A Gravitational-Wave Data Analysis Primer for the IndIGO Mock Data Challenge

P. Ajith,[1,*] Satya Mohapatra,[2,†] and Archana Pai[3,‡]

[1] *LIGO Laboratory and Theoretical Astrophysics, California Institute of Technology, Pasadena, CA 91125, USA*
[2] *University of Massachusetts, Amherst, MA 01003, USA*
[3] *Indian Institute of Science Education and Research, Thiruvananthapuram, India*

(Id: DAPrimer.tex 88 2011-08-24 15:14:03Z ajith ; August 24, 2011)

This is an introduction to the gravitational-wave (GW) data analysis for the participants of the IndIGO Mock Data Challenge.

I. INTRODUCTION

The General Theory of Relativity predicts the existence of gravitational waves (GWs). Generation of GWs is analogous to the generation of electromagnetic waves: while changes in the electric field (acceleration of charges) produce electromagnetic waves, changes in the gravitational 'field' (acceleration of masses) produce GWs.

Although any (non-spherical) accelerated motion of masses can produce GWs, those produced by the motion of terrestrial sources (or "non-relativistic" astronomical objects such as planets) are too weak to be detectable by any conceivable technology. Indeed, gravitation is the weakest long range force of nature, which is evident from the smallness of G! Still there exist a number of astronomical sources that can produce GWs that are detectable using the current cutting-edge technology. These include violent astrophysical phenomena such as the coalescence of black-hole binaries, gravitational collapse of massive stars resulting in supernovae, rapidly rotating neutron stars etc., and various energetic processes that might have happened in the early Universe. By decoding the emitted GW signal, it is possible to extract the physical properties of the source, such as the component masses, spins, distance and energetics. Thus, GW astronomy will complement the electromagnetic astronomy bringing new information about our Universe. It is fair to say that GW astronomy will open a completely new window to the Universe.

In astronomy, the sky is viewed in different bands of the electromagnetic spectrum ranging from radio waves (frequency $\sim 10^8$ Hz) to gamma-rays (10^{20} Hz), spanning 12 orders of magnitude in frequency). The GW frequency spectrum covers an impressive 20 orders of magnitude — ranging from 10^{-16} Hz (produced by processes in the early Universe such as inflation) to $10^3 - 10^4$ Hz (produced by supernovae, neutron-star oscillations etc.). Just as in the electromagnetic astronomy, the detection and analysis methods used in GW astronomy are different in different frequency bands.

When GWs pass through the Earth, they distort the geometry of the space-time. Observing the tiny distortions — the "strain" — in the space-time geometry is the key to the detection of GWs. Laser interferometry provides a precise method for measuring such small deformations. In a laser interferometer of arm-length L , a coherent laser beam is split by a beam splitter and sent in two orthogonal directions. These beams are reflected back by two mirrors, which are in turn recombined to produce an interference pattern. Gravitational waves induce a relative length change δL between the two orthogonal arms of an interferometer, which produce a change in the interference pattern. The output is recorded in terms of the strain ($h = \delta L/L$) produced by GWs.

An international collaboration of scientists with diverse expertise is involved in this GW hunt using km-scale arm-length ($L \sim$ kms) interferometers as GW antennas. These detectors — LIGO [1], Virgo [2], GEO 600 [3], TAMA 300 [4] – are among the most sensitive measurement devices ever constructed by

*Electronic address: ajith@caltech.edu
†Electronic address: satya@physics.umass.edu
‡Electronic address: archana@iisertvm.ac.in

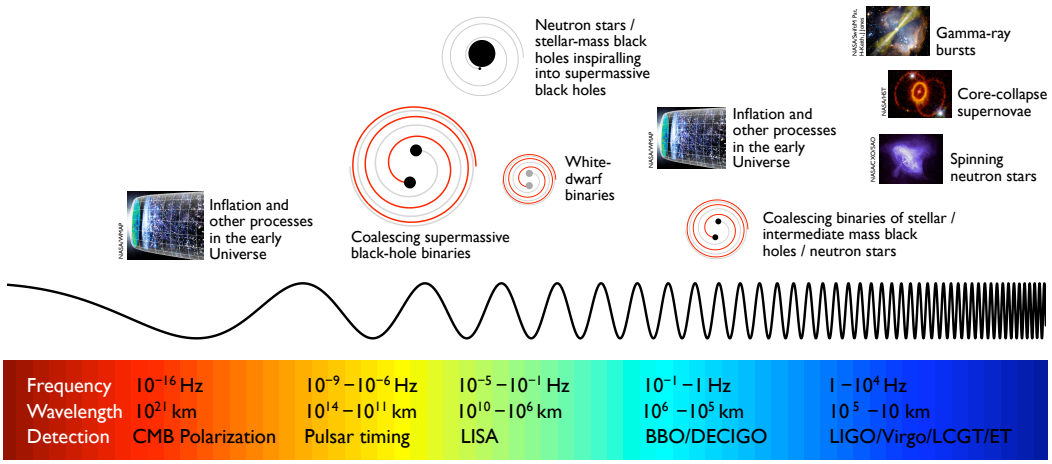

FIG. 1: An overview of the GW spectrum, sources and detectors in different frequency bands.

(wo)mankind. These detectors are sensitive in the frequency band of 10 Hz to a few kHz. Astronomical sources that can produce detectable GWs in this frequency band include the coalescence of neutron-star or stellar-mass black-hole binaries, galactic supernovae, rapidly spinning neutron stars etc. Figure 1 gives an overview of the GW spectrum.

II. GRAVITATIONAL-WAVE DATA ANALYSIS

Unlike electromagnetic astronomy, GW astronomy will not be producing images but would give out a time-series data where a signal or many signals would be hidden in the interferometer's noise. The challenge would be to identify the signal (if any), separate the signal, and to locate the source. It is more of a "finding the needle in a haystack" problem. It is this nature which makes GW astronomy unique as it requires novel algorithms, and most of the astronomy is done on computers. In this exercise, we will give a flavor of what a GW astronomer does! We will be focusing on a particular class of GW sources, called inspiralling compact binaries.

A. Gravitational-wave signal from an inspiralling compact binary star

Among the most promising sources of GWs for the ground-based interferometric antennas are binaries of compact stars such as black holes and neutron stars. Neutron stars and black holes are highly dense objects: A neutron star with mass equal to that of the Sun will have a radius of around 15 km, while a black hole with same mass will have a radius of 3 km. Thus, such objects can be treated as point particles when the orbital separation is large.

Consider a system of two compact stars with masses m_1 and m_2 in a circular orbit. Such a system emits GWs due to its non-spherical kinetic energy (via time varying quadrupole moment). Due to the loss in the energy, the stars inspiral to each other and the orbital frequency of the system increases with time following Newtonian mechanics; in particular Kepler's laws. The inspiral continues due to the continuous emission of GWs. The emitted gravitational waveform is a "chirp" waveform (similar to the chirping of birds) with both amplitude and frequency increasing with time. When the compact objects are widely separated, the problem can be treated perturbatively. In the leading order, called "Newtonian approximation", the GW signal (called the *Newtonian chirp*) can be computed as:

$$h(t) = A(t)\cos[\varphi(t) + \varphi_0] \quad t \geq t_0\,. \tag{1}$$

The amplitude $A(t)$ depends on a particular combination of the masses, called the *chirp mass* $\mathcal{M}$, the instantaneous frequency $F(t)$ of GWs, the luminosity distance D to the source, and a factor $\mathcal{G}$ that depends on the location of the source in the sky and its orientation with respect to the detector.

$$A(t) = \mathcal{G}\,\frac{4\mathcal{M}^{5/3}\pi^{2/3}F(t)^{2/3}}{D}. \tag{2}$$

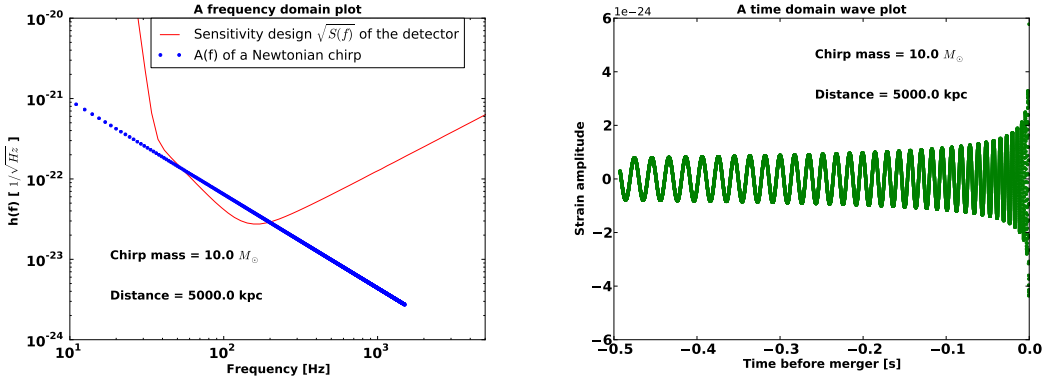

FIG. 2: Frequency domain and time domain plots of a Newtonian chirp. The frequency domain plot is shown along with the strain sensitivity of a 4 km detector ($\sqrt{S(f)}$)for comparison. For more discussion about strain sensitivity / PSD see section A 2. Note the increasing frequency and amplitude in the time domain waveform.

For simplicity, we shall assume $\mathcal{G} = 1$ which implies that the binary is conveniently oriented giving circular polarization and the source is located along the direction where the detector shows maximum directional sensitivity. The chirp mass can be expressed in terms of the total mass $M \equiv m_1 + m_2$ and reduced mass $\mu \equiv m_1 m_2/M$ as $\mathcal{M} = \mu^{3/5} M^{2/5}$.

The GW signal phase $\varphi(t)$ can be expressed as

$$\varphi(t) = \varphi_0 + 2\pi \int_0^{t-t_0} F(t')\, dt'. \tag{3}$$

The constant φ_0 is the GW phase when the signal enters the detector at the time instant $t = t_0$ and frequency $F = F_0$. The gravitational wave frequency depends on source parameters by

$$F(t) = F_0 \left(1 - \frac{t - t_0}{T_c}\right)^{-3/8}, \qquad T_c \equiv \frac{5}{256\,(\pi F_0)^{8/3} \mathcal{M}^{5/3}} \tag{4}$$

where T_c is the chirp duration. It can be seen that the frequency $F(t)$ monotonically sweeps from lower to higher frequencies. As expected, more massive stars inspiral much faster than less massive stars which is indicative in the above equation. Given F_0 (which depends on the lower frequency cut-off of the interferometer; see, e.g. the left plot of Figure 2), the Newtonian chirp is characterized by four parameters; namely $\mathcal{M}, t_0, \varphi_0$ and D.

All expressions in this Section are written in geometrized units, in which $G = c = 1$. Mass and distance have units of seconds. Physical units can be obtained by replacing a mass $\mathcal{M}$ by $G\mathcal{M}/c^3$, and a distance D by cD. In geometrical units, $1M_\odot = 4.92549095 \times 10^{-6}$ s and 1pc $= 1.0292712503 \times 10^8$ s. To give an order of magnitude of the quantities involved, we set $m_1 = m_2 = 1.4M_\odot$; $\mathcal{M} = 1.22M_\odot$ located at the distance of 100 kpc. It would emit GWs of amplitude $A \sim 10^{-22}$ at frequency 10 Hz.

$$A(t) = 3.6 \times 10^{-22} \left(\frac{D}{100\text{kpc}}\right)^{-1} \left(\frac{\mathcal{M}}{1.22M_\odot}\right)^{5/3} \left(\frac{F(t)}{10\text{Hz}}\right)^{-2/3}. \tag{5}$$

Later, we will see that it is useful to express the time-domain signal given in Eqs. (1)–(4) in frequency domain. The Fourier transform of $h(t)$ can be computed by employing the *stationary phase approximation* [5]. At leading order, the Fourier transform $\tilde{h}(f) \equiv A(f)\, e^{i\Psi(f)}$ is given by:

$$A(f) = \frac{\mathcal{M}^{5/6}}{D\,\pi^{2/3}} \sqrt{\frac{5}{24}}\, f^{-7/6}, \qquad \Psi(f) = 2\pi f t_0 - \varphi_0 - \pi/4 + \frac{3}{128}\,(\pi \mathcal{M} f)^{-5/3}. \tag{6}$$

The Fourier frequency f may not be confused with the instantaneous frequency $F(t)$ of GWs. We can plot the time-domain waveform $h(t)$ and the frequency-domain amplitude $A(f)$ using Eqs. (1) – (6). The plots are shown in Figure 2. A python code to make these plots can be found here: http://gw-indigo.org/mdc-2011/tools/plot_chirp.py.

B. Signal detection in the noisy data

A general class of signal detection problems involve detecting signals in the random noise of the instrument. The problem becomes hard when the signal is buried deep inside the noise and does not stand out in the time-series [7]. This problem is encountered in identifying a submarine in sonar data, aircraft in radar signal, fingerprint matching etc. The special stream which addresses these problems is called as Statistical Signal Analysis.

The data $x(t)$ can be modeled as an addition of a signal $h(t)$ to be detected and random noise $n(t)$; i.e., $x(t) = h(t) + n(t)$. Given the data stream $x(t)$, there exists two possibilities; signal is present or absent in the data. Since $x(t)$ is random, one can attach probabilities to these outcomes, namely, $P_1(x = n + h)$ and $P_0(x = n)$ respectively. Naively, one could think that if the measured values of the data are more probable when signal is present than when it is absent *i.e.* $P_1(x) > P_0(x)$, we should claim that we have detected signal. However, there is a small trouble. Since noise is random, at times it mimics the signal. Such events are called *false alarms* (falsely identifying an event as a signal event). We need to account for the false alarm rates. The probability of false alarms (FAP) is defined as

$$\text{FAP} = \int_{x_0}^{\infty} P_0(x)\, dx\,. \tag{7}$$

The FAP is the sum of all probabilities for the data to cross the *threshold value* x_0 in absence of signal. Calculation of the FAP, and the threshold x_0 requires modelling the noise distribution. Now revisiting the previous argument, we can claim a detection when $x > x_0$, or alternatively, $\Lambda > \Lambda_0$, where

$$\Lambda \equiv \frac{P_1(x)}{P_0(x)}\,, \quad \text{and,} \quad \Lambda_0 \equiv \frac{P_1(x_0)}{P_0(x_0)}. \tag{8}$$

The ratio Λ is called the *likelihood ratio*. It can be shown that, testing the condition $\Lambda > \Lambda_0$ amounts to maximizing the *signal detection probability* (SDP)

$$\text{SDP} = \int_{x_0}^{\infty} P_1(x)\, dx\,, \tag{9}$$

for a given value of FAP [6]. The signal detection probability is the sum of all the probabilities for the data to cross the threshold x_0 in presence of signal, *i.e.* for $\Lambda > \Lambda_0$.

1. *Noise properties*

As we discussed in Section II B, estimating the FAP requires modelling the noise distribution. The instrument noise $n(t)$ of duration T is modeled as a *stationary, Gaussian* random process of mean zero and variance σ^2. This implies that noise at various time instances are identically distributed Gaussian random variables with mean zero and variance σ^2. For all such stationary processes, the moments of the distribution remain constant with time. In reality, stationarity assumption may be too simplistic, but it is sufficient for this exercise.

- *White Noise*: As the name suggests, white noise is analogous to white light. Just like white light has equal contributions from all its constituent colors, white noise has equal power contribution from all the frequencies. If we denote the Fourier transform of $n(t)$ as $\tilde{n}(f)$, white noise has the property

$$E[\tilde{n}(f)\tilde{n^*}(f')] = \frac{T}{2}\, S_0\, \delta(f - f')\,, \tag{10}$$

  where $E[.]$ is the expectation/mean of a quantity and * denotes complex conjugation. Note that the power contribution (S_0) is same for all the frequencies.

- *Colored Noise:* On the contrary, colored noise carries unequal noise power from its constituent frequencies. Depending upon the functional dependence, the noise power is low at certain frequencies and high at certain frequencies *i.e.*

$$E[\tilde{n}(f)\tilde{n^*}(f')] = \frac{T}{2} S(f)\delta(f - f'), \tag{11}$$

where $S(f)$ is called the one sided *power spectral density* (PSD), with dimensions of Hz^{-1}. The interferometer noise is the sum total of noise due to its subsystems such as suspension system, laser, electronics, vacuum system etc. and hence it is expected to be colored. For example, seismic isolation has high noise at low frequencies (< 20 Hz) whereas quantum noise from laser has high noise at high frequencies ($f > 1000$ Hz). As a result the $S(f)$ looks like a broad 1-D valley when plotted against frequency; see Figure 5. The bandwidth of the initial interferometers is $20 - 1000$ Hz with the most sensitive frequencies in the range ($f \sim 70 - 500$ Hz).

2. *Matched filtering and signal-to-noise ratio*

In case a *known* signal $h(t)$ buried in stationary Gaussian noise, the optimal technique for signal extraction is the *matched filtering*, which involves cross-correlating the data with a *template* of the signal. It can be seen that under the aforementioned assumptions of the noise, the the likelihood ratio is equivalent to using a matched filter.

The correlation function between two time series $x(t)$ and $\hat{h}(t)$ for a time shift τ is defined as:

$$R(\tau) = \int_{-\infty}^{\infty} x(t)\, \tilde{h}^{\star}(t-\tau)\, dt\,, \tag{12}$$

where * denotes complex conjugation. The *Correlation Theorem* (see Table I) provides an efficient way of computing the correlation function: $R(\tau)$ is the inverse Fourier transform of $\tilde{x}(f)\tilde{h}^{\star}(f)$.

$$R(\tau) = \int_{-\infty}^{\infty} \tilde{x}(f)\tilde{h}^{\star}(f)\, e^{i2\pi f\tau} df. \tag{13}$$

This is the optimal filter for detecting a known signal buried in Gaussian *white noise*. On the other hand, if the noise is *colored*, then the corresponding matched filter output is

$$R(\tau) = 2\int_{-\infty}^{\infty} \frac{\tilde{x}(f)\tilde{h}^{\star}(f)}{S(f)}\, e^{i2\pi f\tau} df, \tag{14}$$

where $S(f)$ is the one-sided PSD of the detector noise (see Eq. 11 for definition). Since, in our case the signal $h(t)$ and noise $n(t)$ are real valued functions, the following relations hold: $\tilde{n}(-f) = \tilde{n}(f)^{\star}$, and $\tilde{h}(-f) = \tilde{h}(f)^{\star}$. This means that positive frequencies contain all the relevant information, and we can restrict the integration into positive frequencies. Hence Eq.(14) can be written as

$$R(\tau) = 4\int_{0}^{\infty} \frac{\tilde{x}(f)\tilde{h}^{\star}(f)}{S(f)}\, e^{i2\pi f\tau} df. \tag{15}$$

The optimality is evident as it gives more weight for frequencies where the detector is more sensitive and hence enhancing the *signal-to-noise ratio* (SNR). The optimal SNR is obtained when the template exactly matches with the signal.

$$\mathrm{SNR}^2 \equiv ||h||^2 \equiv 4 \int_{0}^{\infty} \frac{|\tilde{h}(f)|^2}{S(f)} df. \tag{16}$$

In the ideal scenario where the noise is stationary Gaussian, if the SNR is greater than a predetermined threshold, which corresponds to an acceptably small FAP, a detection can be claimed.

C. Detection of an unknown Newtonian chirp in colored Gaussian noise

We saw in the previous section that signal detection can be declared if the SNR is greater than a predetermined threshold. But computing the matched filter requires the exact knowledge of the signal. Although we can compute the expected gravitational waveform as a function of the source parameters, the parameters of the signal that is buried in the data is not known *a priori*. Thus, we need to maximize the SNR over all the parameters describing the signal: $\mathcal{M}, D, t_0, \varphi_0$.

- *Maximization over D*: We note from Eqs.(2) and (6) that D appears only in the amplitude, as a linear scaling factor. The SNR can be maximized over D by just using *normalized templates* $\hat{h}(f) \equiv \tilde{h}(f)/||h||$ in the filtering (see Eqs. 15 and 16).

TABLE I: Correspondence between continuous and discrete time as well as Fourier domain

Continuous Domain	**Discrete domain**
Time series: $x(t); 0 < t < T$	$x_j \equiv x(t_j),\ t_j = j\Delta,\ j = 1, \ldots, N$
Frequency series: $\tilde{x}(f); 0 < f < \infty$	$x_k \equiv x(f_k),\ f_k = k/(N\Delta),\ k = 0, \ldots, N/2$
Fourier Transform (FT): $\tilde{x}(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i f t} dt$	Discrete Fourier Transform (DFT): $\tilde{x}_k = \sum_{j=1}^{N} x_j e^{-2\pi i j k/N}$
Inverse Fourier Transform (IFT): $x(t) = \int_{-\infty}^{\infty} \tilde{x}(f) e^{2\pi i f t} df$	Discrete Inverse Fourier Transform (DIFT): $x_j = \frac{1}{N} \sum_{k=1}^{N} \tilde{x}_j e^{2\pi i j k/N}$
Parseval Theorem: $\int_{-\infty}^{\infty} y(t)\, z(t)\, dt = \int_{-\infty}^{\infty} \tilde{y}(f)\, \tilde{z}^{\star}(f)\, df$	$N \sum_{j=1}^{N} y_j\ z_j = \sum_{k=1}^{N} \tilde{y}_k\ \tilde{z}_k^*$
Correlation Theorem: $\int_{-\infty}^{\infty} y(t)\, z^{\star}(t - t')\, dt = \mathrm{IFT}\{\tilde{y}(f)\ \tilde{z}^{*}(f)\}$	$\sum_{j=1}^{N} y(j)\, z^{\star}(j - j') = \mathrm{DIFT}\{\tilde{y}_k\ \tilde{z}_k^*\}$
Power Spectral Density $S(f)$	$\frac{2}{N^2} \frac{T}{} S_k, S_k = E[\lvert\tilde{n}_k\rvert^2]$

- *Maximization over* φ_0: It can be seen from Eq.(1) that, when $\varphi_0 \to 0$, the optimal template is $\cos\varphi(t)$ and when $\varphi_0 \to \pi/2$, the optimal template is $\sin\varphi(t)$. Making use of the orthogonality of sine and cosine functions, the obvious way to maximize the SNR over φ_0 is to compute the correlation of the data with a sine template and a cosine template, and add the SNRs in quadrature. i.e., $\mathrm{SNR}^2 = \mathrm{SNR}_0^2 + \mathrm{SNR}_{\pi/2}^2$, where SNR_0 [$\mathrm{SNR}_{\pi/2}$] is the SNR obtained using the cosine [sine] template. These templates can be generated in the frequency domain by setting $\varphi = 0$ [$\pi/2$] in Eq.(6).
- *Maximization over* t_0 is effected by just taking the maximum value of the correlation function $R(\tau)$.
- *Maximization over* $\mathcal{M}$: There is no simple analytical trick to maximize the SNR over the chirp mass $\mathcal{M}$! So, one has to create a "bank" of templates with different values of $\mathcal{M}$ and find out which value of $\mathcal{M}$ gives the maximum SNR.

D. Discrete Fourier Transform

Till now, we treated signal as a continuous function of time or frequency. However, data recorded at the output of an interferometer is sampled with a fixed sampling rate and is finite in length of time T. Hence it is necessary to work in the discrete domain rather than continuous domain. Let $f_s = 1/\Delta$ be the sampling frequency and $N = f_s T$ be the number of samples. In Table I, we summarize the correspondence between discrete and continuous domain.

III. THE MOCK DATA CHALLENGE

In this challenge, participants are required to identify inspiralling binary black hole GW signals buried in the detector noise. The data set consisting of a few tens of simulated GW signals from inspiralling binary black holes in simulated noise of a future GW observatory called IndIGO is available at the following link: http://www.gw-indigo.org/mdc_data_set. This hypothetical detector is assumed to have sensitivity comparable to that of the Initial LIGO [1] detectors. Each frame file from the link

above contains a channel called `I1:INDIGO-STRAIN` storing 3600 seconds of "strain" data. The frame file also contains related information, such as the sampling rate of the data and the GPS time-stamps. The data is stationary Gaussian distributed, with a one-sided PSD of:

$$S(f) = \begin{cases} 9 \times 10^{-46} \left[(4.49x)^{-56} + 0.16x^{-4.52} + 0.52 + 0.32x^2\right], & \text{if } f \geq 40 \text{ Hz} \\ \infty, & \text{if } f < 40 \text{ Hz}. \end{cases} \tag{17}$$

where $x \equiv f/f_m$, $f_m = 150$ Hz. In actual searches, the PSD is estimated from the data (see Appendix A for a simple exercise). Still, in order to make the analysis simpler, we suggest to use this analytical fit to the PSD in computing the matched filter.

The main task in this challenge is to:

- Identify the GPS times of signal using a matched filtering search or an "excess-power" search on the data set. A data analysis group will get `10 points` for each correctly identified signal within `0.5 s` of actual GPS time of the signal. If the identified signal is within `1 s` of the actual injected time, then the group will get `5 points`. Participants are encouraged to find as many signals as they can. If they report a time which does not have any signal within `1 s` then there is no point for that.
- They should also report the signal-to-noise ratio (SNR) and the estimated chirp mass of the signal. A correct chirp mass within `20 %` accuracy will carry `10 points` each.
- We suggest an SNR threshold of 5.5 to be used for the identification of a signal.

Participating groups are required to submit a report via email to mdc2011@gw-indigo.org at the end of their analysis. In the report the groups are supposed to briefly describe the methods they used to identify the signals along with a result table. A sample result table is show below. The first line contains the actual parameters of a signal that is present in the data set. Participants may use these values to verify their search algorithms.

```
# GPS_time          estimated_SNR       estimated_chirp_mass (in solar_mass)
977875997.91        15                  19
9********           **                  **
9********           **                  **
9********           **                  **
9********           **                  **
9********           **                  **
```

---

[1] URL http://www.ligo.caltech.edu/.
[2] URL http://www.virgo.infn.it/.
[3] URL http://www.geo600.org/.
[4] URL http://tamago.mtk.nao.ac.jp/.
[5] K. Thorne, in *Three Hundred Years of Gravitation*, edited by S. Hawking and W. Israel (Cambridge University Press, Cambridge, U.K.; New York, U.S.A., 1987), pp. 330–458.
[6] *The Neyman-Pearson Lemma*, URL http://en.wikipedia.org/wiki/NeymanPearson_lemma.
[7] Students are encouraged to play the Black-hole hunter game at http://www.blackholehunter.org/

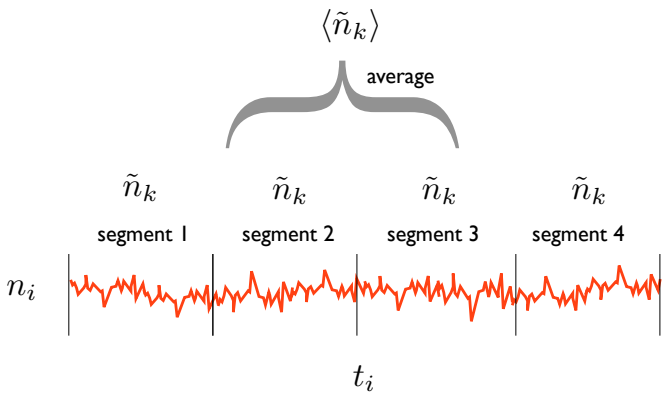

FIG. 3: A schematic diagram showing the splitting of data into smaller segments, computing FFTs of each segments, and averaging to compute the PSD.

Appendix A: Introduction to statistical analysis and signal processing

This section provides some warm-up exercises in statistics and signal processing, along with a list of MATLAB/Octave/Python functions which are useful for each exercise.

1. Random numbers, probability distributions

- Generate a set of random numbers n_i drawn from Gaussian distribution. Estimate the mean and variance. Make a histogram of $\{n_i\}$ and plot the expected probability distribution (Gaussian) computed from the mean and variance. What is the probability $P(n_i > \lambda)$ for one of them to be greater than a threshold λ ?

  Useful functions (MATLAB/Octave): `randn, mean, var, normpdf, normcdf, hist`

  (Python): `numpy.random.randn, numpy.mean, numpy.var, numpy.random.normal, scipy.stats.norm.pdf,scipy.stats.norm.cdf, numpy.histogram`

- What is the probability distribution of the set of numbers $p_i = n_i^2$? Plot the histogram of $\{p_i\}$ along with the expected probability distribution computed from the mean and variance. Determine the probability $P(p_i > \lambda)$ for one of them to be greater than a threshold λ.

  Useful functions (MATLAB/Octave): `chi2pdf, chi2cdf`

  (Python): `scipy.stats.chi2.pdf,scipy.stats.chi2.cdf`

2. Fast Fourier transform, power spectral density estimation

- Assume that $\{n_i\}$ is a time-series data vector sampled at a sampling rate Δ. Compute the Fast Fourier Transform (FFT) of n_i. Plot the magnitude of the FFT. Repeat the analysis using a window function (say, Hanning window). What is the effect of the window function? Why is the FFT 'noisy'?

  The frequency range of the Fourier transform $\tilde{n}_k$ is limited by the sampling rate due to the *Nyquist theorem*. The maximal useful frequency $f_N \equiv f_s/2$ is called Nyquist frequency, where $f_s \equiv 1/\Delta$ is the sampling frequency. The frequency resolution of the FFT is given by $\Delta f = f_s/N$, where N is the number of samples in $\{n_i\}$. The way the frequency bins of $\tilde{n}_k$ are ordered depends on the particular FFT implementation. In Octave, the frequency bins are ordered as $[0, \Delta f, 2\Delta f, ..., f_N, -f_N + \Delta f, -f_N + 2\Delta f, ..., -\Delta f]$.

  Useful functions (MATLAB/Octave): `fft, hanning`

  (Python): `numpy.fft, numpy.hanning`

- Compute the power spectral density (PSD) of the data by appropriately normalizing the FFT and averaging over many noise realizations. What is the effect of changing the number of samples used to compute one FFT? What is the effect of the number of segments averaged over in order to compute the PSD?

  Since the time-domain data that we use is real, the following relation holds in the Fourier domain: $\tilde{n}(-f) = \tilde{n}(f)^*$. This means that just the positive frequencies contain all the information needed to reconstruct the data. The normalization used to compute the PSD depends on the normalization used in the particular FFT implementation. In Octave, the *one-sided* (positive frequencies only) PSD can be computed as

$$S_k = \frac{2\left\langle|\tilde{n}_k|^2\right\rangle \Delta}{N}, \tag{A1}$$

  where N is the number of samples used to compute *one* FFT, and angular brackets denote ensemble averages. In the case of a windowed FFT, N should be replaced by $\sum w_i^2$ where w_i is the window function used.

3. Correlation function, matched filter

- Construct a data stream n_i of Gaussian white noise (using `randn` function) with zero mean and unit variance. Multiply the data with 10^{-23}, thus creating a noise vector with variance 10^{-46} (comparable to the noise variance of current interferometers — but note that actual detector noise is *not* white).
- Create a time vector with sampling frequency $f_s = \Delta^{-1} = 2048$ Hz.
- Add a Newtonian chirp signal (see Section II A) to the noise: $x_i = n_i + h_i$. Choose $\mathcal{M} = 8M_\odot, D = 50$ Mpc, $F_0 = 40$ Hz.
- Compute the correlation function between the data and the normalized template (see Eq.12). The inverse FFT can be computed using the `ifft` function.
- Find the peak of the correlation function and use that as your detection statistic. This is the optimal filter for Newtonian chirp signals in white noise.
- Change the distance D to the binary. Increasing D will result in a decrease in the SNR, and vice-versa. The ability to detect the signal depends on the SNR.

Appendix B: Software

We recommended using high-level programming languages such as Matlab/GNU-Octave or Python. For all practical purposes, "high-level" just means they are easy to learn and use! Additionally, Octave and Python are free software, and come preinstalled with many Unix-like operating systems. Indeed, the participants are free to use any language of their preference. But reading the binary files containing the GW data (frame files) is particularly easy with these languages. We will also assume that the participants are using a Unix-like operating system (GNU/Linux, Solaris, Mac OS X, BSD etc.). Some useful resources for Matlab/Octave/Python are given below:

1. Matlab

- Matlab page: http://www.mathworks.com/products/matlab/
- Matlab documentation: http://www.mathworks.com/help/techdoc/
- Introduction to Matlab:
  1. http://www.mathworks.com/moler/intro.pdf
  2. http://www.mccormick.northwestern.edu/docs/efirst/matlab.pdf
  3. http://www.physics.byu.edu/Courses/Computational/phys330/matlab.pdf

2. Octave

- Octave home page: http://www.gnu.org/software/octave
- Octave documentation: http://www.gnu.org/software/octave/doc/interpreter
- Introduction to Octave:
  1. http://linuxgazette.net/109/odonovan.html
  2. http://www.math.utah.edu/docs/info/octave_3.html
  3. http://www.maritime-engineers.com/Documentation/octave-intro.pdf

3. Python

If you are using a Unix like operating system there is high chance that your OS has already Python installed. In the shell terminal type `python`. For this primer we will assume you are using a version of Python $\geq$ 2.4. If your OS does not have Python, you can install the appropriate version from the Python homepage.

- Python home page: http://www.python.org/
- Python by itself comes pretty light-weight. To be able to do a wide variety of numerical manipulation on arrays we will need `numpy`. Similarly `scipy` package comes with necessary scientific subroutines to do data analysis. To be able to graphs with Python we need `matplotlib`. Sub-packages for Python:
  1. `numpy`: http://numpy.scipy.org/
  2. `scipy`: http://www.scipy.org/
  3. `matplotlib`: http://matplotlib.sourceforge.net/

You might also need several other sub-packages of Python. There is a company, Enthought scientific computing solutions, which bundles the latest Python distribution with all the different sub-packages. For academic use (using an academic email) it will let you download this bundled Python distribution free of cost.

- Enthought Python download page: http://www.enthought.com/products/edudownload.php

There are several learning resources for Python in the web. We list a few here for your reference.

- General Python tutorial: http://docs.python.org/tutorial/
- `numpy` tutorial: http://www.scipy.org/Tentative_NumPy_Tutorial
- `scipy` tutorial: http://docs.scipy.org/doc/scipy/reference/tutorial/index.html
- `numpy` for MATLAB users: http://www.scipy.org/NumPy_for_Matlab_Users
- `scipy` cookbook: http://www.scipy.org/Cookbook

4. Frame library

The data collected by GW observatories worldwide is stored in a particular binary format, called *frame format*. A frame is a unit of information from the interferometer data over a finite time interval. The *frame library* is a collection of tools to read and manipulate frame files. To install the Frame library, first download the latest version from: http://lappweb.in2p3.fr/virgo/FrameL/.

The example below shows how to download, compile and install the frame library. Executing the following commands in a Unix-type system should work:

```
wget http://lappweb.in2p3.fr/virgo/FrameL/libframe-8.15.tar.gz
tar xzvf libframe-8.15.tar.gz
pushd libframe-8.15
./configure --prefix=SUITABLE_FOLDER_WHERE_YOU_WANT_TO_INSTALL
make
make install
popd
```

To be able to link frame library with other programs it is useful to have `pkg-config` program. Most probably your OS should have `pkg-config` pre-installed. If not you can download and install it from: http://www.freedesktop.org/wiki/Software/pkg-config. The following example demonstrates the usefulness of `pkg-config`.

```
$ export PKG_CONFIG_PATH=FOLDER_WHERE_YOU_INSTALLED_FRAMELIB/lib/pkgconfig:$PKG_CONFIG_PATH
$ pkg-config --libs --cflags-only-I libframe
```

And as an output of the above command you will get something similar to what is show below.

```
-I/opt/local/include  -L/opt/local/lib -lFrame
```

This header and library locations will help to link other programs with the frame library. There are wrapper codes to use the frame library to read/write frame files in Matlab/Octave/Python.

5. Frame library interface to Matlab/Octave/Python

This section describes how to read data into Matlab/Octave/Python. Mostly we will assume that the platform is Linux. But these instructions should work in any Unix-type system with minor changes.

a. Matlab

The Frame library also provides a "mex" file which can be used to read the data stored in the frame files to Matlab. First we need to compile the source code to create the mex file. Start Matlab, and go to the following directory: `~/libframe-8.15/matlab` You should be able to see the file `frgetvect.c` here. Now compile the mex file in Matlab command window:

```
>> mex frgetvect.c FOLDER_WHERE_YOU_INSTALLED_FRAMELIB/lib/libFrame.so -I../src
```

This will create a mex file `frgetvect.mexa64` (the extension depends on the system architecture). Copy `frgetvect.mexa64` and `frgetvect.m` to your working directory or add `~/libframe-8.15/matlab` to your Matlab path. Make sure that Matlab can access the mex file by typing:

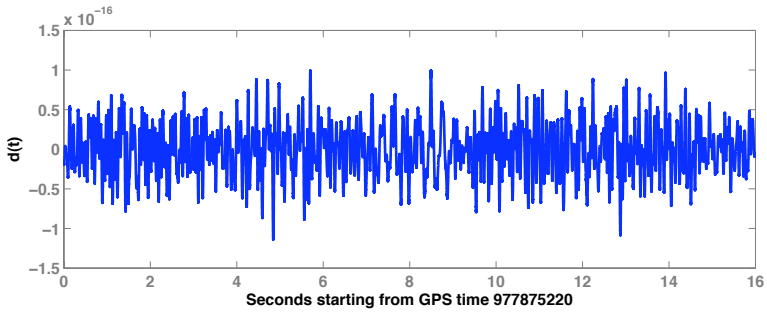

FIG. 4: An example of the time series data retrieved from frame files.

```
>> which frgetvect
```

in the Matlab command window. Matlab should return the location of the newly compiled mex file. Type `help frgetvect` to see how to use this function. Here is an example:

```
>> [d, t] = frgetvect('I-INDIGO-977875220-3600.gwf', 'I1:INDIGO-STRAIN', 977875220, 16, 0);
Warning: frgetvect:info:Opening I-INDIGO-977875220-3600.gwf for channel I1:INDIGO-STRAIN
    (t0=977875220.00, duration=16.00)
>> figure; plot(t, d); xlabel('Seconds starting from 977875220'); ylabel('d(t)');
```

This will plot 16 seconds of time-series data starting from GPS seconds 977875220, as shown in Figure 4.

b. Octave

Go to the directory ∼`/libframe-8.15/octave`, and open the `Makefile` using a text editor. In the `Makefile` change the following variables to point to the location of your newly installed frame library

```
FRAME_INC = FOLDER_WHERE_YOU_INSTALLED_FRAMELIB/include
FRAME_LIB = FOLDER_WHERE_YOU_INSTALLED_FRAMELIB/lib
```

Now type `make` in the shell. This will create, among other files, a binary file called `loadframe.oct`. Copy `loadframe.oct` and `loadframe.o` to your working directory or add ∼`/libframe-8.15/octave` to your Octave path. Do `help loadframe` in the Octave command line to see how to use this function. Here is an example:

```
octave:1> [d, fs] = loadframe("I-INDIGO-977875220-3600.gwf", "I1:INDIGO-STRAIN", 1, 977875220);
octave:2> length(d)/fs
ans =  3600
```

Note that, unlike the case of the Matlab mex file, here we cannot specify the length of data to be read in a single call: `loadframe` loads the entire data in the frame file (3600 seconds) into the vector `d`, and hence is rather slow. Also, $f_s = 1/\Delta t$ is the sampling frequency of the data.

c. Python

`Pylal` is a package which offers several python based routines to do GW data analysis. Here we will use a strip-down version of `pylal` only to be used to read the frame data. Assuming you have `numpy`, `pkg-config` and `wget` you can just download the following script and run it. This shell script will install both the frame library and the pylal python wrapper for you in any Unix like OS: http://gw-indigo.org/mdc-2011/tools/install_python_tools.sh. By default this installs in the following directory: ∼`/indigo_python/install`. To use the python frame reading wrapper do:

```
source ~/indigo_python/install/pylal/etc/pylal-user-env.sh
```

Then you can start Python. In Python prompt do:

```
>>> from pylal import Fr
```

You can load a gwf file into a python array using `frgetvect` function, e.g.

```
frame = Fr.frgetvect("I-INDIGO-977875220-3600.gwf","I1:INDIGO-STRAIN")
```

Now let's look at this array:

```
>>> frame
(array([  3.85241898e-17,   3.70307674e-17,   3.55302314e-17, ...,
        -3.90825860e-17,  -3.74059633e-17,  -3.56520304e-17]), 977875220.0, (’’,), ’’)
```

The first index holds time series strain data (Note: unlike MATLAB/Octave, Python starts counting array index from 0):

```
>>> frame[0]
array([  3.85241898e-17,   3.70307674e-17,   3.55302314e-17, ...,
        -3.90825860e-17,  -3.74059633e-17,  -3.56520304e-17])
```

The second index shows the GPS start time of the frame.

```
>>> frame[1]
977875220.0
```

The fourth index shows the inverse of sampling rate, i.e. spacing between the time-series strain data.

```
>>> frame[3]
(0.000244140625,)
>>> frame[3][0]
0.000244140625
>>> 1/frame[3][0]
4096.0
```

This particular frame file has 3600 s of data, which we can verify by,

```
>>> len(frame[0])*frame[3][0]
3600.0
```

Rather than loading the full data we can load part of the data, e.g. in the example shown below only `10 s` the data is being loaded from a start time of `977875255`.

```
>>> frame = Fr.frgetvect("I-INDIGO-977875220-3600.gwf","I1:INDIGO-STRAIN",977875255,10)

>>> print frame[1]
977875255.0
>>> print len(frame[0])*frame[3][0]
10.0
```

In the example below we show how to do some plots in Python. We will first load `100` s of data then plot the time-series. After that we will use the PSD function in Python's matplotlib library to draw the spectra. The plots are show in fig: 5. More details on PSD is given in section A 2.

```
##########################
### Python code to do produce time-series and PSD from the frame file
### This code can also be downloaded from
### http://gw-indigo.org/mdc-2011/tools/plot_frame.py
# import necessary python modules
import numpy
from pylal import Fr
# matplotlib is the plotting library for python
from matplotlib.mlab import psd
import matplotlib
# Use the "Agg" option only if you are running this code outside python prompt
# i.e. in a shell $ python THIS_CODE.py
# if you running this code inside python prompt, comment out the "Agg" option
matplotlib.use("Agg")
# pylab gives python some MATLAB like feel
import pylab

# load 10 s of data starting from GPS time = 977875220
frame = Fr.frgetvect("I-INDIGO-977875220-3600.gwf","I1:INDIGO-STRAIN",977875220,100)
```

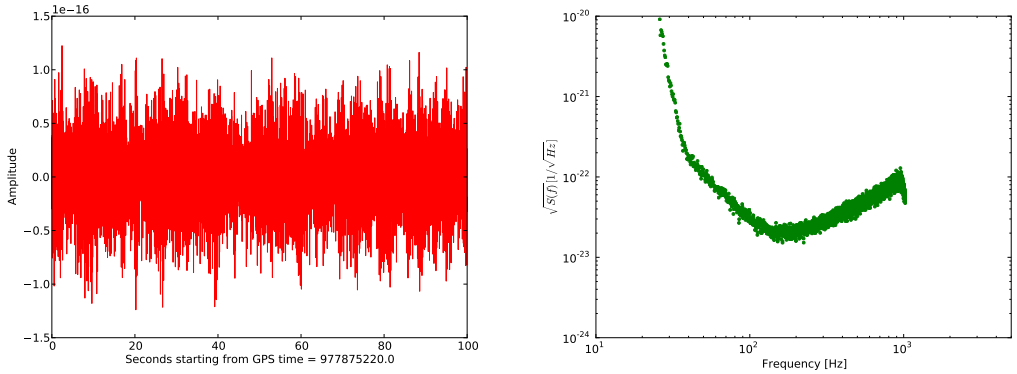

FIG. 5: An example of the time series data and PSD plot with Python.

```
t = numpy.linspace(0,100,len(frame[0]))

# estimate PSD from the data
spectra, freqs = psd(frame[0],NFFT=int(4/frame[3][0]),Fs=1/frame[3][0],noverlap=0,sides="twosided")

#plot time series
pylab.plot(t,frame[0],"r-")
pylab.xlabel("Seconds starting from GPS time = " + str(frame[1]))
pylab.ylabel("Amplitude")
pylab.savefig("timeseries.pdf")
pylab.close()

#plot spectra
pylab.loglog(freqs,numpy.sqrt(spectra),"g.")
pylab.ylim(1e-24,1e-20)
pylab.xlim(1e1,5e3)
pylab.xlabel("Frequency [Hz]")
pylab.ylabel("$\sqrt{S(f)}$" + "$ [ 1/\sqrt{Hz}  ] $")
pylab.savefig("spectra.pdf")
pylab.close()
```