Statistics, Algorithms and Experimental design Final Assignment

In this exam we will analyze LIGO data to find the gravitational waves transient, caused by the coalescence of two neutron stars (GW170817). You are given a true 2048-second segment of Hanford LIGO data, sampled at 4096 Hz (down-sampled from the original data, which is at 16kHz). Along with this PDF, you should have

1. (`strain.npy`), readable by `NumPy`, containing the strain.
2. `gw_search_functions` containing helpful functions, constants, and skeletons for functions for you to complete. It is provided both as `.py` and `.txt` formats, to avoid uploading issues.
3. The timestamps corresponding to the strain are not uploaded, due to size, and are instead provided in `gw_search_functions`.

***Submission Instructions**: Submit a single PDF file and a single `Python` file. Include your name in the file names. In each section, the object that is required to include in your PDF (plot, number etc.) is written in bold. If you think of any other statement, figure or code that will help the grader examine your work, please include it. Submit the Python file with functions completed (if you used them), or other functions you find important.*

Introduction: data model and SNR calculation

Under the null hypothesis and signal hypothesis, the data model is

$$H_0 \ : \ s(t) = n(t) \tag{0.1}$$
$$H_1 \ : \ s(t) = n(t) + h(t) \tag{0.2}$$

The noise $n(t)$ is approximately stationary and Gaussian with a certain power spectral density $S_n(f)$. This is only approximately true, for two reasons. First, the spectral shape changes smoothly on a time scale of a few seconds. Second, there are "glitches" in the data, which are unexplained time-localized loud noises. Under the Gaussian noise approximation, the likelihood of waveform h at strain data s is:

$$\ln \mathcal{L} = \Re\langle h, s\rangle - \frac{1}{2}\langle h, h\rangle \tag{0.3}$$

with the inner products $\langle \cdot, \cdot \rangle$ mean:

$$\langle a, b\rangle = \sum_f \frac{a(f)b^*(f)}{S_n(f)}\mathrm{d}f = \sum_f \tilde{a}(f)\tilde{b}^*(f)\mathrm{d}f \tag{0.4}$$

where the tilde stands for the series with whitened filter applied.

The strain signal at the detector is a sum of 2 polarizations, weighted by the detector response to each polarization:

$$h(f) = F_+ h_+(f) + F_\times h_\times(f) \tag{0.5}$$

when, under some simplifications and approximations, the plus and cross polarizations are related to each other by

$$h_\times(f) = i h_+(f) \tag{0.6}$$

Thus, the detector responses can be treated as a complex amplitude and maximized over. Following this logic, we define the complex-overlap time-series as a combination of two real overlap time-series. Using the "inverted convolution" notation for the correlation:

$$z(t) = z_{\cos}(t) + i z_{\sin}(t) = (\tilde{s} * \overleftarrow{\tilde{h}_+})(t) + i(\tilde{s} * \overleftarrow{\tilde{h}_\times})(t) \tag{0.7}$$

Using a normalization such that

$$\langle h_+ | h_+ \rangle = 1, \tag{0.8}$$

the Signal-to-Noise (SNR) timeseries is

$$\mathrm{SNR}^2(t) = |z(t)|^2 = |z_{\cos}(t)|^2 + |z_{\sin}(t)|^2, \tag{0.9}$$

and the log-likelihood is

$$\log \mathcal{L} = \frac{\mathrm{SNR}^2}{2} \tag{0.10}$$

Questions

1. Load the time domain data and Fourier transform it.
2. Estimate the ASD and create the whitening filter. **Create a log-log plot of the ASD from 20Hz onward**. **Create a log-log plot of the whitening filter from 20Hz onward. Plot the entire whitened strain data**.
3. Create a single template for a search, with arbitrarily selected masses of $m_1 = 1.5$ and $m_2 = 1.25$ (in solar masses). **Plot the time-domain template, such that it is localized in the middle of the time-axis. Fix the plot such that the waveform features are visible**.
4. Generate the complex-overlap time-series. **Plot a histogram with the real and imaginary parts of the complex-overlap, in a segment of data without an obvious glitch. Overlay the theoretical predictions.**
5. Create the SNR^2 time series. To verify your results, use the estimated ASD to draw mock data without a GW transient. Create the same SNR^2 time-series on this data. **On the same figure, plot the histograms of the SNR^2 of the real data and of the mock data. Overlay the theoretical prediction**.
6. Create a test-statistic to detect glitches. Use it to remove glitches from the SNR^2 timeseries. **Plot the cleaned SNR^2 timeseries histogram. Overlay the theoretical prediction**.

7. Now you know how to conduct a search with a single template. We now go on to prepare a bank of templates. The first step is to find a good linear basis to work with. Draw 2^8 mass samples (see given functions). Create the waveform for each and perform SVD to find their phase basis-vectors. Choose the number of basis-vectors you want to use. **Plot the basis-vectors against frequency.** Make sure to read the guidance before performing the SVD.
8. Calculate the inner product between waveforms at different coordinate-distance ($\sum_\alpha |\Delta c_\alpha|^2$). **Plot their overlap against their distance, and the theoretical prediction**.
9. Generate a template bank for the search. **On the same plot, present the relevant region in coordinates space and a scatter plot of the coordinates of the templates in the bank**. Refer to the details for a suggested algorithm for this task.
10. Repeat the search (sections 4-6, without repeating their plots) for each template in the bank individually (including glitch-removal). For each interval of 0.1 seconds, record which template gave the maximal SNR, and what was that SNR. **Plot the time-series of maximal SNR^2 in per 0.1 seconds. Plot a histogram of the maximal values per 0.1 seconds** *Before using the entire bank, try a small subset and see that the results make sense. The entire search could take several minutes, depending on hardware.*
11. If you detected an event, **report its time, the masses of the template and an estimation or a upper bound of the false-alarm rate for such SNR**. Consider the number of templates you used and the fact that waveforms have typical auto-correlation length of 1 ms.
12. **Create a spectogram (using e.g. `matplotlib.pyplot.specgram`), localized in time and frequency around the event you found**. This may take several iterations and manually fixing the parameters of the `specgram` function, see guidence below.

Comments

Runtime

All code in this exam should not take a long time to run. End-to-end, it should take about 5 minutes, with more than half of the runtime dedicated to the search itself (see Section 11).

FFT and RFFT

Since the time domain data and waveforms are real, use `rfft` and `rfftfreq` to work with only positive frequencies. The inverse, `irfft`, enforces real output, so constructing complex time-domain signals (e.g., Eq. (0.7)) requires careful handling.

Normalizations

A large portion of this exam involves computing and comparing test statistic values. Using incorrect normalization can significantly distort the results. Be deliberate when applying normalization conditions (e.g., Eq. (0.8)). Monitor the resulting values, including their means and variances, to ensure the computations are consistent and correct.

Guidance

1.

To avoid edge effects, before performing the (real) FFT of the data, apply a smooth window, meaning multiply by a function that gradually zeros at the ends. It is recommended to use a Tukey window with α parameter 0.1 (see `scipy.signal.tukey`).

2.

Use the Welch method (see `scipy.signal.welch`) to estimate the ASD of the data. This function divides data into slices, performs an FFT on each, and averages over the squared absolute value per frequency. Use ~64-second slices, with 50% overlap. Use median averaging to be less sensitive to outliers (such as glitches). The function will apply a window to the slices by default.

Note that the estimated ASD frequency resolution is defined by the window length. Interpolate it (in frequency domain) so it will be defined on the same frequency axis as the strain.

The whitening window is approximately the inverse of the ASD, except it is also high-passed above 20Hz. The high-pass filter should be smooth, e.g. zero below 20Hz, one above 21Hz, and gradually increasing between 20Hz and 21Hz like sine-squared.

3.

Use the code given to create a single template of the plus-polarization waveform:

$$h_+(f) = A(f)e^{i\Psi(f)}, \tag{0.11}$$

with masses $m_1 = 1.5$ and $m_2 = 1.25$ (solar masses) and zero-spins. The amplitude is

$$A(f) \propto f^{-7/6} \tag{0.12}$$

and the phase is given by a sequence of power-laws, with coefficients given by the post-Newtonian approximation:

$$\Psi(f; m_1, m_2) = \sum_i f^{p_i} b_i(m_1, m_2) \tag{0.13}$$

The $p_i = 0, 1$ powers are related to somewhat arbitrary time-convention and orbital-phase convention. Make sure the waveform is well localized by seeing that the peak of the waveform time-series is roughly at the beginning or the end of the time-series.

4.

See equations (0.5)-(0.9) for details about the complex-overlap timeseries and SNR.

5.

Random data can be drawn from the PSD $S_n(f)$ by drawing complex $n(f)$ such that

$$\Re(n(f)) \sim \mathcal{N}(0, S_n(f)/2) \tag{0.14}$$

$$\Im(n(f)) \sim \mathcal{N}(0, S_n(f)/2) \tag{0.15}$$

Make sure to use proper normalization. The best way to make sure is to estimate the PSD from the mock-data, and see the resulting PSD is approximately the same as the PSD you used.

6.

Glitches are short periods of time with strong power, not coming from the stationary noise nor an astrophysical GW transient. Since their shape is not related to the shape of GW transient, they will fail a signal-consistency test. This test is defined per-template. We will create a $h_{\rm low}$ and $h_{\rm high}$:

$$h_{\rm low}(f) = \begin{cases} h_+(f) & f < \bar{f} \\ 0 & f > \bar{f} \end{cases} \tag{0.16}$$

$$h_{\rm high}(f) = \begin{cases} 0 & f < \bar{f} \\ h_+(f) & f > \bar{f} \end{cases} \tag{0.17}$$

where $\bar{f}$ is defined as the mid-point of the template accumulated SNR^2:

$$\sum_{f=0}^{\bar{f}} \frac{|h_+|^2}{S_n(f)} \mathrm{d}f = \sum_{f=\bar{f}}^{f_{\rm max}} \frac{|h_+|^2}{S_n(f)} \mathrm{d}f \tag{0.18}$$

$h_{\rm low}$ and $h_{\rm high}$ are normalized to have unity norm ($\langle h_{\rm low}|h_{\rm low}\rangle = \langle h_{\rm high}|h_{\rm high}\rangle = 1$). This means that their complex-overlaps $z_{\rm low}(t)$, $z_{\rm high}(t)$ should be complex-normal random variables with variance of 1. The glitch-test $g(t)$ is defined as:

$$g(t) = \frac{1}{2}|z_{\rm low} - z_{\rm high}|^2(t) \tag{0.19}$$

Under the noise hypothesis or under signal consistent with h_+, g follows a $\chi^2(2)$ distribution. In the presence of a glitch, $z_{\rm low}$ and $z_{\rm high}$ will have large amplitudes and different phases, which will lead to a large $g(t)$.

To mark an element of the timeseries as a glitch, it has to both have SNR^2 larger than some value, which you will set by observing the SNR^2 histogram, AND that $g(t)$ has false-positive (probability to reject a measurement under the no-glitch hypothesis) of 1%.

7 & 8.

Here are instructions on how to create phase-unit vectors and coordinates from a moderately large set of samples. Use the functions given to draw mass samples and convert them into post-Newtonian coefficients, and to phases. The outcome will define a template-bank for a search. While an actual search usually includes many different banks, we will focus on a single one.

The main idea is that the h_+ waveform can be represented by an amplitude and phase (equations (0.12) -(0.13)). For a small enough region of parameter space, the amplitudes are the same and phases are a combination of a common phase evolution and some deviation, that can be written as a linear combination of orthonormal phase functions:

$$\Psi_i(f) = \bar{\Psi}(f) + \sum_\alpha c_\alpha \psi_\alpha(f) \tag{0.20}$$

where the common-evolution is the mean over samples per frequency, and orthonormality is defined with weights that are the whitened-amplitude:

$$\sum_f \frac{A^2(f)}{S_n(f)}\psi_i(f)\psi_j(f) = \delta_{i,j} \tag{0.21}$$

Given two normalized templates, $h_i(f)$, $i = 1, 2$, the match between the templates is

$$\langle h_i|h_j\rangle = \sum_f \frac{A_i A_j}{S_n(f)} e^{i(\Psi_1(f)-\Psi_2(f))}\mathrm{d}f \tag{0.22}$$

To second order in $\Delta\Psi = \Psi_i - \Psi_j$:

$$\langle h_i|h_j\rangle \approx \sum_f \frac{A^2(f)}{S_n(f)}\left(1 + i\Delta\Psi(f) - \frac{1}{2}(\Delta\Psi(f))^2\right)\mathrm{d}f \tag{0.23}$$

Removing a constant from the phases will not change any SNR calculation. Therefore, we will project out a constant term from each $\Psi_i(f)$:

$$\Psi_i(f) \to \Psi_i(f) - \sum_f \frac{A^2(f)}{S_n(f)}\Psi_i(f)\mathrm{d}f \tag{0.24}$$

This transformation will nullify the $i\Delta\Psi(f)$ term in (0.24). Similarly, any linear ($\propto f$) component in the phase can be attributed to a time-shift, without changing the values of the SNR^2 timeseries, only their time-position. A function that removes the constant and linear component is provided: `gw_search_functions.phases_to_linear_free_phases`.

Now, the imaginary component disappears, and the second-order term can be written using the orthonormal basis:

$$\langle h_i|h_j\rangle \approx 1 - \frac{1}{2}\sum_\alpha (\Delta c_\alpha)^2 \tag{0.25}$$

We can create an orthonormal basis by taking the phases of the samples, and performing an SVD on the matrix X (which has the shape of number of samples times the number of frequencies):

$$X_{i,j} = \Psi_i(f_j) \cdot \frac{A(f_j)}{S_n^{1/2}(f_j)} \tag{0.26}$$

Just like we applied the weights to turn the phase into a linear basis, we will need to divide by the weights to turn the linear basis to the wanted ψ_α. Looking at the eigenvalues would make it clear how many new coordinates we'll need to describe the waveforms.

When implementing this section in code, please note that the phase evolution is rather smooth. You can reduce the complexity without introducing large errors by working in a frequency grid that starts at 20Hz and has increments, say, 2^{-4} Hz. Performing SVD with the full frequency grid could be too much for a reasonable computer.

9.

We want to construct an actual bank of templates to search with. We want the bank to cover the entire coordinate space, but without being wastefully dense. There is no point in searching with two templates that are too similar. It is possible to do it by creating a regular grid in the coordinate space, and constraining the templates to the relevant regions. You are free to do it, if you wish. Another way to do it is presented below.

Draw 2^{13} samples of masses. Create their phases, remove the common phase evolution (use the same one found in the previous section) and find their coordinates (i.e. the projection of their phases on the phase vectors, as done in the previous sections). Then use the c_α distance between points as a measure of similarity. Iterate over the samples. Add a sample to the bank if it is far enough from all other samples already in the bank:

$$d_i^2 = \min_{j \in \text{subset}} \sum_\alpha \left(c_\alpha^{(i)} - c_\alpha^{(j)}\right)^2 \geq 0.1 \tag{0.27}$$

It can be shown that this number corresponds to a maximal 10% decrease in SNR^2 due to using the wrong template. For the SNR^2 values expected here, this is good enough.

If you choose to create the bank using this method, you can plot the result by overlaying two scatter plots: the first with all 2^{13} samples, with some transparency to the markers (e.g. `alpha=0.5`, see `matplotlib.pyplot.plot`), and another for the bank, with non-transparent markers.

10.

It is recommended to use `tqdm` to create a progress bar when performing a `for-loop` of initially unknown completion time.

```
from tqdm import tqdm

for i in tqdm(range(1000)):
    pass
```