Gravitational-Wave Data Analysis Exercises – Day 3

Peter S. Shawhan
CGWA Summer School, May 30, 2012

Today you'll continue to track down the inspiral signal that you searched for yesterday, using the same data file (`day2data.txt`) and template (`day2template.txt`) but different filtering methods. Recall that the sampling rate for these files is 16384 Hz.

1. Matched filtering in the time domain [if you didn't do this yesterday]

a. Calculate the correlation between the (original unfiltered) data and the template for all possible offsets of the template. You will probably want to use a '`for`' loop to do this, and inside the loop take the dot product of the template with a section of the data. Store the results of the correlation for each offset in an array (one correlation value per offset). This takes a long time! It will help to pre-create the correlation array, e.g. by doing:
```
corr=zeros(1,length(data)-length(template));
```

b. Plot the correlation array that you just calculated. Do you see any evidence for a signal?

2. Down-sampling [if you didn't do this yesterday]

So far, you've looked for a signal whose frequency band is far below the Nyquist frequency. In other words, the sampling rate is much higher than it needs to be, and you can down-sample (or "decimate") the time series to reduce the amount of data you have to work with.

a. Calculate the RMS for your original unfiltered data and record it for later reference.

b. Try down-sampling your time series by simply selecting every 16^{th} sample. (This produces a time series with 262144/16 = 16384 samples with a sampling rate of 16384/16 = 1024 Hz.) Calculate the RMS for this filtered time series; how does it compare with what you found in part a? Do you understand why?

c. Before resampling, you really should use an anti-alias filter to remove all of the noise power at frequencies above the new Nyquist frequency. So, construct a 6^{th}-order Butterworth low-pass filter with a cutoff frequency of 512 Hz and apply it to your original time series, and then select every 16^{th} sample from the output time series. Plot the resulting time series.

d. Calculate the RMS for this properly down-sampled time series; how does it compare with what you found in part a?

e. Use `pwelch` to estimate the PSD for your down-sampled time series. Note that the frequency scale is different from before because the Nyquist frequency has changed.

f. Down-sample your template time series too, and repeat the time-domain matched filtering (section 1 from today) with the down-sampled data and template. (This should go much faster than before!) Plot the output. The signal should stand out as clearly as it did before.

3. Matched filtering in the frequency domain

a. Calculate the FFT of the original data time series.

b. You also need the FFT of the template. However, you first need to extend the template to have the same length as the data time series so that its FFT will have the same frequency spacing. You can simply extend it with zeros; we call this "zero-padding". Then calculate the FFT of the extended time series.

c. Multiply (bin-by-bin) the FFT of the data by the complex conjugate of the FFT of the template, and do an inverse FFT using Matlab's '`ifft`' function. This gives you the filter output time series.

d. Plot the output time series. How does it compare to what you got from time-domain matched filtering? How much faster was it to compute?

4. Optimal matched filtering using frequency weighting

The matched filtering you did in the previous parts is optimal only if the noise is white, which isn't true in this case. To do optimal filtering, you have to use the power spectrum of the noise. In the real world, we use additional data from nearby times to estimate the PSD of the noise, and we use a median (not an average) of several periodograms to reduce the uncertainty while avoiding a biased estimate if there is a strong signal in addition to the noise (which there is in this case!). Since you don't have any additional data, I'll simply tell you the PSD I used to generate the noise:

$$S_n(f) = (0.09) \times \left| \frac{f}{40\,\mathrm{Hz}} \right|^{-12} + (0.09) \times \left| \frac{f}{40\,\mathrm{Hz}} \right|^{-2} + (0.01) \times \left| \frac{f}{120\,\mathrm{Hz}} \right|$$

In addition, I cut off the noise below 20 Hz, since the expression above goes to infinity as $f \to 0$. Constructing the frequency-domain array with this function, following the Matlab convention with "negative frequencies" to the right of the positive frequencies, is kind of tricky, so I have done it for you and put the final PSD estimate into a file:

a. Get a copy of the file `day3psd.txt` and `load` it into Matlab.

b. Multiply (bin-by-bin) the FFT of the data by the complex conjugate of the FFT of the template, and then divide by the PSD array that you just loaded. Now do an inverse FFT to get the filter output time series.

c. Plot the output time series. How does it compare to what you got from "non-optimal" filtering in the time domain and in the frequency domain? Do you understand why?

d. What is the signal-to-noise ratio in the non-optimal case and in the optimal case?

e. What if you estimate the PSD from this data (which contains the signal) rather than using the function I gave you? How much does that degrade the signal-to-noise ratio?