CGWAS Exercises - GW Burst Analysis

July 8 2015

DYI excess power algorithm

The first stage in any burst detection algorithm is an event trigger generator (ETG) that identifies discrete excess power in time and frequency. You will code up the first stages of your own excess power ETG using h(t) data from the LIGO interferometers taken during the previous science run (S6), courtesy of the LIGO Open Science Center (losc.ligo.org).

The following code instructions use python and the GWpy package. If you do not have this installed on your computer already, you should make use of the CGWAS virtual machine.

A note of caution before starting: if you copy and paste code snippets directly from this pdf you will get syntax errors for nonstandard symbols - especially for single quotes. Try typing things out. Tab complete should work for many things.

Step 1 - Grab data

We will be downloading S6 LIGO data from the LIGO Open Science Center.

Go to `https://losc.ligo.org/timeline/show/S6/H1_BURST_CAT1*L1_BURST_CAT1/964650354/505/1/`.

Click on 'Download this data'.

We'll look at H1 first - click on 'H1' after 'Get strain data'.

Click on 'Frame'. If you're using QupZilla on the CGWAS virtual machine, you'll get a pop up with the option to save the file. Do so, and choose to save in the default location (cgwas). This will save the file in the same directory you're ported to when you open a new terminal with the virtual machine.

If you take some other path to download the data, make note of the file path or move the file somewhere convenient.

Open an ipython session from a terminal:

```
$ ipython
```

Read in the data using the following time interval and code (note this assumes the data file is stored in your current directory):

```
from gwpy.timeseries import TimeSeries
data = TimeSeries.read('H-H1_LOSC_4_V1-964648960-4096.gwf',
   'H1:LOSC-STRAIN', start=964650450, end=964650505,
   format='lalframe')
```

This might take a minute. Note that tab complete also works for the name of the file.

Step 2 - Plotting a first glimpse

You now have a h(t) time series stored as 'data' in your ipython session. Next we'll crack it open and see what's there.

First, plot a time series.

```
TSplot = data.plot()
TSplot.set_ylabel('Gravitational-wave strain amplitude')
TSplot.set_title('S6 LIGO Hanford h(t) data')
TSplot.show()
```

A figure window should pop up. If you want to save it or interact with it you can use the figure tools in the lower left.

Do you spot obvious excess power features in this time series? If so, when? (Note the time after 964650450 in seconds.)

Step 3 - 'Tiling' the data in frequency

Next we'll use multiple bandpass filters targeting different frequency ranges[1]. We'll design our bandpass filters using scipy's Butterworth filter function, scipy.signal.butter[2].

First, import the required function:

```
from scipy.signal import butter
```

Next we define a new function that intakes 1) our lower and upper corner frequencies, 2) the data sampling frequency, and 3) the desired filter order and outputs the zeros, poles, and gain of the appropriate IIR filter transfer function[3].

(When you're inputting this to ipython, remember that consistent indentation within a loop or function definition is important. Also, setting order=2 makes second order filters the default.)

[1] A bandpass filter acts just as it sounds - a band of frequency is allowed to pass (between two *corner* frequencies) and all other frequencies are filtered out. The aggressiveness in the shape of the filter is controlled by the filter order.

[2] Butterworth filters are a good choice because they are maximally flat in the 'pass' frequency region, whether for a bandpass, lowpass, or high pass filter.

[3] An IIR, or infinite impulse response, filter is a digital filter that generally gives a better frequency response with the same order filter than an FIR, or finite impulse response, filter.

```
def butter_bandpass(lowcut, highcut, fs, order=2, output='zpk'):
     nyq = 0.5 * fs
     low = lowcut / nyq
     high = highcut / nyq
     b, a = butter(order, [low, high], btype='band')
     return b, a
```

And we need define another new function to apply this to the data. This function should intake some data, the desired bandpass corner frequencies, and the filter order. Then we want it to apply the appropriate filter as defined in our previous function and return the filtered data.

```
def butter_bandpass_filter(data, lowcut, highcut, order=2):
    filt = butter_bandpass(lowcut, highcut,
         data.sample_rate.value, order=order, output='zpk')
    return data.filter(*filt)
```

Now we can simply call butter_bandpass_filter to apply an arbitrary bandpass filter to the data.

Let's first try plotting the data after applying a second order bandpass filter with corner frequencies at 1250 and 1350 Hz.

```
filtdata = butter_bandpass_filter(data, 1250, 1350, order=2)
filtplot = filtdata.plot()
filtplot.set_ylabel('Gravitational-wave strain amplitude')
filtplot.set_title('S6 LIGO Hanford h(t): 1250-1350 Hz bandpass')
filtplot.show()
```

You may need scale the y-axis some to make it clear. This can be done using set_ylim:

```
filtplot.set_ylim(-1e-19, 1e-19)
filtplot.refresh()
```

Make note of any excess power features that are newly apparent.

Next, plot the data using a second order bandpass filter with corner frequencies at 500 and 600 Hz. Make note of any excess power features that

are newly apparent.

Note: if you play around with these filters further be careful if you increase the filter order. There's a known bug in scipy that can make higher order filters unstable (to be fixed soon in scipy 0.16!).

Step 4 - A more complete picture

Using a normalized spectrogram will make discrete bursts of excess power much easier to see than the unfiltered time series from step 1.

```
specgram = data.spectrogram(2, fftlength=1, overlap=0.5) ** (1/2.)
medratio = specgram.ratio('median')
specplot = medratio.plot(norm='log', vmin=0.1, vmax=10)
specplot.add_colorbar(label='Amplitude relative to median')
specplot.set_title('S6 LIGO Hanford h(t): normalized spectrogram')
specplot.show()
```

What excess power features do you see now? Estimate the time after the start time (in seconds) and the central frequency (in Hz). Compare this with features you'd identified previously.

Warning: do not close this figure if you're planning to complete the next advanced step, or look at Livingston data for coincident excess power. Otherwise you'll need to repeat your work from the first plot step (the up arrow should make this less painful, if it happens.)

Advanced step: Design your own filters

Design additional bandpass filters that will target any other excess power features you identified in your normalized spectrogram. You should be able to make effective filters by changing the filter corner frequencies and perhaps the filter order of butter_bandpass_filter. Test them by applying them and verifying a feature emerges where you would expect in time.

Some advice: start with the higher frequency events first for practice and skip the events below 200Hz altogether unless you're up for a challenge (in navigating what is less than optimal about scipy's filtering).

Note: if you see a spike in the first second or so of data, that's probably an edge effect from a wonky scipy filter. Ignore this spike.

Looking for coincident excess power

If you have time, repeat steps 1, 2, and 4 for Livingston data from the same time. (The data can be downloaded from the same link.) Can you identify any burst events coincident in time and frequency?

Troubleshooting

If you see this error: TclError: this isn't a Tk application
that probably means you closed a figure and then tried to show or refresh it. You need to go back to the line where you plotted the data (i.e. plot = data.plot()) and start from there. (It's best not to close figure windows if you think you may want to alter them or view them again.)

Additional information and resources

The python package you just used, GWpy, is a very handy way to access and interact with LIGO data and analysis triggers. Check out the documentation here: `https://gwpy.github.io/docs/stable/index.html`.

Another great resource to easily find and download LIGO data from recent science runs is the LIGO Open Science Center (which you also used in this exercise). You can find more tutorials on using LOSC data here: `https://losc.ligo.org/tutorials/` or ask Jonah Kanner while you're at Caltech.