axis labels. This is the time series you'll use for most of the rest of today's exercises. Also, zoom in on just the first 0.05 seconds of it.

e. Calculate the mean and RMS values of your time series from part d.

2. Fourier domain representation

a. Use Matlab's `fft` function to take the Fourier Transform of the time series from above, storing it in a new array.

b. Plot the periodogram, i.e. the squares of the amplitudes of the Fourier components, using a logarithmic *y* axis. Zoom in to get a closer look at the peaks in the periodogram. Can you explain why there are as many as there are, and why they have different widths?

c. Figure out Matlab's convention for arranging the different frequency components in the array generated by `fft`.

d. Check Parseval's theorem using your time-series and frequency-domain arrays.

e. Re-do parts a and b, but this time use a windowed version of your data. Specifically, apply a Hann window to your data (multiply it by $(1-\cos(2\pi t))/2$), plot that, and *then* do the fft. Zoom in on the peaks in the periodogram to see how they have changed.

3. Power spectra

a. Read the information about Matlab's `pwelch` function, then use it to estimate the power spectrum for your time series. Is this a one-sided or a two-sided power spectrum? [`pwelch` is part of Matlab's Signal Processing Toolbox. You can find the `pwelch` documentation on the web here: http://www.mathworks.com/access/helpdesk/help/toolbox/signal/pwelch.html .]

b. Zoom in to get a closer look at the peaks. How do their widths compare to what you saw in the periodograms you plotted in item 2?

4. Filtering

a. Apply a simple first-difference filter, i.e. $y_k = x_k - x_{k-1}$, to your time series, and plot the output time series.

b. Calculate analytically what the transfer function of this simple filter should be.

c. Use `pwelch` to estimate the power spectrum for the filtered time series, and compare it to what you got in section 3b.

d. Butterworth filters are a pretty standard class of all-purpose IIR filters. Figure out how to use Matlab's `butter` function to design a 4^{th}-order Butterworth band-pass filter to select the frequency range 65 to 85 Hz; that produces the a_k and b_k coefficients that define the filter. Then use Matlab's `filter` function to apply it to your original data. (Note: be careful about what N you pass to the `butter` function to specify the filter order. Matlab uses that N differently for band-pass filters than it does for high- or low-pass filters!)

5. Down-sampling [if you have time]

a. Returning to your time series from part 1, try down-sampling it by simply selecting every 16^{th} sample. (This produces a time series with 16384/16 = 1024 samples.) Calculate the RMS for this filtered time series; how does it compare with the RMS for the original?

b. Now return to the original time series and apply a low-pass filter with a corner frequency of 512 Hz, and THEN select every 16^{th} sample. What is the RMS of this down-sampled time series now that a decent anti-alias filter has been used prior to down-sampling?