MULTI-INTERFEROMETER SEARCH METHODS FOR GRAVITATIONAL WAVES

By

SHAWN EUGENE SEADER

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science in Physics

WASHINGTON STATE UNIVERSITY
Department of Physics and Astronomy

December 2005

To the faculty of Washington State University:

The members of the Committee appointed to examine the thesis of SHAWN EUGENE SEADER find it satisfactory and recommend that it be accepted.

_____
(Chair)

_____

_____

_____

ACKNOWLEDGMENT

Becoming a member of the LIGO Scientific Collaboration has afforded me the opportunity of working with many great physicists in the field of gravitational wave detection. I can't possibly name everyone who has helped me along the way but would like to thank the few people with whom I have worked most closely. I would like to thank Patrick Brady for his always present words of encouragement. Then Steven Fairhurst, whose foresight has no doubt saved me from wasting countless hours writing computer codes. Also, I would like to thank Steve for the many meetings and telecons that have occurred between us in an effort to incorporate my work into the current search pipeline. I would like to thank Duncan Brown for also meeting with me several times both in person and over the phone to help in solving some of the more complicated bugs that crept up in the coding process. Special thanks is in order for Dr. Bose. I can not recall a time when Dr. Bose was ever unwilling to meet with me or explain something that I was having difficulty with. I thank him for not only that but also for facilitating the best possible work environment and working relationship between us. I truly could not have hoped for a better situation in which to work under to complete my thesis.

I would also like to thank the many great professors with whom I have had the opportunity of learning from both at Washington State as well as Boise State. I feel like I have been in good hands through the entire course of my studies.

And finally I would like to thank my family for their constant support and encouragement. I thank my Mom for instilling in me a good work ethic and always pushing me to get better grades. I thank my Dad for providing me with the tools necessary at a young age to help me become interested in mathematics and science. And last but not least I thank Heidy for her love and support over the years without which I would surely have lost track of my goal.

MULTI-INTERFEROMETER SEARCH METHODS FOR

GRAVITATIONAL WAVES

Abstract

by Shawn Eugene Seader, M.S.
Washington State University
December 2005

Chair: Sukanta Bose

The theory of general relativity predicts the existence of gravitational waves, which are ripples on the fabric of space-time caused by the bulk motion of very large amounts of mass-energy. In 1993, Hulse and Taylor received the Nobel Prize for their work on a binary pulsar system which indirectly proved the existence of gravitational waves. The challenge of directly detecting these waves still remains unmet today. The fabric of space-time is extremely rigid, so only extreme systems of mass can produce sizable gravity waves on it. Some of the more promising systems include binary star systems with neutron star and black hole constituents. Given that these systems are at extra-galactic distances, the waves from them will be very weak upon arriving at Earth. Only very recently has it become possible to build a detector that can measure such a small effect.

There is currently a network of laser interferometric gravitational wave observatories being built and commissioned around the globe with one goal in mind: to detect gravitational waves. To date, much of the data from these observatories has been analyzed on a single interferometer level for several reasons both political and scientific. It is possible to combine the outputs of several interferometers to improve the chances of a successful detection. This work focuses mainly on the application of this technique to the laser interferometric detectors in searching for signals from inspiraling compact binaries.

Employing such a technique can lead to considerable computational cost. To reduce this cost, a hierarchical analysis method can be used. Work done on this method is also presented here. Additionally, a search method for unmodeled sources is developed using a technique known as aperture synthesis.

Contents

Chapter 1

Introduction

Much of what we know about the universe comes from looking at electromagnetic radiation from a variety of astrophysical sources such as stars, gas clouds, and accretion disks. Gravitational waves will bring us the complementary knowledge which we at present lack. Many sources from which we can not detect an electromagnetic emission emit gravitational waves which can be detected. These waves come from exotic objects such as black holes, neutron stars, white dwarfs, and supernovae. Even the big bang is thought to have furnished the universe with a cosmic gravitational wave background. These waves will tell us something about the bulk motion of very large concentrations of mass or energy, whereas the electromagnetic waves we observe typically only tell us about the thermodynamic state of a very small concentration of matter. The gravitational waves from numerous sources throughout the universe combine to form a sort of symphony of the cosmos. Their wavelengths are typically of the order of the same size as their sources, unlike electromagnetic waves which have very small wavelengths of the order of the

size of the atom. Gravity-waves are therefore more analogous to sound, since we cannot form a picture of the source from them. Sound waves are vibrations of air molecules typically, whereas gravity waves are vibrations of the fabric of space-time. Due to their very long wavelengths and the fact that they are weakly interacting, gravitational waves traverse the universe virtually undisturbed by matter, unlike electromagnetic waves which are scattered and absorbed by intervening matter.

One of the most promising candidates for detection by Earth-based gravity wave detectors, or antennas, is the inspiraling compact binary system which this work primarily deals with. A typical binary is composed of two neutron stars whose masses can be up to three times the mass of our Sun and whose radii are about ten kilometers. As these stars orbit about their common center of mass, they lose energy due to the emission of gravitational waves. As they lose energy, their orbits shrink and they spiral toward one another in a process referred to as the inspiral phase. Once the innermost stable circular orbit is reached, the stars plunge toward one another and merge in what is known as the merger phase. After merging into a single star, there is a period of intense vibration known as the ringdown phase where the newly formed star is essentially acting like a struck bell.

The waveforms for the gravitational waves emitted in both the inspiral and ringdown phases are well known. This knowledge makes it possible to

employ a signal detection technique known as matched filtering to search for the true gravity waves in the data from a detector or multiple detectors. To apply the technique, a discrete bank of filters or templates is built to populate all the parameter space upon which the signal depends. The correlation between each filter and each chunk of data is then computed and compared to some threshold value. Since the data is composed of primarily Gaussian distributed noise and possibly a true gravity wave signal, only a probabilistic statement about a detection can be made. Any true signal that may be present in the data will be buried deep in noise. To make this signal show up with greater strength, one can additionally form the optimal combination of several detectors data streams to perform what is known as a coherent analysis. In such an analysis, the amplitude modulation, polarization, inclination, sky-posttion, and time-delay information from the different sites in the network are required to be consistent. The formalism for this type of analysis was developed in [1]. This work extends that formalism to the second post-Newtonian order of the inspiral phase of a compact binary's evolution.

Matched filtering works well, but can be computationally expensive. Some ways to combat this issue are discussed alongside some of the details of the search algorithms we use and some description of the computer codes that ultimately run the searches across many cluster computing sites. A significant part of this thesis went into the development of computer codes necessary to

do the coherent type of search for a network of interferometers. Additionally, much research was done on implementing a hierarchical search to reduce the computational cost. In this type of search the data is first filtered with a coarse bank of templates, then any candidate events are followed up with filtering using a fine bank localized around each candidate.

When the waveforms are not well known, matched filtering is not possible. This is the case for the merger phase of the binary's evolution. There are many techniques that have been developed for searching data for the presence of unmodeled signals [2, 3]. In this work a new formalism is developed for a coherent search for power bursts in the data of a network of interferometers. While not developed here, it is then possible to search over all three evolutionary phases of the binary system by stitching together the known waveforms of the inspiral and ringdown phases of the binaries evolution with this burst type analysis for the merger phase.

An introduction to gravitational waves as well as a description of an interferometric detector is given in Chapter 2. A formalism for combining the outputs of a network of interferometric detectors is developed in Chapter 3. The computer codes for implementing the multi-interferometer search are discussed in Chapter 4 along with the search algorithm and some results of testing. Chapter 5 discusses a two step hierarchical multi-interferometer search designed to reduce the computational costs of performing a multi-

interferometer search. The results of some Monte Carlo studies are also presented there. In Chapter 6 a multi-interferometer search formalism for unmodeled signals is developed. A conclusion is given in Chapter 7.

Chapter 2

Gravitational Waves and Detectors

General relativity is a theory which tells us how to measure distances in the vicinity of dense concentrations of mass or mass-energy. It is a geometric theory of gravity, wherein gravity is nothing but a manifestation of curved space-time. Mass, or energy, tells space-time how to curve and curved space-time tells mass how to move. All of the theory can be encapsulated quite elegantly in one equation:

$$\mathbf{G} = \frac{8\pi G}{c^4}\,\mathbf{T} \quad . \tag{2.1}$$

On the left hand side of this equation is the Einstein tensor, which describes the curvature of the surrounding space-time. On the right hand side are some physical constants and the stress-energy tensor $\mathbf{T}$. This equation is a direct statement of the equivalence between space-time curvature and mass-energy.

Since both tensors in the equation have two indices that each take four possible values, this one equation is really a set of sixteen, nonlinear, coupled partial differential equations. Due to symmetry, only ten of these equations are unique.

2.1 Gravitational Waves

In space-time, the four dimensions that form the fabric in which we all live, an event is a single point given by a set of four numbers or coordinates, three for the spatial dimensions and one for time. The distance between two events, ds, is known as an interval and is given by:

$$ds^2 = g_{\mu\nu} dx^\mu dx^\nu \quad , \tag{2.2}$$

where the Einstein summation convention is implied on the right hand side. Unless otherwise noted, when an index appears as both a superscript and subscript on the same side of an equation, a summation over the full range of that index is implied. This quantity is invariant under a Lorentz transformation, that is, an observer in any inertial frame will measure the same interval for two events. In this equation, $g_{\mu\nu}$ is known as the metric, and is a set of sixteen numbers that describe the curvature of the surrounding space-time. It gets its name from the fact that it tells us how to measure distances in

curved space-time.

In very weak gravity, the metric can be decomposed into the flat space metric, or the Minkowski metric, plus a small perturbation,

$$g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu} \tag{2.3}$$

where $\eta_{\mu\nu}$ is the flat space metric given by

$$(\eta_{\mu\nu}) = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

and $h_{\mu\nu}$ is matrix of small perturbations. In this approximation of very weak gravity, the flat space metric can be used to raise and lower indices,

$$h^{\mu\nu} = \eta^{\nu\beta} h^{\mu}{}_{\beta} \quad . \tag{2.5}$$

Now, by defining a new tensor $\bar{h}^{\mu\nu}$, as the trace-reverse of $h_{\mu\nu}$:

$$\bar{h}^{\mu\nu} = h^{\mu\nu} - \frac{1}{2}\eta^{\mu\nu} h^{\alpha}{}_{\alpha} \quad , \tag{2.6}$$

and by choosing a suitable gauge (the Lorentz gauge), it is possible to show

that the Einstein tensor can be written:

$$G^{\mu\nu} = -\frac{1}{2}\Box\bar{h}^{\mu\nu} \tag{2.7}$$

where the box denotes the wave operator,

$$\Box \equiv \nabla^2 - \frac{1}{c^2}\frac{\partial^2}{\partial t^2} \quad . \tag{2.8}$$

Now it is easy to see that in the weak field limit, equation (2.1) becomes

$$\Box\bar{h}^{\mu\nu} = -\frac{16\pi G}{c^4}\, T^{\mu\nu} \quad . \tag{2.9}$$

In vacuum, which is primarily what gravitational waves travel through, all the components of the stress-energy tensor are zero, giving:

$$\Box\bar{h}^{\mu\nu} = 0 \quad . \tag{2.10}$$

This is the well known three-dimensional wave equation and has solutions which are plane waves:

$$\bar{h}^{\mu\nu} = A^{\mu\nu} exp\,(ik_\sigma x^\sigma) \quad . \tag{2.11}$$

It is shown in a book by Schutz [4] that there are only two independent components or polarizations that a gravity wave can have, termed *plus* and *cross* for the way in which they affect matter that they pass near. The effect

of each of the polarizations is shown in Figure 2.1 for a wave traveling into the page. The top row shows the effect of the *plus* polarization stretching and squeezing in the vertical and horizontal directions as the wave passes. Likewise, the bottom row shows the effect of the *cross* polarization as the wave passes. Notice that the two act 45 degrees apart, in contrast to the two polarizations of an electromagnetic wave which are 90 degrees apart. This fundamental difference is due to the fact that the mediating particles, the graviton for gravity waves and photon for electromagnetic waves, have different spins. Both electromagnetic waves and gravity waves travel at the speed of light.

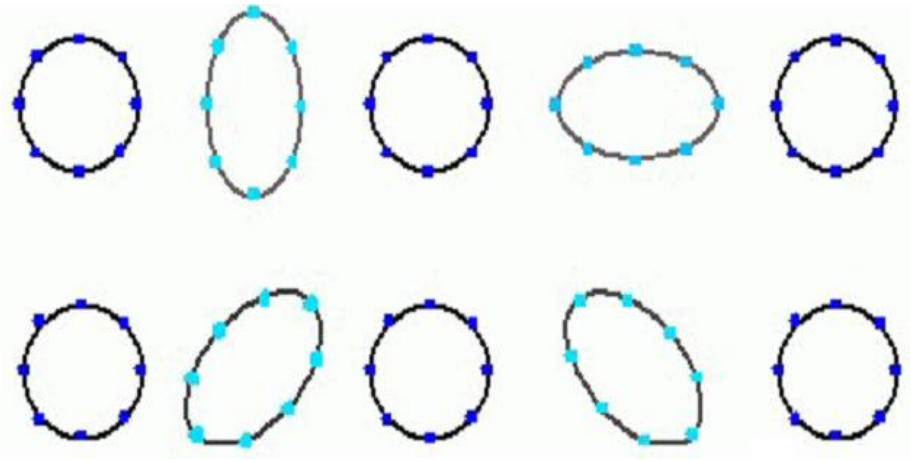


Figure 2.1: Effect of two polarizations on a ring of test masses.

In electromagnetic theory, it is dipole radiation which dominates. In contrast

to this, the dominating form of gravitational radiation is quadrupolar in nature. One can show that conservation of momentum prohibits dipolar gravitational radiation [5]. In *Gravitation* [5], it is shown that the power output of a source emitting gravitational radiation is given by

$$P_{mass\ quadrupole} = \frac{1}{5}\left\langle \dddot{I}_{jk}\dddot{I}^{jk}\right\rangle \quad , \tag{2.12}$$

where the overdot denotes differentiation with respect to time, the angular brackets denote a time average, and I_{jk} is the mass quadrupole moment given by

$$I_{jk} = \int \rho\left(x_j x_k - \frac{1}{3}\delta_{jk}r^2\right)d^3x \quad . \tag{2.13}$$

Not surprisingly, it can be shown that the $\bar{h}^{\mu\nu}$ also depend on the quadrupole moment. Often times, as is the case here, it is convenient to choose a coordinate system that remains at rest with respect to a free particle in the gravity-wave field. This choice of coordinate system is referred to as the *transverse-traceless*, or TT, gauge. Working in these coordinates, and neglecting any contribution to the radiation from higher multi-pole moments, Einstein first showed that $\bar{h}^{\mu\nu}$ is given by the quadrupole formula [5]

$$\bar{h}^{TT}_{\mu\nu} = \frac{2G}{c^4 r}\ \ddot{I}^{\ TT}_{\mu\nu} \quad , \tag{2.14}$$

where r is the source-observer distance.

Armed with the quadrupole formula, one can compute the waveforms of gravitational waves for various sources simply by performing the necessary time derivatives on the appropriate quadrupole moment describing the sources distribution of mass. After arriving at the formulae, one typically will apply higher order corrections termed post-Newtonian, or PN, corrections to the phase of the waveforms. Since we will ultimately use these waveforms to construct a bank of templates to search for signals in data we do not want to suffer appreciable loss due to the fact that our waveforms do not match any true gravity wave signal because they are not calculated accurately enough. There are not only higher order multipoles that can contribute to the gravity-wave field but also some relativistic corrections to the quadrupole formula. Since the technique of matched filtering is sensitive to the phase of the signal we need not correct the amplitude of the waveform. The waveforms for the plus and cross polarizations to the first post-Newtonian order are given in the next chapter.

2.2 Detectors

A new generation of gravitational wave detectors have now begun taking data. These new detectors are essentially large scale Michelson interferometers. Below is a very simplistic sketch of their design.

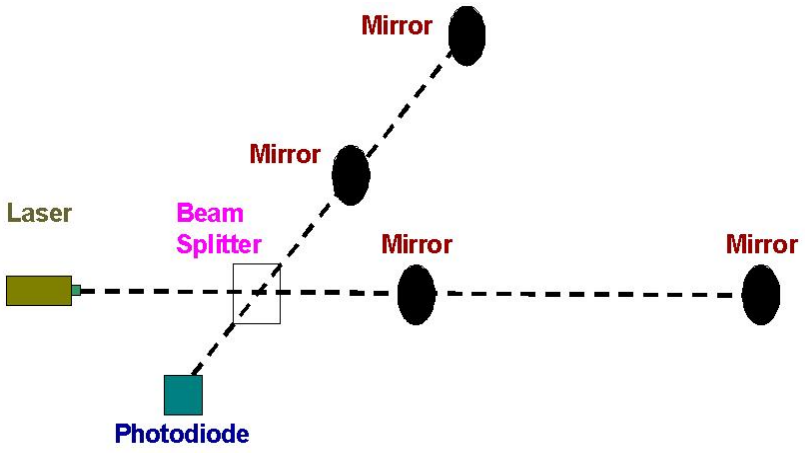


Figure 2.2: Simplified schematic of an interferometric detector.

In the United States, we are fortunate to have three such interferometers and have access to a fourth by virtue of a scientific collaboration. The Laser Interferometric Gravitational Wave Observatory (LIGO) Scientific Collaboration (LSC) is a worldwide collaboration and includes two detectors at Hanford, Washington, a detector at Livingston, Louisiana, and a fourth detector in Hannover, Germany. This collaboration has also begun working with scientists at other observatory sites, most notably those in Japan, Italy, and Australia. The detectors in the United States were commissioned first and have been continually improved upon to benefit from the best possible sensitivity. At Hanford, one detector has arms which are two kilometers long while the other, along with the detector at Louisiana, has arms that are 4 kilometers long.

The basic principle on which the detectors work is that of interference of light waves. A high power laser beam is split by a beam splitter, after which the two halves travel down the two different arms of the interferometer. They then bounce off the end mirror and travel back to be recombined at the beam splitter and detected at a series of photodiodes. If the waves travel different distances then interference will result and a series of bright and dark interference fringes will be seen. If there is an impinging gravitational wave it will act to stretch or squeeze the surrounding space-time, effectively causing the arms to have different lengths. This then leads to an overall phase difference in the light of the two arms. To see this effect, consider a plus polarized wave traveling perpendicular to the plane formed by the two detector arms termed x-arm and y-arm. For this scenario, equation (2.2) becomes

$$c^2 dt^2 = \left[1 + h_+ (t)\right] dx^2 + \left[1 - h_+ (t)\right] dy^2 + dz^2 \quad . \tag{2.15}$$

One can then calculate the amount of phase accumulated by the beam in a round trip down the x-arm,

$$\begin{aligned} \phi_x &= \int_0^{\tau_{rt}} 2\pi f_l \, dt \\ &= \frac{2\pi f_l}{c} \left[\int_0^L \sqrt{1 + h_+} \, dx - \int_L^0 \sqrt{1 + h_+} \, dx \right] \\ &\simeq \frac{4\pi f_l L}{c} \left(1 + \frac{h_+}{2} \right) \quad , \end{aligned} \tag{2.16}$$

where τ_{rt} is the round trip time, f_l is the laser frequency, L is the length of the arm and the binomial approximation has been used due to the perturbative nature of h_+. The extra amount of phase accumulation due to the presence of the gravity-wave, $\delta\phi_x$, is then

$$\delta\phi_x = \frac{2\pi h_+ L}{\lambda_l} \quad , \tag{2.17}$$

where λ_l is the wavelength of the laser. A similar calculation for the y-arm can be done to give

$$\delta\phi_y = -\frac{2\pi h_+ L}{\lambda_l} \quad , \tag{2.18}$$

to give an overall phase shift between the arms of (subtract eqn. (2.18) from (2.17))

$$\Delta\phi = \frac{4\pi h_+ L}{\lambda_l} \quad . \tag{2.19}$$

It is this phase difference that allows us to search for gravitational waves. To further amplify the phase difference, a Fabry-Perot cavity is set up in each arm by the use of a secondary mirror. This cavity holds the light for about 200 bounces, allowing the phase difference to build up to a more substantial value.

All of the laser interferometric detectors suffer from the same noise sources. To complicate this matter further, they are broadband detectors and are therefore susceptible to noise sources of many different frequencies. Displayed

in Figure 2.2 is the design sensitivity curve for the LIGO interferometer. On the y-axis is the dimensionless strain, or the change in arm length divided by the total arm length. At low frequencies we are bound by seismic noise and typically high pass the data at around 60 Hz. At the high frequency end, the quantum mechanical effect of photon shot noise is the dominating source of noise and we therefore low pass our data at around a kilohertz. Scattered across this band there are known sources of noise such as power line harmonics that occur at distinct frequencies. These are typically notched out of the data. After removing all these known noise sources, we are left with data that has a mostly Gaussian and stationary noise distribution. These two criterion are usually assumed valid in the formulation of a data analysis strategy. There are then vetoes in place that monitor the validity of the assumptions and allow us to discern relatively clean segments of data from polluted segments. In addition to those vetoes, there are hundreds of environmental channels at each site that monitor all different kinds of noise from many types of sources such as fluctuations in the local magnetic field, earthquakes, and many others.

Each detector is built with the goal of isolation from noise and amplification of any true signal in mind. To combat seismic noise, which can make it difficult to keep the detectors in operation, all the optics are placed on

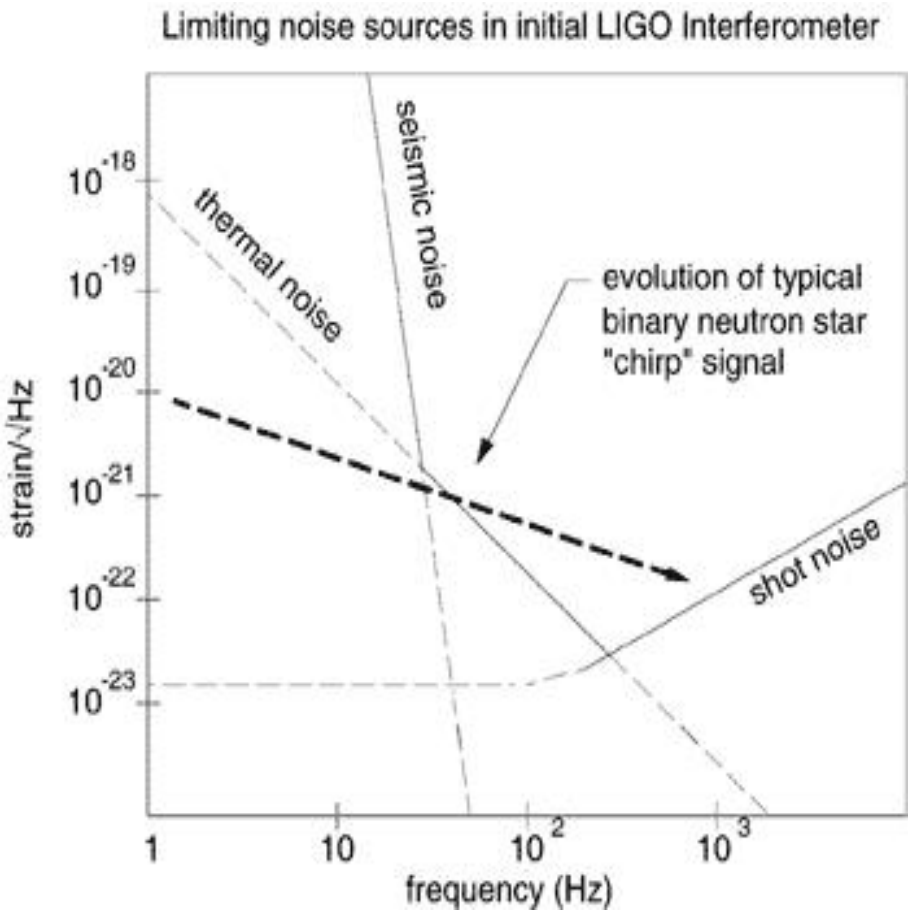


Figure 2.3: Prominent noise sources in LIGO.

tables with very robust seismic isolation. Because we are trying to measure arm length differences that are less than the size of the atomic nucleus, it is absolutely critical that the laser beam be pure. Prior to entering the beam splitter, the beam will travel through a mode cleaner and several other optics that ensure its purity. To make a true gravity wave signal show up more clearly, a relatively high powered laser is used. The laser can not however be made arbitrarily powerful because at some point the thermal noise in the

mirrors will start to dominate. There are many other details involved but are beyond the scope of this thesis.

Chapter 3

The Coherent Search

The technique of matched filtering involves correlating the detector output with a waveform that is constructed from the expected signal. The technique is derived from a more general and well known method of maximum likelihood detection [6]. This method involves the computation of a likelihood ratio and the comparison of this value to some threshold that is set based upon our understanding of the false alarm probability and detection probability. If the likelihood ratio is above the threshold value, then we will claim a detection, and if it is below then we will not claim a detection. There are two kinds of mistakes that we can make in such an analysis. A false alarm could occur if we claim a detection when there is not actually a true astrophysical signal present. A false dismissal results when we do not claim a detection in the presence of a true signal. Clearly, we would like to avoid making either of these mistakes. The setting of the threshold is the key. If

we set the threshold too low, we will get many spurious events which are due purely to noise. If we set the threshold too high we risk missing out on true signals that may be weak. To set the threshold appropriately, the false alarm probability and detection probability must be understood.

An additional difficulty is that the parameter space of signal characteristics is very large. This space must be adequately covered by a bank of templates so as not to miss out on any signals. A geometric formalism for creating this template bank has been put forth [7]. Typically, we set a requirement on the maximum distance that can occur between a template and a signal in the parameter space. Alternatively, one can view this as a constraint on the amount of signal to noise that we will suffer by filtering a signal with a template that is at this maximum distance away from the signal in parameter space. The notion of a signal to noise ratio and some of these other concepts will be developed below.

3.1 Mathematical Framework

Our first aim is to obtain a quantity that defines the *response* of an arbitrary network of broadband detectors to an incoming gravitational wave. In this quest, it is necessary to understand how the responses of arbitrarily oriented and arbitrarily located individual detectors to such a wave relate to one

another. This is aided by introducing the three different frames of reference that arise naturally in such a problem, namely, (i) the wave frame, (ii) the network frame, and (iii) the frame of a representative detector in the network. We define these reference frames in terms of the following right-handed, orthogonal, three-dimensional Cartesian coordinates:

(i) *Wave frame*: We associate with this frame the coordinates (X, Y, Z). The gravitational wave, which is assumed to be weak and planar, is taken to travel along the positive Z-direction; then X and Y denote the axes of the polarization ellipse of the wave.

(ii) *Network frame*: There is no unique definition for this frame. For Earth-based detectors being discussed here, if the network has a large number of detectors (say, $M > 3$), a convenient choice is a frame attached to the center of the Earth. Let the coordinate system that defines this frame be (x_E, y_E, z_E). The x_E axis lies along the line joining Earth's center and the equatorial point that lies on the meridian passing through Greenwich, England. It points radially outwards. The z_E axis is chosen to lie in the direction of the line passing through the center of Earth and the north pole. The y_E axis is chosen to form a right-handed coordinate system with the x_E and z_E axes.

For a network consisting of $M \leq 3$ detectors, certain calculations can be simplified by using the fact that the corner stations (or hubs) of all the detectors will lie on a single plane. Specifically, for $M = 3$ we define the network frame such that one of the detectors is at its origin, a second detector

is on one of the coordinate axes, say, z, and the third lies on one of the coordinate planes containing the z axis, say, the $x - z$ plane.

(*iii*) *Detector frame*: Let $(x_{(I)}, y_{(I)}, z_{(I)})$ (with I = 1,2,...,M) denote the orthogonal coordinate frame attached to the detector labeled I. The $(x_{(I)}, y_{(I)})$ plane contains the detector arms and is assumed to be tangent to the surface of the Earth. The $x_{(I)}$ axis bisects the angle between the detector's arms. The direction of the $y_{(I)}$ axis is chosen in such a way that $(x_{(I)}, y_{(I)}, z_{(I)})$ form a right-handed coordinate system with the $z_{(I)}$ axis pointing radially out of Earth's surface.

Apart from the above choices for frames, we define a fourth frame, namely, the frame of a "fiducial" detector (henceforth referred to as the "fide"). This frame serves as a reference with respect to which the locations or orientations of each of the detectors in a network shall be specified. Indeed, we will develop our whole formalism for a general network using the fide frame as a reference. When one considers specific cases of networks, it may prove useful to identify the fide frame with one of the three frames defined above, depending upon suitability.

Physical quantities in these frames are related by orthogonal transformations that rotate one frame into another. These orthogonal transformations are defined in terms of three sets of Euler angles that specify the orientation of one frame with respect to another. To understand these relations, let $\mathcal{O}$ be an orthogonal transformation matrix. Let (ϕ, θ, ψ) be the Euler angles through which one must rotate the fide frame to align it with the wave frame.

Then

$$\mathsf{x}_{\rm wave} = \mathcal{O}(\phi, \theta, \psi)\mathsf{x}_{\rm fide} \quad , \tag{3.1}$$

where $\mathsf{x}_{\rm fide}$ denotes the axes of the fide frame and $\mathsf{x}_{\rm wave}$ those of the wave frame. $\mathcal{O}$ is an orthogonal matrix as defined in Ref. [8]. Similarly, if $\boldsymbol{\alpha}_{(I)} = (\alpha_{(I)}, \beta_{(I)}, \gamma_{(I)})$ are the Euler angles that rotate the fide frame to the frame of the I-th detector, then

$$\mathsf{x}_{\rm detector_{(I)}} = \mathcal{O}(\alpha_{(I)}, \beta_{(I)}, \gamma_{(I)})\mathsf{x}_{\rm fide} \quad , \tag{3.2}$$

where $\mathsf{x}_{\rm detector_{(I)}}$ denote the axes of the I-th detector frame.

The following convention for symbols in this chapter will be used, unless otherwise specified. When it is useful to keep track of the complex nature of a network-based or individual detector-based *variable* we denote it by an uppercase Roman letter, whereas the lower case letters are reserved for real variables. Note that quantities such as the gravitational constant, G, though written in upper case, are not complex since they do not represent any inherent characteristic of the network or an individual detector. On the other hand, we shall not use an uppercase letter to denote a complex quantity when its complex nature is apparent from other means, such as by the use of a tilde, e.g., in $\tilde{n}$, which denotes the, in general, complex Fourier transform of the real quantity n. Network-based vectors are displayed in the Sans Serif font. The *label* I in the superscript or subscript of a variable denotes a (real) natural number that associates it with a particular detector. It ranges from

1 to M, where M is the total number of detectors in a network. It can be considered as a vector index over detectors. We use the index I for several of the network variables. However, certain quantities that do not obviously display a vector character, but still pertain to a detector are denoted by enclosing the index in parentheses, such a (I). Again, the Einstein summation convention over repeated indices is used for brevity, unless explicitly stated.

As described previously, a gravitational wave can be represented by a metric tensor fluctuation, $h_{\mu\nu}$, about the background space-time which we take to be flat. The μ and ν denote space-time indices and range from zero to four. In the TT gauge, the non-vanishing components of $h_{\mu\nu}$ in the wave frame are $h_{xx} = -h_{yy} \equiv h_+$, $h_{xy} = h_{yx} \equiv h_\times$. Here, h_+ and $h_\times$ are the two linear-polarization components of the wave. When a metric fluctuation specifically represents a gravitational wave, its spatial part is identified as twice the wave tensor, w_{ij}, where i and j refer to space indices and take values $1, 2$, and 3 (see Ref. [9]). In the TT gauge, the wave tensor is a symmetric trace-free (STF) tensor of rank 2 [10]. In any arbitrary frame, the wave tensor can be expressed in terms of its circular-polarization components as,

$$w^{ij}(t) = \frac{1}{2}\left[(h_+(t) + ih_\times(t))e_R^{ij} + (h_+(t) - ih_\times(t))e_L^{ij}\right] \quad , \tag{3.3}$$

where $e_{R,L}^{ij}$ are the right and left-circular polarization tensors, respectively. The $e_{R,L}^{ij}$ are both second rank STF tensors and obey the orthonormality

conditions,

$$e^{ij}_{L,\,R}\, e^{*}_{L,\,R\,ij} = 1\,, \quad e^{ij}_{L,\,R}\, e^{*}_{R,\,L\,ij} = 0 \quad , \tag{3.4}$$

where a star denotes complex conjugation. The reality of the wave tensor ensures that $e^{ij\,*}_{L} = e^{ij}_{R}$. Thus, the wave-tensor expression (3.3) simplifies to

$$w^{ij}(t) = \Re\left[(h_{+}(t) + ih_{\times}(t))\, e^{ij}_{R}\right] \quad , \tag{3.5}$$

where $\Re[A]$ denotes the real part of a complex quantity A.

In an arbitrary reference frame, the polarization tensor can be expressed as

$$e^{ij}_{L} = m^{i} m^{j} \quad , \tag{3.6}$$

where the m^k is the k-th component of a complex null vector m in that reference frame. It is defined as

$$m^{k} = \frac{1}{\sqrt{2}}(e^{k}_{X} + ie^{k}_{Y}) \quad , \tag{3.7}$$

where $\mathbf{e}_X$ and $\mathbf{e}_Y$ are unit vectors in the X and Y direction of the wave axes, respectively [9].

The I-th detector tensor, d^{I}_{ij}, is given by

$$d^{I}_{ij} = n_{(I)1i} n_{(I)1j} - n_{(I)2i} n_{(I)2j} \quad , \tag{3.8}$$

where $\mathsf{n}_{(I)1}$ and $\mathsf{n}_{(I)2}$ are the unit vectors along the arms of the I-th interfer-

ometer, which may not have orthogonal arms.

When a network consists of detectors around globe, there is in general a time delay between the arrival time of a true signal at each site. Let $\tau_{(I)}(\theta,\phi)$ be the relative delay between the arrival times at the I-th detector and the fide, where the source direction is given by (θ,ϕ). If $\hat{\mathsf{n}}(\theta,\phi)$ is the unit vector along the direction of the wave, i.e., $\hat{\mathsf{n}}(\theta,\phi)=\hat{Z}$, then

$$\tau_{(I)}(\theta,\phi)=\frac{(\mathsf{r}_{(I)}-\mathsf{r}_{(f)})\cdot\hat{\mathsf{n}}(\theta,\phi)}{c} \quad , \tag{3.9}$$

where $\mathsf{r}_{(I)}$ and $\mathsf{r}_{(f)}$ are the position vectors of the I-th detector and fide, respectively, with respect to any given reference frame. Note that $\tau_{(I)}(\theta,\phi)$ can take positive as well as negative values.

The signal in the I-th detector is the scalar

$$s^I(t)=w^{ij}(t-\tau_{(I)})d^I_{ij} \quad , \tag{3.10}$$

which, by definition, is invariant under coordinate transformations. Above, $w^{ij}(t)$ is the wave tensor at the location of the fide at time t. It is a function of $h_+(t)$ and $h_\times(t)$, which define the amplitudes of the two polarization components at time t and at the location of the fide. The above definition shows that the signal depends on the projections of the polarization tensors, $e^{ij}_{L,\,R}$, onto the I-th detector tensor, d^I_{ij}. These projections are

$$F^I=e^{ij}_L d^I_{ij}, \qquad\qquad F^{I*}=e^{ij}_R d^I_{ij} \quad , \tag{3.11}$$

which are the beam-pattern functions for the left- and right-circular polarizations, respectively. They depend on the direction of the source, the orientation and the arm-opening angle of the detector. Owing to any motion of the source with respect to the detector this orientation can change with time. Hence, in general, F^I are functions of time. Since we are only concerned with short-duration signals, we will assume these functions to be independent of time (which is a very good approximation). Using the above definition of the beam-pattern functions and the wave-tensor expression (3.5) in Eq. (3.10), we find the signal to be

$$s^I(t) = \Re\left[\left(h^I_+(t) + ih^I_\times(t)\right) F^{I*}\right] \quad , \tag{3.12}$$

where $h^I_+(t) \equiv h_+(t - \tau_{(I)})$ and $h^I_\times(t) \equiv h_\times(t - \tau_{(I)})$ are the time-delayed amplitudes of the two polarizations of the wave at detector I.

The signal from an inspiraling binary will typically not stand above the broadband noise of the interferometric detectors making the concept of an absolutely certain detection unattainable. Only probabilities can be assigned to the presence of an expected signal. In the absence of prior probabilities, such a situation demands a decision strategy that maximizes the detection probability for a given false alarm probability. This is termed as the Neyman-Pearson criterion [6]. Such a criterion implies that the decision must be based on a statistic called the likelihood ratio (LR). It is defined as the ratio of the probability that a signal is present in an observation to the probability that it

is not. This is the criterion we employ in formulating our detection strategy.

In order to define a strategy to search for signals in a noisy environment, it is important to recognize the characteristics of the noise. Here, we assume that the noise, $n^I(t)$, in the I-th detector (a) has a zero mean and (b) is mostly stationary[1] and statistically as well as algebraically independent of the noise in any other detector. These requirements are mathematically summarized, respectively, as:

$$\overline{n^I(t)} = 0, \tag{3.13}$$

$$\overline{\tilde{n}^*_I(f)\tilde{n}^J(f')} = s_{h(I)}(f)\delta(f-f')\delta^J_I \quad , \tag{3.14}$$

where the over-bar on a symbol denotes the ensemble average of that quantity and the tilde denotes the Fourier transform,

$$\tilde{n}^I(f) = \int_{-\infty}^{\infty} n^I(t)e^{-2\pi ift}dt\,. \tag{3.15}$$

Also, $s_{h(I)}(f)$ is the one sided power-spectral-density (PSD) of the I-th detector. Note that $s_{h(I)}(f)$ is the Fourier transform of the auto-covariance of the noise in detector I. We also assume the noise to be additive. This implies that when a signal is present in the data, then $x^I(t)$ is given by

$$x^I(t) = s^I(t) + n^I(t) \quad , \tag{3.16}$$

[1]In reality, detector noise contains non-Gaussian and non-stationary components. Such features can be accommodated in our treatment, by using vetoing techniques of the kind described in Ref. [11].

otherwise $x^I(t) = n^I(t)$. Here, $x^I(t)$ is the data of the $I-th$ detector, $s^I(t)$ is the signal, and $n^I(t)$ is the noise as above. Finally, we assume that the noises are Gaussian, i.e., the two moments in Eq. (3.13) and Eq. (3.14) are sufficient to completely characterize the noises statistically.

An important tool in the theory of detection of known signals in noisy environments is the cross correlation between a signal template and a detector's output. In order to define it, consider two real, sufficiently smooth, and absolutely integrable functions of time, namely, $a(t)$ and $b(t)$. We can assume that the signal template, $s^I(t)$, and the detector outputs, $x^I(t)$, belong to this category of functions to a good approximation. A cross correlation can be represented in terms of an inner product, which is defined as

$$\langle a,\, b\rangle_{(I)} = 2\Re \int_0^\infty df\, \frac{\tilde{a}^*(f)\tilde{b}(f)}{s_{h(I)}(f)} \quad , \tag{3.17}$$

where $\tilde{a}(f)$ and $\tilde{b}(f)$ are the Fourier transforms of $a(t)$ and $b(t)$, respectively, and the * denotes complex conjugation. In order to obtain the correlation between a complex function $A(t) = a_1(t) + ia_2(t)$ and a real function $b(t)$, we adopt the following convention to define the inner product

$$\langle A, b\rangle \equiv \langle a_1, b\rangle - i\langle a_2, b\rangle\,. \tag{3.18}$$

This definition is consistent with the convention of (3.17) where the complex conjugation is performed on the first entry in the inner product.

For a network of M detectors, the data consist of M data trains, $\{x^I(t)|I =$

$1, 2, ..., M$ and $t \in [0, T]\}$. The network matched-template can be obtained naturally by the maximum-likelihood method, where the decision whether the signal is present or not is made by evaluating the likelihood ratio (LR) for the network [6]. Under the assumptions made on the noise, the network LR, denoted by λ, is just a product of the individual detector LRs. In addition, for Gaussian noise, the logarithmic likelihood ratio (LLR) for the network is just the sum of the LLRs of the individual detectors [12],

$$\ln \lambda = \sum_{I=1}^{M} \ln \lambda_{(I)} \quad , \tag{3.19}$$

where

$$\ln \lambda_{(I)} = \langle s^I, x^I \rangle_{(I)} - \frac{1}{2} \langle s^I, s^I \rangle_{(I)} \, . \tag{3.20}$$

The network LLR takes a compact form in terms of the network inner-product,

$$\langle \mathsf{s}, \mathsf{x} \rangle_{NW} = \sum_{I=1}^{M} \langle s^I(t), x^I(t) \rangle_{(I)} \quad , \tag{3.21}$$

where

$$\mathsf{s}(t) = \big(s^1(t), s^2(t), \cdots, s^M(t) \big) \tag{3.22}$$

is the network template-vector, which comprises of individual detector templates as its components, and

$$\mathsf{x}(t) = \big(x^1(t), x^2(t), \cdots, x^M(t) \big) \tag{3.23}$$

is the network data-vector.

It can be shown by using the Schwarz inequality that the network template, s, defined above, yields the maximum signal-to-noise (SNR) amongst all linear templates and, hence, is the matched template. As shown in Ref. [12], in terms of the above definitions, the network LLR takes the following simple form:

$$\ln \lambda = \langle \mathsf{s}, \mathsf{x} \rangle_{NW} - \frac{1}{2} \langle \mathsf{s}, \mathsf{s} \rangle_{NW} \quad , \tag{3.24}$$

which is a function of the source parameters that determine s. Different selections of source-parameter values and, therefore, different values of s result in varying magnitudes of the LLR. The selection that gives the maximum value stands the best chance for beating the pre-set threshold on the LLR. Since scanning the complete source-parameter manifold for the maximum of LLR is computationally very expensive, we propose to perform its maximization analytically over as many parameters as possible. This requires the knowledge of the analytic dependence of the network matched-template on source parameters. This is what we seek below.

3.2 The Signal

The Inspiral Search Code (ISC), discussed in Chapter 4, has the functionality to search in the data from a single detector for the 2PN waveforms of inspiraling, non-spinning compact binaries. It does this by evaluating the detection

statistic (and the χ^2 statistic discussed later) and comparing it with a pre-set threshold. It might also be possible to extend this code to search for other waveforms, such as the ones corresponding to the 2.5 PN approximants. In any case, the ISC depends on the existence of a bank of templates for the corresponding waveform. Alternatively, one can implement the Fast Chirp Transform (FCT) code to evaluate the detection statistic. In either case, one needs to know beforehand the form of the signal in a given detector, and how it relates to the response of other detectors to the same source. The form of the network detection statistic given in the next section basically remains unaffected by the order of the PN approximation to which the waveforms are computed, as long as we use the *restricted* waveforms (no amplitude corrections). Therefore, for simplicity, we shall use the chirp expression obtained in the post^1-Newtonian, quadrupole approximation to express the detector responses. The 2PN results have also been worked out but the expressions are unwieldy to write down given their extreme complexity and size. However, the end results for the 2PN case are given.

We assume that the binary is at a luminosity distance of r from the Earth[2]. Further, we let m_1 and m_2 be the masses of the individual stars. Then, in the post^1-Newtonian approximation the two corresponding GW linear polarization components in the wave frame, at the location of the fide,

[2]Here, r is not to be confused with the magnitude of a detector position-vector, which always carries as an index the label of the detector, i.e., (I) or (f).

are

$$h_+(t; r, \delta_c, t_c, \xi_{1,2}) = \frac{2\mathcal{N}}{r} a^{-1/4}(t; t_c, \xi_1) \frac{1+\cos^2\epsilon}{2} \cos[\chi(t; t_c, \xi_{1,2}) + \delta_c] \quad (3.25)$$

$$h_\times(t; r, \delta_c, t_c, \xi_{1,2}) = \frac{2\mathcal{N}}{r} a^{-1/4}(t; t_c, \xi_1) \cos\epsilon \, \sin[\chi(t; t_c, \xi_{1,2}) + \delta_c] \quad , \quad (3.26)$$

where

$$\mathcal{N} \equiv \left[\frac{2G^{5/3}\mathcal{M}^{5/3}(\pi f_s)^{2/3}}{c^4}\right] \quad (3.27)$$

is a constant appearing in the chirp amplitude having the dimensions of length and where $\xi_{1,2}$ is sometimes used for brevity to mean both ξ_1 and ξ_2. It depends on the binary's 'chirp' mass, $\mathcal{M} \equiv (\mu^3 M^2)^{1/5}$, and a fiducial chirp frequency, f_s. Here, μ is the reduced mass of the binary and M is the total mass. Usually, f_s is taken to be the lowest frequency in the bandwidth of a detector - the seismic cut-off - hence the reason for the subscript s. This choice of the fiducial frequency maximizes the duration of tracking the chirp because the chirp frequency increases monotonically with time.

Quantities closely related to the chirp mass are the so-called chirp times,

$$\xi_1 = 34.5 \left(\frac{\mathcal{M}}{M_\odot}\right)^{-5/3} \left(\frac{f_s}{40\,\mathrm{Hz}}\right)^{-8/3} \text{ sec.} \quad , \quad (3.28)$$

$$\xi_2 = 0.0837 \left(\frac{743}{84} + \frac{11\mu}{M}\right) \left(\frac{M_\odot}{\mu}\right) \left(\frac{f_s}{40}\right)^{-2} \text{ sec.} \quad , \quad (3.29)$$

whose sum equals the time duration for which the chirp exists in a detector's sensitivity window from the time of arrival until the time of final coalescence.

The time of arrival, t_a, is defined as the time when the instantaneous frequency of the chirp equals the fiducial frequency, i.e., $f(t_a) = f_s$. Formally, the coalescence time, t_c, is the time at which the chirp frequency diverges. The corresponding phase of the wave-form at t_c is δ_c. We define the quantity

$$a(t; t_c, \xi_1) = \frac{t_c - t}{\xi_1} \quad , \tag{3.30}$$

and the instantaneous frequency is given by an implicit equation,

$$t = \xi_1 + \xi_2 - \xi_1 \left(x^{-8/3}\right) - \xi_2 \left(x^{-2}\right) \quad , \tag{3.31}$$

where $x = f(t; \xi_1, \xi_2)/f_s$. Finally, the instantaneous phase of the waveform is $\chi(t) + \delta_c$, where

$$\chi(t; f_s, t_c, \xi_1, \xi_2) \equiv -2\pi \int_t^{t_c} f(t'; f_s, t_c, \xi_1, \xi_2) dt' \quad . \tag{3.32}$$

The two GW polarization amplitudes at the I-th detector site are obtained by substituting t with $(t - \tau_{(I)})$ in Eqs. (3.25), (3.26), (3.30), (3.31), and (3.32).

A chirp signal registers itself in a detector's output only after its instantaneous frequency crosses the seismic cut-off of that detector. Thus, a signal arrives in the I-th detector's bandwidth when its instantaneous frequency reaches $f = f_{s(I)}$ and it lasts there for a time duration equaling $\Xi_{(I)} = \xi_1 \left(f_{s(I)}/f_s\right)^{-8/3} + \xi_2 \left(f_{s(I)}/f_s\right)^{-2}$. Alternatively put, the chirp wave-

form at the I-th detector starts at $t = t_c + \tau_{(I)} - \Xi_{(I)}$ and ends at $t = t_c + \tau_{(I)}$.

In order to formulate a strategy for detecting a chirp, it helps to isolate the factors in the two polarizations, $h_{+,\times}$, that are time dependent from those that are not. To this end, we define two mutually orthogonal normalized waveforms s^I_0 and $s^I_{\pi/2}$, with $s^I_{0,\pi/2}(t) = s_{0,\pi/2}(t - \tau_{(I)})$. Their complex combination, $S^I = s^I_0 + i s^I_{\pi/2}$, is a normalized complex signal; it has the simple form:

$$S^I(t; t_c, \xi_1, \xi_2) \equiv \frac{a^{-1/4}(t - \tau_{(I)}; \xi_1)}{g_{(I)}\sqrt{\xi_1}} e^{i\chi(t-\tau_{(I)};\xi_1,\xi_2)}\,. \tag{3.33}$$

Here, $g_{(I)}$ is a normalization factor such that

$$\langle S^I, S^I \rangle_{(I)} = 2\,. \tag{3.34}$$

Now an expression for the normalization factor, $g_{(I)}$, will be obtained.

In the stationary-phase approximation (SPA), the Fourier transform of $S^I(t)$ for positive frequencies is,

$$\begin{aligned} \tilde{S}^I(f; t_c, \xi_1, \xi_2) &= \int_{-\infty}^{\infty} S^I(t; t_c, \xi_1, \xi_2) e^{-2\pi i f t} dt \\ &= \frac{2}{g_{(I)}} \sqrt{\frac{2}{3 f_s}} \left(\frac{f}{f_s}\right)^{-7/6} \exp\left[i\Psi_{(I)}(f; f_s, t_c, \xi_{1,2})\right] \end{aligned} \tag{3.35}$$

where

$$\Psi_{(I)}(f; f_s, t_c, \xi_1, \xi_2) = -2\pi f_s \times \left[\frac{f}{f_s}t_c + \frac{f}{f_s}\tau_{(I)} + \frac{3}{5}\xi_1\left(\frac{f}{f_s}\right)^{-5/3} + \xi_2\left(\frac{f}{f_s}\right)^{-1}\right] + \frac{\pi}{4} \equiv \Psi(f; f_s, t_c, \xi_1, \xi_2) - 2\pi f\tau_{(I)} \quad , \tag{3.36}$$

for the 1PNchirp. [3] Note that $\Psi_{(I)} = \Psi$ for vanishing time-delay ($\tau_{(I)} = 0$). Thus, Ψ defines the phase in the Fourier transform of the normalized complex signal at the fide, in the SPA. The normalization condition (3.34) implies that,

$$g^2_{(I)} = \frac{4}{3}{f_{s(I)}}^{4/3}\int_{f_{s(I)}}^{\infty}\frac{df}{f^{7/3}s_{h(I)}(f)} \quad , \tag{3.37}$$

where $f_{s(I)}$ is the seismic cut-off for the I-th detector.

The signal due to a 1PN chirp at the I-th detector can be expressed in terms of $S^I(t)$. In the special case of the face-on binary (i.e., $\epsilon = 0$), the signal at the detector is given by

$$s^I(t) = 2\kappa\Re\left[g_{(I)}F^*_I S^I(t)e^{i\delta_c}\right] \quad , \tag{3.38}$$

where $\kappa \equiv \mathcal{N}\sqrt{\xi}_1/r$ and ϵ is the orientation angle of binary's orbital plane. Note that δ_c is detector independent and separates out as a phase factor in the expression for $s^I(t)$.

[3] For the restricted 2PN waveform, the signal gets modified through terms appearing essentially in the phase $\Psi(f)$. These terms depend on powers of f that are different from those appearing in Eq. (3.36) and are parameterized by both ξ and the 1.5PN chirp time.

The generalization of (3.38) for arbitrary value of ϵ is straightforward. In this case, s^I can be expressed as follows:

$$s^I(t) = 2\kappa\, \Re\left[(E_I^* S^I) e^{i\delta_c}\right] \quad , \tag{3.39}$$

where we have defined the extended beam-pattern functions

$$E^I = g_{(I)} \left[\frac{1+\cos^2\epsilon}{2}\Re(F^I) + i\cos\epsilon\Im(F^I)\right] . \tag{3.40}$$

Here, $\Re(F^I)$ and $\Im(F^I)$ are the real and imaginary parts of the detector beam pattern functions, respectively. In the limit $\epsilon \to 0$, the signal in Eq. (3.39) reduces to that in (3.38). E^I depends on the source-direction angles, $\{\theta, \phi\}$, the inclination and polarization angles $\{\epsilon, \psi\}$, as well as on the orientation of the I-th detector relative to the fide, given by the Euler angles $\boldsymbol{\alpha}_{(I)}$. Also, E^I depends on the signal-normalization factor $g_{(I)}$, which expresses the sensitivity of the detector to the incoming signal. Thus, the signal at a detector depends on a total of nine independent parameters, namely, $\{r, \delta_c, \psi, \epsilon, t_c, \xi_1, \xi_2, \theta, \phi\}$. The ranges of the four angles are as follows: $\psi \in [0, 2\pi]$, $\epsilon \in [0, \pi]$, $\phi \in [0, 2\pi]$, and $\theta \in [0, \pi]$. The 2PN signal depends on an additional parameter, *viz.*, the 1.5PN chirp time.

The total energy in a signal that is accessible to a network is the network

scalar $\langle \mathsf{s}, \mathsf{s} \rangle_{NW}$, and is given by

$$\begin{aligned} \langle \mathsf{s}, \mathsf{s} \rangle_{NW} &= \sum_{I=1}^{M} \langle s^I(t), s^I(t) \rangle_{(I)} \\ &= 4\kappa^2 \sum_{I=1}^{M} {E^*}_I E^I \equiv \mathsf{b}^2. \end{aligned} \tag{3.41}$$

The quantity $\sum_{I=1}^{M} {E^*}_I E^I \equiv \mathsf{E} \cdot \mathsf{E} = \| \mathsf{E} \|^2$ is the $\mathcal{L}^2$ norm of E^I in $\mathcal{C}^M$. The above analysis also suggests the normalization for the network signal. The signal vector with unit norm is defined by $\hat{\mathsf{s}} \equiv \mathsf{s}/\mathsf{b}$. Its components are

$$\hat{s}^I = \Re \left[({Q_I}^* S^I) e^{i\delta_c} \right] \quad , \tag{3.42}$$

where

$$Q^I \equiv \frac{E^I}{\| \mathsf{E} \|} \, . \tag{3.43}$$

Note that the network vector $\mathsf{Q} = (Q^1, Q^2, ..., Q^M)$ lies in the M dimensional complex space $\mathcal{C}^M$ and has a unit norm, i.e.,

$$\| \mathsf{Q} \|^2 = 1 \, . \tag{3.44}$$

Like E, even Q depends on $\{\psi, \epsilon, \theta, \phi; \boldsymbol{\alpha}_{(1)}, ..., \boldsymbol{\alpha}_{(M)}\}$. But for a given choice of values for the angles $\{\theta, \phi; \boldsymbol{\alpha}_{(1)}, ..., \boldsymbol{\alpha}_{(M)}\}$, one can prove that Q is always restricted to lie in a two-dimensional complex subspace of $\mathcal{C}^M$ [10]. This subspace is termed as the "helicity" plane, $\mathcal{H}$. Thus, given a network of detectors (with their fixed orientations), choosing a source direction is tantamount to

choosing a specific helicity plane in $\mathcal{C}^M$. Furthermore, by varying ψ and ϵ one picks out different "directions" in this plane along which Q lies.

3.3 The Network Statistic

In the case of a single detector, the LLR is a functional of the data as measured by that detector. For a network of M detectors, one needs to compute the statistic in terms of the network data vector x. When our assumptions about the statistical properties of detector noise are valid, the appropriate network LLR is given by Eq. (3.24). The optimal network statistic is obtained by maximizing this LLR over the eight physical parameters that define the signal. It is this maximized LLR that must be compared with a pre-set threshold, corresponding to a given false alarm probability. How such a maximization can be performed over b, δ_c, ψ, and ϵ is shown in [10]. Further maximization over the time of final coalescence (or, analogously, over the time of arrival) can be performed by using FFTs in a way that is very similar to what is done in the case of a single detector [1]. Letting

$$C_I^* \equiv \langle S^I, X^I \rangle_{(I)} \quad , \tag{3.45}$$

we can write the LLR maximized over the four parameters, Λ, as

$$\Lambda = \| \, \mathsf{C}_{\mathcal{H}} \, \|^2 \quad , \tag{3.46}$$

where $\mathsf{C}_{\mathcal{H}}$ is is the correlation vector of (3.45) projected onto the helicity plane by a suitable choice of ϵ, and ψ. Performing these maximizations yields maximum likelihood estimates of the parameters. The results are

$$\hat{\delta}_c = Arg(\mathsf{C} \cdot \mathsf{Q}) \quad , \tag{3.47}$$

$$\hat{\mathsf{b}} = \sum_{I=1}^{M} \langle \hat{s}^I, x^I \rangle_{(I)} \quad , \tag{3.48}$$

$$\hat{\psi} = \frac{1}{4} Arg(w) \quad , \tag{3.49}$$

$$\hat{\epsilon} = \cos^{-1}\left[(1 - \sqrt{\| \, w \, \|}) / (1 - \sqrt{\| \, w \, \|}) \right] \quad , \tag{3.50}$$

where the over hats denote the value of the parameter that maximizes the statistic. Here, w is given by

$$w \equiv \frac{C_{+2}}{C_{-2}}, \tag{3.51}$$

where

$$C_{\pm 2} = \mathsf{C} \cdot \hat{\mathsf{D}}_{\pm 2} \quad , \tag{3.52}$$

and where $\hat{\mathsf{D}}_{\pm 2}$ is a vector on the space of detectors with components given by

$$D^I_{\pm 2} = -i g_{(I)} T_{\pm 2}{}^{s}(\phi, \theta, 0) \left[T^{2*}{}_{s}(\boldsymbol{\alpha}_{(I)}) - T^{-2*}{}_{s}(\boldsymbol{\alpha}_{(I)}) \right] \quad . \tag{3.53}$$

Here, the s index takes values $0, \pm 1, \pm 2$, and the T's are Gel' Fand functions described in more detail in the final chapter as well as in [13].

A basis for the helicity plane can be formed by first decomposing D_{+2} into its real and imaginary parts

$$\mathsf{D}_{+2} = \mathsf{d}_1 + \mathsf{i}\mathsf{d}_2 \quad , \tag{3.54}$$

where d_1 and d_2 are real vectors on the detector space. The unit basis vectors can then be formed by

$$\hat{v}^{\pm} \equiv (\hat{\mathsf{d}}_1 \pm \hat{\mathsf{d}}_2)/ \parallel \hat{\mathsf{d}}_1 \pm \hat{\mathsf{d}}_2 \parallel \quad , \tag{3.55}$$

where the hat denotes unit magnitude vectors. The maximized statistic Λ can be written in terms of these basis vectors as

$$\Lambda = |\hat{v}^{+} \cdot \mathsf{C}|^2 + |\hat{v}^{-} \cdot \mathsf{C}|^2 \quad . \tag{3.56}$$

Here, $(\hat{v}^{+}, \hat{v}^{-})$ are two orthonormal real vectors. They form a basis in which any complex vector in $\mathcal{H}$ can be expanded. The vectors $\hat{v}^{\pm}$ depend on the detector orientations and the source-direction angles, i.e.,

$$\hat{v}^{\pm} = \hat{v}^{\pm}(\theta, \phi; \boldsymbol{\alpha}_{(1)}, ..., \boldsymbol{\alpha}_{(M)}) \quad , \tag{3.57}$$

(for details, refer [10, 1]). Thus, the statistic (3.56) remains to be maximized

over $\{\theta, \phi\}$. This is done in practice by using a grid on this two-dimensional space of source-direction angles.

To use the statistic (3.56) for a coherent search using the LIGOs and GEO, the detector in Germany, one first picks a discrete set of source directions. For each of these directions, one computes the basis vectors $\hat{v}^{\pm}$, which can be viewed as a new beam pattern function. A Mathematica notebook that performs this calculation over specified θ and ϕ steps has been written and is given in Appendix B. Separately, one constructs C by collecting the cross-correlation functions $C^{(I)}$ that are obtained for a given template by running the Inspiral Search Code through the individual data streams. For each template and source direction, one uses the computed $C^{(I)}$ and $\hat{v}^{\pm}$ in Eq. (3.56) to obtain the value of the network statistic, as a function of τ. To compute the network statistic (3.56) over the full parameter space, one needs a code that repeats the above steps for all templates and for each (discretely sampled) source direction. Such a code has been developed and will be discussed in a Chapter 4.

3.4 Distribution of the Test Statistic

The maximum-likelihood method involves computing the likelihood ratio for given data and comparing it with a predetermined threshold. In some cases it is more useful to replace the likelihood ratio by another quantity derived from

it. When the likelihood ratio is a function of several parameters, it is often possible to maximize it analytically over some of these parameters, as shown above. In our case, such a maximization led to a reduced statistic derived from the LLR. We call this statistic Λ. To detect the presence of a signal in the data one must, therefore, compute the values that Λ takes at all the grid points on the space of the remaining parameters. Each of these values must then be compared with the threshold, Λ_0, to infer the presence or absence of a signal. The value of Λ_0 can be obtained via the Neyman-Pearson decision criterion [6], given the predetermined value of the false-alarm probability, Q_0, associated with the event of detection of the signal. When $\Lambda < \Lambda_0$, we conclude that the signal is absent in the data, whereas when $\Lambda > \Lambda_0$, the detection of the signal is announced.

To compute the false-alarm probability, Q_0, and the detection probability, Q_d, we need to know the probability distribution of Λ in the absence of the signal, i.e., $p_0(\Lambda)$, and in the presence of the signal, i.e., $p_1(\Lambda)$. We note that as evident in equation (3.56), Λ can be written as the sum of squares of four Gaussian random variables, $c_0^{\pm}$ and $c_{\pi/2}^{\pm}$ [1]. If our assumed properties of the detector noises are valid, then in the absence of a signal, i.e., for hypothesis H_0 or the null hypothesis, each of the random variables has a mean equal to zero and variance of unity [1]. One can then conclude from standard literature (see, e.g., Ref. [6]) that the probability density function for Λ, under the H_0 hypothesis, is a χ^2 distribution with four degrees of freedom,

and is given by

$$p_0(\Lambda) = \frac{\Lambda}{4} \exp(-\Lambda/2)\,. \tag{3.58}$$

The false alarm probability Q_0 is then obtained to be

$$Q_0 = \int_{\Lambda_0}^{\infty} p_0(\Lambda) d\Lambda = \left(1 + \frac{\Lambda_0}{2}\right) \exp(-\Lambda_0/2)\,. \tag{3.59}$$

The value of Q_0, which is inferred from astrophysical estimates of event rates and detector sensitivities, determines the detection threshold Λ_0 through the above equation.

The detection probability is obtained from the probability distribution $p_1(\Lambda)$. In order to calculate $p_1(\Lambda)$, we need the norm of the average network correlation-vector when the template gives a perfect match with the data. Assuming that the strength of the signal in the data is b, i.e., if $x^I(t) = \mathsf{b}\hat{s}^I(t) + n^I(t)$, then the average value of the network correlation-vector is $\bar{\mathsf{C}} = \mathsf{b}\mathsf{Q}e^{-i\delta_c}$. Therefore,

$$\| \overline{\mathsf{C}} \|^2 = \mathsf{b}^2 \quad , \tag{3.60}$$

for which we obtain (see, e.g., Ref. [6])

$$p_1(\Lambda) = \frac{1}{2} \left(\frac{\sqrt{\Lambda}}{\mathsf{b}}\right)^{1/2} \exp\left[-\frac{\Lambda + \mathsf{b}^2}{2}\right] I_1(\mathsf{b}\sqrt{\Lambda}) \quad , \tag{3.61}$$

where I_1 is the modified Bessel function. The detection probability itself is

$$Q_d = \int_{\Lambda_0}^{\infty} p_1(\Lambda) d\Lambda \quad , \tag{3.62}$$

which we now obtain in the large SNR limit. In terms of the network statistic, $Ł \equiv \sqrt{\Lambda}$, this asymptotic limit amounts to the condition $\mathsf{b}Ł >> 1$, and Eq. (3.61) approximates to a Gaussian distribution:

$$p_1(Ł) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{(Ł - \mathsf{b})^2}{2}\right] . \tag{3.63}$$

Thus, in the large SNR limit the network statistic is a Gaussian with mean approximately equal to the network strength of the signal, b. The false-alarm rate is typically taken to be one per year and the detection probability taken to be 95%. Then, assuming a sampling rate of about 2 kHz, we arrive at a false-alarm probability of $Q_0 \sim 1.5 \times 10^{-11}/n_{tot}$, where n_{tot} is the total number of templates used. For the sake of this calculation, we assumed that output samples in the correlation vector are uncorrelated. The correlation between these samples will reduce Q_0 but this does not make appreciable difference to the thresholds and sensitivities [14]. The threshold Λ_0 is then computed using Eq. (3.59).

The detection statistic is obtained by maximizing Λ over ϑ^α, which comprises of the five parameters $(t_c, \xi_1, \xi_2, n_3, n_1)$, where the last two define the projections of the normal sky-position vector on the z- and x-axis, respectively, of the fiducial detector. We define Λ as the *network* rectified out-

put of a template with parameters $\vartheta^{\alpha}_{m_\alpha}$, where m_α are the discretization indices. Therefore, matched-filtering the data yields the rectified outputs $\Lambda(t_c, \vartheta^{i}_{m_i}) = \Lambda_{m_i}(t_c)$. When the data is a discrete time-series, the rectified outputs are denoted by $\Lambda^{j}_{m_i}$, where j is the time index. The detection statistic is then given by

$$Ł = \Lambda^{j}_{m_i}\big|_{j,m_i}\,. \tag{3.64}$$

A two-step search involves first matched filtering the data by coarsely spaced templates followed by a second step where only the data corresponding to the candidate-events from the first step are filtered using fine spaced templates. Whereas increasing the template spacing at the first step lowers the computational costs of that step, it actually increase the costs of the second step by increasing the number of false-alarms masquerading as candidate events from the first step. Thus, there exists an optimal ratio of the first- and second-step template spacings for which the computational costs are minimized. This type of search is discussed in Chapter 5.

3.5 False-Alarm Probability

Here we ask: Given a template, with parameters ϑ_{m_i}, what is the probability that matched filtering the data with it will lead to a false-alarm? Addressing this question needs determining the joint probability distribution of the complete set rectified outputs of this template, $\{\Lambda_{m_i}\}$, under the null hypothesis. For a time-series data set with N_p points, let us denote the complete set

of rectified outputs as $\{\Lambda^0_{m_i}, \Lambda^1_{m_i}, ..., \Lambda^{N_p-1}_{m_i}\}$ Then, under this hypothesis, a rectified output of a template is itself the sum of the squares of two uncorrelated, Gaussian random variables that have vanishing means. Thus, $\Lambda^j_{m_i}$, has the Rayleigh distribution [14] $R(\Lambda^j_{m_i})$, defined as

$$R(\Lambda^j_{m_i}) = \Lambda^j_{m_i} e^{-\left(\Lambda^j_{m_i}\right)^2/2}. \tag{3.65}$$

Let the joint probability distribution of the rectified outputs be $P_b(\Lambda^0_{m_i}, \Lambda^1_{m_i}, ..., \Lambda^{N_p-1}_{m_i})$, where b takes the binary values 0 and 1 under hypotheses H_0 and H_1, respectively. Thus, P_0 is a multivariate Rayleigh distribution, and is related to the multivariate χ^2 distribution. Given a detection threshold, Ł = Ł$_0$, the false-alarm probability, Q_0, is the area under the curve P_0 for $\Lambda^j_{m_i} > \Lambda_0$, for all j. Thus,

$$Q_0(\text{Ł}_0) = 1 - \int_0^{\Lambda_0} ... \int_0^{\Lambda_0} d\Lambda^0_{m_i} ... \Lambda^{N_p-1}_{m_i} \; P_0(\Lambda^0_{m_i}, \Lambda^1_{m_i}, ..., \Lambda^{N_p-1}_{m_i}) \tag{3.66}$$

As was noted in Ref. [14], it is non-trivial to compute the above integral. This is primarily because P_0 itself is non-trivial owing to non-vanishing covariances among the $\Lambda^j_{m_i}$. Monte Carlo studies to estimate the False Alarm probability are discussed in Chapter 5.

3.6 Detection Probability

Under the alternative hypothesis in which a signal is present in the data, the four Gaussian random variables into which the statistic can be decomposed, call them $c_0^{\pm}$ and $c_{\pi/2}^{\pm}$, no longer have zero mean. Furthermore, the distribution of $\Lambda_{m_i}^j$ goes over to the Rician [14] (which is related to the more well-known non-central chi-square; if r has a Rician distribution, then r^2 has the non-central chi-square distribution),

$$\chi'(\Lambda_{m_i}^j) = \Lambda_{m_i}^j \exp\left[-\left({\Lambda_{m_i}^j}^2 + \mu_{m_i}^2\right)\right] I_0(\mu_{m_i}\Lambda_{m_i}^j) \quad , \tag{3.67}$$

where I_0 is the modified Bessel function of the first kind and

$$\mu_{m_i}^2 = \left(\overline{c_{i\ 0}^{\pm}}\right)^2 + \left(\overline{c_{i\ \pi/2}^{\pm}}\right)^2 . \tag{3.68}$$

The detection probability is defined as the area under the curve P_1 for $\Lambda_{m_i}^j > \Lambda_0$, for all j. Thus,

$$Q_d(Ł_0) = 1 - \int_0^{\Lambda_0} ... \int_0^{\Lambda_0} d\Lambda_{m_i}^0 ... \Lambda_{m_i}^{N_p-1}\ P_1(\Lambda_{m_i}^0, \Lambda_{m_i}^1, ..., \Lambda_{m_i}^{N_p-1}) \tag{3.69}$$

Because of the difficulty in arriving at an expressions for the probability density functions P_0 and P_1, we typically perform Monte Carlo studies to estimate them as discussed in Chapter 5.

3.7 Template Bank in ξ_1, ξ_2, θ, and ϕ

Recall that the LLR is a function of nine parameters, namely $\{r, \delta_c, \epsilon, \psi, t_c, \xi_1, \xi_2, \theta, \phi\}$ for the 1PN chirp. As Ł can be maximized over t_c numerically via the FFT, we only need to lay the templates in the rest of the P-dimensional parameter space, comprising of $\{\xi_1, \xi_2, \theta, \phi\}$. In other words, we need to compute the metric γ_{ij} in the P-dimensional subspace. It is determined by projecting the metric $g_{\alpha\beta}$ onto the subspace orthogonal to t_c. We then obtain

$$\gamma_{ij} = g_{ij} - \frac{g_{0i} g_{0j}}{g_{00}} \,. \tag{3.70}$$

The number of templates is obtained as follows. We compute the proper volume of the parameter space with the metric γ_{ij} and multiply the volume by the number density of the templates. Fixing the value of μ determines the grid spacing of the network templates in the parameter space. The number density, $\rho_P(\mu)$, which is the number of templates per unit proper volume, is given by

$$\rho_P(\mu) = \left(\frac{1}{2} \sqrt{\frac{P}{\mu}} \right)^P \,. \tag{3.71}$$

It is defined to be uniform over the whole parameter space and, therefore, its use is applicable as long as the curvature of the astrophysically interesting region of the manifold described by γ_{ij} is sufficiently small, and the effects arising from the boundary of the region are negligible.

The total volume of the parameter space is

$$\mathcal{V} = \int_{\mathcal{P}} \sqrt{\det \parallel \gamma_{ij} \parallel} \; d^P\vartheta \,. \tag{3.72}$$

Thus, the total number of templates required is , $n_f = \mathcal{V} \times \rho_P(\mu)$. In general, the total number of templates depends on the source parameters $\{\epsilon, \psi\}$. In order to scan the parameter space (for a given mismatch μ) for each pair of values $\{\epsilon, \psi\}$, we must maximize the volume $\mathcal{V}$ over $\{\epsilon, \psi\}$. This is tantamount to choosing the finest bank of templates. For simple cases, this is straightforward and has been implemented in some examples in section 3.9 of this chapter. In general, however, such a maximization is non-trivial to perform.

3.8 Metric on the Parameter Space

The metric $g_{\alpha\beta}$ defined on this four-dimensional space is related to the amount of drop in the statistic, Ł, and is obtained by expanding the statistic about the maximum. The squared statistic can be rewritten as,

$$\begin{aligned} \text{Ł}^2 &= |\hat{v}^{+\prime} \cdot \mathsf{C}|^2 + |\hat{v}^{-\prime} \cdot \mathsf{C}|^2 \\ &\equiv p'^J_I C^I C^*_J \,. \end{aligned} \tag{3.73}$$

The quantity ${p'_I}^J$ is a projection tensor given by,

$$ {p'_I}^J \equiv {v'}^{+J} v'^{+}_I + {v'}^{-J} v'^{-}_I \quad , \tag{3.74} $$

which projects a vector in $\mathcal{C}^M$ on the helicity plane spanned by $\hat{v}'^{\pm}$. It obeys the identities

$$ {p'_I}^J {p'_J}^K = {p'_I}^K \, , \qquad {p'_I}^J {p'_J}^I = 2 \quad , \tag{3.75} $$

which are consistent with its being a projection tensor on a two-dimensional plane. The primed coordinates refer to the template.

Let ϑ^α and $\vartheta'^\alpha = \vartheta^\alpha + \Delta\vartheta^\alpha$ be the parameters corresponding to the signal and the network template, respectively. For computing the metric, one takes normalized templates for the signal as well as the template, so that the maximum value of Ł is unity when the parameters of the signal and template match. In the absence of noise, we have

$$ C^*_I = \langle S'^I, s^I \rangle_{(I)} \simeq e^{i\delta_c} Q^*_I \langle S'^I, S^I \rangle_{(I)} \quad , \tag{3.76} $$

where, the S'^I denotes the template. The above expression is exact within the SPA. So the statistic can be written as,

$$ \text{Ł}^2 = {p'_I}^J Q^I Q^*_J \Theta_{(I)(J)} \quad , \tag{3.77} $$

where

$$\begin{aligned}\Theta_{(I)(J)} &\equiv \langle S^I(t;\vartheta'^\mu), S^I(t;\vartheta^\mu)\rangle^*_{(I)} \langle S^J(t;\vartheta'^\mu), S^J(t;\vartheta^\mu)\rangle_{(J)} \\ &= \int_{f_{s(I)}}^{\infty} df \mathcal{A}_{(I)}(f) \exp\left(-i\Phi_{(I)}(f;\vartheta^\mu,\Delta\vartheta^\mu)\right) \\ &\qquad \times \int_{f_{s(J)}}^{\infty} df \mathcal{A}_{(J)}(f) \exp\left(i\Phi_{(J)}(f;\vartheta^\mu,\Delta\vartheta^\mu)\right) \qquad (3.78)\end{aligned}$$

is the product of the individual ambiguity functions of the I-th and J-th detectors. It is a measure of how distinguishable the two wave-forms, i.e., the signal and the template, are. Here,

$$\mathcal{A}_{(I)}(f) \equiv \frac{4}{3 f_s g^2_{(I)}} \frac{1}{s_{h(I)}(f)} \left(\frac{f}{f_s}\right)^{-7/3} \quad , \qquad (3.79)$$

which satisfies the normalization condition $\int_{f_{s(I)}}^{\infty} \mathcal{A}_{(I)}(f) df = 1$. Note that in the limit of $\Delta\vartheta^\alpha \to 0$, the projection tensor of the filter is same as that of the projection tensor of the signal, *i.e.* ${p'_I}^J \to {p_I}^J$. In this limit, the projection tensor of the signal obeys the relation ${p_I}^J Q^I Q^*_J = 1$ and noting that if the S^I's are normalized to unity, we can see from Eq. (3.77) that $Ł^2 \to 1$ as desired.

The correlation phase, $\Phi_{(I)}(f;\vartheta^\mu,\Delta\vartheta^\mu)$, is given by (see Eq. (3.36)):

$$\Phi_{(I)}(f;\vartheta^\mu,\Delta\vartheta^\mu) = \Psi_{(I)}(f;f_s,t_c,\xi_1,\xi_2) - \Psi_{(I)}(f;f_s,t'_c,\xi'_1,\xi'_2)\,. \qquad (3.80)$$

The correlation phase includes the contribution from the differential time-

delay between the signal and the template. Instead of using (θ, ϕ) to specify the direction to the source we use the components n_1 and n_3 of the unit vector $\hat{\mathsf{n}} \equiv (n_1, n_2, n_3)$ to do so. The time delays in units of f_s^{-1} are,

$$f_s \tau_{(I)}(n_3, n_1) = \left[r_{(I)1} n_1 + r_{(I)2}(1 - n_3^2 - n_1^2)^{1/2} + r_{(I)3} n_3 \right] \quad , \tag{3.81}$$

where $\mathsf{r}_{(I)}$ is the position vector of the I-th detector's hub and is henceforth measured in units of the "fiducial wavelength", $\lambda_s \equiv c/f_s$. Since we choose to measure the time delays with respect to the fide, we must have $\mathsf{r}_{(f)} = 0$. Thus, we may write the correlation phase as

$$\Phi_{(I)}(f; \vartheta^\mu, \Delta\vartheta^\mu) \equiv 2\pi \varphi_{(I)\alpha}(f; \vartheta^\mu) \Delta\vartheta^\alpha \quad , \tag{3.82}$$

where $\vartheta^\alpha \equiv \{f_s t_c,\, f_s \xi_1,\, f_s \xi_2,\, n_3,\, n_1\}$ is a quintet of dimensionless parameters and

$$\varphi_{(I)1}(f; \vartheta^1) = \left(\frac{f}{f_s} \right) \quad , \tag{3.83}$$

$$\varphi_{(I)2}(f; \vartheta^2) = \frac{3}{5} \left(\frac{f}{f_s} \right)^{-5/3} \quad , \tag{3.84}$$

$$\varphi_{(I)3}(f; \vartheta^3) = \left(\frac{f}{f_s} \right)^{-1} \quad , \tag{3.85}$$

$$\varphi_{(I)4}(f; \vartheta^4) = \left[r_{(I)3} - r_{(I)2} \frac{n_3}{(1 - n_3^2 - n_1^2)^{1/2}} \right] \frac{f}{f_s} \quad , \tag{3.86}$$

$$\varphi_{(I)5}(f; \vartheta^5) = \left[r_{(I)1} - r_{(I)2} \frac{n_1}{(1 - n_3^2 - n_1^2)^{1/2}} \right] \frac{f}{f_s} \quad . \tag{3.87}$$

To obtain $g_{\alpha\beta}$ we Taylor expand the network statistic about the peak at $\Delta\vartheta^{\mu} = 0$ to obtain

$$Ł(\boldsymbol{\vartheta}, \boldsymbol{\Delta\vartheta}) \approx 1 + \frac{1}{2}\left(\frac{\partial^2 Ł}{\partial\Delta\vartheta^{\alpha}\partial\Delta\vartheta^{\beta}}\right)\Bigg|_{\Delta\boldsymbol{\vartheta}=0} \Delta\vartheta^{\alpha}\Delta\vartheta^{\beta} \quad ,$$

where $\boldsymbol{\vartheta}$ is the five-dimensional signal parameter-vector. The metric is then defined as [1]

$$\begin{aligned} g_{\alpha\beta}(\vartheta) &\equiv -\frac{1}{2}\left(\frac{\partial^2 Ł}{\partial\Delta\vartheta^{\alpha}\partial\Delta\vartheta^{\beta}}\right)\Bigg|_{\Delta\boldsymbol{\vartheta}=0} && (3.88) \\ &\approx \frac{1}{4}{p_I}^J \Re[Q^I Q^*_J] g_{(I)(J)\alpha\beta} \quad , && (3.89) \end{aligned}$$

where we used the fact that both p^J_I and $g_{(I)(J)\alpha\beta}$ are symmetric under the interchange of I and J. Also, $g_{(I)(J)\alpha\beta} = -\big(\partial^2\Theta_{(I)(J)}/\partial\Delta\vartheta^{\alpha}\partial\Delta\vartheta^{\beta}\big)|_{\Delta\vartheta^{\gamma}=0}$. The reality of $g_{\alpha\beta}$ is now manifest. Owing to the linearity of Φ in $\Delta\vartheta^{\alpha}$, the metric $g_{\alpha\beta}$ depends only on its first derivatives. Therefore, it can be easily shown that

$$g_{(I)(J)\alpha\beta} = \langle\Phi_{\alpha}\Phi_{\beta}\rangle_{(I)} + \langle\Phi_{\alpha}\Phi_{\beta}\rangle_{(J)} - \langle\Phi_{\alpha}\rangle_{(I)}\langle\Phi_{\beta}\rangle_{(J)} - \langle\Phi_{\beta}\rangle_{(I)}\langle\Phi_{\alpha}\rangle_{(J)} \quad , \quad (3.90)$$

where the suffix α denotes the derivative with respect to $\Delta\vartheta^{\alpha}$. The angular bracket denotes the average over a given frequency range. For the frequency

range of $[f_s, \infty]$, the average value of the function $X_{(I)}(f)$ is denoted as,

$$\langle X \rangle_{(I)} = \int_{f_{s(I)}}^{\infty} \mathcal{A}_{(I)}(f) X_{(I)}(f) df \quad , \tag{3.91}$$

where within the angular brackets we have dropped the subscript on X simply because the same subscript appears outside those brackets. In other words, we have reduced redundancy by introducing the notation: $\langle X \rangle_{(I)} = \langle X_{(I)} \rangle_{(I)}$. We observe that (3.90) is a generalization of Owen's formula in Ref. [7], wherein the metric for the single-detector case was derived. It is not difficult to understand the origin of the different factors in the expression for $g_{\alpha\beta}$. This metric gets contribution from every pair of detectors in a network, including the diagonal terms (i.e., terms with $I = J$), through the "coupling" metric $g_{(I)(J)\alpha\beta}$. The magnitude of each of these contributions is determined by their respective coupling strengths in the form of coefficients, ${p_I}^J \Re[\big(Q^I Q^*_J\big)]$, which depend on the four angles $\{\epsilon, \psi, \theta, \phi\}$. This is because these coefficients essentially arise from the extended beam-pattern functions of the detectors, which, apart from depending on the signal amplitude through ϵ, determine how sensitive a given detector is to a source direction and ψ. The above expressions allow one to calculate the parameter space metric for any Earth-based network. However, since the metric is non-flat (as opposed to a flat metric for a single-detector "network"), the template spacings $\Delta\boldsymbol{\vartheta}$ will depend on the location, $\boldsymbol{\vartheta}$, of the template. The general expressions for the

moment functionals are:

$$\langle \Phi_\alpha \rangle_{(I)} = 2\pi \varphi_{(I)s\alpha} j_{(I)}(7 - 3m_\alpha) \quad , \tag{3.92}$$

where $\varphi_{(I)s\alpha} \equiv \varphi_{(I)\alpha}(f_{(I)s}; \vartheta^\alpha)$ and m_α is the power of f on which $\varphi_{(I)\alpha}$ depends and $j_{(I)}(q)$ is q-th noise moment of the noise-curve corresponding to the I-th detector and is defined in Appendix A. Similarly,

$$\langle \Phi_\alpha \Phi_\beta \rangle_{(I)} = 4\pi^2 \varphi_{(I)s\alpha} \varphi_{(I)s\beta} j_{(I)}(7 - 3(m_\alpha + m_\beta)) \quad . \tag{3.93}$$

3.9 Examples of Detector Networks: Results

Before considering the case of the actual network of laser-interferometric detectors being built around the globe, we shall first consider some idealized cases that are simple to analyze. Such an exercise is meant to provide us with some useful estimates on the number of templates required. We begin by verifying that in the case of a single detector the formalism developed in the previous sections yields the results expected from earlier studies [15, 7]. Then we apply our formalism to cases of networks with two and three identical detectors, respectively, with identical noise power spectral densities(PSDs). Assuming a common noise simplifies computation of the metric on the signal parameter space. Purely for the purposes of obtaining estimates we choose the noise PSDs in these cases to be that of LIGO I , with $f_s = 40$ Hz. For a description of the noise curves see [16]. Here, cases of non-coincident as well

as arbitrarily oriented detectors are also studied. Subsequently, we generalize these analyses to obtain estimates for realistic cases of networks comprising of the LIGO and VIRGO detectors: These include two-detector networks, which pair up the two LIGOs or VIRGO with one of the LIGOs, and the three-detector network that includes VIRGO and both the LIGOs. In Table 3.1 at the end of the chapter, we summarize our results for the required numbers of templates.

3.9.1 The One-Detector 'Network'

It is instructive to start with the one detector case since it lays the foundation for the M-detector case, the analysis of which is the final goal. We first verify that our solution gives the expected estimates for $M = 1$. For a single detector, we have $\vartheta^0 = f_s t_c$, $\vartheta^1 = f_s \xi_1$ and $\vartheta^2 = f_s \xi_2$. The network statistic is

$$\begin{aligned} \text{Ł} &= |C_1^* Q^1| = |C_1| \\ &= \left| \int_{f_s}^{\infty} \mathcal{A}(f) \exp\left(-i\Phi(f; \vartheta^\mu, \Delta\vartheta^\mu)\right) df \right| \quad , \end{aligned} \tag{3.94}$$

where the phase Φ can be derived from Eqs. (3.80) and (3.36) by setting $I = 1$ and the time-delay term to zero in those equations, respectively.

For the above statistic, the *exact* metric $g_{\alpha\beta}$ can be obtained from Eq.

(3.88). Using the scaling $\Phi = 2\pi\tilde{\Phi}$, the scaled metric is

$$\tilde{g}_{\alpha\beta} = \frac{1}{2}[\langle\tilde{\Phi}_\alpha\tilde{\Phi}_\beta\rangle - \langle\tilde{\Phi}_\alpha\rangle\langle\tilde{\Phi}_\beta\rangle] \quad , \tag{3.95}$$

where the moment functionals can be expressed in terms of the noise moments listed in Appendix A. The metric then reduces to

$$\tilde{g}_{\alpha\beta} = \frac{1}{2}\begin{pmatrix} k_1 & k_3 & k_4 \\ k_3 & k_2 & k_5 \\ k_4 & k_5 & k_6 \end{pmatrix} \quad , \tag{3.96}$$

where k_1, k_2, k_3, k_4, k_5, and k_6 are certain useful combinations of noise moments and are defined in Appendix A (here we have dropped the detector index I from those combinations for obvious reasons). Projecting orthogonal to ϑ^0, we find

$$\tilde{\gamma}_{ij} = \frac{1}{2}\begin{pmatrix} k_2 - k_3^2/k_1 & k_5 - k_3k_4/k_1 \\ k_5 - k_3k_4/k_1 & k_6 - k_4^2/k_1 \end{pmatrix} \quad , \tag{3.97}$$

The parameter-space in this case is two-dimensional. The volume of this two-dimensional space is given by:

$$\begin{aligned} \mathcal{V} = 4\pi^2\tilde{\mathcal{V}} &= 4\pi^2 f_s^2 \int \sqrt{\det \parallel \tilde{\gamma}_{ij} \parallel}\, d\xi_1 d\xi_2 \\ &= 4\pi^2 f_s^2 \sqrt{\det \parallel \tilde{\gamma}_{ij} \parallel} \int d\xi_1 d\xi_2 \, . \end{aligned} \tag{3.98}$$

Since the number density of templates here is $\rho_2(\mu) = 1/(2\mu)$, the number of templates (including 2 sets of templates for searching over δ_c) is just

$$n_{tot} = 4\pi^2 (f_s^2/\mu) k \int d\xi_1 d\xi_2 \,. \tag{3.99}$$

where

$$k \equiv \frac{1}{2k_1} \sqrt{k_1^2 k_2 k_6 - k_1 k_2 k_4^2 - k_1 k_3^2 k_6 - k_1^2 k_5^2 + 2 k_1 k_3 k_4 k_5} \,. \tag{3.100}$$

The value of n_{tot} for this case I is given in Table 3.1.

3.9.2 Two-detector networks

(a) Two non-coincident, identical detectors with identical noise PSDs and identical orientations

We consider a network of two identical detectors with identical noise PSDs. We make the following choice of coordinates. We choose one of the two detectors to be the fide. The z axis of the fide is chosen along the line joining the two detectors. Then the second detector is taken to be located at $(0, 0, z_2)$, with an orientation identical to that of the fide, i.e., $\alpha_{(2)} = \beta_{(2)} = \gamma_{(2)} = 0$. Owing to the same orientations, the beam-pattern functions of the two detectors are identical. If the detectors were located at the same place, then the resulting network would have mimicked a single detector, but with a higher sensitivity. Here, however, we consider spatially separated detectors,

where the relative time delay, $\tau_{(2)}$, provides partial information about the source-direction, namely, θ. The time delay in units of f_s^{-1} is given by

$$f_s \tau_{(2)} = z_2(\hat{\mathsf{z}} \cdot \hat{\mathsf{n}}) \quad , \tag{3.101}$$

where we measure z_2 in units of the fiducial wavelength, λ_s (which is ≈ 7500 km for $f_s = 40$ Hz). Thus, the network statistic in this case is

$$\begin{aligned} Ł &= |\mathsf{C} \cdot \mathsf{Q}'| & (3.102) \\ &= \frac{1}{\sqrt{2}} |C_1^* + C_2^*| \, . & (3.103) \end{aligned}$$

Note that $Q'^1 = Q'^2$ and $|Q'^1| = |Q'^2| = 1/\sqrt{2}$. This means that we have no information about ϵ and ψ. For a two-detector network, any given value of the time delay corresponds to more than one source direction, all of which lie on the surface of a cone whose axis coincides with the line joining the two detectors. Only when the source lies on the line passing through the two detectors is the time delay single-valued, and is of maximum magnitude for a given pair of detectors (note that we have allowed $\tau_{(I)}$ to be negative as well). The value of the time delay $\tau_{(2)}$ determines the opening angle of the cone. Thus, the azimuthal direction angle ϕ of the wave remains undetermined in the case of two detectors. Only θ can be estimated from the time delay that appears in the phase difference of the detector responses.

As in the single detector case, the exact metric $g_{\alpha\beta}$ can be obtained di-

rectly from Eq. (3.88). The corresponding scaled metric is

$$\tilde{g}_{\alpha\beta} = \frac{1}{4}\sum_{I,J}\frac{1}{2}\big[\langle\tilde{\Phi}_\alpha\tilde{\Phi}_\beta\rangle_{(I)} - \langle\tilde{\Phi}_\alpha\rangle_{(I)}\langle\tilde{\Phi}_\beta\rangle_{(J)}\big] \quad , \tag{3.104}$$

where the $\Phi_{(I)}$ are defined in Eq. (3.80). The metric on the four-dimensional parameter space $\{f_s t_c, f_s\xi_1, f_s\xi_2, n_3\}$ can now be given in terms of the noise moments as

$$\tilde{g}_{\alpha\beta} = \frac{1}{2}\begin{pmatrix} k_1 & k_3 & k_4 & z_2k_1/2 \\ k_3 & k_2 & k_5 & z_2k_3/2 \\ k_4 & k_5 & k_6 & z_2k_4/2 \\ z_2k_1/2 & z_2k_3/2 & z_2k_4/2 & z_2^2(k_1+j(1))/4 \end{pmatrix} \quad , \tag{3.105}$$

where $j(1)$ is a noise moment defined in Eq. (A.3).[4] After maximization $\tilde{g}_{\alpha\beta}$ over the time of coalescence, we deduce the metric $\tilde{\gamma}_{ij}$ to be

$$\tilde{\gamma}_{ij} = \frac{1}{2}\begin{pmatrix} k_2 - k_3^2/k_1 & k_5 - k_3k_4/k_1 & 0 \\ k_5 - k_3k_4/k_1 & k_6 - k_4^2/k_1 & 0 \\ 0 & 0 & z_2^2 j(1)/4 \end{pmatrix} . \tag{3.106}$$

Here, the vanishing γ_{13} and γ_{23} imply that there is no covariance between ξ_1 and θ as well as between ξ_2 and θ . This is however not true when the noise curves are assumed to be different for the two detectors, as can be seen below.

[4]We have dropped the detector index I from the noise moment combinations k_1, k_2, and k_3 (see Appendix A) since here the noise moments are identical for the two detectors.

The volume on the 3-D parameter space $\{f_s\xi_1, f_s\xi_2, n_3\}$ is obtained by integrating $\sqrt{\det \parallel \tilde{\gamma} \parallel}$ over n_3 , ξ_1 , and ξ_2. The proper volume of interest is

$$\mathcal{V} = (2\pi)^3\tilde{\mathcal{V}} = 4\pi^3 f_s^2 k z_2 \sqrt{2j(1)} \int d\xi_1 d\xi_2 \quad , \tag{3.107}$$

For the two-detector network, the number density of templates is $\rho_3(\mu) = (1/8)(3/\mu)^{3/2}$. Therefore, the number of templates is

$$n_{tot} = 3\pi^3 z_2 f_s^2 k \sqrt{6j(1)/\mu^3} \int d\xi_1 d\xi_2 \, . \tag{3.108}$$

The value of n_{tot} for this case is given in Table 3.1.

(b) Two non-coincident, identical detectors with identical noise PSDs, but with different orientations

We make the choice of coordinates identical to that in the previous subsection (a). Since the two detectors have different orientations, the beam-pattern functions for the two detectors differ, i.e., $Q^1 \neq Q^2$. This has the implication that more information about the signal parameters, namely, ϵ and ψ, can be obtained. Since here we have only two dimensions on the network space to contend with, the network correlation-vector, C, always lies in $\mathcal{H}$ and no projection is required. Therefore, the problem of maximization of the LLR over the angles $\{\epsilon, \psi\}$ reduces to aligning Q' along C. Thus, the

network statistic simplifies to

$$Ł = \| \mathsf{C} \| = \left(|C^1|^2 + |C^2|^2 \right)^{1/2} . \tag{3.109}$$

The metric $\tilde{g}_{\alpha\beta}$ on the parameter space with coordinates $\{f_s t_c, f_s \xi_1, f_s \xi_2, n_3\}$ is obtained from Eq. (3.88) to be exactly

$$\tilde{g}_{\alpha\beta} = \frac{1}{2} \sum_I |Q^I|^2 \left[\langle \tilde{\Phi}_\alpha \tilde{\Phi}_\beta \rangle_{(I)} - \langle \tilde{\Phi}_\alpha \rangle_{(I)} \langle \tilde{\Phi}_\beta \rangle_{(I)} \right] \ , \tag{3.110}$$

where the $\tilde{\Phi}_{(I)}$ are the same as in case (a) above. Setting $|Q^1|^2 \equiv \eta$ and, therefore, $|Q^2|^2 = 1 - \eta$, the metric in terms of η and the noise moments is

$$\tilde{g}_{\alpha\beta} = \frac{1}{2} \begin{pmatrix} k_1 & k_3 & k_4 & z_2(1-\eta)k_1 \\ k_3 & k_2 & k_5 & z_2(1-\eta)k_3 \\ k_4 & k_5 & k_6 & z_2(1-\eta)k_4 \\ z_2(1-\eta)k_1 & z_2(1-\eta)k_3 & z_2(1-\eta)k_4 & z_2^2(1-\eta)k_1 \end{pmatrix} . \tag{3.111}$$

The associated $\tilde{\gamma}_{ij}$ is then given by

$$\tilde{\gamma}_{ij} = \frac{1}{2} \begin{pmatrix} k_2 - k_3^2/k_1 & k_5 - k_3 k_4/k_1 & 0 \\ k_5 - k_3 k_4/k_1 & k_6 - k_4^2/k_1 & 0 \\ 0 & 0 & z_2^2 \eta(1-\eta) k_1 \end{pmatrix} . \tag{3.112}$$

We note that the $\sqrt{\det \| \tilde{\gamma}_{\rm ij} \|}$ depends on η, which is a function of source-direction (θ, ϕ) as well as the angles $\{\epsilon, \psi\}$. Thus, for every single source-

direction we have a two-parameter family of metrics $\tilde{\gamma}_{ij}$ (dependent on ϵ and ψ).

Clearly, the template spacings in this case will vary with their locations. For the purpose of obtaining estimates, we perform a simplification by opting to choose a bank of templates that is the finest over these two parameters. To do so, we first maximize det $\parallel \tilde{\gamma}_{\rm ij} \parallel$ over ϵ and ψ and then compute the volume. The parameters $\{\epsilon, \psi\}$ appear in the determinant only through the factor $\sqrt{\eta(1-\eta)}$. The value of η for which the determinant is maximized is $\eta = 1/2$. It has been proven in [1] that one can always find a physically allowed pair of $\{\epsilon, \psi\}$ such that the value $\eta = 1/2$ is attainable for any given pair of $\{\theta, \phi\}$ and for any orientations of the detectors. We use this value of η to compute the parameter-space volume.

The proper volume after multiplying by the appropriate scaling factor is

$$\mathcal{V} = 4\pi^3 k f_s^2 z_2 \sqrt{2k_1} \int d\xi_1 d\xi_2 \quad . \tag{3.113}$$

As in case (a), the number of templates is arrived at by multiplying the proper volume by the number density of templates (with an extra factor of 2 for the search over t_c). The result is:

$$n_{tot} = 3\sqrt{6k_1/\mu^3}\pi^3 z_2 f_s^2 k \int d\xi_1 d\xi_2 \quad . \tag{3.114}$$

(c) General case of two detectors

Here, we typically consider the case of a network comprising of the VIRGO detector and one of the LIGO detectors, say, the one at Louisiana, for concreteness. (The results do not differ much if we replace in our calculations the numbers corresponding to the detector at Louisiana with those describing the 4 km long Hanford detector.) Here, we have a case in which the seismic cut-offs are different. Labeling the VIRGO detector as 1 and LIGO as 2, we have $f_{s(1)} = 16$ Hz and $f_{s(2)} = 40$ Hz. The important implication of this is that the signal in the VIRGO detector will last longer by a factor $(f_{s(1)}/f_{s(2)})^{-8/3} \sim 11.5$. Thus, until the chirp reaches the frequency of 40 Hz, essentially it is only one detector, namely, VIRGO that contributes to the SNR. We now compute the metric and the number of templates for this network.

The expression for the scaled metric in this case is the same as the one in Eq. (3.110). Let z_2 denote the distance between the detectors as in case (a) and let $|Q^1|^2 = \eta$ as in (b). Also, we take the fiducial frequency to be $f_s = f_{s(2)}$, without any loss of generality. The metric components are then given by

$$\tilde{g}_{11} = \frac{1}{2}\left[k_{1(2)} + \eta(\varrho^2 k_{1(1)} - k_{1(2)})\right] \quad , \tag{3.115}$$

$$\tilde{g}_{12} = \tilde{g}_{21} = \frac{1}{2}\left[k_{3(2)} + \eta(\varrho^{-2/3} k_{3(1)} - k_{3(2)})\right] \quad , \tag{3.116}$$

$$\tilde{g}_{13} = \tilde{g}_{31} = \frac{1}{2}\left[k_{4(2)} + \eta(k_{4(1)} - k_{4(2)})\right] \quad , \tag{3.117}$$

$$\tilde{g}_{14} = \tilde{g}_{41} = \frac{1}{2}(1-\eta) z_2 k_{1(2)} \quad , \tag{3.118}$$

$$\tilde{g}_{22} = \frac{1}{2}\left[k_{2(2)} + \eta(\varrho^{-10/3} k_{2(1)} - k_{2(2)})\right] \quad , \tag{3.119}$$

$$\tilde{g}_{23} = \tilde{g}_{32} = \frac{1}{2}\left[k_{5(2)} + \eta(\varrho^{-8/3} k_{5(1)} - k_{5(2)})\right] \quad , \tag{3.120}$$

$$\tilde{g}_{24} = \tilde{g}_{42} = \frac{1}{2}(1-\eta) z_2 k_{3(2)} \quad , \tag{3.121}$$

$$\tilde{g}_{33} = \frac{1}{2}\left[k_{6(2)} + \eta(\varrho^{-2} k_{6(1)} - k_{6(2)})\right] \quad , \tag{3.122}$$

$$\tilde{g}_{34} = \tilde{g}_{43} = \frac{1}{2}(1-\eta) z_2 k_{4(2)} \quad , \tag{3.123}$$

$$\tilde{g}_{44} = \frac{1}{2}(1-\eta) z_2^2 k_{1(2)} \quad , \tag{3.124}$$

where we have used $\varrho_{(1)} = f_{s(1)}/f_{s(2)} \equiv \varrho$ and $\varrho_{(2)} = 1$. For the network under consideration $\varrho = 0.4$, and $k_{1(I)}$, $k_{2(I)}$, $k_{3(I)}$, $k_{4(I)}$, $k_{5(I)}$, and $k_{6(I)}$ for VIRGO and LIGO-I are listed in Appendix A . From Eq. (3.88) we compute $\tilde{\gamma}_{ij}$. We find that $\sqrt{\det \parallel \tilde{\gamma}_{ij} \parallel} = z_2 B(\eta)$, where $B(\eta)$ is a smoothly varying function of η that attains its maximum value of ~ 0.011 at $\eta \sim 0.69$.

The volume of the parameter space is

$$\mathcal{V} \simeq 0.176 \pi^3 z_2 f_s^2 \int d\xi_1 d\xi_2 \quad , \tag{3.125}$$

and the corresponding number of templates is

$$n_{tot} \simeq \frac{0.23\pi^3}{\mu^{3/2}} z_2 f_s^2 \int d\xi_1 d\xi_2 \quad , \tag{3.126}$$

where n_{tot} is listed in Table 3.1.

3.9.3 Three-detector networks

(a) Three non-coincident identical detectors with identical noise PSDs and identical orientations

We consider a network of three detectors with identical noise PSDs. The detectors are spatially separated and have identical orientations. Such a situation will be difficult, if not impossible to realize on a spherical Earth. However, in this simple case our goal is to obtain order of magnitude estimates for the number of templates required. We treat one of the three detectors to be the fide. We choose the coordinate system of the fide as follows: The z-axis of the fide is along the line joining the fide and one of the remaining detectors. Thus, the second detector is located at $(0, 0, z_2)$. The x-axis is chosen such that the plane formed by the network coincides with the $x - z$ plane. The spatial coordinates of the third detector are $(x_3, 0, z_3)$. The detectors have identical orientations and therefore identical antenna-pattern functions, i.e., $Q'^1 = Q'^2 = Q'^3$. Then, the network statistic simplifies to

$$\begin{aligned} \text{Ł} &= |\mathsf{C} \cdot \mathsf{Q}'| \\ &= \frac{1}{\sqrt{3}} |C_1^* + C_2^* + C_3^*| \,. \end{aligned} \tag{3.127}$$

Note that, the network of three spatially separated detectors provide two independent relative time delays, $\tau_{(2)}$ and $\tau_{(3)}$, which determine the two possible

source directions as follows. For each pair of detectors in such a network, the time delays draw a circle in the sky for possible source locations. The intersections of two such circles determine two possible source directions. Here, the time delays in units of f_s^{-1} are $f_s\tau_{(2)} = z_2(\hat{\mathsf{z}}\cdot\hat{\mathsf{n}})$ and $f_s\tau_{(3)} = x_3(\hat{\mathsf{x}}\cdot\hat{\mathsf{n}}) + z_3(\hat{\mathsf{z}}\cdot\hat{\mathsf{n}})$. We measure z_2, z_3, and x_3 in units of fiducial wavelength λ_s.

The exact scaled metric is obtained via Eq. (3.88) to be

$$\tilde{g}_{\alpha\beta} = \frac{1}{9}\sum_{I,J}\frac{1}{2}\big[\langle\tilde{\Phi}_\alpha\tilde{\Phi}_\beta\rangle_{(I)} - \langle\tilde{\Phi}_\alpha\rangle_{(I)}\langle\tilde{\Phi}_\beta\rangle_{(J)}\big] \quad , \tag{3.128}$$

where the $\tilde{\Phi}_{(I)}$ can be obtained from Eq. (3.82) by using for the time delays, $f_s\Delta\tau_{(1)} = 0$, $f_s\Delta\tau_{(2)} = z_2\Delta n_3$, and $f_s\Delta\tau_{(3)} = x_3\Delta n_1 + z_3\Delta n_3$. The resulting components of the symmetric metric $\tilde{g}_{\alpha\beta}$, on the space of the variables $\{f_s t_c, f_s\xi_1, f_s\xi_2, n_3, n_1\}$, are computed. Not surprisingly, $\tilde{g}_{11}$, $\tilde{g}_{12}$, $\tilde{g}_{13}$, $\tilde{g}_{22}$, $\tilde{g}_{23}$, $\tilde{g}_{33}$, and their symmetric counterparts are identical to the components of the one detector metric. The new components are:

$$\tilde{g}_{14} = \tilde{g}_{41} = \frac{1}{6}(z_2 + z_3)k_1 \quad , \tag{3.129}$$

$$\tilde{g}_{15} = \tilde{g}_{51} = \frac{1}{6}x_3k_1 \quad , \tag{3.130}$$

$$\tilde{g}_{24} = \tilde{g}_{42} = \frac{1}{6}(z_2 + z_3)k_3 \quad , \tag{3.131}$$

$$\tilde{g}_{25} = \tilde{g}_{52} = \frac{1}{6}x_3k_3 \quad , \tag{3.132}$$

$$\tilde{g}_{34} = \tilde{g}_{43} = \frac{1}{6}(z_2 + z_3)k_4 \quad , \tag{3.133}$$

$$\tilde{g}_{35} = \tilde{g}_{53} = \frac{1}{6} x_3 k_4 \quad , \tag{3.134}$$

$$\tilde{g}_{44} = \frac{1}{18}[3(z_2^2 + z_3^2) j(1) - (z_2 + z_3)^2 j(4)^2] \quad , \tag{3.135}$$

$$\tilde{g}_{45} = \tilde{g}_{54} = \frac{1}{18} x_3 [3 z_3 j(1) - (z_2 + z_3) j(4)^2] \quad , \tag{3.136}$$

$$\tilde{g}_{55} = \frac{1}{18} x_3^2 [3 j(1) - j(4)^2] \quad . \tag{3.137}$$

Maximization over $f_s t_c$ gives

$$\tilde{\gamma}_{ij} = \frac{1}{2} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \tag{3.138}$$

where

$$A_{11} = \begin{pmatrix} k_2 - k_3^2/k_1 & k_5 - k_3 k_4/k_1 \\ k_5 - k_3 k_4/k_1 & k_6 - k_4^2/k_1 \end{pmatrix} \tag{3.139}$$

$$A_{12} = A_{21} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{3.140}$$

and A_{22} has components:

$$A_{22}^{11} = \frac{1}{9}[3(z_2^2 + z_3^2) j(1) - (z_2 + z_3)^2 (k_1 + j(4)^2)] \quad , \tag{3.141}$$

$$A_{22}^{12} = A_{22}^{21} = \frac{1}{9} x_3 [3 z_3 j(1) - (z_2 + z_3)(k_1 + j(4)^2)] \quad , \tag{3.142}$$

and

$$A_{22}^{22} = \frac{1}{9} x_3^2 [3j(1) - j(4)^2 - k_1/2] \quad . \tag{3.143}$$

The proper volume is then

$$\begin{aligned} \mathcal{V} &= (2\pi)^4 \tilde{\mathcal{V}} \\ &= \frac{64\pi^4}{9} j(1) k \mathcal{A} f_s^2 [3 + k_1 (z_2^2 + z_3^2 - z_2 z_3)/j(1) z_2^2]^{1/2} \\ &\qquad \times \int d\xi_1 d\xi_2 \quad , \end{aligned} \tag{3.144}$$

where $\mathcal{A} = \frac{1}{2} x_3 z_2$ is the area of the network. For the three-detector network discussed here, the parameter space is four-dimensional and the number density of templates is $\rho_4(\mu) = \frac{1}{\mu^2}$.This gives the total number of templates as:

$$\begin{aligned} n_{tot} &= 2(8 f_s \pi^2 / 3\mu)^2 k \mathcal{A} j(1) [3 + k_1 (z_2^2 + z_3^2 - z_2 z_3)/j(1) z_2^2]^{1/2} \\ &\qquad \times \int d\xi_1 d\xi_2 \, . \end{aligned} \tag{3.145}$$

The number of templates is listed in Table 3.1.

(b) The network comprising of LIGO (Livingston), LIGO (Hanford), and VIRGO (Pisa) with their respective noise curves.

We finally discuss the case of a non-coincident three-detector network involving the LIGO and VIRGO detectors with their respective noise curves. We denote such a network as LHV. The detector noise PSDs are represented

by analytical fits given in Ref. [16]. The locations and orientations of the detectors on the globe can be found in [1]. In order to compute the metric we choose the fide frame to be the network frame with L at its origin, H lying on the z-axis, and V lying in the $x-z$ plane. In units of λ_s, the dimensionless position-vectors of the detectors are given by

$$\mathbf{r}_L = (0,0,0) \ , \qquad \mathbf{r}_H = (0,0,0.40) \ , \qquad \mathbf{r}_V = (1.05,0,0.29) \, . \quad (3.146)$$

Note that since the y-component of each of these vectors is zero in such a frame (i.e., $r_{(I)2} = 0$, for I =1, 2, 3), the moment functionals given in Eq. (A.9) simplify considerably. An inspection of that equation shows that in this case the noise moments do not depend on the direction to the source and, hence, the $g_{(I)(J)\alpha\beta}$ are constants. Therefore, this choice of the fide frame simplifies the computations involved. The metric $g_{\alpha\beta}$ however, depends on n_1 and n_3 through Q. The orientations of the detectors can easily be obtained [1]. In the fide (network) frame, they are given by

$$\begin{aligned} \{\alpha_L, \beta_L, \gamma_L\} &= \{38.11^\circ, 256.35^\circ, 107.43^\circ\} \ , \\ \{\alpha_H, \beta_H, \gamma_H\} &= \{38.09^\circ, 283.54^\circ, 196.88^\circ\} \ , \\ \{\alpha_V, \beta_V, \gamma_V\} &= \{320.34^\circ, 275.92^\circ, 159.02^\circ\} \, . \end{aligned} \quad (3.147)$$

The metric components for this case are extremely lengthy and complex. A Mathematica notebook exists that computes these components and forms

the projected metric γ_{ij}. The number of templates are then calculated and are given in the following table along with the numbers for all the other cases (sequential by sub-section).

Table 3.1: n_{tot} Results

Network Case	1PN n_{total}	2PN n_{total}
Case I	2.45×10^5	4.97×10^5
Case II	1.55×10^7	3.14×10^7
Case III	7.58×10^6	1.53×10^7
Case IV	1.79×10^8	3.46×10^8
Case V	2.41×10^9	4.74×10^9

These were computed for LIGO I noise and with a mass range from 0.5 to 3 solar masses. Here Case I is the one detector case, Case II is the first two detector case discussed and so forth in the order in which the different networks were presented above. It is important to note that the 2PN phase was parameterized as in [16], and depended therefore on two coordinates different than ξ_1 and ξ_2. In those coordinates the metric components do not vary greatly and so it is a convenient space in which to calculate the numbers of templates. Even so however, the determinant of the metric does not factor out of the integral as it did for all but the last of the 1PN cases. The metric components at 2PN depend on the parameters or position in parameter space. This fact, combined with the fact that the parameter space has a complicated shape led to the writing of a Mathematica notebook to numerically integrate to find the proper volumes. As can be seen, there is about a factor of two

difference in the number of templates required between the 1PN and 2PN cases. This is consistent with the one detector results in [16].

Chapter 4

Search Algorithm and Computer Codes

Considerable effort over the last several years has gone into the development of a package of computer codes to analyze data. There are essentially two main components to this package: the LIGO Algorithm Library (LAL), and LALapps. Both the LAL and LALapps are divided into different sections for each of the very different types of searches that are done, e.g. searches for inspiral signals, pulsars, stochastic gravity-wave background, and burst type searches. Since this thesis primarily deals with inspiral signals, Here, focus will be on the inspiral codes within both LAL and LALapps. LAL contains the codes that define functions whereas LALapps contains the codes that use those functions to perform searches. The majority of these codes are written in standard C, some however are written in Python and even C++.

Those codes for which this author is a primary author have been included in appendices which contain the codes themselves. Here, without explaining all the small details, a description of how the codes operate, what they need, and what they return will be given. Only those codes which form the core of the coherent inspiral analysis will be discussed.

4.1 LAL

There are three main components to this library: header files, functions, and test codes. The header files contain all structure and function prototypes. LAL contains many structures and types which make time and frequency analysis more organized and simple. Many of these are tailored toward working with time series, frequency series, power spectra, and many other objects frequently encountered in the field of signal processing. Many functions also exist that take as arguments these structures and types, and perform calculations upon them. This philosophy of modularity makes it very easy to test codes and eliminate errors. The code for these functions is contained in a collection of C-files that use the associated header files containing the prototypes. A series of test codes are also written that test the behavior of each function at the time in which the codes are compiled.

In Appendix C is a printout of the header file CoherentInspiral.h which

contains all the structures and function prototypes used in the coherent inspiral analysis. A current version of the documentation for the whole packages is maintained at:
http://www.lsc-group.phys.uwm.edu/daswg/projects/lal.html. Explanation for each structure and function will not be given, instead the search code that uses them will be discussed in some detail. In Appendix D is a printout of another LAL code, CoherentInspiralInput.c, which contains a function used to output the complex filter data that the coherent statistic depends upon. Appendix E contains the LAL code CoherentInspiralFilter.c which contains all the functions necessary to compute the coherent statistic and estimate some parameters. It should also be noted that there were additional modifications necessary to many other files in this library to facilitate this search.

4.2 LALapps

LALapps contains all the search related codes. The inspiral package within lalapps contains codes for creating template banks, injecting signals into data, analyzing events, as well as the codes that do the actual search. The inputs that the coherent search code needs are obtained by running various other codes. The coherent statistic depends on a combination of the real and imaginary parts of the complex filter data from multiple interferometers. When the standard search for inspiral signals is done by the running of the

inspiral code, knowledge of the phase is lost since the signal to noise ratio for a single interferometer is the modulus of the complex filter data. Rather than write an entire new code to do the filtering at the single interferometer level to output the complex data with phase information intact, the standard inspiral search code was modified to do the task. An extra parse option was added that when enabled will cause the code to output a complex time series of filter data. Documentation that explains how these codes work in great detail is available at:
http://www.lsc-group.phys.uwm.edu/daswg/projects/lalapps.html. Some of the basics of the core codes will be discussed below.

To perform any search, one must have data. To obtain real detector data from any of the interferometers in the LSC, one must be a member of the LSC. It is not however necessary to have real data to perform a search. The codes have been written in such a way as to make it very easy to test them on synthesized data. Much of our signal processing strategy is based on having Gaussian stationary noise so it worthwhile to test our codes for this optimal noise situation. Some of the tests discussed in a subsequent chapter on hierarchical search methods were done with synthesized Gaussian data using some of the standard search codes. Typically however, the codes are run on real data which is contained at many different cluster computing sites all over the world. The raw data is sampled at 16384 Hz, but for the inspiral analysis we typically downsample to 4096 Hz.

Next, a template bank must be built that covers the parameter space wherein the signals we are interested in lie. A code exists that will build such a template bank for a given chunk of data based on the specification of some minimal correlation between a template and the farthest possible signal. The number of templates one can obtain by running this code depends on many factors. If the data is very noisy, the number of templates required will be relatively high in order to meet the minimal correlation condition. Also, as the specified parameter ranges get larger, more templates will be required to cover the space. After running the code, an xml file is output which essentially contains a list of parameters for each template in the bank. This file is now used when the inspiral search code is run. From the list of parameters, each template or signal is calculated by the inspiral code just prior to filtering.

Now one has the option of injecting fake signals into the data, possibly for testing purposes. There are two ways in which this can be done: hardware injections and software injections. Hardware injections can occur only when the data is being initially taken by the interferometers. These injections involve actually moving the mirrors of the interferometer so as to mimic an impinging gravitational wave. Software injections can be done any time and involve simply adding a signal into the already collected data via the software. Several codes exist that can inject various inspiral type signals. These

codes offer much freedom in how many signals to inject as well as in the parameters of the injected signals. The codes return xml files very similar to those returned by the template bank code described above. They contain a list of parameters for each injection that gets synthesized into a signal and added to the data by the inspiral search code if desired.

The inspiral code is run with the data, template bank, and possibly a specified injection file. This code resamples the data if necessary, forms the template bank from the template bank parameter lists, computes the power spectrum of the data, and correlates the templates with the data, recording any events above the specified signal to noise ratio threshold. As mentioned above, this code can be run with the option of outputting the complex filter data. The nature of these chunks of complex data will be discussed further in the next section. Suffice it to say for now however, that this is absolutely necessary if one wishes to perform a coherent inspiral search using the coherent code. The complex data is written out as a time series.

In Appendix F is a printout of the coherent code, coherent_inspiral.c. There are essentially two different modes this code can be run in: pipeline mode, and test mode. The only difference is the way in which the input files are provided to the code. In the pipeline mode, the code globs, or searches, for files containing the correct complex filter data. In the test mode, the user provides the name of a file containing the complex filter time series data for

each interferometer in the desired network. Since the code is doing a coherent analysis which is inherently multi-interferometer, it will print a friendly message and exit if one attempts to run it for a network with a single interferometer.

The coherent code must also be given an event file, formatted in a specific way, that essentially tells it how to read in the given files containing the complex data. For each block of complex data in these files, the code will first determine what type of network it is dealing with. The code can operate on data for up to four detectors, but there are differences in how the coherent statistic is calculated depending on both the number of detectors in the network and the number of separate detector sites in the network (it is possible to have two detectors at the same site and is a reality at the detector site in Hanford, Washington). There are essentially 7 different possibilities labeled Case 1, Case 2A, Case 2B, Case 3A, Case 3B, Case 4A, and Case 4B, where the number signifies the number of detectors in the network and the letter signifies whether or not the network has two detectors that are at the same site ("A" if it does, "B" if it does not). The formulae for the coherent statistics are arrived at from the formalism developed in the previous chapter with the identification that C is the complex filter output.

After determining what case it is operating under, the code may need to read in the beam pattern coefficients. These are generated in the geocentric

frame by a Mathematica notebook given in Appendix B. These come into play only for networks that have three or more sites. The data is then read in, and the coherent statistic is computed as a time series. Whenever the statistic rises above a user defined threshold, and event is recorded with the option to cluster based on SNR and time window (to avoid getting hundreds of events spaced milliseconds apart corresponding to one actual signal in the data or maybe one noise burst). The code also has the ability to estimate some parameters of the signal coherently as outlined in the previous chapter. Table 4.1 lists what the coherent code outputs for each different network case where Y stands for yes and N stands for no.

Table 4.1: Parameter Estimation

Parameter	2A	2B	3A	3B	4A	4B
End T(s)	Y	Y	Y	Y	Y	Y
End T(ns)	Y	Y	Y	Y	Y	Y
Mass 1	Y	Y	Y	Y	Y	Y
Mass 2	Y	Y	Y	Y	Y	Y
M	Y	Y	Y	Y	Y	Y
η	Y	Y	Y	Y	Y	Y
Eff Dist	Y	Y	Y	Y	Y	Y
Coh SNR	Y	Y	Y	Y	Y	Y
Cone $\angle$	N	Y	Y	N	N	N
θ	N	N	N	Y	Y	Y
ϕ	N	N	N	Y	Y	Y
ϵ	N	N	N	Y	Y	Y
ψ	N	N	N	Y	Y	Y
δ_c	N	N	N	Y	Y	Y

The η in the table is a mass parameter found by dividing the reduced mass by the total mass. The distance in Table 4.1 is computed by anology with the single detector effective distance estimates. For the single interferometer, it has been shown that the effective distance in megaparsecs is given by

$$D_{eff} = \frac{\sigma_{noise}}{\rho} \quad , \tag{4.1}$$

where σ_{noise} is the noise variance and ρ is the single detector signal to noise ratio. By analogy then, the coherent effective distance estimate is given by

$$D_{eff}^{coh} = \frac{||\mathsf{b}||}{||\mathsf{C}||} \quad , \tag{4.2}$$

where b and C are network vectors and were discussed in the previous chapter.

Gravitational waves travel at the speed of light as predicted by Einstein's theory. Given that, a gravitational wave would not necessarily reach two separated sites at the same time. There would be some finite time delay of the order of tens of milliseconds and this delay would of course depend on the sky position of the source. Since we do not know this sky position in advance, when we form the coherent statistic we test for signals of all possible time delays up to the maximum light travel time between the given detector sites. Consider a simplistic two detector description of the algorithm. First compute the light travel time between two sites based on their stored position values. Then determine how many data points will fit into this time period based on the sampling rate which the user provides. This is now the time

point window for which to search for signals with. For each timepoint in the first detector's data time series, the coherent statistic is computed for each timepoint in the second detectors data contained one time point window before and after the timepoint in the first detector. Only the coherent statistic which was largest after doing all these calculations is kept. Adding more detectors to the network simply adds an additional shell to the algorithm. This gets very computationally intense for three or more detectors because of the way the total number of calculations gets compounded. This fact prohibits us from doing a full blown coherent search over all the data.

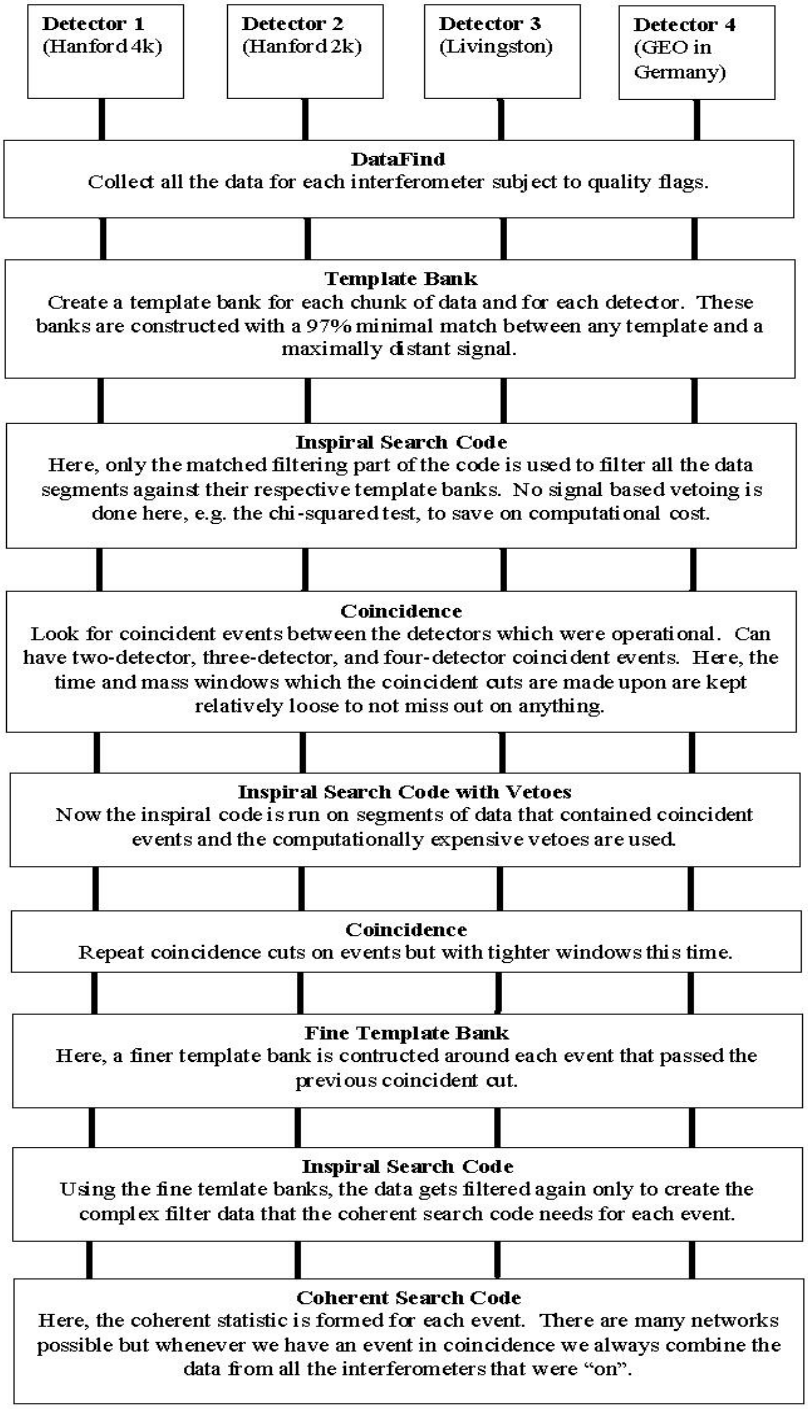
4.3 The Search Pipeline

The search pipeline is composed of many of the codes discussed already in the section on LALapps. This pipeline was arrived at by members of the Inspiral Upper Limits (IUL) group of the LSC. These codes are run in a particular succession to analyze the data but also to be computationally as efficient as possible at the same time. Even with maximal efficiency, the use of large computer clusters is still necessary to analyze a given data set on the time scale of about a week. Within the LSC there are several computing clusters available in many different locations such as Hanford, Livingston, Penn State, University of Wisconsin Madison (UWM), and Caltech to name some. The data is archived at these sites and so they are kept under very

strict security. These clusters employ a software that was developed at UWM called Condor to manage jobs. To run the inspiral search pipeline essentially requires the writing of a file known as a Directed Acyclic Graph, or DAG, that the Condor job scheduler uses to set up all the parent-child relationships among jobs at varying levels to ensure they get run in the correct order. Some parts of the pipeline need results from previous parts.

The DAG for the inspiral search pipeline is generated by running a Python script in LALapps. This script has been modified appropriately so that a coherent analysis step is now in place and being used. Figure 4.3 shows the steps of the inspiral pipeline. All of the data that is taken is typically not worth analyzing. Some of it is polluted with noise bursts from many possible sources and so some sections of the data become flagged for the various types of known pollution. Prior to running the pipeline over a data set, one must generate a list of GPS time intervals of data that is to be used in the analysis subject to constraints on some of these different pollution flags. These lists are then then used in the very first step of the pipeline to locate the specified data on disk. Once it is located, it is divided up into analysis chunks. A series of template bank jobs then ensue. Template banks are constructed for each chunk based on the noise power spectrum and a 97% minimal correlation.

Figure 4.1: Inspiral analysis search pipeline.



Once the template banks are in place, the inspiral code can be run with them along with the data to search for signals. At this inspiral stage of the pipeline we do not apply any computationally expensive vetoes such as the chi squared veto discussed previously. After running the inspiral code over the many segments of data, we are left with large preliminary event or trigger lists for each interferometer. At this stage we perform a coincidence analysis on these triggers both in time and in recovered mass. We expect that if one interferometer registered an event at a particular time and estimated a particular mass, that other interferometers should recover an event with a very similar time and similar recovered mass. Typically, we set aside ten percent of the data, interspersed throughout a given data set, to tune some of these pipeline parameters such as the appropriate time and mass window to use. This data is known as the *playground* data. We always err on the safe side so as not to miss out on any true events that may be hiding in the noise.

After doing the coincident analysis, we then re-run the inspiral codes over those triggers that were left over after the coincidence cut. This time however, we apply our computationally expensive vetoes. Since we have much fewer jobs now, it is acceptable to use the vetoes. This marks a huge savings over applying the vetoes at the first inspiral stage. Any triggers that result are then put through an additional coincident analysis very similar to the first. Triggers surviving this stage are convincing candidates and warrant a coherent followup. Around each of these triggers we construct a fine template

bank that will now act as an event file. In Appendix G can be found the CoherentBank.c code which is used to build this bank. Since the coherent statistic must be formed of filter outputs that from the different interferometers corresponding to the same filter, the coherent bank code must perform a commensuration. A given event could ring off a different template in each interferometer and they would therefore all have different complex data. The inspiral code is then run on this much reduced set of triggers and with these small, fine template banks to generate the complex filter data that the coherent code needs. Since we really only care about the part of the data very close to these events, the complex data is typically only output in 4 second chunks around the GPS time of the event in the bank file. This speeds up the coherent step drastically.

After the complex data is generated, the coherent statistic is constructed for each of the events and its parameters are coherently estimated. If we record a value above threshold at this stage, we would claim a detection but follow up with many more studies before becoming absolutely convinced. It is also worth noting that in the future it may be possible to exchange the complex data snippets with other observatories not in the LSC. Of course the data is very sensitive and has come at high cost so other observatories would not be keen on exchanging whole data sets of raw data. They may possibly want to exchange complex filtered snippets, which are much less sensitive, for mutual benefit. This makes the coherent analysis more marketable.

4.4 Testing the Code

Some testing and validation of the code has taken place. This is typically done by performing injections in data and searching for them with the coherent code. In the studies presented here, three large signals were injected into a 600 second long chunk of playground data for the two Hanford detectors and the Livingston detector. To generate the complex filter data, each data chunk was filtered with a template that very closely resembled the injected signal so as not to suffer much loss in SNR due to template-signal mismatch. The three injections were placed 200 seconds apart so as to lessen their effect on the noise PSD estimate. The end times of the injected signals at the two sites are given in Table 4.2, where H stands for the Hanford location and L for the Livingston. There is a variety of end time differences between the two sites, indicating a variety in source sky positions.

Table 4.2: Injected GPS End Times

Inj #	H End T(s)	H End T(ns)	L End T(s)	L End T(ns)
1	751976335	20549769	751976335	20499738
2	751976335	17828239	751976335	11089951
3	751976335	21077676	751976335	19083738

After filtering with the single detector inspiral code (to generate the complex filter data), the single detector results are obtained and given in Table 4.3. There, rather than give the GPS times, the timing error which is the absolute value of the difference between the injected end time and the end

time recovered by the inspiral code is given. It should be noted that for this study the sample rate was 4096 Hz so the timing resolution is about a quarter of a millisecond.

Table 4.3: Parameters Recovered by Inspiral Code

Injection #	Detector	SNR	Timing Error (ms)
1	Livingston	22.52	0.50
1	Hanford 4k	81.02	0.45
1	Hanford 2k	25.65	0.45
2	Livingston	14.64	0.38
2	Hanford 4k	63.97	0.48
2	Hanford 2k	19.47	0.48
3	Livingston	19.22	0.45
3	Hanford 4k	67.92	0.41
3	Hanford 2k	19.77	0.41

With the complex data in hand, it is then possible to do some coherent analysis. There are a few different networks possible given the three detectors used in this analysis. The networks chosen for analysis are H1-H2, or the Hanford 4k and 2k respectively, H1-L, and H1-H2-L. This gives Case 2A, Case 2B, and Case 3A respectively. The coherent SNR's obtained are given in Table 4.4. The recovered times from the coherent code are all the same as the time of the peak SNR in Hanford. This is because of the way the search algorithm is constructed. Hanford is chosen as the fiducial detector if it is in the network.

In the case of the H1-H2 network, one can not simply compare the coherent SNR obtained with two respective single detector SNR's. This is because

Table 4.4: Coherent SNR's Recovered

Case	SNR for Inj 1	SNR for Inj 2	SNR for Inj 3
Case 2A: H1-H2	106.67	83.43	87.69
Case 2B: H1-L	84.09	65.62	70.58
Case 3A: H1-H2-L	109.2	84.70	89.77

when there are two detectors in a network that are at the same site, the coherent statistic will depend on phase cross terms and is not obtained by a simple sum of squares as it is for cases 2B and 3A (beyond 3A the beam pattern coefficients come into play). To aid in seeing how the coherent SNR is arrived at for case 2A, Table 4.5 gives the real and imaginary parts, x and y respectively, of the complex filter output at the time in which the coherent snr was recovered.

Table 4.5: Real and Imaginary Parts of Complex Filter Output

Injection #	Component	H1	H2
1	x	-29.65	-9.99
1	y	75.40	23.63
2	x	28.81	8.02
2	y	57.11	17.74
3	x	-65.50	-19.09
3	y	-17.97	-5.15

Given the values in Table 4.5, one can compute the coherent statistic as

$$\rho_{coh} = \sqrt{\rho_{H1}^2 + \rho_{H2}^2 + x_{H1}x_{H2} + y_{H1}y_{H2}} \quad , \tag{4.3}$$

where $\rho_{H1,H2}$ are the single detector SNR's. If one performs this calculation with the single detector SNR's given in Table 4.3 and the complex filter components give in Table 4.5, then the result is 96.44, 75.6, and 79.67 for the coherent SNR's for injection 1, 2, and 3 respectively. The reason that these values are lower than the recovered coherent SNR's given in Table 4.4 is a result of the fact that the complex filter components were taken at the time of the recovered coherent SNR. In the past, a study was carried out by this author on the effect of including a small timing bias between detectors at the same site when combining their data to form the coherent statistic. That study showed that because of timing inaccuracy, a small error should be allowed so that a higher coherent SNR can be recovered. This is presently allowed for in the coherent code and so the values obtained by it are slightly higher.

For cases 2B and 3A, the values for the coherent SNR can be directly compared to the single detector SNR's. There, the coherent SNR is calculated as

$$\rho_{coh} = \sqrt{\rho_L + \rho_{H1}} \quad , \tag{4.4}$$

for case 2B considered here, and as

$$\rho_{coh} = \sqrt{\rho_{H1-H2} + \rho_L} \quad , \tag{4.5}$$

for case 3A, where ρ_{H1-H2} is the coherent SNR for the H1-H2 network. Plugging in the appropriate values for case 2B gives coherent SNR's of 84.09,

65.62, and 70.58 for injection 1, 2, and 3 respectively. These values match exactly those given in Table 4.4. Similarly, plugging in the appropriate values for case 3A gives coherent SNR's of 109.2, 84.7, and 89.77 for the three injections. Again, these values exactly match those given in Table 4.4. Similar studies have been done for all the remaining cases with results that are as expected. Other large scale tests also indicate that the codes now reside in a relatively bug-free state.

Chapter 5

Hierarchical Search Methods

The goal of a hierarchical search is not only to detect an astrophysical signal, but also to save on the computational resources needed to do so. The basic idea is to first filter a given segment of data with a somewhat coarse template bank but then follow up each event with a second stage, finer template bank centered around the event. If there are not many first stage events or crossers, then this will clearly save computationally over doing one search with a very large fine template bank. Work on this topic has been done for signals from inspiraling compact binaries up to 1.5 post-Newtonian order and on a single interferometer level [14, 17]. This work will apply the techniques developed in the those previous works but for a network of interferometers.

5.1 One Step Search

Before analyzing how a two step search can be done, it is instructive to first look at the single step search. One could form a bank of templates to cover the parameter space but will obviously want to use the smallest number of templates possible to reduce the amount of work or computation. Some criteria for the placement of the templates must be formulated. One natural thing to do would be to require that every signal having a strength S greater than some minimum signal strength S_{min} be detected with some detection probability Q_d greater than a given minimum detection probability $Q_{d,min}$, and with a false alarm probability Q_0 below some specified value $Q_{0,max}$. Clearly, for larger signal strengths it should be possible to use larger spacings and still satisfy all the criteria.

The false alarm probability and detection probability are both functions of the signal to noise ratio threshold. In Mohanty's paper [17], approximate expressions for these probabilities were arrived at based in part on theory and some on the results of Monte Carlo studies. As you allow the signal to depend on more and more parameters however, as it does in the coherent search, the problem of calculating the false alarm and detection probabilities for a bank of templates becomes intractable.

5.2 Two Step Search

In a two step search the criterion would be similar but now we seek to minimize the total number of templates from the first step and second step combined. There are two thresholds at play now, one for the first stage and one for the second stage. If the first stage threshold is set too low, then we will be flooded with threshold crossers and would need to employ many fine templates at the second stage. On the other hand if we set the first stage threshold too high we risk losing out on signals.

If we assume that there are no true signals present in the data, then due to noise alone we should expect to have a number of first stage crossers n_c^{av} given by

$$n_c^{av} = Q_0(\eta^{(1)}) \times n_t^{(1)} \quad , \tag{5.1}$$

where $\eta^{(1)}$ is the signal to noise ratio threshold at the first stage, $Q_0(\eta^{(1)})$ is the probability of a crossing for a single template at the first stage, and $n_t^{(1)}$ is the total number of first stage templates. The average number of templates we would use for both the first and second stage together n_t^{av} is given by

$$n_t^{av} = (n_c^{av} + q) \times M + n_t^{(1)} \quad , \tag{5.2}$$

where q is the average number of signals that may occur in a given length of data and M is the fixed number of second stage templates which are used for each crosser of the first stage. Both n_c^{av} and n_t^{av} are a function of the

first stage spacing of the templates in the different directions of the parameter space. The average number of first stage crossers rises as the spacings increase, but the average number of total templates may start out large for very small first stage spacings, being dominated by the number of first stage templates, but then will get smaller as the first stage spacings get larger. A point will be reached, however, when the spacings are so large that we start getting many first stage crossers. At that point the number of templates at the second stage will start to dominate and increase the total average number of templates. So there is a tradeoff between the two, and some set of spacings for which the total number of templates is minimized.

5.3 Monte Carlo Studies

As mentioned before, in the case of the 2PN coherent search, the signal depends on a large number of parameters. To make the problem more manageable, a two detector case is considered in which a hierarchy only in the individual masses of the binaries is being done. To get a handle on the first and second stage template spacings that one can get away with, one must first have a good handle on the detection and false alarm probabilities. Also mentioned above, the task of deriving their functional forms is intractable. This is where we rely on Monte Carlo studies to estimate them.

To estimate the false alarm probability of a two detector network as a function of the coherent SNR threshold, the coherent code was run thousands of times with a fixed, small template bank on the space of the masses of the binaries. Each time, Gaussian noise was synthesized using different random number seeds to ensure, as much as possible, no statistical bias. There were no injections made into the data since we are trying to estimate the false alarm which would happen only when there actually isnt a signal in the data. For each run, the threshold was kept very low so that many different threshold cuts could be made subsequently. A Mathematica notebook was then written to take all the resulting data and, by making a series of different threshold cuts, determine the false alarm probability as a function of threshold. The false alarm probability is the same for each template and the spacings do not matter so they were kept fixed for this part of the analysis.

After the false alarm probability was adequately understood, the same series of runs were performed but with a signal injected in the data each time to estimate the detection probability. The signal was injected so as to always lie in the middle of the square bank for the worst possible scenario. This would give us the safest set of first and second stage spacings. Here, however, the spacings of the templates were altered, since the detection probability depends on them as well as on the threshold. A Mathematica notebook was then written to determine the set of first and second stage spacings that leads to a minimum number of templates.

To determine the best set of spacings from all the data resulting from the Monte Carlo simulations, the following algorithm was employed. First, given a minimum signal strength S_{min}, a minimum detection probability $Q_{d,min}$, and a maximum false alarm probability $Q_{0,max}$, a one step template bank and threshold are set up using the algorithm presented above in the one step search section. It may be possible to do this by using the data for many of the spacing sets chosen. Next, for every set of spacings that Monte Carlo simulations have been done for, the first stage threshold $\eta^{(1)}$ is calculated such that the detection probability is equal to the minimum tolerated $Q_{d,min}$. Next, n_t^{av} is calculated according to the prescription in equation (5.2) , and stored. The set of first and second stage spacings that minimize this number must be the best set to use given the studies that have been done. Below, plots for the average number of first stage crossers and the total average number of templates as functions of the spacings in the binary's masses are given.

Examining Figs. 5.1 and 5.2, each quantity is plotted versus the cell area which is equivalent to the product of the two mass parameter spacings. As expected, n_c^{av} will start out high for small cell areas because for small cell areas the number of first stage templates required is very large (the number of templates is inversely proportional to the cell area). As the cell area is

Figure 5.1: Average number of crossers from the first stage.

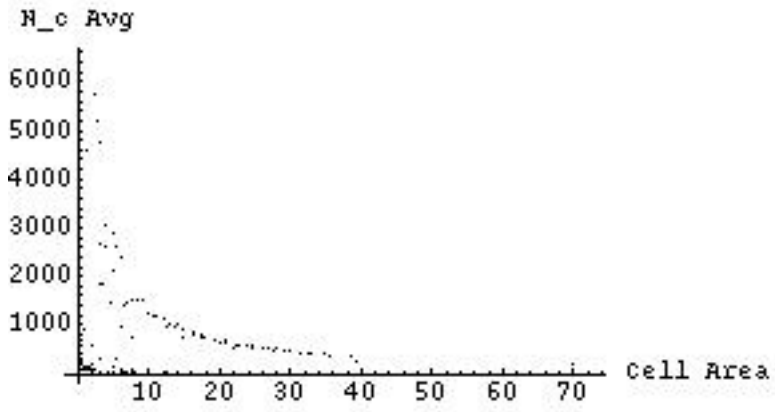
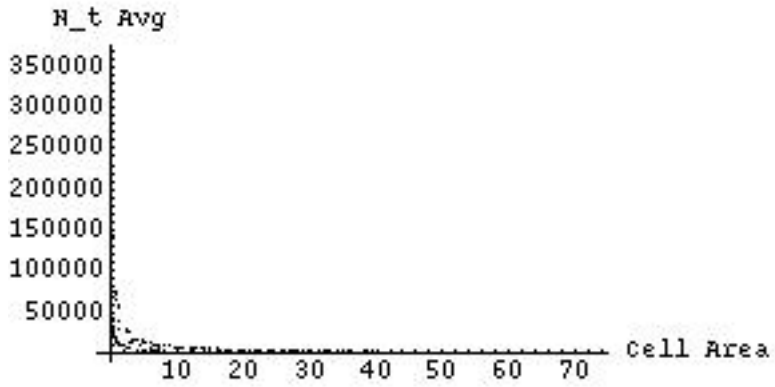


Figure 5.2: Total number of templates (1st and 2nd stages combined).



increased then the number of first stage templates is reduced, so the number of first stage crossers drops off. If we were to go to larger cell areas then we would expect to reach a point where n_c^{av} starts to increase again due to the fact that we would be forced to drop our first stage threshold very low and hence would be flooded with crossers. The behaviour for n_t^{av} should follow the same pattern. When the cell sizes are very small then the number of first stage templates is very large and will dominate the total number of templates, n_t^{av}. Then, when the cell area gets larger, n_t^{av} should bottom out at some desirable minimum then start to increase again due to the fact that for large cell areas we are forced to drop our first stage threshold. At that point we will have many first stage crossers and would be forced to employ many second stage templates around them. So for large cell areas the n_t^{av} becomes dominated by the number of second stage templates.

These plots were obtained after many days of running Monte Carlo studies. Although it is not apparent that we have reached the minimum number of templates from the plots, these results can still be useful. Even if we use the largest cell area that was tested, we would still benefit from much savings in the overall computational cost.

Chapter 6

Excess Power Statistic

The energy output during the merger phase of a massive binary black hole coalescence can be substantially higher than that yielded by its inspiral phase. Such events are also visible to much larger distances than binary neutron star coalescences, and offer better chances of detection with the first generation of Earth-based interferometers. However, we do not yet have reliable enough waveforms for the merger phase. Here we extend the work of Flanagan and Hughes in modeling this phase as a burst and coherently analyzing the data from multiple interferometers for detecting such a signal. Although this strategy is not as efficient as matched filtering (i.e., if reliable templates were present), it is powerful enough in discerning for consistency in polarization and sky-position information available in the signals at the different detectors.

The waveforms for the merger phase of the evolution of a binary black hole system are in the preliminary stages and impractical for use in matched filtering. Since we expect a large amount of gravitational radiation to be emitted in this phase[2], an alternative detection strategy must be adopted. Several detection statistics have been proposed to search for unmodeled bursts [18, 3, 19]. Flanagan and Hughes[20] first proposed the use of an excess power statistic for detection of such bursts. Anderson et al.[2] have since shown this statistic to be optimal when one assumes only knowledge of the duration of the signal and its frequency band. In our work we seek a statistic that can be applied to any network of interferometric detectors. Our method will differ from those previously proposed in the way that we arrive at the effective gravitational wave strains for a network. The statistic will also lend itself to the detection of some other burst type sources such as supernovae, which are poorly modeled.

If a signal is well modeled, then matched filtering is the optimal method for analyzing the data. In matched filtering one computes the cross correlation of the detector output with the template. In our formulation, we begin with the analogous cross correlation between network template and data vectors as presented in a paper by Pai et al.[21]. To arrive at the effective network strains we re-write this statistic as a sum of cross-correlations between a single template and an effective network strain for the two gravitational wave strains. The excess power statistic for the network is then simply the cross correlation of the network effective strain with itself. We arrive at

this network excess power statistic under the assumption that the detector noises are stationary and Gaussian with vanishing means. We furthermore assume that there are no noise correlations between detectors (i.e. the noise covariance matrix is diagonal). Our analysis can be easily extended to allow for noise correlations between detectors. Finally, we show how this excess power statistic can be analytically maximized over two of the source parameters, namely, the inclination of the binary orbit, ϵ, and the polarization angle, ψ. This is useful in mergers where gravitational waves are emitted with a dominant polarization component. In the absence of such a component one must average the statistic over ψ and ϵ.

6.1 Formulation of the Statistic

The convention for symbols in this chapter is the same as that employed in Chapter 3. The signal from a binary merger may not stand above the broadband noise of the interferometric detectors; the concept of an absolutely certain detection does not exist in such a case. Only probabilities can be assigned to the presence of an expected signal. In the absence of prior probabilities, such a situation demands a decision strategy that maximizes the detection probability for a given false alarm probability. This is termed as the Neyman-Pearson criterion [6]. Such a criterion implies that the decision must be based on a statistic called the likelihood ratio (LR). It is defined as the ratio of the probability that a signal is present in an observation to

the probability that it is not. This is the criterion we employ in formulating our detection strategy.

In order to define a strategy to search for signals in a noisy environment, it is important to recognize the characteristics of the noise. Here, we assume that the noise, $n^I(t)$, in the I-th detector (a) has a zero mean and (b) is mostly stationary and statistically as well as algebraically independent of the noise in any other detector. These requirements are mathematically summarized in equations (3.13) and (3.14). In what follows, let $w_{(I)}(f)$ be one-sided power-spectral-density (PSD) of the I-th detector. Note that $w_{(I)}(f)$ is the Fourier transform of the auto-covariance of the noise in detector I. We also assume the noise to be additive. This implies that when a signal is present in the data, then $h^I(t)$ is given by

$$h^I(t) = s^I(t) + n^I(t) \quad , \tag{6.1}$$

otherwise $h^I(t) = n^I(t)$. Finally, we assume that the noises are Gaussian, i.e., the two moments in Eqs. (3.13) and (3.14) are sufficient to completely characterize the noises statistically[1].

As previously mentioned, an important tool in the theory of detection of known signals in noisy environments is the cross correlation between a signal template and a detector's output. In order to define it, consider two real, sufficiently smooth, and absolutely integrable functions of time, namely, $A(t)$

[1]In reality, detector noise contains non-Gaussian components. Such features can be accommodated in our treatment by using robust methods discussed in [22]

and $B(t)$. In what follows we assume that the signal template, $s^I(t)$, and the detector outputs, $h^I(t)$, belong to this category of functions to a good approximation. A cross correlation can be represented in terms of an inner product, which is defined in equation (3.17).

For a network of M detectors, the data consist of M data trains, $\{h^I(t)|I = 1, 2, ..., M \text{ and } t \in [0, T]\}$. The network matched-template can be obtained naturally by the maximum-likelihood method, where the decision whether the signal is present or not is made by evaluating the likelihood ratio (LR) for the network [6]. Under the assumptions made on the noise, the network LR, denoted by λ, is the product of the individual detector LRs. In addition, for Gaussian noise, the logarithmic likelihood ratio (LLR) for the network is just the sum of the LLRs of the individual detectors [12] as in equation (3.19). The network LLR takes a compact form in terms of the network inner-product as in equation (3.21). It can be shown by using the Schwarz inequality that the network template, s, defined above yields the maximum signal-to-noise (SNR) amongst all linear templates and, hence, is the matched template. As shown in Ref. [12], in terms of the above definitions, the network LLR takes the following simple form as given in equation (3.24), which is a function of the source parameters that determine s. To formulate the network effective strain we begin with a piece of this network LLR, namely the cross correlation between the signal and the data.

A convenient form of the signal at the I-th detector is [21] :

$$s^I(t) = 2\kappa\, \Re\left[(E_I^* S^I) e^{i\delta_c}\right] \quad , \tag{6.2}$$

where the extended beam-pattern functions are given by

$$E^I = g_{(I)} {T_2}^p(\psi, \epsilon, 0) {D_p}^I, \qquad p = \pm 2, \tag{6.3}$$

S^I is the normalized complex signal, δ_c is the initial phase of the gravitational wave, and κ denotes the inherent strength and depends inversely on the luminosity distance to the source. Here, $g_{(I)}$ is a normalization factor such that

$$\langle S^I, S^I \rangle_{(I)} = 1\,, \tag{6.4}$$

the ${T_2}^p$'s are the Gel'fand functions[13], and

$${D_p}^I = -i {T_p}^s(\phi, \theta, 0)\left[{T_s}^{2*}(\alpha_{(I)}, \beta_{(I)}, \gamma_{(I)}) - T_s^{-2*}(\alpha_{(I)}, \beta_{(I)}, \gamma_{(I)})\right], \tag{6.5}$$

with $s = 0, \pm 1, \pm 2$. Here, θ and ϕ are the source direction angles while the $\alpha_{(I)}$'s, $\beta_{(I)}$'s, and $\gamma_{(I)}$'s are the Euler orientation angles of their respective detectors with respect to a reference or fiducial detector (fide), which is chosen to be the $I = 1$ detector.

Let the Fourier transform of the I-th signal $S^I(t)$ be

$$\tilde{S}^I(f; t_c, \tau_{(I)}, \vartheta^\mu) = \int_{-\infty}^{\infty} S^I(t; t_c, \tau_{(I)}, \vartheta^\mu) e^{-2\pi i f t} dt \quad . \tag{6.6}$$

Here, ϑ^μ are source parameters, t_c is the time of coalescence, and $\tau_{(I)}$ is the relative delay between the arrival times at the I-th detector and the fide. Note that $\tau_{(I)}$ can be positive or negative and depends on the source sky position. In the frequency domain, the time-delay separates out as a phase factor

$$\tilde{S}^I(f; t_c, \tau_{(I)}, \xi) = \tilde{S}^I(f; t_c, \xi) e^{2\pi i f \tau_{(I)}} \quad . \tag{6.7}$$

So far, the time delay between detectors is manifested only in the template signal vector and not in the data. We could just as easily make all the templates correspond to a time delay of zero and shift the data around. It is easy to see simply by looking at the definition for the cross correlation that the results will be identical regardless of where the time delay is in computing the cross correlation between the template and the data. We can then write:

$$\sum_{I=1}^{M} \langle s^I(t; -\tau_I), h^I(t; 0) \rangle_{(I)} = \sum_{I=1}^{M} \langle s^I(t; 0), h^I(t; \tau_I) \rangle_{(I)}. \tag{6.8}$$

Next, the signal can be decomposed as

$$\begin{aligned} s^I &= \kappa E_I^* S^I e^{i\delta_c} + \kappa E_I S^{I*} e^{-i\delta_c} \\ &=: Z_L^I + Z_R^I \quad . \end{aligned} \tag{6.9}$$

Now the above network cross correlation can be re-written as

$$\begin{aligned}\sum_{I=1}^{M}\langle s^I(t;0), h^I(t;\tau_I)\rangle_{(I)} &= \sum_{I=1}^{M}\langle Z_L^I + Z_R^I, h^I(t;\tau_I)\rangle_{(I)} \\ &= \sum_{I=1}^{M}\langle Z_L^I, h^I(t;\tau_I)\rangle_{(I)} + \sum_{I=1}^{M}\langle Z_R^I, h^I(t;\tau_I)\rangle_{(I)} \end{aligned} \quad (6.10)$$

The expressions for the network effective strains can be obtained by trading off amplitude terms from the left hand side of the cross correlation to the right hand side. After some simplification, one obtains:

$$\sum_{I=1}^{M}\langle s^I(t;0), h^I(t;\tau_I)\rangle_{(I)} = \sum_{\alpha=L,R}\langle Z_\alpha^1, \sum_{I=1}^{M} Y_\alpha^I\rangle_{(1)}. \quad (6.11)$$

The two components for the network effective strain are given by $\sum_{I=1}^{M} Y_\alpha^I$, where the Y_α^I can be defined through their Fourier transforms as

$$\tilde{Y}_L^I(f;\tau_{(I)}) = \frac{E_I}{E_1}\frac{g_{(1)}w_{(1)}}{g_{(I)}w_{(I)}}\tilde{h}^I(f;\tau_{(I)}) \quad , \quad (6.12)$$

$$\tilde{Y}_R^I(f;\tau_{(I)}) = \frac{E_I^*}{E_1^*}\frac{g_{(1)}w_{(1)}}{g_{(I)}w_{(I)}}\tilde{h}^I(f;\tau_{(I)}) \quad , \quad (6.13)$$

where quantities with an index of 1 ($I = 1$) refer to the first, or fide, detector.

Here, we will show how one can formulate an excess power statistic from these network effective strains. We now obtain the optimal statistic for coherently searching for the left-circular polarization component in the data from multiple detectors. Our strategy can be straightforwardly extended to

search the right-circular polarization component as well. The excess power statistic for the left polarization can then be written as

$$\mathcal{E} = \left\langle \sum_{I=1}^{M} Y_L^I, \sum_{J=1}^{M} Y_L^J \right\rangle_{(1)} \quad . \tag{6.14}$$

To simplify matters, define

$$C^{IJ} \equiv \int_{-\infty}^{\infty} \frac{w_{(1)} \tilde{h}^{I*}(f;\tau_{(I)}) \tilde{h}^{J}(f;\tau_{(J)})}{w_{(I)} w_{(J)}} df \quad , \tag{6.15}$$

which is Hermitian in indices I and J. We then have

$$\mathcal{E} = \sum_{I,J=1}^{M} \frac{g_{(1)}^2}{g_{(I)} g_{(J)}} \Re \left[\frac{E_I^* E_J}{|E_1|^2} C^{IJ} \right] \quad . \tag{6.16}$$

Note that for a single detector this statistic is proportional to the single detector statistic given in reference [2].

6.2 Maximization over ψ and ϵ

To handle the ψ and ϵ dependence there are two possibilities: the statistic can be averaged over them or it can be maximized over them. In an event such as a supernova explosion, a substantial part of the gravitational-wave emission will likely be unpolarized. In that case it may be more appropriate to average over ψ and ϵ. For sources that may have a dominant polarization component, such as binary black hole mergers, it will be optimal to maximize

the statistic over ψ and ϵ. Since the averaging simply amounts to taking the integral average of the statistic over the two variables in question, it is more straightforward to handle and will be discussed elsewhere [23]. Here we discuss only the maximization of the statistic.

To aid in the analytical maximization of $\mathcal{E}$ over ψ and ϵ, all factors containing their dependence must first be separated out. It is easy to see from expression (6.16), that the ψ and ϵ dependence is contained entirely within the detector extended beam pattern functions. We are always free to choose the orientation coordinate system for the first, or fide, detector since all the other detectors are oriented with respect to it. Some choices allow for greater simplification of the statistic than others. We make the following choices:

$$\gamma_{(1)} = 0, \qquad \beta_{(1)} = \frac{\pi}{2}, \qquad \phi - \alpha_{(1)} = \frac{\pi}{2}. \tag{6.17}$$

Making these choices leads to a much more simplified expression for E_1:

$$E_1 = -\frac{1}{2} g_{(1)} \sin(2\theta)({T_2}^2 + {T_2}^{-2}) \quad . \tag{6.18}$$

The statistic can now be written as

$$\mathcal{E} = \frac{4}{\sin^2(2\theta)\left|{T_2}^2 + {T_2}^{-2}\right|^2} \sum_{I,J=1}^{M} \frac{1}{g_{(I)} g_{(J)}} \Re\left[E_I^* E_J C^{IJ}\right] \quad . \tag{6.19}$$

Using (6.3) to substitute for the extended beam-pattern functions and after

some algebraic simplification, one finally obtains

$$\mathcal{E} = \Theta_{++} + 2\Re\left(W\Theta_{+-}\right) - |W|^2\,\Theta_{--} \quad , \tag{6.20}$$

where we defined

$$W \equiv \frac{{T_2}^2 - {T_2}^{-2}}{{T_2}^2 + {T_2}^{-2}} \tag{6.21}$$

and

$$\Theta_{\pm\pm} \equiv \frac{2}{\sin^2(2\theta)} \sum_{I,J=1}^{M} (D_2^I \pm D_{-2}^I)(D_2^J \pm D_{-2}^J) C^{IJ} C^{IJ} \quad . \tag{6.22}$$

Above, the Θ_{++} and Θ_{--} are real quantities, whereas Θ_{+-} is a complex quantity. Also, note that all the dependence on ψ and ϵ is now in W.

The ψ, ϵ variables map onto two new variables, namely, $|W|$ and $\arg W$. Since these variables are only functions of ψ and ϵ, maximizing the statistic with respect to ψ and ϵ is equivalent to maximizing $\mathcal{E}$ over $|W|$ and $\arg W$. We first maximize over $\arg W$ to obtain:

$$\mathcal{E}\big|_{\arg \hat{W}} = \Theta_{++} + 2|W||\Theta_{+-} - |W|^2\Theta_{--} \quad , \tag{6.23}$$

where

$$\arg \hat{W} = -\arg \Theta_{+-} \quad . \tag{6.24}$$

The $\hat{x}$ denotes the value of x that maximizes $\mathcal{E}$. The $\arg \hat{W}$ that maximizes the statistic is related to $\hat{\psi}$ and $\hat{\epsilon}$. Using equation (6.24), one can obtain the

following condition that $\hat{\epsilon}$ must satisfy:

$$\cos^4\hat{\epsilon} - \eta\cos^3\hat{\epsilon} - 2\cos^2\hat{\epsilon} - \eta\cos\hat{\epsilon} + 1 = 0 \quad , \tag{6.25}$$

where

$$\eta = \frac{2(1+\tan^2 2\psi)}{\tan 2\psi}\tan(\arg\Theta_{+-}) \quad . \tag{6.26}$$

Now, the above reduced statistic can be maximized over $|W|$ to obtain

$$\mathcal{E}|_{\arg\hat{W},|\hat{W}|} = \Theta_{++} + \frac{|\Theta_{+-}|^2}{\Theta_{--}} \quad , \tag{6.27}$$

where

$$|\hat{W}| = \frac{|\Theta_{+-}|}{\Theta_{--}} \quad . \tag{6.28}$$

The above equation implies that $\hat{\psi}$ must satisfy

$$\frac{\frac{1}{4}(\cos^2\hat{\epsilon}+1)^2\sin^2 2\hat{\psi} + \cos^2\hat{\epsilon}\cos^2 2\hat{\psi}}{\frac{1}{4}(\cos^2\hat{\epsilon}+1)^2\cos^2 2\hat{\psi} + \cos^2\hat{\epsilon}\sin^2 2\hat{\psi}} = \frac{|\Theta_{+-}|^2}{\Theta_{--}^2} \quad . \tag{6.29}$$

For a given network of detectors, (6.25) and (6.29) can be solved simultaneously for the ψ and ϵ that maximize the statistic. Note that these solutions provide maximum-likelihood estimates of the two parameters ψ and ϵ.

6.3 Conclusion

We have arrived at the excess power statistic for an arbitrary network of detectors. We have also shown how one can maximize this statistic over two of the source parameters, namely ϵ and ψ. This analysis can easily be extended to the case where noise correlations are allowed between detectors. For that case, the noise covariance matrix will no longer be diagonal. However, expressions obtained here for the excess power statistic should remain of the same form with the h's replaced by the eigenvectors obtained by diagonalizing the noise covariance matrix. We also hope to combine this statistic for the merger phase of the event evolution with the optimal statistics from the inspiral and ringdown phases to arrive at one "coherent" statistic that one can use to track an event through all three evolutionary phases.

Chapter 7

Conclusion

Until very recently, the sensitivities of the detectors have not been good enough for us to really expect a sizable event rate of neutron star coalescences. As the sensitivities improve, we are able to see the same sources out to greater distances and therefore expect a higher event rate. Even at the relatively low sensitivities however, data was recorded in a series of science runs termed S1, S2, S3, and recently S4. For the S1 and S2 analyses, due to the poor sensitivities there were no strong candidate events. Because of this, the analysis was geared toward providing an upper limit on the event rate to some confidence interval [24].

For S3 and beyond the sensitivities are much improved over S1 and S2 sensitivities. Now we are starting to include more and more galaxies and should have a sizable event rate. Although the analysis pipeline is only slightly different, the scientific statement at the end will hopefully now be

a claim of detection rather than an upper limit on the event rate. For S3 and beyond, the coherent analysis is in place and should help to improve our chances of detection further. We also understand the detectors more fully, having analyzed two data sets to completion. There are now improved vetoes in place that ensure a much higher quality of data.

To realize the extra benefit of the coherent analysis, data from all detectors must be used whenever they are on. This means that even if we have events coincident in only two detectors when three are actually on, we will combine data from all three to form the coherent statistic. If there is a true gravity wave present in the data, then forming the coherent statistic should yield a high value since the phase and amplitude of the signal will be consistent in all the detectors of the network. If the phases and amplitudes are not consistent then the coherent statistic will suffer a loss and a relatively low value will be obtained. This loss has been explicitly shown with the coherent code on two separate injected signals. By injecting two signals that are largely different in two different detectors we can see that the coherent statistic is indeed relatively small due to inconsistent amplitude and phase. Additionally, the coherent statistic is a reliable discriminator for detectors that are co-located like the two at Hanford. There a noise burst could easily masquerade as a signal in both detectors but the phase will typically not be consistent.

After much painstaking code writing, the coherent search has been implemented in a computationally efficient manner. By analyzing the data at

the single detector level and creating a list of coincident events to run the coherent code over, there is huge savings over doing a full blown coherent search with essentially no loss of sensitivity. This can be viewed as a sort of hierarchical search much like that discussed in Chapter 5. An additional complication that enters into the problem for networks that have three sites is the fact that the coherent statistic depends on a set of beam pattern coefficients which vary with sky position. Since we obviously can not populate the space with a continuous set of coefficients, we expect some loss due to the fact that we have a discrete bank. This loss was characterized, and, by setting an acceptable loss, banks were set up to be as coarse as possible.

After writing the coherent code itself, numerous problems were dealt with in incorporating the code into the search pipeline that gets run across cluster computing sites. This required the writing of a Python script to set up the necessary parent-child relationships that should exist between the coherent jobs and everything that happens above them in the pipeline. Once successful, large scale testing of the coherent code became possible.

The orientation and location of a detector determines how different polarizations of gravitational waves will affect it's arms for a given source position and inclination. When the coherent statistic is computed, consistency in this polarization and inclination, as well as consistent time delay and sky-position information, also need to be met for relatively large value for the result. It is this thought that prompted the formulation of the excess power statistic in Chapter 6. After reading through other papers on the subject, a

statistic that incorporated both the polarization and inclination information could not be found. The dependence of the statistic on the polarization and inclination comes about naturally in the formalism presented in Chapter 6. Furthermore, using a statistic that depends on those parameters may also prove useful in estimating Stoke's parameters for the gravitational radiation present in the data of a network.

Appendix A

Noise Curves and Noise Moments

We define the moments of the I-th detector's noise curve as,

$$i_{(I)}(q) = s_{h(I)}(f_{0(I)}) \int_1^{f_{c(I)}/f_{s(I)}} dx \frac{x^{-q/3}}{s_{h(I)}(x f_{s(I)})} \quad , \tag{A.1}$$

where $f_{0(I)}$ denotes the "knee" frequency of that detector; this is the frequency at which the sensitivity of the detector is the highest. On the other hand, $f_{c(I)}$ is its high-frequency cut-off and $f_{s(I)}$ is the seismic cut-off. The noise moment, $i_{(I)}(7)$, is related to the normalization, $g_{(I)}$, as follows:

$$g_{(I)}^2 = \frac{8}{3 s_{h(I)}(f_{0(I)})} \varrho_{(I)}^{-4/3} i_{(I)}(7) \quad , \tag{A.2}$$

where $\varrho_{(I)} \equiv f_{s(I)}/f_s$. Since our templates are normalized using the above factor, we find the following ratio useful in our calculations of the parameter-

space metric:

$$j_{(I)}(q) \equiv i_{(I)}(q)/i_{(I)}(7)\,. \tag{A.3}$$

In this paper, we evaluate these noise moments using the analytical fits to noise PSDs of different detectors given in Ref. [16]. For LIGO I we have

$$s_h(f) = s_0/3\,\left[(f_0/f)^4 + 2(f/f_0)^2\right] \quad , \tag{A.4}$$

and for VIRGO

$$s_h(f) = s_0/4\,\left[290(f_s/f)^5 + 2(f_0/f) + 1 + (f/f_0)^2\right] \quad , \tag{A.5}$$

where the parameters are listed in Table A.1. There, s_0 denotes the minimum value of $s_h(f)$, and f_0 is the frequency at which this minimum occurs. We take $s_h(f)$ to be infinite below the seismic cut-off frequency f_s. The parameter f_c is the high frequency cutoff.

Table A.1: Parameters of the Noise Fit

Parameter	LIGO I	VIRGO
s_0 (Hz^{-1})	4.4×10^{-46}	1.1×10^{-45}
f_0 (Hz)	175	475
f_s (Hz)	40	16
f_c (Hz)	1300	2750

There are certain combinations of the noise moments that appear frequently in the expression for the metric on the parameter space relevant for

a network. In order to simplify these expressions, we define the following noise-moment combinations:

$$
\begin{aligned}
k_{1(I)} &\equiv \left[j_{(I)}(1) - j^2_{(I)}(4)\right], \\
k_{2(I)} &\equiv 9\left[j_{(I)}(17) - j^2_{(I)}(12)\right]/25, \\
k_{3(I)} &\equiv 3\left[j_{(I)}(9) - j_{(I)}(4)j_{(I)}(12)\right]/5 \qquad \text{(A.6)} \\
k_{4(I)} &\equiv \left[1 - j_{(I)}(4)j_{(I)}(10)\right] \qquad \text{(A.7)} \\
k_{5(I)} &\equiv 3\left[j_{(I)}(15) - j_{(I)}(10)j_{(I)}(12)\right]/5 \qquad \text{(A.8)} \\
k_{6(I)} &\equiv \left[j_{(I)}(13) - j^2_{(I)}(10)\right], \quad ,
\end{aligned}
$$

which are, in general, different for detectors with different noise PSDs.

The moment functionals $\langle\Phi_\alpha\rangle_{(I)}$ and $\langle\Phi_\alpha\Phi_\beta\rangle_{(I)}$ defined in Eq. (3.92) and (3.93) for the I-th detector, can be given in terms of the moments of its noise

curve. They are (for $n_2 \neq 0$)

$$
\begin{aligned}
\langle \Phi_0 \rangle_{(I)} &= 2\pi \varrho_{(I)}\ j_{(I)}(4) \quad , \\
\langle \Phi_1 \rangle_{(I)} &= 6\pi \varrho_{(I)}^{-5/3}\ j_{(I)}(12)/5 \quad , \\
\langle \Phi_2 \rangle_{(I)} &= 2\pi \varrho_{(I)}^{-1}\ j_{(I)}(10) \quad , \\
\langle \Phi_{3,4} \rangle_{(I)} &= 2\pi \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right] \varrho_{(I)}\ j_{(I)}(4) \\
&= \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right] \langle \Phi_0 \rangle_{(I)} \quad , \\
\langle (\Phi_0)^2 \rangle_{(I)} &= 4\pi^2 \varrho_{(I)}^2\ j_{(I)}(1) \quad , \\
\langle \Phi_0 \Phi_1 \rangle_{(I)} &= 12\pi^2 \varrho_{(I)}^{-2/3}\ j_{(I)}(9)/5 \quad , \\
\langle \Phi_0 \Phi_2 \rangle_{(I)} &= 4\pi^2 \quad , \\
\langle \Phi_0 \Phi_{3,4} \rangle_{(I)} &= 4\pi^2 \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right] \varrho_{(I)}^2\ j_{(I)}(1) \\
&= \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right] \langle (\Phi_0)^2 \rangle_{(I)} \quad , \\
\langle (\Phi_1)^2 \rangle_{(I)} &= 36\pi^2 \varrho_{(I)}^{-10/3}\ j_{(I)}(17)/25 \quad , \\
\langle \Phi_1 \Phi_2 \rangle_{(I)} &= 12\pi^2 \varrho_{(I)}^{-8/3}\ j_{(I)}(15)/5 \quad , \\
\langle \Phi_1 \Phi_{3,4} \rangle_{(I)} &= 12\pi^2 \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right] \varrho_{(I)}^{-2/3}\ j_{(I)}(9)/5 \\
&= \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right] \langle \Phi_0 \Phi_1 \rangle_{(I)} \quad , \\
\langle (\Phi_2)^2 \rangle_{(I)} &= 4\pi^2 \varrho_{(I)}^{-2}\ j_{(I)}(13) \quad , \\
\langle \Phi_2 \Phi_{3,4} \rangle_{(I)} &= 4\pi^2 \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right] \\
&= \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right] \langle \Phi_0 \Phi_2 \rangle_{(I)} \quad , \\
\langle (\Phi_{3,4})^2 \rangle_{(I)} &= 4\pi^2 \varrho_{(I)}^2 \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right]^2\ j_{(I)}(1) \\
&= \left[r_{(I)3,1} - r_{(I)2} n_{3,1}/n_2 \right]^2 \langle (\Phi_0)^2 \rangle_{(I)} \quad , \\
\langle \Phi_3 \Phi_4 \rangle_{(I)} &= 4\pi^2 \varrho_{(I)}^2 \left[r_{(I)3} - r_{(I)2} n_3/n_2 \right] \left[r_{(I)1} - r_{(I)2} n_1/n_2 \right] j_{(I)}(1) \\
&= \left[r_{(I)3} - r_{(I)2} n_3/n_2 \right] \left[r_{(I)1} - r_2^I n_1/n_2 \right] \langle (\Phi_0)^2 \rangle_{(I)} \quad , \qquad \text{(A.9)}
\end{aligned}
$$

which shows that all the moment functionals are expressible in terms of eight independent noise-moments, $j_{(I)}(q)$. These are the ones corresponding to $q = 1$, 4, 9, 10, 12, 13, 15, 17. The values of these noise moments and the combinations (A.6) for relevant noise PSD's are listed in Table A.

For a network of three or less detectors a plane can always be arranged to contain the hubs of all the detectors. This makes it possible to choose the fide frame (or the network frame, in this case) in such a way that $r_{(I)2} = 0$

for all I. With this choice the moment functionals reduce to

$$\begin{aligned}
\langle \Phi_0 \rangle_{(I)} &= 2\pi \varrho_{(I)}\ j_{(I)}(4) \ , \\
\langle \Phi_1 \rangle_{(I)} &= 6\pi \varrho_{(I)}^{-5/3}\ j_{(I)}(12)/5 \ , \\
\langle \Phi_2 \rangle_{(I)} &= 2\pi \varrho_{(I)}^{-1}\ j_{(I)}(10) \ , \\
\langle \Phi_{3,4} \rangle_{(I)} &= r_{(I)3,1} \langle \Phi_0 \rangle_{(I)} \ , \\
\langle (\Phi_0)^2 \rangle_{(I)} &= 4\pi^2 \varrho_{(I)}^{2}\ j_{(I)}(1) \ , \\
\langle \Phi_0 \Phi_1 \rangle_{(I)} &= 12\pi^2 \varrho_{(I)}^{-2/3}\ j_{(I)}(9)/5 \ , \\
\langle \Phi_0 \Phi_2 \rangle_{(I)} &= 4\pi^2 \ , \\
\langle \Phi_0 \Phi_{3,4} \rangle_{(I)} &= r_{(I)3,1} \langle (\Phi_0)^2 \rangle_{(I)} \ , \\
\langle (\Phi_1)^2 \rangle_{(I)} &= 36\pi^2 \varrho_{(I)}^{-10/3}\ j_{(I)}(17)/25 \ , \\
\langle \Phi_1 \Phi_2 \rangle_{(I)} &= 12\pi^2 \varrho_{(I)}^{-8/3}\ j_{(I)}(15)/5 \ , \\
\langle \Phi_1 \Phi_{3,4} \rangle_{(I)} &= r_{(I)3,1} \langle \Phi_0 \Phi_1 \rangle_{(I)} \ , \\
\langle (\Phi_2)^2 \rangle_{(I)} &= 4\pi^2 \varrho_{(I)}^{-2}\ j_{(I)}(13) \ , \\
\langle \Phi_2 \Phi_{3,4} \rangle_{(I)} &= r_{(I)3,1} \langle \Phi_0 \Phi_2 \rangle_{(I)} \ , \\
\langle (\Phi_{3,4})^2 \rangle_{(I)} &= r_{(I)3,1}^2 \langle (\Phi_0)^2 \rangle_{(I)} \ , \\
\langle \Phi_3 \Phi_4 \rangle_{(I)} &= r_{(I)3} r_{(I)1} \langle (\Phi_0)^2 \rangle_{(I)} \, . \qquad \text{(A.10)}
\end{aligned}$$

The moment functionals simplify significantly in this case.

Table A.2: Noise moments and some of their combinations. In evaluating these, we take the values of $f_{s(I)}$, $f_{0(I)}$, and $f_{c(I)}$ as given in Table A.1.

Quantity	LIGO I	VIRGO
j(1)	22.43	168.9
j(4)	4.128	8.038
j(9)	0.443	0.395
j(10)	0.306	0.274
j(12)	0.157	0.150
j(13)	0.117	0.118
j(15)	0.069	0.077
j(17)	0.045	0.055
k_1	5.386	104.3
k_2	0.0073	0.012
k_3	-0.124	-0.489
k_4	-0.262	-1.198
k_5	0.013	0.022
k_6	0.023	0.043
k	0.000572	0.00168

Appendix B

Beam Coefficient Calculator

```
(* BeamCoeffCalculator.nb:
      A Mathematica Notebook for calculating beam pattern
      coefficients *)

(* This notebook will be used to compute the v (beam-pattern)
coefficients for the various detectors as outlined in eq. 4.10
of PRD 64, 042004.  This notebook outputs 5 files, one for
each detector site including Hanford, Livingston, GEO, VIRGO,
TAMA. These files are needed by coherent_inspiral.c when the
network it is being used for has 3 or 4 sites(not necessarily
detectors...). These files will need to be formatted before
use with the coherent code.  Basically remove everything so
that you are left with only 4 columns of numbers only
delimited by white space. (remove commas and braces..)  *)

(* The following two params, numpts1, numpts2,
  fix the spacing in theta and phi.  numpts1 fixes the theta
  spacing, and numpts2 fixes the phi spacing.
  A larger numpts means a smaller spacing. *)
numpts1=9;
numpts2=9;
(* Orientation angles connecting the fide and detector frames
described in \
aforementioned paper. *)
alphaH=.665145;
```

```
betaH=4.474152;
gammaH=1.875007;
alphaL=.664796;
betaL=4.948707;
gammaL=3.436204;
alphaV=5.590988;
betaV=4.815712;
gammaV=2.775423;
alphaG=5.696929;
betaG=4.709946;
gammaG=1.926844;
alphaT=0.2909464;
betaT=3.289422;
gammaT=5.695009;
(* We have now specified all the parameters that we need,
  so crank out the coefficients... *)
(* Now construct the spin weighted spherical harmonics and the
 Gel'fand \
functions that depend on them *)
(* First, the basis vectors... *)

nullm={1/(Sqrt[2])*(Cos[phi]-I*Cos[theta]*Sin[phi]),
      1/(Sqrt[2])*(Sin[phi]+I*Cos[theta]*Cos[phi]),1/(Sqrt[2])
*I*Sin[theta]};

mVectH={Exp[-I*gammaH]*1/(Sqrt[2])*(Cos[alphaH]-I*Cos[betaH]
*Sin[alphaH]),
      Exp[-I*gammaH]*1/(Sqrt[2])*(Sin[alphaH]+I*Cos[betaH]
*Cos[alphaH]),
      Exp[-I*gammaH]*1/(Sqrt[2])*I*Sin[betaH]};

mVectL={Exp[-I*gammaL]*1/(Sqrt[2])*(Cos[alphaL]-I*Cos[betaL]
*Sin[alphaL]),
      Exp[-I*gammaL]*1/(Sqrt[2])*(Sin[alphaL]+I*Cos[betaL]
*Cos[alphaL]),
      Exp[-I*gammaL]*1/(Sqrt[2])*I*Sin[betaL]};

mVectV={Exp[-I*gammaV]*1/(Sqrt[2])*(Cos[alphaV]-I*Cos[betaV]
```

```
*Sin[alphaV]),
      Exp[-I*gammaV]*1/(Sqrt[2])*(Sin[alphaV]+I*Cos[betaV]
*Cos[alphaV]),
      Exp[-I*gammaV]*1/(Sqrt[2])*I*Sin[betaV]};
mVectG={Exp[-I*gammaG]*1/(Sqrt[2])*(Cos[alphaG]-I*Cos[betaG]
*Sin[alphaG]),
      Exp[-I*gammaG]*1/(Sqrt[2])*(Sin[alphaG]+I*Cos[betaG]
*Cos[alphaG]),
      Exp[-I*gammaG]*1/(Sqrt[2])*I*Sin[betaG]};
mVectT={Exp[-I*gammaT]*1/(Sqrt[2])*(Cos[alphaT]-I*Cos[betaT]
*Sin[alphaT]),
      Exp[-I*gammaT]*1/(Sqrt[2])*(Sin[alphaT]+I*Cos[betaT]
*Cos[alphaT]),
      Exp[-I*gammaT]*1/(Sqrt[2])*I*Sin[betaT]};
(* A1\[Rule]A5 are the spherical harmonics *)
 A1={{1*0.25*Sqrt[15/(2*Pi)],I*0.25*Sqrt[15/(2*Pi)],
        0},{I*0.25*Sqrt[15/(2*Pi)],-1*0.25*Sqrt[15/(2*Pi)],0}
,{0,0,0}};
A2={{0,0,-1*0.25*Sqrt[15/(2*Pi)]},{0,
        0,-I*0.25*Sqrt[15/(2*Pi)]},{-1*0.25*Sqrt[15/(2*Pi)],
-I*0.25*
          Sqrt[15/(2*Pi)],0}};
A3={{-1*0.5*Sqrt[5/(4*Pi)],0,0},{0,-1*0.5*Sqrt[5/(4*Pi)],0},
{0,0,
        Sqrt[5/(4*Pi)]}};
A4=-Conjugate[A2];
A5=Conjugate[A1];
(* Now construct all the needed Gel'fand functions *)
B1=Sqrt[8*Pi/15]*Sum[Sum[A1[[i,j]]*nullm[[i]]*nullm[[j]],
{j,1,3}],{i,1,3}];
B2=Sqrt[8*Pi/15]*Sum[Sum[-A2[[i,j]]*nullm[[i]]*nullm[[j]],
{j,1,3}],{i,1,3}];
B3=Sqrt[8*Pi/15]*Sum[Sum[A3[[i,j]]*nullm[[i]]*nullm[[j]],
{j,1,3}],{i,1,3}];
B4=Sqrt[8*Pi/15]*Sum[Sum[-A4[[i,j]]*nullm[[i]]*nullm[[j]],
{j,1,3}],{i,1,3}];
B5=Sqrt[8*Pi/15]*Sum[Sum[A5[[i,j]]*nullm[[i]]*nullm[[j]],
{j,1,3}],{i,1,3}];
```

```
C1=Sqrt[8*Pi/15]*Sum[Sum[A1[[i,j]]*mVectH[[i]]*mVectH[[j]],
{j,1,3}],{i,1,3}];
C2=Sqrt[8*Pi/15]*Sum[Sum[-A2[[i,j]]*mVectH[[i]]*mVectH[[j]],
{j,1,3}],{i,1,3}];
C3=Sqrt[8*Pi/15]*Sum[Sum[A3[[i,j]]*mVectH[[i]]*mVectH[[j]],
{j,1,3}],{i,1,3}];
C4=Sqrt[8*Pi/15]*Sum[Sum[-A4[[i,j]]*mVectH[[i]]*mVectH[[j]],
{j,1,3}],{i,1,3}];
C5=Sqrt[8*Pi/15]*Sum[Sum[A5[[i,j]]*mVectH[[i]]*mVectH[[j]],
{j,1,3}],{i,1,3}];
D1=Sqrt[8*Pi/15]*Sum[Sum[A1[[i,j]]*mVectL[[i]]*mVectL[[j]],
{j,1,3}],{i,1,3}];
D2=Sqrt[8*Pi/15]*Sum[Sum[-A2[[i,j]]*mVectL[[i]]*mVectL[[j]],
{j,1,3}],{i,1,3}];
D3=Sqrt[8*Pi/15]*Sum[Sum[A3[[i,j]]*mVectL[[i]]*mVectL[[j]],
{j,1,3}],{i,1,3}];
D4=Sqrt[8*Pi/15]*Sum[Sum[-A4[[i,j]]*mVectL[[i]]*mVectL[[j]],
{j,1,3}],{i,1,3}];
D5=Sqrt[8*Pi/15]*Sum[Sum[A5[[i,j]]*mVectL[[i]]*mVectL[[j]],
{j,1,3}],{i,1,3}];
E1=Sqrt[8*Pi/15]*Sum[Sum[A1[[i,j]]*mVectV[[i]]*mVectV[[j]],
{j,1,3}],{i,1,3}];
E2=Sqrt[8*Pi/15]*Sum[Sum[-A2[[i,j]]*mVectV[[i]]*mVectV[[j]],
{j,1,3}],{i,1,3}];
E3=Sqrt[8*Pi/15]*Sum[Sum[A3[[i,j]]*mVectV[[i]]*mVectV[[j]],
{j,1,3}],{i,1,3}];
E4=Sqrt[8*Pi/15]*Sum[Sum[-A4[[i,j]]*mVectV[[i]]*mVectV[[j]],
{j,1,3}],{i,1,3}];
E5=Sqrt[8*Pi/15]*Sum[Sum[A5[[i,j]]*mVectV[[i]]*mVectV[[j]],
{j,1,3}],{i,1,3}];
N1=Sqrt[8*Pi/15]*Sum[Sum[A1[[i,j]]*mVectG[[i]]*mVectG[[j]],
{j,1,3}],{i,1,3}];
N2=Sqrt[8*Pi/15]*Sum[Sum[-A2[[i,j]]*mVectG[[i]]*mVectG[[j]],
{j,1,3}],{i,1,3}];
N3=Sqrt[8*Pi/15]*Sum[Sum[A3[[i,j]]*mVectG[[i]]*mVectG[[j]],
{j,1,3}],{i,1,3}];
N4=Sqrt[8*Pi/15]*Sum[Sum[-A4[[i,j]]*mVectG[[i]]*mVectG[[j]],
{j,1,3}],{i,1,3}];
```

```
N5=Sqrt[8*Pi/15]*Sum[Sum[A5[[i,j]]*mVectG[[i]]*mVectG[[j]],
{j,1,3}],{i,1,3}];
P1=Sqrt[8*Pi/15]*Sum[Sum[A1[[i,j]]*mVectT[[i]]*mVectT[[j]],
{j,1,3}],{i,1,3}];
P2=Sqrt[8*Pi/15]*Sum[Sum[-A2[[i,j]]*mVectT[[i]]*mVectT[[j]],
{j,1,3}],{i,1,3}];
P3=Sqrt[8*Pi/15]*Sum[Sum[A3[[i,j]]*mVectT[[i]]*mVectT[[j]],
{j,1,3}],{i,1,3}];
P4=Sqrt[8*Pi/15]*Sum[Sum[-A4[[i,j]]*mVectT[[i]]*mVectT[[j]],
{j,1,3}],{i,1,3}];
P5=Sqrt[8*Pi/15]*Sum[Sum[A5[[i,j]]*mVectT[[i]]*mVectT[[j]],
{j,1,3}],{i,1,3}];
mVectConjH=Conjugate[mVectH];
mVectConjL=Conjugate[mVectL];
mVectConjV=Conjugate[mVectV];
mVectConjG=Conjugate[mVectG];
mVectConjT=Conjugate[mVectT];
F1=Sqrt[8*Pi/15]*
      Sum[Sum[A1[[i,j]]*mVectConjH[[i]]*mVectConjH[[j]],
{j,1,3}],{i,1,3}];
F2=Sqrt[8*Pi/15]*
      Sum[Sum[-A2[[i,j]]*mVectConjH[[i]]*mVectConjH[[j]],
{j,1,3}],{i,1,3}];
F3=Sqrt[8*Pi/15]*
      Sum[Sum[A3[[i,j]]*mVectConjH[[i]]*mVectConjH[[j]],
{j,1,3}],{i,1,3}];
F4=Sqrt[8*Pi/15]*
      Sum[Sum[-A4[[i,j]]*mVectConjH[[i]]*mVectConjH[[j]],
{j,1,3}],{i,1,3}];
F5=Sqrt[8*Pi/15]*
      Sum[Sum[A5[[i,j]]*mVectConjH[[i]]*mVectConjH[[j]],
{j,1,3}],{i,1,3}];
G1=Sqrt[8*Pi/15]*
      Sum[Sum[A1[[i,j]]*mVectConjL[[i]]*mVectConjL[[j]],
{j,1,3}],{i,1,3}];
G2=Sqrt[8*Pi/15]*
      Sum[Sum[-A2[[i,j]]*mVectConjL[[i]]*mVectConjL[[j]],
{j,1,3}],{i,1,3}];
```

```
G3=Sqrt[8*Pi/15]*
      Sum[Sum[A3[[i,j]]*mVectConjL[[i]]*mVectConjL[[j]],
{j,1,3}],{i,1,3}];
G4=Sqrt[8*Pi/15]*
      Sum[Sum[-A4[[i,j]]*mVectConjL[[i]]*mVectConjL[[j]],
{j,1,3}],{i,1,3}];
G5=Sqrt[8*Pi/15]*
      Sum[Sum[A5[[i,j]]*mVectConjL[[i]]*mVectConjL[[j]],
{j,1,3}],{i,1,3}];
H1=Sqrt[8*Pi/15]*
      Sum[Sum[A1[[i,j]]*mVectConjV[[i]]*mVectConjV[[j]],
{j,1,3}],{i,1,3}];
H2=Sqrt[8*Pi/15]*
      Sum[Sum[-A2[[i,j]]*mVectConjV[[i]]*mVectConjV[[j]],
{j,1,3}],{i,1,3}];
H3=Sqrt[8*Pi/15]*
      Sum[Sum[A3[[i,j]]*mVectConjV[[i]]*mVectConjV[[j]],
{j,1,3}],{i,1,3}];
H4=Sqrt[8*Pi/15]*
      Sum[Sum[-A4[[i,j]]*mVectConjV[[i]]*mVectConjV[[j]],
{j,1,3}],{i,1,3}];
H5=Sqrt[8*Pi/15]*
      Sum[Sum[A5[[i,j]]*mVectConjV[[i]]*mVectConjV[[j]],
{j,1,3}],{i,1,3}];
R1=Sqrt[8*Pi/15]*
      Sum[Sum[A1[[i,j]]*mVectConjG[[i]]*mVectConjG[[j]],
{j,1,3}],{i,1,3}];
R2=Sqrt[8*Pi/15]*
      Sum[Sum[-A2[[i,j]]*mVectConjG[[i]]*mVectConjG[[j]],
{j,1,3}],{i,1,3}];
R3=Sqrt[8*Pi/15]*
      Sum[Sum[A3[[i,j]]*mVectConjG[[i]]*mVectConjG[[j]],
{j,1,3}],{i,1,3}];
R4=Sqrt[8*Pi/15]*
      Sum[Sum[-A4[[i,j]]*mVectConjG[[i]]*mVectConjG[[j]],
{j,1,3}],{i,1,3}];
R5=Sqrt[8*Pi/15]*
      Sum[Sum[A5[[i,j]]*mVectConjG[[i]]*mVectConjG[[j]],
```

```
{j,1,3}],{i,1,3}];
S1=Sqrt[8*Pi/15]*
      Sum[Sum[A1[[i,j]]*mVectConjT[[i]]*mVectConjT[[j]],
{j,1,3}],{i,1,3}];
S2=Sqrt[8*Pi/15]*
      Sum[Sum[-A2[[i,j]]*mVectConjT[[i]]*mVectConjT[[j]],
{j,1,3}],{i,1,3}];
S3=Sqrt[8*Pi/15]*
      Sum[Sum[A3[[i,j]]*mVectConjT[[i]]*mVectConjT[[j]],
{j,1,3}],{i,1,3}];
S4=Sqrt[8*Pi/15]*
      Sum[Sum[-A4[[i,j]]*mVectConjT[[i]]*mVectConjT[[j]],
{j,1,3}],{i,1,3}];
S5=Sqrt[8*Pi/15]*
      Sum[Sum[A5[[i,j]]*mVectConjT[[i]]*mVectConjT[[j]],
{j,1,3}],{i,1,3}];
T2S0={B1,B2,B3,B4,B5};
T2S1={C1,C2,C3,C4,C5};
T2S2={D1,D2,D3,D4,D5};
T2S3={E1,E2,E3,E4,E5};
T2S4={N1,N2,N3,N4,N5};
T2S5={P1,P2,P3,P4,P5};
TmVectLS1={F1,F2,F3,F4,F5};
TmVectLS2={G1,G2,G3,G4,G5};
TmVectLS3={H1,H2,H3,H4,H5};
TmVectLS4={R1,R2,R3,R4,R5};
TmVectLS5={S1,S2,S3,S4,S5};
DD1=Sum[-I*T2S0[[n]]*(Conjugate[T2S1[[n]]]-
Conjugate[TmVectLS1[[n]]]),{n,1,
       5}];
DD2=Sum[-I*T2S0[[n]]*(Conjugate[T2S2[[n]]]-
Conjugate[TmVectLS2[[n]]]),{n,1,
       5}];
DD3=Sum[-I*T2S0[[n]]*(Conjugate[T2S3[[n]]]-
Conjugate[TmVectLS3[[n]]]),{n,1,
       5}];
DD4=Sum[-I*T2S0[[n]]*(Conjugate[T2S4[[n]]]-
Conjugate[TmVectLS4[[n]]]),{n,1,
```

```
        5}];
DD5=Sum[-I*T2S0[[n]]*(Conjugate[T2S5[[n]]]-
Conjugate[TmVectLS5[[n]]]),{n,1,
        5}];
Dplus={DD1,DD2,DD3,DD4,DD5};
d1vect=Re[Dplus];
d2vect=Im[Dplus];
d1hatvect={d1vect[[1]]/Sqrt[d1vect.d1vect],
d1vect[[2]]/Sqrt[d1vect.d1vect],
      d1vect[[3]]/Sqrt[d1vect.d1vect],d1vect[[4]]/Sqrt[d1vect.
d1vect],
      d1vect[[5]]/Sqrt[d1vect.d1vect]};
d2hatvect={d2vect[[1]]/Sqrt[d2vect.d2vect],
d2vect[[2]]/Sqrt[d2vect.d2vect],
      d2vect[[3]]/Sqrt[d2vect.d2vect],d2vect[[4]]/Sqrt[d2vect.
d2vect],
      d2vect[[5]]/Sqrt[d2vect.d2vect]};
vpvect=(d1hatvect+
          d2hatvect)/(Sqrt[((d1hatvect+d2hatvect).(d1hatvect+
d2hatvect))]);
vmvect=(d1hatvect-
          d2hatvect)/(Sqrt[((d1hatvect-d2hatvect).(d1hatvect-
d2hatvect))]);
varray=Array[vs,{numpts1+1,numpts2+1,4}];
For[i=1,i\[LessEqual]numpts1+1,i++,theta=(Pi/numpts1)*(i-1);
  thetadegrees=theta*(180/Pi);
  For[j=1,j\[LessEqual]numpts2+1,j++,phi=(2*Pi/numpts2)*(j-1);
    value1=vpvect[[1]];value2=vmvect[[1]];phidegrees=phi*
(180/Pi);
    varray[[i,j]]={thetadegrees,phidegrees,value1,value2};
    Clear[phi,value1,value2,phidegrees]];Clear[theta,
thetadegrees]]
flattenedvarray=Flatten[varray,1];
For[i=1,i\[LessEqual](numpts1+1)*(numpts2+1),i++,
  Write["HanfordBeamCoeff.dat",flattenedvarray[[i]]]]
varray2=Array[vs2,{numpts1+1,numpts2+1,4}];
Clear[value1,value2];
For[i=1,i\[LessEqual]numpts1+1,i++,theta=(Pi/numpts1)*(i-1);
```

```
  thetadegrees=theta*(180/Pi);
  For[j=1,j\[LessEqual]numpts2+1,j++,phi=(2*Pi/numpts2)*(j-1);
    value1=vpvect[[2]];value2=vmvect[[2]];phidegrees=phi*
(180/Pi);
    varray2[[i,j]]={thetadegrees,phidegrees,value1,value2};
    Clear[phi,value1,value2,phidegrees]];Clear[theta,
thetadegrees]]
flattenedvarray2=Flatten[varray2,1];
For[i=1,i\[LessEqual](numpts1+1)*(numpts2+1),i++,
  Write["LivingstonBeamCoeff.dat",flattenedvarray2[[i]]]]
varray3=Array[vs3,{numpts1+1,numpts2+1,4}];
Clear[value1,value2];
For[i=1,i\[LessEqual]numpts1+1,i++,theta=(Pi/numpts1)*(i-1);
  thetadegrees=theta*(180/Pi);
  For[j=1,j\[LessEqual]numpts2+1,j++,phi=(2*Pi/numpts2)*(j-1);
    value1=vpvect[[3]];value2=vmvect[[3]];phidegrees=phi*
(180/Pi);
    varray3[[i,j]]={thetadegrees,phidegrees,value1,value2};
    Clear[phi,value1,value2,phidegrees]];Clear[theta,
thetadegrees]]
flattenedvarray3=Flatten[varray3,1];
For[i=1,i\[LessEqual](numpts1+1)*(numpts2+1),i++,
  Write["VirgoBeamCoeff.dat",flattenedvarray3[[i]]]]
varray4=Array[vs4,{numpts1+1,numpts2+1,4}];
Clear[value1,value2];
For[i=1,i\[LessEqual]numpts1+1,i++,theta=(Pi/numpts1)*(i-1);
  thetadegrees=theta*(180/Pi);
  For[j=1,j\[LessEqual]numpts2+1,j++,phi=(2*Pi/numpts2)*(j-1);
    value1=vpvect[[4]];value2=vmvect[[4]];phidegrees=phi*
(180/Pi);
    varray4[[i,j]]={thetadegrees,phidegrees,value1,value2};
    Clear[phi,value1,value2,phidegrees]];Clear[theta,
thetadegrees]]
flattenedvarray4=Flatten[varray4,1];
For[i=1,i\[LessEqual](numpts1+1)*(numpts2+1),i++,
  Write["GeoBeamCoeff.dat",flattenedvarray4[[i]]]]
varray5=Array[vs5,{numpts1+1,numpts2+1,4}];
Clear[value1,value2];
```

```
For[i=1,i\[LessEqual]numpts1+1,i++,theta=(Pi/numpts1)*(i-1);
  thetadegrees=theta*(180/Pi);
  For[j=1,j\[LessEqual]numpts2+1,j++,phi=(2*Pi/numpts2)*(j-1);
    value1=vpvect[[5]];value2=vmvect[[5]];phidegrees=phi*
(180/Pi);
    varray5[[i,j]]={thetadegrees,phidegrees,value1,value2};
    Clear[phi,value1,value2,phidegrees]];Clear[theta,
thetadegrees]]
flattenedvarray5=Flatten[varray5,1];
For[i=1,i\[LessEqual](numpts1+1)*(numpts2+1),i++,
  Write["TamaBeamCoeff.dat",flattenedvarray5[[i]]]]
```

Appendix C

CoherentInspiral.h

```
/*-----------------------------------------------------------------------
 *
 * File Name: CoherentInspiral.h
 *
 * Author: Bose, S., Seader, S. E.
 *
 * Revision: $Id: CoherentInspiral.h,v 1.11 2005/07/26 16:20:59 sseader Exp $
 *
 *-----------------------------------------------------------------------
 */

#if 0
<lalVerbatim file="CoherentInspiralHV">
Author: Bose, S., Seader, S. E.
$Id: CoherentInspiral.h,v 1.11 2005/07/26 16:20:59 sseader Exp $
</lalVerbatim>
#endif

#ifndef _COHERENTINSPIRALH_H
#define _COHERENTINSPIRALH_H

#include <lal/LALRCSID.h>
#include <lal/LALStdlib.h>
#include <lal/LALStdio.h>
#include <lal/LALConstants.h>
#include <lal/AVFactories.h>
#include <lal/LALDatatypes.h>
#include <lal/DataBuffer.h>
#include <lal/LIGOMetadataTables.h>
#include <lal/LALInspiral.h>
#include <lal/FindChirp.h>
#include <lal/LALInspiralBank.h>

#ifdef  __cplusplus
extern "C" {
#pragma }
#endif
```

```
NRCSID (COHERENTINSPIRALH, "$Id: CoherentInspiral.h,v 1.11 2005/07/26 16:20:59
sseader Exp $");

#if 0
#endif
/* <lalErrTable> */
#define COHERENTINSPIRALH_ENULL 1
#define COHERENTINSPIRALH_ENNUL 2
#define COHERENTINSPIRALH_EALOC 3
#define COHERENTINSPIRALH_ENUMZ 4
#define COHERENTINSPIRALH_ESEGZ 5
#define COHERENTINSPIRALH_ECHIZ 6
#define COHERENTINSPIRALH_EDTZO 7
#define COHERENTINSPIRALH_EFREE 8
#define COHERENTINSPIRALH_ERHOT 9
#define COHERENTINSPIRALH_ECHIT 10
#define COHERENTINSPIRALH_ESMSM 11
#define COHERENTINSPIRALH_EZDET 12
#define COHERENTINSPIRALH_MSGENULL "Null pointer"
#define COHERENTINSPIRALH_MSGENNUL "Non-null pointer"
#define COHERENTINSPIRALH_MSGEALOC "Memory allocation error"
#define COHERENTINSPIRALH_MSGENUMZ "Invalid number of points in segment"
#define COHERENTINSPIRALH_MSGESEGZ "Invalid number of segments"
#define COHERENTINSPIRALH_MSGECHIZ "Invalid number of chi squared bins"
#define COHERENTINSPIRALH_MSGEDTZO "deltaT is zero or negative"
#define COHERENTINSPIRALH_MSGEFREE "Error freeing memory"
#define COHERENTINSPIRALH_MSGERHOT "coherentSNR threshold is negative"
#define COHERENTINSPIRALH_MSGECHIT "Chisq threshold is negative"
#define COHERENTINSPIRALH_MSGESMSM "Size mismatch between vectors"
#define COHERENTINSPIRALH_MSGEZDET "Number of detectors should be greater than 1 and less than 5"
/* </lalErrTable> */



/* --- structure for describing a binary insipral event ------------------ */
/* <lalVerbatim file="FindChirpHInspiralEvent"> */


/* </lalVerbatim> */
#if 0


/* --- parameter structure for the coherent inspiral filtering function ---- */
/* <lalVerbatim file="CoherentInspiralHCoherentInspiralFilterParams"> */
typedef struct
tagCoherentInspiralInitParams
{
  UINT4                         numDetectors;
  UINT4                         numSegments;
  UINT4                         numPoints;
  UINT4                         numBeamPoints;
  UINT4                         cohSNROut;
}
CoherentInspiralInitParams;
/* </lalVerbatim> */
```

```
typedef struct
tagDetectorVector
{
  UINT4                   numDetectors;
  LALDetector            *detector;
}
DetectorVector;


typedef struct
tagDetectorBeamArray
{
  UINT4                    numBeamPoints;
  REAL4TimeSeries         *thetaPhiVs;/* 4D array: theta,phi,v+,v- */
}
DetectorBeamArray;


typedef struct
tagCoherentInspiralBeamVector
{
  UINT4                   numDetectors;
  DetectorBeamArray      *detBeamArray;
}
CoherentInspiralBeamVector;


typedef struct
tagCoherentInspiralFilterParams
{
  INT4                          numTmplts;
  UINT4                         maximizeOverChirp;
  UINT4                         numDetectors;
  UINT4                         numSegments;
  UINT4                         numPoints;
  UINT4                         numBeamPoints;
  REAL4                         fLow;
  REAL4                         deltaT;
  REAL4                         cohSNRThresh;
  REAL4                         segNorm[4];
  REAL4                         templateNorm;
  INT4                          segmentLength; /* time points */
  UINT4                         cohSNROut;
  UINT2Vector                  *detIDVec; /* Note: H1, H2 are from same site,different detectors */
  DetectorVector               *detectorVec; /*stores detectors' site info */
  REAL4TimeSeries              *cohSNRVec;
}
CoherentInspiralFilterParams;


/* --- input to the CoherentInspiral filtering functions --------- */
/* <lalVerbatim file="CoherentInspiralHCoherentInspiralCVector"> */
typedef struct
tagCoherentInspiralCVector
{
  UINT4                   numDetectors;
  COMPLEX8TimeSeries     *cData;
```

```
}
CoherentInspiralCVector;
/* </lalVerbatim> */


/* <lalVerbatim file="CoherentInspiralHCoherentInspiralFilterInput"> */
typedef struct
tagCoherentInspiralFilterInput
{
  InspiralTemplate            *tmplt;
  CoherentInspiralBeamVector  *beamVec;
  CoherentInspiralCVector     *multiCData;
}
CoherentInspiralFilterInput;
/* </lalVerbatim> */



/*
 *
 * function prototypes for memory management functions
 *
 */

void
LALCoherentInspiralFilterInputInit (
    LALStatus                      *status,
    CoherentInspiralFilterInput    **input,
    CoherentInspiralInitParams      *params
    );

void
LALCoherentInspiralFilterInputFinalize (
    LALStatus                      *status,
    CoherentInspiralFilterInput    **input
    );

void
LALCoherentInspiralFilterParamsInit (
    LALStatus                      *status,
    CoherentInspiralFilterParams   **output,
    CoherentInspiralInitParams      *params
    );

void
LALCoherentInspiralFilterParamsFinalize (
    LALStatus                      *status,
    CoherentInspiralFilterParams   **output
    );

/*
 *
 * function prototypes for coherent inspiral filter function
 *
 */
```

```
void
LALCoherentInspiralEstimatePsiEpsilonCoaPhase (
    LALStatus                           *status,
    INT4                                 caseID[6],
    REAL4                                segNorm[4],
    REAL4                                theta,
    REAL4                                phi,
    COMPLEX8                             cData[4],
    REAL4                               *inclination,
    REAL4                               *polarization,
    REAL4                               *coaPhase
    );

void
LALCoherentInspiralEstimateDistance (
    LALStatus                           *status,
    REAL4                                segNorm[4],
    REAL4                                templateNorm,
    REAL4                                deltaT,
    INT4                                 segmentLength,  /* time pts */
    REAL4                                coherentSNR,
    REAL4                               *distance
    );

void
LALCoherentInspiralFilterSegment (
    LALStatus                           *status,
    MultiInspiralTable                  **eventList,
    CoherentInspiralFilterInput         *input,
    CoherentInspiralFilterParams        *params
    );



#ifdef  __cplusplus
#pragma {
}
#endif

#endif /* _COHERENTINSPIRALH_H */
```

Appendix D

CoherentInspiralInput.c

```
/*-----------------------------------------------------------------------
 *
 * File Name: CoherentInspiralInput.c
 *
 * Author: Seader, S. E.  Brown, D.
 *
 * Revision: $Id: CoherentInspiralInput.c,v 1.7 2005/10/10 18:18:22 sseader Exp $
 *
 *-----------------------------------------------------------------------
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#include <time.h>
#include <math.h>

#include <lal/LALRCSID.h>
#include <lal/LALConfig.h>
#include <lal/LALStdio.h>
#include <lal/LALStdlib.h>
#include <lal/LALError.h>
#include <lal/LALDatatypes.h>
#include <lal/AVFactories.h>
#include <lal/LALConstants.h>
#include <lal/DataBuffer.h>
#include <lal/LIGOMetadataTables.h>
#include <lal/LIGOMetadataUtils.h>
#include <lal/Date.h>
#include <lal/Units.h>
#include <lal/FindChirp.h>
#include <lal/FindChirpSP.h>
#include <lal/FindChirpBCV.h>
```

```
#include <lal/FindChirpBCVSpin.h>
#include <lal/FindChirpChisq.h>
#include <lal/StochasticCrossCorrelation.h>
#include <lal/DetectorSite.h>
#include <lal/Random.h>
#include <lal/LALInspiral.h>
#include <lal/CoherentInspiral.h>
#include <lal/LALStatusMacros.h>

NRCSID( COHERENTINSPIRALINPUTC, "$Id: CoherentInspiralInput.c,v 1.7 2005/10/10 18:18:22
sseader Exp $");

#define rint(x) (floor((x)+0.5))
double modf( double value, double *integerPart );

void
LALFindChirpCreateCoherentInput(
     LALStatus                  *status,
     COMPLEX8TimeSeries         **coherentInputData,
     COMPLEX8TimeSeries         *input,
     SnglInspiralTable          *templt,
     REAL4                      coherentSegmentLength,/*in seconds*/
     INT4                       corruptedDataLength /*in timepoints */
     )
{
  COMPLEX8TimeSeries      *cohInputData = NULL;
  LIGOTimeGPS              end_time;
  UINT8                    eventID = 0;
  INT4                     numPoints = 0;
  REAL4                    cohSegLength = 0.0;
  INT4                     inputEpochSeconds = 0;
  INT4                     inputEpochNanoSeconds = 0;
  REAL8                    deltaT = 0.0;
  REAL8                    intpart = 0.0;
  REAL8                    fracpart = 0.0;
  REAL8                    tempTime = 0.0;
  INT4                     crupDataLength = 0;
  INT4                     cohSegEnd = 0;
  INT4                     cohSegStart = 0;
  INT4                     nonCorruptEnd = 0;
  INT4                     nonCorruptStart = 0;
  INT4                     overlap         = 0;
  INT4                     fullCohSegLength = 0;
  INT4                     eventTimePoint = 0;

  INITSTATUS( status, "LALFindChirpCreateCoherentInput",
      COHERENTINSPIRALINPUTC );
  ATTATCHSTATUSPTR( status );

  /* Ensure that arguments are reasonable */

  /* check that the output handle exists, but points to a null pointer */
  ASSERT( coherentInputData, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );
  ASSERT( !*coherentInputData, status, FINDCHIRPH_ENNUL, FINDCHIRPH_MSGENNUL );

  /* check that a valid COMPLEX8TimeSeries is input */
  ASSERT( input, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );
```

```
  ASSERT( input->data, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );
  ASSERT( input->data->data, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );
  ASSERT( input->deltaT > 0, status, FINDCHIRPH_EDTZO, FINDCHIRPH_MSGEDTZO );
  ASSERT( input->epoch.gpsSeconds > 0, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );
  ASSERT( input->epoch.gpsNanoSeconds >= 0, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );
  ASSERT( input->data->length, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );

  /* check that a valid snglInspiralTable is input */
  ASSERT( templt, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );
  ASSERT( templt->end_time.gpsSeconds > 0, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );
  ASSERT( templt->end_time.gpsNanoSeconds >= 0, status, FINDCHIRPH_ENULL, FINDCHIRPH_MSGENULL );

  /* check for valid lengths */
  ASSERT( coherentSegmentLength > 0, status, FINDCHIRPH_ENUMZ, FINDCHIRPH_MSGENUMZ );
  ASSERT( corruptedDataLength >= 0, status, FINDCHIRPH_ENUMZ, FINDCHIRPH_MSGENUMZ );


  /* Get necessary info from input structures */

  end_time = templt->end_time;
  eventID  = templt->event_id->id;
  numPoints = input->data->length;
  cohSegLength = coherentSegmentLength;
  inputEpochSeconds = input->epoch.gpsSeconds;
  inputEpochNanoSeconds = input->epoch.gpsNanoSeconds;
  deltaT = input->deltaT;
  crupDataLength = corruptedDataLength;

  /* Now determine if the event lies in corrupted data */

  nonCorruptStart = crupDataLength;
  nonCorruptEnd = numPoints - crupDataLength;
  cohSegStart = (INT4) rint( ((end_time.gpsSeconds + end_time.gpsNanoSeconds*1.0e-9) -
(inputEpochSeconds + inputEpochNanoSeconds*1.0e-9)-cohSegLength)/deltaT - 1.0);
  cohSegEnd = cohSegStart + 2* (INT4)cohSegLength/deltaT + 1;
  eventTimePoint = (INT4) rint( ((end_time.gpsSeconds + end_time.gpsNanoSeconds*1.0e-9) -
(inputEpochSeconds + inputEpochNanoSeconds*1.0e-9))/deltaT - 1.0);

  if( eventTimePoint < nonCorruptEnd && eventTimePoint >= nonCorruptStart )
    {
      /* The coherent chunk is outside of the corrupted data */

      cohInputData = *coherentInputData = (COMPLEX8TimeSeries *)
LALCalloc(1, sizeof(COMPLEX8TimeSeries) );

      fullCohSegLength = 2*cohSegLength/deltaT;

      LALCCreateVector(status->statusPtr, &(cohInputData->data), fullCohSegLength);
      CHECKSTATUSPTR( status );
      /* store C-data snippet and its associated info */
      memcpy(cohInputData->name, input->name, LALNameLength * sizeof(CHAR) );
      memcpy(cohInputData->data->data, &(input->data->data[cohSegStart]), fullCohSegLength *
sizeof(COMPLEX8) );
      cohInputData->deltaT = deltaT;
      tempTime = inputEpochSeconds + inputEpochNanoSeconds*1.0e-9 + cohSegStart * deltaT;
      fracpart = modf(tempTime, &intpart);
      cohInputData->epoch.gpsSeconds = (INT4) intpart;
```

```
      cohInputData->epoch.gpsNanoSeconds = (INT4) (fracpart*1.0e9);
    }
  else
    {
      /* return a null pointer cohInputData */
    }
  /* normal exit */
  DETATCHSTATUSPTR( status );
  RETURN( status );
}
```

Appendix E

CoherentInspiralFilter.c

```
/*-----------------------------------------------------------------------
 *
 * File Name: CoherentInspiralFilter.c
 *
 * Author: Bose, S., Seader, S. E.
 *
 * Revision: $Id: CoherentInspiralFilter.c,v 1.13 2005/08/29 16:54:00 sseader Exp $
 *
 *-----------------------------------------------------------------------
 */

#include <math.h>
#include <string.h>

#include <lal/LALRCSID.h>
#include <lal/LALConfig.h>
#include <lal/LALStdlib.h>
#include <lal/LALStdio.h>
#include <lal/LALConstants.h>
#include <lal/AVFactories.h>
#include <lal/DataBuffer.h>
#include <lal/FindChirp.h>
#include <lal/FindChirpSP.h>
#include <lal/DetectorSite.h>
#include <lal/StochasticCrossCorrelation.h>
#include <lal/LIGOMetadataTables.h>
#include <lal/SkyCoordinates.h>
#include <lal/Date.h>
#include <lal/CoherentInspiral.h>

#define rint(x) (floor((x)+0.5))
double modf( double value, double *integerPart );

NRCSID (COHERENTINSPIRALFILTERC, "$Id: CoherentInspiralFilter.c,v 1.13 2005/08/29 16:54:00
sseader Exp $");


static REAL4 cartesianInnerProduct(REAL4 x[3], REAL4 y[3])
```

```
{
  return x[0]*y[0] + x[1]*y[1] + x[2]*y[2];
}

void
LALCoherentInspiralFilterInputInit (
    LALStatus                       *status,
    CoherentInspiralFilterInput    **input,
    CoherentInspiralInitParams      *params
    )
{
  UINT4                            i, l;
  UINT4                            length = 4; /*length of thetaPhiVs vectors*/
  CoherentInspiralFilterInput     *inputPtr = NULL;
  CoherentInspiralBeamVector      *beamVecPtr = NULL;
  DetectorBeamArray               *detBeamArray = NULL;
  CoherentInspiralCVector         *cVecPtr = NULL;

  INITSTATUS( status, "LALCoherentInspiralFilterInputInit",
      COHERENTINSPIRALFILTERC );
  ATTATCHSTATUSPTR( status );

  /*
   *
   * ensure that the arguments are reasonable
   *
   */

  /* check that the output handle exists, but points to a null pointer */
  ASSERT( input, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );
  ASSERT( !*input, status, COHERENTINSPIRALH_ENNUL, COHERENTINSPIRALH_MSGENNUL );

  /* check that the parameter structure exists */
  ASSERT( params, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );

  /* check that the number of detectors is greater than 1 */
  ASSERT( params->numDetectors > 1, status,
      COHERENTINSPIRALH_EZDET, COHERENTINSPIRALH_MSGEZDET );

  /* check that the number of detectors is less than 5 */
  ASSERT( params->numDetectors < 5, status,
  COHERENTINSPIRALH_EZDET, COHERENTINSPIRALH_MSGEZDET );

  /* check that the number of data segments is positive */
  ASSERT( params->numSegments > 0, status,
      COHERENTINSPIRALH_ESEGZ, COHERENTINSPIRALH_MSGESEGZ );

  /* check that the number of points is positive */
  ASSERT( params->numPoints > 0, status,
      COHERENTINSPIRALH_ENUMZ, COHERENTINSPIRALH_MSGENUMZ );


  inputPtr= *input = (CoherentInspiralFilterInput *)
    LALCalloc(1, sizeof(CoherentInspiralFilterInput) );
  if ( !inputPtr )
    {
      ABORT( status, COHERENTINSPIRALH_EALOC, COHERENTINSPIRALH_MSGEALOC);
```

```
    }


  /*
   *
   * create the CoherentInspiralBeamVector structure if needed
   *
   */


  if( params->numBeamPoints != 0 )
    {
      /* allocate memory for an array  */
      beamVecPtr = inputPtr->beamVec = (CoherentInspiralBeamVector *)
LALCalloc(1, sizeof(CoherentInspiralBeamVector));

      if ( !beamVecPtr )
{
  LALFree( inputPtr);
  *input = NULL;
  ABORT( status, COHERENTINSPIRALH_EALOC, COHERENTINSPIRALH_MSGEALOC);
}

      /* set the number of detectors in the vector */
      beamVecPtr->numDetectors = params->numDetectors;

      detBeamArray = beamVecPtr->detBeamArray = (DetectorBeamArray *)
LALCalloc(1, beamVecPtr->numDetectors*sizeof(DetectorBeamArray));
      if ( !detBeamArray )
{
  LALFree( beamVecPtr );
  beamVecPtr = NULL;
  inputPtr->beamVec = NULL;
  LALFree( inputPtr );
  *input = NULL;
  ABORT( status, COHERENTINSPIRALH_EALOC, COHERENTINSPIRALH_MSGEALOC);
}


      /* set the number of beamPoints in the vector */
      for ( l = 0 ; l < beamVecPtr->numDetectors ; l++) {
detBeamArray[l].numBeamPoints = params->numBeamPoints;

detBeamArray[l].thetaPhiVs = (REAL4TimeSeries *)
  LALCalloc(1, params->numBeamPoints*sizeof(REAL4TimeSeries));

BEGINFAIL(status) {
  LALFree( detBeamArray );
  beamVecPtr->detBeamArray = NULL;
  LALFree( beamVecPtr );
  inputPtr->beamVec = NULL;
  LALFree( inputPtr );
  *input = NULL;
}
ENDFAIL( status );
      }
```

```
      /* set the number of fields in the thetaPhiVs to 4: theta,phi,v+,and v-*/
      for ( l = 0 ; l < beamVecPtr->numDetectors ; l++) {
for ( i = 0 ; i < detBeamArray[l].numBeamPoints ; i++ ) {
  LALCreateVector(status->statusPtr, &(detBeamArray[l].thetaPhiVs[i].data), length);

  BEGINFAIL( status ) {
    for( l = 0; l < beamVecPtr->numDetectors; l++) {
      LALFree( detBeamArray[l].thetaPhiVs );
    }
    LALFree( detBeamArray );
    beamVecPtr->detBeamArray = NULL;
    LALFree( beamVecPtr );
    inputPtr->beamVec = NULL;
    LALFree( inputPtr );
    *input = NULL;
  }
  ENDFAIL( status );
}
      }

    }
  /*
   *
   * create the CoherentInspiralCVector structure
   *
   */

  /* allocate memory for the cData vector  */

      cVecPtr = (*input)->multiCData = (CoherentInspiralCVector *)
LALCalloc(1, sizeof(CoherentInspiralCVector));
      if ( !cVecPtr )
{
  if (params->numBeamPoints != 0)
    {
      for ( l = 0 ; l < beamVecPtr->numDetectors ; l++) {
for ( i = 0 ; i < detBeamArray[l].numBeamPoints ; i++ ) {
  TRY( LALDestroyVector(status->statusPtr,
&(detBeamArray[l].thetaPhiVs[i].data)), status);
}
      }
      for( l = 0; l < beamVecPtr->numDetectors; l++) {
LALFree( detBeamArray[l].thetaPhiVs );
      }
      LALFree( detBeamArray );
      beamVecPtr->detBeamArray = NULL;
      LALFree( beamVecPtr );
      inputPtr->beamVec = NULL;
    }
          LALFree( inputPtr );
          *input = NULL;
          ABORT( status, COHERENTINSPIRALH_EALOC, COHERENTINSPIRALH_MSGEALOC);
        }

      /* set the number of detectors in the vector */
      cVecPtr->numDetectors = params->numDetectors;
```

```
      cVecPtr->cData = (COMPLEX8TimeSeries *)
        LALCalloc(1, cVecPtr->numDetectors*sizeof(COMPLEX8TimeSeries));
      if ( !(cVecPtr->cData) )
        {
          LALFree( cVecPtr );
          (*input)->multiCData = NULL;
          if( params->numBeamPoints != 0)
    {
      for ( l = 0 ; l < beamVecPtr->numDetectors ; l++) {
        for ( i = 0 ; i < detBeamArray[l].numBeamPoints ; i++ ) {
          TRY( LALDestroyVector(status->statusPtr,
&(detBeamArray[l].thetaPhiVs[i].data)), status);
        }
      }
      for( l = 0; l < beamVecPtr->numDetectors; l++) {
        LALFree( detBeamArray[l].thetaPhiVs );
      }
      LALFree( detBeamArray );
      beamVecPtr->detBeamArray = NULL;
      LALFree( beamVecPtr );
      inputPtr->beamVec = NULL;
    }
          LALFree( inputPtr );
          *input = NULL;
          ABORT( status, COHERENTINSPIRALH_EALOC, COHERENTINSPIRALH_MSGEALOC);
        }

      for ( i = 0 ; i < cVecPtr->numDetectors ; i++ ) {
        LALCCreateVector(status->statusPtr, &(cVecPtr->cData[i].data),
 params->numPoints);
        BEGINFAIL( status ) {
          LALFree( cVecPtr->cData );
          LALFree( cVecPtr );
          (*input)->multiCData = NULL;
          if( params->numBeamPoints != 0)
    {
      for ( l = 0 ; l < beamVecPtr->numDetectors ; l++) {
        for ( i = 0 ; i < detBeamArray[l].numBeamPoints ; i++ ) {
          TRY( LALDestroyVector(status->statusPtr,
&(detBeamArray[l].thetaPhiVs[i].data)), status);
        }
      }
      for( l = 0; l < beamVecPtr->numDetectors; l++) {
        LALFree( detBeamArray[l].thetaPhiVs );
      }
      LALFree( detBeamArray );
      beamVecPtr->detBeamArray = NULL;
      LALFree( beamVecPtr );
      inputPtr->beamVec = NULL;
    }
          LALFree( inputPtr );
          *input = NULL;
        }
        ENDFAIL( status );
      }
```

```
  BEGINFAIL( status ) {
    for ( i = 0 ; i < cVecPtr->numDetectors ; i++ ) {
    TRY( LALCDestroyVector(status->statusPtr, &(cVecPtr->cData[i].data)),
 status);
    }
    LALFree( cVecPtr->cData );
    LALFree( cVecPtr );
    (*input)->multiCData = NULL;
    if( params->numBeamPoints != 0)
      {
for ( l = 0 ; l < beamVecPtr->numDetectors ; l++) {
  for ( i = 0 ; i < detBeamArray[l].numBeamPoints ; i++ ) {
    TRY( LALDestroyVector(status->statusPtr,
&(detBeamArray[l].thetaPhiVs[i].data)), status);
  }
}
for( l = 0; l < beamVecPtr->numDetectors; l++) {
  LALFree( detBeamArray[l].thetaPhiVs );
}
LALFree( detBeamArray );
beamVecPtr->detBeamArray = NULL;
LALFree( beamVecPtr );
inputPtr->beamVec = NULL;
      }
    LALFree( inputPtr );
    *input = NULL;
    }
    ENDFAIL( status );

  /* normal exit */
  DETATCHSTATUSPTR( status );
  RETURN( status );
}

void
LALCoherentInspiralFilterInputFinalize (
    LALStatus                       *status,
    CoherentInspiralFilterInput    **input
    )
{
  UINT4                            i,l;
  CoherentInspiralBeamVector      *beamVecPtr;
  DetectorBeamArray               *detBeamArray;
  CoherentInspiralFilterInput     *inputPtr;
  CoherentInspiralCVector         *cVecPtr;

  INITSTATUS( status, "LALCoherentInspiralFilterInputFinalize",
      COHERENTINSPIRALFILTERC );
  ATTATCHSTATUSPTR( status );


  /*
   * check that the arguments are reasonable
   *
   */

  ASSERT( input, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );
```

```
  ASSERT( *input, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );


  inputPtr= *input;

  /*
   *
   * destroy the contents of the CoherentInspiralCVector structure if necessary
   *
   */

  if( inputPtr->multiCData )
    {
      cVecPtr = (*input)->multiCData;

      for ( l = 0 ; l < cVecPtr->numDetectors ; l++ ) {
if (cVecPtr->cData[l].data != NULL ) {
  LALCDestroyVector( status->statusPtr, &(cVecPtr->cData[l].data) );
  CHECKSTATUSPTR( status );
}
      }

      LALFree( cVecPtr->cData );
      LALFree( cVecPtr );
      (*input)->multiCData = NULL;

    }
  /*
   *
   * destroy the contents of the CoherentInspiralBeamVector structure if needed
   *
   */

  if( inputPtr->beamVec )
    {
      beamVecPtr = inputPtr->beamVec;
      detBeamArray = beamVecPtr->detBeamArray;

      /* destroy vector  */
      if ( beamVecPtr != NULL ) {
for ( l = 0 ; l < beamVecPtr->numDetectors ; l++ ) {
  for  ( i = 0 ; i < detBeamArray[l].numBeamPoints ; i++) {
    if (inputPtr->beamVec->detBeamArray[l].thetaPhiVs != NULL ) {
      LALDestroyVector (status->statusPtr, &(detBeamArray[l].thetaPhiVs[i].data));
      CHECKSTATUSPTR( status );
    }
  }
  LALFree(detBeamArray[l].thetaPhiVs);
  detBeamArray[l].thetaPhiVs = NULL;
}
  }

      LALFree(detBeamArray);
      beamVecPtr->detBeamArray = NULL;
      LALFree(beamVecPtr);
      inputPtr->beamVec = NULL;
    }
```

```
  LALFree( inputPtr );
  *input = NULL;

  /* normal exit */
  DETATCHSTATUSPTR( status );
  RETURN( status );

}


void
LALCoherentInspiralFilterParamsInit (
    LALStatus                       *status,
    CoherentInspiralFilterParams   **output,
    CoherentInspiralInitParams      *params
    )
{
  CoherentInspiralFilterParams     *outputPtr;
  INT4                              networkLength = 6; /*CHECK: hardwired
 to 6 for now*/

  INITSTATUS( status, "LALCoherentInspiralFilterParamsInit",
      COHERENTINSPIRALFILTERC );
  ATTATCHSTATUSPTR( status );



  /*
   *
   * ensure that the arguments are reasonable
   *
   */


  /* check that the output handle exists, but points to a null pointer */
  ASSERT( output, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );
  ASSERT( !*output, status, COHERENTINSPIRALH_ENNUL, COHERENTINSPIRALH_MSGENNUL );

  /* check that the parameter structure exists */
  ASSERT( params, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );
 /* check that the number of detectors is greater than 1 */
  ASSERT( params->numDetectors > 1, status,
      COHERENTINSPIRALH_EZDET, COHERENTINSPIRALH_MSGEZDET );

  /* check that the number of detectors is less than 5 */
  ASSERT( params->numDetectors < 5, status,
  COHERENTINSPIRALH_EZDET, COHERENTINSPIRALH_MSGEZDET );

  /* check that the number of data segments is positive */
  ASSERT( params->numSegments > 0, status,
      COHERENTINSPIRALH_ESEGZ, COHERENTINSPIRALH_MSGESEGZ );

  /* check that the number of points is positive */
  ASSERT( params->numPoints > 0, status,
      COHERENTINSPIRALH_ENUMZ, COHERENTINSPIRALH_MSGENUMZ );
```

```
/*
 *
 * allocate memory for the FindChirpFilterParams
 *
 */


/* create the output structure */
outputPtr= *output = (CoherentInspiralFilterParams *)
  LALCalloc(1, sizeof(CoherentInspiralFilterParams) );
if (!outputPtr )
  {
    ABORT( status, COHERENTINSPIRALH_EALOC, COHERENTINSPIRALH_MSGEALOC);
  }

outputPtr->numDetectors = params->numDetectors;
outputPtr->numSegments = params->numSegments;
outputPtr->numPoints = params->numPoints;
outputPtr->numBeamPoints = params->numBeamPoints;

LALU2CreateVector(status->statusPtr, &(outputPtr->detIDVec),
   networkLength);
BEGINFAIL( status ) {
  LALFree( outputPtr );
  *output = NULL;
}
ENDFAIL( status );



/*
 *
 * create vector to store coherent SNR if outputting it
 *
 */

if( params->cohSNROut )
  {
      outputPtr->cohSNRVec = (REAL4TimeSeries *)
        LALCalloc( 1, sizeof(REAL4TimeSeries) );
      LALCreateVector (status->statusPtr, &(outputPtr->cohSNRVec->data),
       params->numPoints);
      BEGINFAIL( status ) {
        TRY( LALU2DestroyVector (status->statusPtr, &(outputPtr->detIDVec)),
     status);
        LALFree( outputPtr->cohSNRVec );
        outputPtr->cohSNRVec = NULL;
        LALFree( outputPtr->detectorVec );
        outputPtr->detectorVec = NULL;
        LALFree( outputPtr );
        *output = NULL;
      }
      ENDFAIL( status );
  }

/*
```

```
   *
   * create detector vector to store detector site IDs
   *
   */


  outputPtr->detectorVec = (DetectorVector *)
    LALCalloc( 1, sizeof(DetectorVector) );

  if ( !(outputPtr->detectorVec) ) {
    ABORT( status, COHERENTINSPIRALH_EALOC, COHERENTINSPIRALH_MSGEALOC);
    BEGINFAIL( status ) {
      if( params->cohSNROut )
{
          TRY( LALDestroyVector( status->statusPtr, &(outputPtr->cohSNRVec->data)),
       status);
          LALFree( outputPtr->cohSNRVec );
          outputPtr->cohSNRVec = NULL;
}
      TRY( LALU2DestroyVector( status->statusPtr, &(outputPtr->detIDVec)),
   status);
      LALFree( outputPtr->detectorVec );
      outputPtr->detectorVec = NULL;
      LALFree( outputPtr );
      *output = NULL;
    }
    ENDFAIL( status );
  }

  outputPtr->detectorVec->detector = (LALDetector *)
    LALCalloc( 1, outputPtr->numDetectors*sizeof(LALDetector));

  if ( !(outputPtr->detectorVec->detector ) ) {
    ABORT( status, COHERENTINSPIRALH_EALOC, COHERENTINSPIRALH_MSGEALOC);
    BEGINFAIL( status ) {
      LALFree( outputPtr->detectorVec );
      if( params->cohSNROut )
{
          TRY( LALDestroyVector( status->statusPtr, &(outputPtr->cohSNRVec->data)),
       status);
          LALFree( outputPtr->cohSNRVec );
          outputPtr->cohSNRVec = NULL;
}
      TRY( LALU2DestroyVector( status->statusPtr, &(outputPtr->detIDVec)),
   status);
      LALFree( outputPtr->detectorVec );
      outputPtr->detectorVec = NULL;
      LALFree( outputPtr );
      *output = NULL;
    }
    ENDFAIL( status );
  }


  /* normal exit */
  DETATCHSTATUSPTR( status );
  RETURN( status );
```

```
}


void
LALCoherentInspiralFilterParamsFinalize (
    LALStatus                       *status,
    CoherentInspiralFilterParams   **output
    )
{
  CoherentInspiralFilterParams     *outputPtr;

  INITSTATUS( status, "LALCoherentInspiralFilterParamsInit",
      COHERENTINSPIRALFILTERC );
  ATTATCHSTATUSPTR( status );

  /*
   *
   * ensure that the arguments are reasonable
   *
   */


  /* check that the output handle exists, but points to a non-null pointer */
  ASSERT( output, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );
  ASSERT( *output, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );

  /* local pointer to output structure */
  outputPtr = *output;


  /* destroy detector vector */
  LALFree( outputPtr->detectorVec->detector );
  outputPtr->detectorVec->detector = NULL;
  LALFree( outputPtr->detectorVec );
  outputPtr->detectorVec = NULL;

  LALU2DestroyVector(status->statusPtr, &(outputPtr->detIDVec) );
  CHECKSTATUSPTR( status );

  /*
   *
   * destroy coherent SNR vector, if it exists
   *
   */
  if ( outputPtr->cohSNRVec ) {
    LALDestroyVector( status->statusPtr, &(outputPtr->cohSNRVec->data) );
    CHECKSTATUSPTR( status );
    LALFree( outputPtr->cohSNRVec );
    outputPtr->cohSNRVec = NULL;
  }

  LALFree( outputPtr );
  *output = NULL;
```

```
  /* normal exit */
  DETATCHSTATUSPTR( status );
  RETURN( status );

}

void
LALCoherentInspiralEstimatePsiEpsilonCoaPhase (
    LALStatus                           *status,
    INT4                                 caseID[6],
    REAL4                                segNorm[4],
    REAL4                                theta,
    REAL4                                phi,
    COMPLEX8                             cData[4],
    REAL4                               *inclination,
    REAL4                               *polarization,
    REAL4                               *coaPhase
    )
{
  INT4                  i = 0;
  INT4                  j = 0;
  INT4                  k = 0;

  REAL4                 alphaBetaGamma[6][3]; /* H1 L V G T H2 */
  REAL4                 alphaBetaGammaH[3] = { 38.11, 256.35, 107.43 };
  REAL4                 alphaBetaGammaL[3] = { 38.09, 283.54, 196.88 };
  REAL4                 alphaBetaGammaV[3] = { 320.34, 275.92, 159.02 };
  REAL4                 alphaBetaGammaG[3] = { 326.41, 269.86, 110.4 };
  REAL4                 alphaBetaGammaT[3] = { 16.67, 188.47, 326.3 };
  REAL4                 gelFandABGPlusRe[6][5];
  REAL4                 gelFandABGMinusRe[6][5];
  REAL4                 gelFandPTZPlusRe[5];
  REAL4                 gelFandPTZMinusRe[5];
  REAL4                 gelFandABGPlusIm[6][5];
  REAL4                 gelFandABGMinusIm[6][5];
  REAL4                 gelFandPTZPlusIm[5];
  REAL4                 gelFandPTZMinusIm[5];
  REAL4                 gelFandPEZPlusRe = 0.0;
  REAL4                 gelFandPEZPlusIm = 0.0;
  REAL4                 gelFandPEZMinusRe = 0.0;
  REAL4                 gelFandPEZMinusIm = 0.0;
  REAL4                 mPTZRe[3] = { 0.0, 0.0, 0.0 };
  REAL4                 mPTZIm[3] = { 0.0, 0.0, 0.0 };
  REAL4                 mABGRe[6][3];
  REAL4                 mABGIm[6][3];
  REAL4                 sphereHarmonicRe[5][3][3]; /* -2 -1 0 1 2 */
  REAL4                 sphereHarmonicIm[5][3][3];
  REAL4                 sphereHarmonicReZero[3][3] = {{1,0,0},{0,-1,0},{0,0,0}};
  REAL4                 sphereHarmonicImZero[3][3] = {{0,-1,0},{-1,0,0},{0,0,0}};
  REAL4                 sphereHarmonicReOne[3][3] = {{0,0,1},{0,0,0},{1,0,0}};
  REAL4                 sphereHarmonicImOne[3][3] = {{0,0,0},{0,0,-1},{0,-1,0}};
  REAL4                 sphereHarmonicReTwo[3][3] = {{-1,0,0},{0,-1,0},{0,0,2}};
  REAL4                 sphereHarmonicImTwo[3][3] = {{0,0,0},{0,0,0},{0,0,0}};
  REAL4                 sphereHarmonicReThree[3][3] = {{0,0,1},{0,0,0},{1,0,0}};
  REAL4                 sphereHarmonicImThree[3][3] = {{0,0,0},{0,0,1},{0,1,0}};
  REAL4                 sphereHarmonicReFour[3][3] = {{1,0,0},{0,-1,0},{0,0,0}};
  REAL4                 sphereHarmonicImFour[3][3] = {{0,1,0},{1,0,0},{0,0,0}};
```

```
  REAL4                 dVectorPlusRe[6]; /* H1 L V G T H2 */
  REAL4                 dVectorPlusIm[6];
  REAL4                 dVectorMinusRe[6];
  REAL4                 dVectorMinusIm[6];
  REAL4                 eVectorRe[6];
  REAL4                 eVectorIm[6];
  REAL4                 eVectorNorm = 0.0;
  REAL4                 qVectorRe[6];
  REAL4                 qVectorIm[6];
  REAL4                 cDotqRe = 0.0;
  REAL4                 cDotqIm = 0.0;
  COMPLEX8              cPlus;
  COMPLEX8              cMinus;
  COMPLEX8              cRatio;  /* cMinus/cPlus */

  INITSTATUS( status, "LALCoherentInspiralEstimatePsiEpsilon",
      COHERENTINSPIRALFILTERC );
  ATTATCHSTATUSPTR( status );

  /* Must have 3 sites to estimate psi and epsilon */
  ASSERT( segNorm[0] && segNorm[1] && segNorm[2], status,
  COHERENTINSPIRALH_ENUMZ, COHERENTINSPIRALH_MSGENUMZ );
  ASSERT( cData[0].re && cData[1].re && cData[2].re, status,
  COHERENTINSPIRALH_ENUMZ, COHERENTINSPIRALH_MSGENUMZ );

  /* Initialize the arrays */
  for( i = 0; i < 6; i++ )
    {
      for( j = 0; j < 3; j++ )
{
  alphaBetaGamma[i][j] = 0.0;
  mABGRe[i][j] = 0.0;
  mABGIm[i][j] = 0.0;
}
    }
  for( i = 0; i < 6; i++ )
    {
      for( j = 0; j < 5; j++ )
{
  gelFandABGPlusRe[i][j] = 0.0;
  gelFandABGMinusRe[i][j] = 0.0;
  gelFandABGPlusIm[i][j] = 0.0;
  gelFandABGMinusIm[i][j] = 0.0;
}
    }
  for( i = 0; i < 5; i++ )
    {
      gelFandPTZPlusRe[i] = 0.0;
      gelFandPTZMinusRe[i] = 0.0;
      gelFandPTZPlusIm[i] = 0.0;
      gelFandPTZMinusIm[i] = 0.0;
    }
  for( i = 0; i < 5; i++ )
    {
      for( j = 0; j < 3; j++ )
{
  for( k = 0; k < 3; k++ )
```

```
    {
      sphereHarmonicRe[i][j][k] = 0.0;
      sphereHarmonicIm[i][j][k] = 0.0;
    }
}
    }
  for( i = 0; i < 6; i++ )
    {
      dVectorPlusRe[i] = 0.0;
      dVectorPlusIm[i] = 0.0;
      dVectorMinusRe[i] = 0.0;
      dVectorMinusIm[i] = 0.0;
      eVectorRe[i] = 0.0;
      eVectorIm[i] = 0.0;
      qVectorRe[i] = 0.0;
      qVectorIm[i] = 0.0;
    }

  /* Set the euler angles for the various sites */
  for( i = 0; i < 3; i++ )
    {
      alphaBetaGamma[0][i] = LAL_PI_180 * alphaBetaGammaH[i];
      alphaBetaGamma[1][i] = LAL_PI_180 * alphaBetaGammaL[i];
      alphaBetaGamma[2][i] = LAL_PI_180 * alphaBetaGammaV[i];
      alphaBetaGamma[3][i] = LAL_PI_180 * alphaBetaGammaG[i];
      alphaBetaGamma[4][i] = LAL_PI_180 * alphaBetaGammaT[i];
      alphaBetaGamma[5][i] = LAL_PI_180 * alphaBetaGammaH[i];
    }

  /* Compute the various Gel'Fand Functions */
  mPTZRe[0] = LAL_SQRT1_2 * cos(phi);
  mPTZRe[1] = LAL_SQRT1_2 * sin(phi);
  mPTZRe[2] = 0;
  mPTZIm[0] = LAL_SQRT1_2 * -cos(theta)*sin(phi);
  mPTZIm[1] = LAL_SQRT1_2 * cos(theta)*cos(phi);
  mPTZIm[2] = LAL_SQRT1_2 * sin(theta);

  for( i = 0; i < 6; i++ )
    {
      mABGRe[i][0] = LAL_SQRT1_2 * (cos(alphaBetaGamma[i][0]) *
            cos(alphaBetaGamma[i][2]) - sin(alphaBetaGamma[i][0])*
            cos(alphaBetaGamma[i][1])*sin(alphaBetaGamma[i][2]));
      mABGRe[i][1] = LAL_SQRT1_2 * (sin(alphaBetaGamma[i][0]) *
            cos(alphaBetaGamma[i][2]) + cos(alphaBetaGamma[i][0])*
            cos(alphaBetaGamma[i][1])*sin(alphaBetaGamma[i][2]));
      mABGRe[i][2] = LAL_SQRT1_2 * sin(alphaBetaGamma[i][1])*
            sin(alphaBetaGamma[i][2]);
      mABGIm[i][0] = LAL_SQRT1_2 * -(cos(alphaBetaGamma[i][0]) *
            sin(alphaBetaGamma[i][2]) + sin(alphaBetaGamma[i][0])*
            cos(alphaBetaGamma[i][1])*cos(alphaBetaGamma[i][2]));
      mABGIm[i][1] = LAL_SQRT1_2 * (-sin(alphaBetaGamma[i][0]) *
            sin(alphaBetaGamma[i][2]) + cos(alphaBetaGamma[i][0])*
            cos(alphaBetaGamma[i][1])*cos(alphaBetaGamma[i][2]));
      mABGIm[i][2] = LAL_SQRT1_2 * sin(alphaBetaGamma[i][1])*
            cos(alphaBetaGamma[i][2]);

    }
```

```
  for( i = 0; i < 3; i++ )
    {
      for( j = 0;j < 3; j++ )
{
  sphereHarmonicRe[0][i][j] = 0.25*sqrt(15/(2*LAL_PI)) * sphereHarmonicReZero[i][j];
  sphereHarmonicIm[0][i][j] = 0.25*sqrt(15/(2*LAL_PI)) * sphereHarmonicImZero[i][j];
  sphereHarmonicRe[1][i][j] = 0.25*sqrt(15/(2*LAL_PI)) * sphereHarmonicReOne[i][j];
  sphereHarmonicIm[1][i][j] = 0.25*sqrt(15/(2*LAL_PI)) * sphereHarmonicImOne[i][j];
  sphereHarmonicRe[2][i][j] = 0.5*sqrt(5/(4*LAL_PI)) * sphereHarmonicReTwo[i][j];
  sphereHarmonicIm[2][i][j] = 0.5*sqrt(5/(4*LAL_PI)) * sphereHarmonicImTwo[i][j];
  sphereHarmonicRe[3][i][j] = -0.25*sqrt(15/(2*LAL_PI)) * sphereHarmonicReThree[i][j];
  sphereHarmonicIm[3][i][j] = -0.25*sqrt(15/(2*LAL_PI)) * sphereHarmonicImThree[i][j];
  sphereHarmonicRe[4][i][j] = 0.25*sqrt(15/(2*LAL_PI)) * sphereHarmonicReFour[i][j];
  sphereHarmonicIm[4][i][j] = 0.25*sqrt(15/(2*LAL_PI)) * sphereHarmonicImFour[i][j];
}
    }

  for( i = 0; i < 3; i++ )
    {
      for( j = 0; j < 3; j++ )
{
  gelFandPTZPlusRe[0] += sphereHarmonicRe[4][i][j]*mPTZRe[i]*mPTZRe[j]
               - sphereHarmonicRe[4][i][j]*mPTZIm[i]*mPTZIm[j]
               - sphereHarmonicIm[4][i][j]*mPTZRe[i]*mPTZIm[j]
               - sphereHarmonicIm[4][i][j]*mPTZIm[i]*mPTZRe[j];

  gelFandPTZPlusIm[0] += sphereHarmonicRe[4][i][j]*mPTZRe[i]*mPTZIm[j]
               + sphereHarmonicRe[4][i][j]*mPTZIm[i]*mPTZRe[j]
               + sphereHarmonicIm[4][i][j]*mPTZRe[i]*mPTZRe[j]
               - sphereHarmonicIm[4][i][j]*mPTZIm[i]*mPTZIm[j];

  gelFandPTZPlusRe[1] += -sphereHarmonicRe[3][i][j]*mPTZRe[i]*mPTZRe[j]
               + sphereHarmonicRe[3][i][j]*mPTZIm[i]*mPTZIm[j]
               + sphereHarmonicIm[3][i][j]*mPTZRe[i]*mPTZIm[j]
               + sphereHarmonicIm[3][i][j]*mPTZIm[i]*mPTZRe[j];

  gelFandPTZPlusIm[1] += -sphereHarmonicRe[3][i][j]*mPTZRe[i]*mPTZIm[j]
               - sphereHarmonicRe[3][i][j]*mPTZIm[i]*mPTZRe[j]
               - sphereHarmonicIm[3][i][j]*mPTZRe[i]*mPTZRe[j]
               + sphereHarmonicIm[3][i][j]*mPTZIm[i]*mPTZIm[j];

  gelFandPTZPlusRe[2] += sphereHarmonicRe[2][i][j]*mPTZRe[i]*mPTZRe[j]
               - sphereHarmonicRe[2][i][j]*mPTZIm[i]*mPTZIm[j]
               - sphereHarmonicIm[2][i][j]*mPTZRe[i]*mPTZIm[j]
               - sphereHarmonicIm[2][i][j]*mPTZIm[i]*mPTZRe[j];

  gelFandPTZPlusIm[2] += sphereHarmonicRe[2][i][j]*mPTZRe[i]*mPTZIm[j]
               + sphereHarmonicRe[2][i][j]*mPTZIm[i]*mPTZRe[j]
               + sphereHarmonicIm[2][i][j]*mPTZRe[i]*mPTZRe[j]
               - sphereHarmonicIm[2][i][j]*mPTZIm[i]*mPTZIm[j];

  gelFandPTZPlusRe[3] += -sphereHarmonicRe[1][i][j]*mPTZRe[i]*mPTZRe[j]
               + sphereHarmonicRe[1][i][j]*mPTZIm[i]*mPTZIm[j]
               + sphereHarmonicIm[1][i][j]*mPTZRe[i]*mPTZIm[j]
               + sphereHarmonicIm[1][i][j]*mPTZIm[i]*mPTZRe[j];
```

```
gelFandPTZPlusIm[3] += -sphereHarmonicRe[1][i][j]*mPTZRe[i]*mPTZIm[j]
              - sphereHarmonicRe[1][i][j]*mPTZIm[i]*mPTZRe[j]
              - sphereHarmonicIm[1][i][j]*mPTZRe[i]*mPTZRe[j]
              + sphereHarmonicIm[1][i][j]*mPTZIm[i]*mPTZIm[j];

gelFandPTZPlusRe[4] += sphereHarmonicRe[0][i][j]*mPTZRe[i]*mPTZRe[j]
              - sphereHarmonicRe[0][i][j]*mPTZIm[i]*mPTZIm[j]
              - sphereHarmonicIm[0][i][j]*mPTZRe[i]*mPTZIm[j]
              - sphereHarmonicIm[0][i][j]*mPTZIm[i]*mPTZRe[j];

gelFandPTZPlusIm[4] += sphereHarmonicRe[0][i][j]*mPTZRe[i]*mPTZIm[j]
              + sphereHarmonicRe[0][i][j]*mPTZIm[i]*mPTZRe[j]
              + sphereHarmonicIm[0][i][j]*mPTZRe[i]*mPTZRe[j]
              - sphereHarmonicIm[0][i][j]*mPTZIm[i]*mPTZIm[j];

gelFandPTZMinusRe[0] += sphereHarmonicRe[4][i][j]*mPTZRe[i]*mPTZRe[j]
              - sphereHarmonicRe[4][i][j]*mPTZIm[i]*mPTZIm[j]
              + sphereHarmonicIm[4][i][j]*mPTZRe[i]*mPTZIm[j]
              + sphereHarmonicIm[4][i][j]*mPTZIm[i]*mPTZRe[j];

gelFandPTZMinusIm[0] += -sphereHarmonicRe[4][i][j]*mPTZRe[i]*mPTZIm[j]
              - sphereHarmonicRe[4][i][j]*mPTZIm[i]*mPTZRe[j]
              + sphereHarmonicIm[4][i][j]*mPTZRe[i]*mPTZRe[j]
              - sphereHarmonicIm[4][i][j]*mPTZIm[i]*mPTZIm[j];

gelFandPTZMinusRe[1] += -sphereHarmonicRe[3][i][j]*mPTZRe[i]*mPTZRe[j]
              + sphereHarmonicRe[3][i][j]*mPTZIm[i]*mPTZIm[j]
              - sphereHarmonicIm[3][i][j]*mPTZRe[i]*mPTZIm[j]
              - sphereHarmonicIm[3][i][j]*mPTZIm[i]*mPTZRe[j];

gelFandPTZMinusIm[1] += sphereHarmonicRe[3][i][j]*mPTZRe[i]*mPTZIm[j]
              + sphereHarmonicRe[3][i][j]*mPTZIm[i]*mPTZRe[j]
              - sphereHarmonicIm[3][i][j]*mPTZRe[i]*mPTZRe[j]
              + sphereHarmonicIm[3][i][j]*mPTZIm[i]*mPTZIm[j];

gelFandPTZMinusRe[2] += sphereHarmonicRe[2][i][j]*mPTZRe[i]*mPTZRe[j]
              - sphereHarmonicRe[2][i][j]*mPTZIm[i]*mPTZIm[j]
              + sphereHarmonicIm[2][i][j]*mPTZRe[i]*mPTZIm[j]
              + sphereHarmonicIm[2][i][j]*mPTZIm[i]*mPTZRe[j];

gelFandPTZMinusIm[2] += -sphereHarmonicRe[2][i][j]*mPTZRe[i]*mPTZIm[j]
              - sphereHarmonicRe[2][i][j]*mPTZIm[i]*mPTZRe[j]
              + sphereHarmonicIm[2][i][j]*mPTZRe[i]*mPTZRe[j]
              - sphereHarmonicIm[2][i][j]*mPTZIm[i]*mPTZIm[j];

gelFandPTZMinusRe[3] += -sphereHarmonicRe[1][i][j]*mPTZRe[i]*mPTZRe[j]
              + sphereHarmonicRe[1][i][j]*mPTZIm[i]*mPTZIm[j]
              - sphereHarmonicIm[1][i][j]*mPTZRe[i]*mPTZIm[j]
              - sphereHarmonicIm[1][i][j]*mPTZIm[i]*mPTZRe[j];

gelFandPTZMinusIm[3] += sphereHarmonicRe[1][i][j]*mPTZRe[i]*mPTZIm[j]
              + sphereHarmonicRe[1][i][j]*mPTZIm[i]*mPTZRe[j]
              - sphereHarmonicIm[1][i][j]*mPTZRe[i]*mPTZRe[j]
              + sphereHarmonicIm[1][i][j]*mPTZIm[i]*mPTZIm[j];

gelFandPTZMinusRe[4] += sphereHarmonicRe[0][i][j]*mPTZRe[i]*mPTZRe[j]
              - sphereHarmonicRe[0][i][j]*mPTZIm[i]*mPTZIm[j]
```

```
                + sphereHarmonicIm[0][i][j]*mPTZRe[i]*mPTZIm[j]
                + sphereHarmonicIm[0][i][j]*mPTZIm[i]*mPTZRe[j];

  gelFandPTZMinusIm[4] += -sphereHarmonicRe[0][i][j]*mPTZRe[i]*mPTZIm[j]
                - sphereHarmonicRe[0][i][j]*mPTZIm[i]*mPTZRe[j]
                + sphereHarmonicIm[0][i][j]*mPTZRe[i]*mPTZRe[j]
                - sphereHarmonicIm[0][i][j]*mPTZIm[i]*mPTZIm[j];
}
    }

  for( k = 0; k < 5; k++)
    {
      for( i = 0; i < 3; i++ )
{
  for( j = 0; j < 3; j++ )
    {
      gelFandABGPlusRe[k][0] += sphereHarmonicRe[4][i][j]*mABGRe[k][i]*mABGRe[k][j]
                    - sphereHarmonicRe[4][i][j]*mABGIm[k][i]*mABGIm[k][j]
                    - sphereHarmonicIm[4][i][j]*mABGRe[k][i]*mABGIm[k][j]
            - sphereHarmonicIm[4][i][j]*mABGIm[k][i]*mABGRe[k][j];

      gelFandABGPlusIm[k][0] += sphereHarmonicRe[4][i][j]*mABGRe[k][i]*mABGIm[k][j]
                    + sphereHarmonicRe[4][i][j]*mABGIm[k][i]*mABGRe[k][j]
                    + sphereHarmonicIm[4][i][j]*mABGRe[k][i]*mABGRe[k][j]
                    - sphereHarmonicIm[4][i][j]*mABGIm[k][i]*mABGIm[k][j];

      gelFandABGPlusRe[k][1] += -sphereHarmonicRe[3][i][j]*mABGRe[k][i]*mABGRe[k][j]
                    + sphereHarmonicRe[3][i][j]*mABGIm[k][i]*mABGIm[k][j]
                    + sphereHarmonicIm[3][i][j]*mABGRe[k][i]*mABGIm[k][j]
                    + sphereHarmonicIm[3][i][j]*mABGIm[k][i]*mABGRe[k][j];

      gelFandABGPlusIm[k][1] += -sphereHarmonicRe[3][i][j]*mABGRe[k][i]*mABGIm[k][j]
                    - sphereHarmonicRe[3][i][j]*mABGIm[k][i]*mABGRe[k][j]
                    - sphereHarmonicIm[3][i][j]*mABGRe[k][i]*mABGRe[k][j]
                    + sphereHarmonicIm[3][i][j]*mABGIm[k][i]*mABGIm[k][j];

      gelFandABGPlusRe[k][2] += sphereHarmonicRe[2][i][j]*mABGRe[k][i]*mABGRe[k][j]
                    - sphereHarmonicRe[2][i][j]*mABGIm[k][i]*mABGIm[k][j]
                    - sphereHarmonicIm[2][i][j]*mABGRe[k][i]*mABGIm[k][j]
                    - sphereHarmonicIm[2][i][j]*mABGIm[k][i]*mABGRe[k][j];

      gelFandABGPlusIm[k][2] += sphereHarmonicRe[2][i][j]*mABGRe[k][i]*mABGIm[k][j]
                    + sphereHarmonicRe[2][i][j]*mABGIm[k][i]*mABGRe[k][j]
                    + sphereHarmonicIm[2][i][j]*mABGRe[k][i]*mABGRe[k][j]
                    - sphereHarmonicIm[2][i][j]*mABGIm[k][i]*mABGIm[k][j];

      gelFandABGPlusRe[k][3] += -sphereHarmonicRe[1][i][j]*mABGRe[k][i]*mABGRe[k][j]
                    + sphereHarmonicRe[1][i][j]*mABGIm[k][i]*mABGIm[k][j]
                    + sphereHarmonicIm[1][i][j]*mABGRe[k][i]*mABGIm[k][j]
                    + sphereHarmonicIm[1][i][j]*mABGIm[k][i]*mABGRe[k][j];

      gelFandABGPlusIm[k][3] += -sphereHarmonicRe[1][i][j]*mABGRe[k][i]*mABGIm[k][j]
                    - sphereHarmonicRe[1][i][j]*mABGIm[k][i]*mABGRe[k][j]
                    - sphereHarmonicIm[1][i][j]*mABGRe[k][i]*mABGRe[k][j]
                    + sphereHarmonicIm[1][i][j]*mABGIm[k][i]*mABGIm[k][j];

      gelFandABGPlusRe[k][4] += sphereHarmonicRe[0][i][j]*mABGRe[k][i]*mABGRe[k][j]
```

```
              - sphereHarmonicRe[0][i][j]*mABGIm[k][i]*mABGIm[k][j]
              - sphereHarmonicIm[0][i][j]*mABGRe[k][i]*mABGIm[k][j]
              - sphereHarmonicIm[0][i][j]*mABGIm[k][i]*mABGRe[k][j];

gelFandABGPlusIm[k][4] += sphereHarmonicRe[0][i][j]*mABGRe[k][i]*mABGIm[k][j]
              + sphereHarmonicRe[0][i][j]*mABGIm[k][i]*mABGRe[k][j]
              + sphereHarmonicIm[0][i][j]*mABGRe[k][i]*mABGRe[k][j]
              - sphereHarmonicIm[0][i][j]*mABGIm[k][i]*mABGIm[k][j];

gelFandABGMinusRe[k][0] += sphereHarmonicRe[4][i][j]*mABGRe[k][i]*mABGRe[k][j]
              - sphereHarmonicRe[4][i][j]*mABGIm[k][i]*mABGIm[k][j]
              + sphereHarmonicIm[4][i][j]*mABGRe[k][i]*mABGIm[k][j]
              + sphereHarmonicIm[4][i][j]*mABGIm[k][i]*mABGRe[k][j];

gelFandABGMinusIm[k][0] += -sphereHarmonicRe[4][i][j]*mABGRe[k][i]*mABGIm[k][j]
              - sphereHarmonicRe[4][i][j]*mABGIm[k][i]*mABGRe[k][j]
              + sphereHarmonicIm[4][i][j]*mABGRe[k][i]*mABGRe[k][j]
              - sphereHarmonicIm[4][i][j]*mABGIm[k][i]*mABGIm[k][j];

gelFandABGMinusRe[k][1] += -sphereHarmonicRe[3][i][j]*mABGRe[k][i]*mABGRe[k][j]
              + sphereHarmonicRe[3][i][j]*mABGIm[k][i]*mABGIm[k][j]
              - sphereHarmonicIm[3][i][j]*mABGRe[k][i]*mABGIm[k][j]
              - sphereHarmonicIm[3][i][j]*mABGIm[k][i]*mABGRe[k][j];

gelFandABGMinusIm[k][1] += sphereHarmonicRe[3][i][j]*mABGRe[k][i]*mABGIm[k][j]
              + sphereHarmonicRe[3][i][j]*mABGIm[k][i]*mABGRe[k][j]
              - sphereHarmonicIm[3][i][j]*mABGRe[k][i]*mABGRe[k][j]
              + sphereHarmonicIm[3][i][j]*mABGIm[k][i]*mABGIm[k][j];

gelFandABGMinusRe[k][2] += sphereHarmonicRe[2][i][j]*mABGRe[k][i]*mABGRe[k][j]
              - sphereHarmonicRe[2][i][j]*mABGIm[k][i]*mABGIm[k][j]
              + sphereHarmonicIm[2][i][j]*mABGRe[k][i]*mABGIm[k][j]
              + sphereHarmonicIm[2][i][j]*mABGIm[k][i]*mABGRe[k][j];

gelFandABGMinusIm[k][2] += -sphereHarmonicRe[2][i][j]*mABGRe[k][i]*mABGIm[k][j]
              - sphereHarmonicRe[2][i][j]*mABGIm[k][i]*mABGRe[k][j]
              + sphereHarmonicIm[2][i][j]*mABGRe[k][i]*mABGRe[k][j]
              - sphereHarmonicIm[2][i][j]*mABGIm[k][i]*mABGIm[k][j];

gelFandABGMinusRe[k][3] += -sphereHarmonicRe[1][i][j]*mABGRe[k][i]*mABGRe[k][j]
              + sphereHarmonicRe[1][i][j]*mABGIm[k][i]*mABGIm[k][j]
              - sphereHarmonicIm[1][i][j]*mABGRe[k][i]*mABGIm[k][j]
              - sphereHarmonicIm[1][i][j]*mABGIm[k][i]*mABGRe[k][j];

gelFandABGMinusIm[k][3] += sphereHarmonicRe[1][i][j]*mABGRe[k][i]*mABGIm[k][j]
              + sphereHarmonicRe[1][i][j]*mABGIm[k][i]*mABGRe[k][j]
              - sphereHarmonicIm[1][i][j]*mABGRe[k][i]*mABGRe[k][j]
              + sphereHarmonicIm[1][i][j]*mABGIm[k][i]*mABGIm[k][j];

gelFandABGMinusRe[k][4] += sphereHarmonicRe[0][i][j]*mABGRe[k][i]*mABGRe[k][j]
              - sphereHarmonicRe[0][i][j]*mABGIm[k][i]*mABGIm[k][j]
              + sphereHarmonicIm[0][i][j]*mABGRe[k][i]*mABGIm[k][j]
              + sphereHarmonicIm[0][i][j]*mABGIm[k][i]*mABGRe[k][j];

gelFandABGMinusIm[k][4] += -sphereHarmonicRe[0][i][j]*mABGRe[k][i]*mABGIm[k][j]
              - sphereHarmonicRe[0][i][j]*mABGIm[k][i]*mABGRe[k][j]
              + sphereHarmonicIm[0][i][j]*mABGRe[k][i]*mABGRe[k][j]
```

```
                    - sphereHarmonicIm[0][i][j]*mABGIm[k][i]*mABGIm[k][j];
    }
}
    }

  /* Dont forget the coefficients... */

  for( i = 0; i < 5; i++ )
    {
      gelFandPTZPlusRe[i] *= sqrt(8*LAL_PI/15);
      gelFandPTZPlusIm[i] *= sqrt(8*LAL_PI/15);
      gelFandPTZMinusRe[i] *= sqrt(8*LAL_PI/15);
      gelFandPTZMinusIm[i] *= sqrt(8*LAL_PI/15);
    }


  for( i = 0; i < 6; i++ )
    {
      for( j = 0; j < 5; j++ )
{
  gelFandABGPlusRe[i][j] *= sqrt(8*LAL_PI/15);
  gelFandABGPlusIm[i][j] *= sqrt(8*LAL_PI/15);
  gelFandABGMinusRe[i][j] *= sqrt(8*LAL_PI/15);
  gelFandABGMinusIm[i][j] *= sqrt(8*LAL_PI/15);
}
    }

  /* Armed with the gelFand functions, the dVectors can be constructed */

  for( i = 0; i < 6; i++ )
    {
      for( j = 0; j < 5; j++ )
{
  dVectorPlusRe[i] += gelFandPTZPlusIm[j] * ( gelFandABGPlusRe[i][j]
                - gelFandABGMinusRe[i][j] ) + gelFandPTZPlusRe[j] *
                ( gelFandABGMinusIm[i][j]  - gelFandABGPlusIm[i][j] );
  dVectorMinusRe[i] += gelFandPTZMinusIm[j] * ( gelFandABGPlusRe[i][j]
                - gelFandABGMinusRe[i][j] ) + gelFandPTZMinusRe[j] *
                ( gelFandABGMinusIm[i][j]  - gelFandABGPlusIm[i][j] );
  dVectorPlusIm[i] += gelFandPTZPlusIm[j] * ( gelFandABGMinusIm[i][j]
                - gelFandABGPlusIm[i][j] ) - gelFandPTZPlusRe[j] *
                ( gelFandABGPlusRe[i][j] - gelFandABGMinusRe[i][j] );
  dVectorMinusIm[i] += gelFandPTZMinusIm[j] * ( gelFandABGMinusIm[i][j]
                - gelFandABGPlusIm[i][j] ) - gelFandPTZMinusRe[j] *
                ( gelFandABGPlusRe[i][j] - gelFandABGMinusRe[i][j] );
}
    }

  /* Now form the C's ( +2 and -2 ) */

  cPlus.re = 0.0;
  cPlus.im = 0.0;
  cMinus.re = 0.0;
  cMinus.im = 0.0;
  cRatio.re = 0.0;
  cRatio.im = 0.0;
```

```
  k = 0;
  for( i = 0; i < 6; i++ )
    {
      if( caseID[i] )
{
  cPlus.re += segNorm[k] * ( cData[k].re * dVectorPlusRe[i]
               - cData[k].im * dVectorPlusIm[i] );
  cPlus.im += segNorm[k] * ( cData[k].re * dVectorPlusIm[i]
+ cData[k].im * dVectorPlusRe[i] );
  cMinus.re += segNorm[k] * ( cData[k].re * dVectorMinusRe[i]
               - cData[k].im * dVectorMinusIm[i] );
  cMinus.im += segNorm[k] * ( cData[k].re * dVectorMinusIm[i]
+ cData[k].im * dVectorMinusRe[i] );
  k++;

}
    }

  cRatio.re = ( cPlus.re * cMinus.re + cPlus.im * cMinus.im ) / ( cPlus.re *
          cPlus.re + cPlus.im * cPlus.im );
  cRatio.im = ( cPlus.re * cMinus.im - cPlus.im * cMinus.re ) / ( cPlus.re *
  cPlus.re + cPlus.im * cPlus.im );

  /* Now the estimates can be computed */

  *inclination = acos(  ( 1 - sqrt( sqrt(cRatio.re*cRatio.re + cRatio.im*cRatio.im) ))/
  ( 1 + sqrt( sqrt(cRatio.re*cRatio.re + cRatio.im*cRatio.im) )) );
  *polarization = 0.25 * atan( cRatio.im / cRatio.re );

  /* To estimate the coaphase, I need to construct the eVectors */

  gelFandPEZMinusRe = 0.25 * (cos(*inclination) - 1)*(cos(*inclination) - 1) *
cos(2 * (*polarization));
  gelFandPEZMinusIm = 0.25 * (cos(*inclination) - 1)*(cos(*inclination) - 1) *
sin(2 * (*polarization));
  gelFandPEZPlusRe = 0.25 * (cos(*inclination) + 1)*(cos(*inclination) + 1) *
cos(2 * (*polarization));
  gelFandPEZPlusIm = -0.25 * (cos(*inclination) + 1)*(cos(*inclination) + 1) *
sin(2 * (*polarization));

  k = 0;
  for( i = 0; i < 6; i++ )
    {
      if( caseID[i] )
{
  eVectorRe[i] = segNorm[k] * (gelFandPEZPlusRe*dVectorPlusRe[i] -
gelFandPEZPlusIm*dVectorPlusIm[i] + gelFandPEZMinusRe*dVectorPlusRe[i] +
gelFandPEZMinusIm*dVectorPlusIm[i]);
  eVectorIm[i] = segNorm[k] * (gelFandPEZPlusRe*dVectorPlusIm[i] +
gelFandPEZPlusIm*dVectorPlusRe[i] + gelFandPEZMinusIm*dVectorPlusRe[i] -
gelFandPEZMinusRe*dVectorPlusIm[i]);
  k++;
}
    }

  /* Now form the Q's */
```

```
  for( i = 0; i < 6; i++ )
    {
      eVectorNorm += eVectorRe[i]*eVectorRe[i] + eVectorIm[i]*eVectorIm[i];
    }

  eVectorNorm = sqrt( eVectorNorm );

  for( i = 0; i < 6; i++ )
    {
      qVectorRe[i] = eVectorRe[i] / eVectorNorm;
      qVectorIm[i] = eVectorIm[i] / eVectorNorm;
    }

  /* Now for the phase estimate */

  k = 0;
  for( i = 0; i < 6; i++ )
    {
      if( caseID[i] )
{
  cDotqRe += cData[k].re * qVectorRe[i] + cData[k].im * qVectorIm[i];
  cDotqIm += cData[k].re * qVectorIm[i] - cData[k].im * qVectorRe[i];
  k++;
}
    }

  *coaPhase = atan( cDotqIm / cDotqRe );

  /* normal exit */
  DETATCHSTATUSPTR( status );
  RETURN( status );
}

void
LALCoherentInspiralEstimateDistance (
    LALStatus                           *status,
    REAL4                                segNorm[4],
    REAL4                                templateNorm,
    REAL4                                deltaT,
    INT4                                 segmentLength,  /* time pts */
    REAL4                                coherentSNR,
    REAL4                               *distance
    )
{
  INT4                i = 0;
  REAL4               sigmaSq[4] = {0.0, 0.0, 0.0, 0.0};
  REAL4               sigmaCoherent = 0.0;

  INITSTATUS( status, "LALCoherentInspiralEstimateDistance",
      COHERENTINSPIRALFILTERC );
  ATTATCHSTATUSPTR( status );

  /* Check the validity of the input params */

  ASSERT( segNorm[0] && segNorm[1], status,
  COHERENTINSPIRALH_ENUMZ, COHERENTINSPIRALH_MSGENUMZ );
  ASSERT( coherentSNR > 0, status,
```

```
      COHERENTINSPIRALH_ESEGZ, COHERENTINSPIRALH_MSGESEGZ );
  ASSERT( templateNorm > 0, status,
      COHERENTINSPIRALH_ESEGZ, COHERENTINSPIRALH_MSGESEGZ );
  ASSERT( deltaT > 0, status,
      COHERENTINSPIRALH_ESEGZ, COHERENTINSPIRALH_MSGESEGZ );
  ASSERT( segmentLength > 0, status,
      COHERENTINSPIRALH_ESEGZ, COHERENTINSPIRALH_MSGESEGZ );

  /* Now estimate the distance coherently */
  for( i = 0; i < 4; i++ )
    {
      sigmaSq[i] = segNorm[i] * templateNorm;
    }
  for( i = 0; i < 4; i++ )
    {
      sigmaCoherent += sigmaSq[i];
    }

  *distance = (2 / coherentSNR) * sqrt( (sigmaCoherent * deltaT) / segmentLength );

  /* normal exit */
  DETATCHSTATUSPTR( status );
  RETURN( status );
}

void
LALCoherentInspiralFilterSegment (
    LALStatus                           *status,
    MultiInspiralTable                  **eventList,
    CoherentInspiralFilterInput         *input,
    CoherentInspiralFilterParams        *params
    )
{
  INT4                              caseID[6] = {0,0,0,0,0,0};
  INT4                              i,q,w,m,j,k,l;
  INT4                              wTemp = 0;
  INT4                              qTemp = 0;
  INT4                              mTemp = 0;
  INT4                              jTemp = 0;
  CHAR                              idtag[6][3] = {"H1","L","V","G","T","H2"};
  INT4                              indexarray[4] = {0,0,0,0};
  CHAR                              caseStr[FILENAME_MAX];
  INT8                              chirpTimeNS = 0;
  UINT4                             numDetectors = 0;
  UINT4                             numSegments = 0;
  UINT4                             numPoints = 0;
  UINT4                             numBeamPoints = 0;
  INT4                              deltaEventIndex = 0;
  INT4                              eventStartIdx = 0;
  INT4                              slidePoints[3] = {0,0,0};
  INT4                              segmentLength = 0;
  REAL4                             buffer = 0; /*account for timing errors*/
  REAL4                             timingError = 0.00025; /* allowed timing error of 2 ms */
  REAL4                             s[4][3];/*up to 4 distances;in 3D space*/
  REAL4                             deltaT = 0.0;
  REAL4                             nHatVect[3] = {0,0,0};
  REAL4                             distance[4] = {0,0,0,0};
```

```
 REAL4                           timeDelay[4] = {0,0,0,0};
 REAL4                           chirpTime = 0;
 REAL4                           cohSNRThresh = 0;
 REAL4                           theta = 0.0;
 REAL4                           phi = 0.0;
 REAL4                           inclination = 0.0;
 REAL4                           polarization = 0.0;
 REAL4                           distanceEstimate = 0.0;
 REAL4                           coaPhase = 0.0;
 REAL8                           tempTime = 0.0;
 REAL8                           fracpart = 0.0;
 REAL8                           intpart = 0.0;
 COMPLEX8                        cDataTemp[4];
 UINT4                           cohSNROut;
 REAL4                           cohSNRLocal = 0;
 LALDetector                    *detector = NULL;
 COMPLEX8TimeSeries             *cData = NULL;
 MultiInspiralTable             *thisEvent = NULL;
 CoherentInspiralBeamVector     *beamVec = NULL;


 INITSTATUS( status, "LALCoherentInspiralFilterSegment",
     COHERENTINSPIRALFILTERC );
 ATTATCHSTATUSPTR( status );

 /*
  *
  * check that the arguments are reasonable
  *
  */


 /* make sure the output handle exists, but points to a null pointer */
 ASSERT( eventList, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );
 ASSERT( !*eventList, status, COHERENTINSPIRALH_ENNUL, COHERENTINSPIRALH_MSGENNUL );

 /* make sure that the parameter structure exists */
 ASSERT( params, status, COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );

 /* check that the filter parameters are reasonable */
 ASSERT( params->cohSNRThresh > 0, status,
 COHERENTINSPIRALH_ERHOT, COHERENTINSPIRALH_MSGERHOT );

/* check that the number of detectors is greater than 1 */
 ASSERT( params->numDetectors > 1, status,
     COHERENTINSPIRALH_EZDET, COHERENTINSPIRALH_MSGEZDET );

 /* check that the number of detectors is less than 5 */
 ASSERT( params->numDetectors < 5, status,
 COHERENTINSPIRALH_EZDET, COHERENTINSPIRALH_MSGEZDET );

 /* check that the number of segments in positive */
 ASSERT( params->numSegments > 0, status,
     COHERENTINSPIRALH_ESEGZ, COHERENTINSPIRALH_MSGESEGZ );

 /* if a cohSNRVec vector has been created, check we can store data in it  */
 if ( params->cohSNROut ) {
```

```
    ASSERT( params->cohSNRVec->data->data, status,
    COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );
    ASSERT( params->cohSNRVec->data, status,
    COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );
  }

  /* make sure that the input structure contains some input */
  ASSERT( input->tmplt, status,
  COHERENTINSPIRALH_ENULL, COHERENTINSPIRALH_MSGENULL );


  /*** read in the parameters ***/
  numDetectors = params->numDetectors;
  numPoints = params->numPoints;
  numSegments = params->numSegments;
  numBeamPoints = params->numBeamPoints;
  cohSNRThresh = params->cohSNRThresh;
  cohSNROut = params->cohSNROut;
  deltaT = params->deltaT;
  segmentLength = params->segmentLength;

  /* if the full coherent snr vector is required, set it to zero */
  if ( cohSNROut ) {
    memset( params->cohSNRVec->data->data, 0, numPoints * sizeof( REAL4 ));
  }

  /*CHECK: hardwired to 6 detectors for now */
  for (l=0 ;l< 6 ; l++)
    {
      caseID[l] = params->detIDVec->data[l];
    }

  /* Now generate the network string for MultiInspiralTables */

  for( i = 0; i < 4; i++ )
    {
      cDataTemp[i].im = 0.0;
      cDataTemp[i].re = 0.0;
    }
  i = 0;
  for (l=0 ;l<6 ;l++)
    {
      if( caseID[l] )
{
  indexarray[i] = l;
  i++;
}
    }
  i = 0;
  j = 0;
  k = 0;
  l = 0;
  switch ( numDetectors )
    {
    case 2:
      i = indexarray[0];
      j = indexarray[1];
```

```
      LALSnprintf( caseStr, FILENAME_MAX * sizeof(CHAR), "%s-%s",idtag[i],idtag[j]);
      break;

    case 3:
      i=indexarray[0];
      j=indexarray[1];
      k=indexarray[2];
      LALSnprintf( caseStr, FILENAME_MAX * sizeof(CHAR), "%s-%s-%s",idtag[i],
idtag[j],idtag[k]);
      break;

    case 4:
      i=indexarray[0];
      j=indexarray[1];
      k=indexarray[2];
      l=indexarray[3];
      LALSnprintf( caseStr, FILENAME_MAX * sizeof(CHAR), "%s-%s-%s-%s",idtag[i],
idtag[j],idtag[k],idtag[l]);
      break;
    }

  /*** get detector beam-pattern information if we have 3 sites ***/

  if( (numDetectors == 3 && !(caseID[0] && caseID[5])) || numDetectors == 4 )
{
  beamVec = input->beamVec;
}

  /* read in the c-data for multiple detectors */
  cData = input->multiCData->cData;

  /*** get detector-site locations */
  detector = params->detectorVec->detector;


  /*Now compute the position vector of all detectors relative to first detector*/

  for ( l=1 ; l < (INT4) numDetectors ; l++) {
    for (i=0;i<3;i++)
      {
s[l][i] = (REAL4) ( detector[l].location[i] - detector[0].location[i]);
      }
  }

  /* calculate the length of the chirp for clustering over chirp-length*/
  {
    REAL4 eta = input->tmplt->eta;
    REAL4 m1 = input->tmplt->mass1;
    REAL4 m2 = input->tmplt->mass2;
    REAL4 fmin = params->fLow;
    REAL4 mtotal = m1 + m2;
    REAL4 c0 = 5*mtotal*LAL_MTSUN_SI/(256*eta);
    REAL4 c2 = 743.0/252.0 + eta*11.0/3.0;
    REAL4 c3 = -32*LAL_PI/3;
    REAL4 c4 = 3058673.0/508032.0 + eta*(5429.0/504.0 + eta*617.0/72.0);
    REAL4 x  = pow(LAL_PI*mtotal*LAL_MTSUN_SI*fmin, 1.0/3.0);
    REAL4 x2 = x*x;
```

```
    REAL4 x3 = x*x2;
    REAL4 x4 = x2*x2;
    REAL4 x8 = x4*x4;

    chirpTime = c0*(1 + c2*x2 + c3*x3 + c4*x4)/x8;

    deltaEventIndex = (UINT4) rint( (chirpTime / deltaT) + 1.0 );
    chirpTimeNS = (INT8) (1e9 * chirpTime);
  }


  buffer = rint( (timingError/deltaT) + 1.0 );

  /* Now construct the appropriate coherent SNR */
  switch (numDetectors)  {
  case 2:
    if(caseID[0] && caseID[5])  /* Network: H1 and H2*/
      {
m = 0;
for (k=0;k<(INT4)numPoints;k++)
  {
    REAL4 cohSNR = 0.0;

    for (m=k-buffer; m<k+buffer; m++)
      {
if(m >=0 && m < (INT4) numPoints)
  {
    cohSNRLocal = sqrt( (cData[0].data->data[k].re +
                         cData[1].data->data[m].re) * (cData[0].data->data[k].re +
                         cData[1].data->data[m].re) + (cData[0].data->data[k].im +
                         cData[1].data->data[m].im) * (cData[0].data->data[k].im +
                         cData[1].data->data[m].im));

    if(cohSNRLocal > cohSNR)
      {
cohSNR = cohSNRLocal;
      }
  }
if( cohSNROut ) params->cohSNRVec->data->data[k]= cohSNR;

if ( cohSNR > cohSNRThresh ) {
  if ( !*eventList ) {
    /* store the start of the crossing */
    eventStartIdx = k;

    /* if this is the first event, start the list */

    thisEvent = *eventList = (MultiInspiralTable *)
      LALCalloc( 1, sizeof(MultiInspiralTable) );

    if ( !thisEvent )
      {
ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
      }

    /* record the data that we need for the clustering algorithm */
    tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
```

```
cData[0].epoch.gpsNanoSeconds + k * deltaT;
    fracpart = modf( tempTime, &intpart );
    thisEvent->end_time.gpsSeconds = (INT4) intpart;
    thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
    thisEvent->snr = cohSNR;
    strcpy(thisEvent->ifos,caseStr);
    thisEvent->mass1 = input->tmplt->mass1;
    thisEvent->mass2 = input->tmplt->mass2;
    thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
    thisEvent->eta = input->tmplt->eta;
                    LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
    thisEvent->eff_distance = distanceEstimate;

    tempTime = 0.0;
    fracpart = 0.0;
    intpart = 0.0;
    fflush( stdout );

  } /* done creating a new event */
  else if (params->maximizeOverChirp && k <= (eventStartIdx +
deltaEventIndex) && cohSNR > thisEvent->snr ) {
    /* if this is the same event, update the maximum */

    tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
    fracpart = modf( tempTime, &intpart );
    thisEvent->end_time.gpsSeconds = (INT4) intpart;
    thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
    thisEvent->snr = cohSNR;
    strcpy(thisEvent->ifos, caseStr);
    thisEvent->mass1 = input->tmplt->mass1;
    thisEvent->mass2 = input->tmplt->mass2;
    thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
    thisEvent->eta = input->tmplt->eta;
    LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
    thisEvent->eff_distance = distanceEstimate;

    tempTime = 0.0;
    fracpart = 0.0;
    intpart = 0.0;
    fflush( stdout );

  }
  else if ( k > (eventStartIdx + deltaEventIndex) ||
!(params->maximizeOverChirp) ) {
    /* clean up this event */
    MultiInspiralTable      *lastEvent = NULL;

    /* allocate memory for the newEvent */
    lastEvent = thisEvent;

    lastEvent->next = thisEvent = (MultiInspiralTable *)
      LALCalloc( 1, sizeof(MultiInspiralTable) );
```

```
    if ( !(lastEvent->next) )
      {
ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
      }

    /* stick minimal data into the event */

    tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
    fracpart = modf( tempTime, &intpart );
    thisEvent->end_time.gpsSeconds = (INT4) intpart;
    thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
    thisEvent->snr = cohSNR;
    strcpy(thisEvent->ifos,caseStr);
    thisEvent->mass1 = input->tmplt->mass1;
    thisEvent->mass2 = input->tmplt->mass2;
    thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
    thisEvent->eta = input->tmplt->eta;
    LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
    thisEvent->eff_distance = distanceEstimate;

    /* Need to initialize the event start index to new value */
    if( k > (eventStartIdx + deltaEventIndex) )
      {
eventStartIdx = k;
      }
    tempTime = 0.0;
    fracpart= 0.0;
    intpart = 0.0;

  }
} /* matches if (cohSNR > cohSNRThresh) */

      } /* matches for(m=k-buffer....) */
  }/* matches for(k=0;... */
      }
    else
      { /* Network: 2 detectors excluding either H1, H2, or both H1 and H2 */
/*Here, the time delay looping must start */

/* Now calculate the distance (in meters) */
distance[1] = sqrt( cartesianInnerProduct(s[1],s[1]) );
timeDelay[1] = distance[1]/LAL_C_SI;
slidePoints[1] = rint( (fabs(timeDelay[1])/deltaT) + 1.0 );

k = 0;
q = 0;
w = 0;

for(k=0;k<(INT4)numPoints;k++)
  {
    REAL4 cohSNR = 0.0;

    for (q = k-slidePoints[1]-buffer; q < k+slidePoints[1]+buffer; q++)
      {
```

```
if(q >= 0 && q < (INT4) numPoints)
  {
    cohSNRLocal = sqrt(cData[0].data->data[k].re *
                       cData[0].data->data[k].re + cData[1].data->data[q].re *
                       cData[1].data->data[q].re + cData[0].data->data[k].im *
                       cData[0].data->data[k].im + cData[1].data->data[q].im *
                       cData[1].data->data[q].im);
    if(cohSNRLocal > cohSNR)
      {
cohSNR = cohSNRLocal;
w = q;
      }
  }
      }
    if( cohSNROut ) params->cohSNRVec->data->data[k] = cohSNR;

    if ( cohSNR > cohSNRThresh ) {
      if ( !*eventList ) {
/* store the start of the crossing */
eventStartIdx = k;

/* if this is the first event, start the list */

thisEvent = *eventList = (MultiInspiralTable *)
  LALCalloc( 1, sizeof(MultiInspiralTable) );

if ( !thisEvent )
  {
    ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
  }

/* record the data that we need for the clustering algorithm */
tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
fracpart = modf( tempTime, &intpart );
thisEvent->end_time.gpsSeconds = (INT4) intpart;
thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
thisEvent->snr = cohSNR;
strcpy(thisEvent->ifos,caseStr);
thisEvent->mass1 = input->tmplt->mass1;
thisEvent->mass2 = input->tmplt->mass2;
thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
thisEvent->eta = input->tmplt->eta;
LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
thisEvent->eff_distance = distanceEstimate;
thisEvent->ligo_angle = acos( LAL_C_SI * deltaT * abs(k-w) / distance[1] );
if( (k-w) > 0 )
  {
    thisEvent->ligo_angle_sig = 1;
  }
else
  {
    thisEvent->ligo_angle_sig = -1;
  }
tempTime = 0.0;
```

```
fracpart = 0.0;
intpart = 0.0;
fflush( stdout );

 } /* done creating a new event */
      else if (params->maximizeOverChirp && k <= (eventStartIdx +
deltaEventIndex) && cohSNR > thisEvent->snr ) {
/* if this is the same event, update the maximum */

tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
fracpart = modf( tempTime, &intpart );
thisEvent->end_time.gpsSeconds = (INT4) intpart;
thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
thisEvent->snr = cohSNR;
strcpy(thisEvent->ifos, caseStr);
thisEvent->mass1 = input->tmplt->mass1;
thisEvent->mass2 = input->tmplt->mass2;
thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
thisEvent->eta = input->tmplt->eta;
LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
thisEvent->eff_distance = distanceEstimate;
thisEvent->ligo_angle = acos( LAL_C_SI * deltaT * abs(k-w) / distance[1] );
if( (k-w) > 0 )
  {
    thisEvent->ligo_angle_sig = 1;
  }
else
  {
    thisEvent->ligo_angle_sig = -1;
  }

tempTime = 0.0;
fracpart = 0.0;
intpart = 0.0;
fflush( stdout );

         }
      else if ( k > (eventStartIdx + deltaEventIndex) ||
!(params->maximizeOverChirp) ) {
/* clean up this event */
MultiInspiralTable      *lastEvent = NULL;

/* allocate memory for the newEvent */
lastEvent = thisEvent;

lastEvent->next = thisEvent = (MultiInspiralTable *)
  LALCalloc( 1, sizeof(MultiInspiralTable) );
if ( !(lastEvent->next) )
  {
    ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
  }

/* stick minimal data into the event */
```

```
tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
fracpart = modf( tempTime, &intpart );
thisEvent->end_time.gpsSeconds = (INT4) intpart;
thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
thisEvent->snr = cohSNR;
strcpy(thisEvent->ifos,caseStr);
thisEvent->mass1 = input->tmplt->mass1;
thisEvent->mass2 = input->tmplt->mass2;
thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
thisEvent->eta = input->tmplt->eta;
LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
thisEvent->eff_distance = distanceEstimate;
thisEvent->ligo_angle = acos( LAL_C_SI * deltaT * abs(k-w) / distance[1] );
if( (k-w) > 0 )
  {
    thisEvent->ligo_angle_sig = 1;
  }
else
  {
    thisEvent->ligo_angle_sig = -1;
  }

/* Need to initialize the event start index to new value */
if( k > (eventStartIdx + deltaEventIndex) )
  {
    eventStartIdx = k;
  }
tempTime = 0.0;
fracpart= 0.0;
intpart = 0.0;

         }
} /* matches if (cohSNR > cohSNRThresh) */
  }
      }
    break;
  case 3: /* Network: 3 detectors including both H1 and H2 */
    if(caseID[0] && caseID[5])
      {
/* Now calculate the distance (in meters) */
distance[1] = sqrt( cartesianInnerProduct( s[1],s[1]) );
timeDelay[1] = distance[1]/LAL_C_SI;
slidePoints[1] = rint( (fabs(timeDelay[1])/deltaT) + 1.0 );

k = 0;
q = 0;
m = 0;
w = 0;

for(k=0;k<(INT4)numPoints;k++)
  {
    REAL4 cohSNR = 0.0;
    for(m=k-buffer;m<k+buffer;m++)
      {
```

```
if(m >=0 && m < (INT4) numPoints)
  {
    for (q = m-slidePoints[1]-buffer;q < m+slidePoints[1]+buffer;q++)
      {
if(q >= 0 && q < (INT4) numPoints)
  {
    cohSNRLocal = sqrt( ((cData[0].data->data[k].re +
                                cData[2].data->data[m].re)*(cData[0].data->data[k].re +
                                cData[2].data->data[m].re) + (cData[0].data->data[k].im
                                + cData[2].data->data[m].im)*(cData[0].data->data[k].im
                                + cData[2].data->data[m].im)) +
                                cData[1].data->data[q].re*cData[1].data->data[q].re +
                                cData[1].data->data[q].im*cData[1].data->data[q].im);

    if(cohSNRLocal > cohSNR)
      {
cohSNR = cohSNRLocal;
w = q;
      }
  }
      }
  }
if( cohSNROut ) params->cohSNRVec->data->data[k] = cohSNR;

if ( cohSNR > cohSNRThresh ) {
  if ( !*eventList ) {
    /* store the start of the crossing */
    eventStartIdx = k;

    /* if this is the first event, start the list */

    thisEvent = *eventList = (MultiInspiralTable *)
      LALCalloc( 1, sizeof(MultiInspiralTable) );

    if ( !thisEvent )
      {
ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
      }

    /* record the data that we need for the clustering algorithm */
    tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
    fracpart = modf( tempTime, &intpart );
    thisEvent->end_time.gpsSeconds = (INT4) intpart;
    thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
    thisEvent->snr = cohSNR;
    strcpy(thisEvent->ifos,caseStr);
    thisEvent->mass1 = input->tmplt->mass1;
    thisEvent->mass2 = input->tmplt->mass2;
    thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
    thisEvent->eta = input->tmplt->eta;
    LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
    thisEvent->eff_distance = distanceEstimate;
    thisEvent->ligo_angle = acos( LAL_C_SI * deltaT * abs(k-w) / distance[1] );
if( (k-w) > 0 )
```

```
  {
    thisEvent->ligo_angle_sig = 1;
  }
else
  {
    thisEvent->ligo_angle_sig = -1;
  }

    tempTime = 0.0;
    fracpart = 0.0;
    intpart = 0.0;
    fflush( stdout );

  } /* done creating a new event */
  else if (params->maximizeOverChirp && k <= (eventStartIdx +
deltaEventIndex) && cohSNR > thisEvent->snr ) {
    /* if this is the same event, update the maximum */

    tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
    fracpart = modf( tempTime, &intpart );
    thisEvent->end_time.gpsSeconds = (INT4) intpart;
    thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
    thisEvent->snr = cohSNR;
    strcpy(thisEvent->ifos, caseStr);
    thisEvent->mass1 = input->tmplt->mass1;
    thisEvent->mass2 = input->tmplt->mass2;
    thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
    thisEvent->eta = input->tmplt->eta;
    LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
    thisEvent->eff_distance = distanceEstimate;
    thisEvent->ligo_angle = acos( LAL_C_SI * deltaT * abs(k-w) / distance[1] );
if( (k-w) > 0 )
  {
    thisEvent->ligo_angle_sig = 1;
  }
else
  {
    thisEvent->ligo_angle_sig = -1;
  }

    tempTime = 0.0;
    fracpart = 0.0;
    intpart = 0.0;
    fflush( stdout );

  }
  else if ( k > (eventStartIdx + deltaEventIndex) ||
!(params->maximizeOverChirp) ) {
    /* clean up this event */
    MultiInspiralTable      *lastEvent = NULL;

    /* allocate memory for the newEvent */
    lastEvent = thisEvent;
```

```
    lastEvent->next = thisEvent = (MultiInspiralTable *)
      LALCalloc( 1, sizeof(MultiInspiralTable) );
    if ( !(lastEvent->next) )
      {
ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
      }

    /* stick minimal data into the event */

    tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
    fracpart = modf( tempTime, &intpart );
    thisEvent->end_time.gpsSeconds = (INT4) intpart;
    thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
    thisEvent->snr = cohSNR;
    strcpy(thisEvent->ifos,caseStr);
    thisEvent->mass1 = input->tmplt->mass1;
    thisEvent->mass2 = input->tmplt->mass2;
    thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
    thisEvent->eta = input->tmplt->eta;
    LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
    thisEvent->eff_distance = distanceEstimate;
    thisEvent->ligo_angle = acos( LAL_C_SI * deltaT * abs(k-w) / distance[1] );
if( (k-w) > 0 )
  {
    thisEvent->ligo_angle_sig = 1;
  }
else
  {
    thisEvent->ligo_angle_sig = -1;
  }

    /* Need to initialize the event start index to new value */
    if( k > (eventStartIdx + deltaEventIndex) )
      {
eventStartIdx = k;
      }
    tempTime = 0.0;
    fracpart= 0.0;
    intpart = 0.0;

  }
} /* matches if (cohSNR > cohSNRThresh) */
      }
  }
      }
    else
      { /* Network: 3 detectors excluding either H1, H2, or both (H1 && H2)*/
/*Now the last 3 cases will involve the looping over the coefficients*/

l = 0;
k = 0;
q = 0;
w = 0;
```

```
for (l=0;l < (INT4) numBeamPoints; l++)
  {
    theta = 0.0;
    phi = 0.0;
    theta = beamVec->detBeamArray[0].thetaPhiVs[l].data->data[0] * (REAL4) LAL_PI_180;
            phi   = beamVec->detBeamArray[0].thetaPhiVs[l].data->data[1] * (REAL4) LAL_PI_180;
    /* position vector to source relative to first detector */
    nHatVect[0] = cos(theta) * sin(phi);
    nHatVect[1] = sin(theta) * sin(phi);
    nHatVect[2] = cos(phi);

    /* Now calculate the distance (in meters) projected along sky-position */
    distance[1] = cartesianInnerProduct(s[1],nHatVect);
    distance[2] = cartesianInnerProduct(s[2],nHatVect);
    timeDelay[1] = distance[1]/LAL_C_SI;
    timeDelay[2] = distance[2]/LAL_C_SI;
    slidePoints[1] = rint( (fabs(timeDelay[1])/deltaT) + 1.0 );
    slidePoints[2] = rint( (fabs(timeDelay[2])/deltaT) + 1.0 );

    for(k=0;k<(INT4)numPoints;k++)
      {
REAL4 cohSNR = 0.0;
qTemp = 0;
wTemp = 0;
for (q = k-slidePoints[1]-buffer;q < k+slidePoints[1]+buffer;q++)
  {
    if(q >= 0 && q < (INT4) numPoints)
      {
for (w = q-slidePoints[2]-buffer; w < q+slidePoints[2]+buffer;w++)
  {
    if (w >= 0 && w < (INT4) numPoints)
      {
      cohSNRLocal = sqrt( (
                                beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] *
                                beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] +
                                beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3] *
                                beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3] )
                                *( cData[0].data->data[k].re*cData[0].data->data[k].re
                                +cData[0].data->data[k].im*cData[0].data->data[k].im )
                                +(beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2]
                                *beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2]
                                + beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3]
                                *beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3])
                                *( cData[1].data->data[q].re*cData[1].data->data[q].re
                                + cData[1].data->data[q].im*cData[1].data->data[q].im)
                                +(beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2]
                                *beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2]
                                + beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3]
                                *beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3])
                                *( cData[2].data->data[w].re*cData[2].data->data[w].re
                                + cData[2].data->data[w].im*cData[2].data->data[w].im)
                                +2*(beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2]
                                *beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] +
                                beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3]
                                *beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3])
                                *(cData[0].data->data[k].re * cData[1].data->data[q].re +
                                cData[0].data->data[k].im * cData[1].data->data[q].im) +
```

```
                                2*(beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2]
                                *beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] +
                                beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3]*
                                beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3])*
                                (cData[0].data->data[k].re*cData[2].data->data[w].re +
                                cData[0].data->data[k].im * cData[2].data->data[w].im) +
                                2*(beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] *
                                beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] +
                                beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3] *
                                beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3]) *
                                (cData[1].data->data[q].re * cData[2].data->data[w].re +
                                cData[1].data->data[q].im *cData[2].data->data[w].im));

if(cohSNRLocal > cohSNR)
  {
    cohSNR = cohSNRLocal;
    qTemp = q;
    wTemp = w;
  }
     }

  }

     }
   if( cohSNROut ) params->cohSNRVec->data->data[k] = cohSNR;

   if ( cohSNR > cohSNRThresh ) {
     if ( !*eventList ) {
       /* store the start of the crossing */
       eventStartIdx = k;

       /* if this is the first event, start the list */

       thisEvent = *eventList = (MultiInspiralTable *)
 LALCalloc( 1, sizeof(MultiInspiralTable) );

       if ( !thisEvent )
 {
   ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
 }

       /* record the data that we need for the clustering algorithm */
       tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
       fracpart = modf( tempTime, &intpart );
       thisEvent->end_time.gpsSeconds = (INT4) intpart;
       thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
       thisEvent->snr = cohSNR;
       strcpy(thisEvent->ifos,caseStr);
       thisEvent->mass1 = input->tmplt->mass1;
       thisEvent->mass2 = input->tmplt->mass2;
       thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
       thisEvent->eta = input->tmplt->eta;

       inclination = 0.0;
       polarization = 0.0;
```

```
       cDataTemp[0].re = cData[0].data->data[k].re;
       cDataTemp[0].im = cData[0].data->data[k].im;
       cDataTemp[1].re = cData[1].data->data[qTemp].re;
       cDataTemp[1].im = cData[1].data->data[qTemp].im;
       cDataTemp[2].re = cData[2].data->data[wTemp].re;
       cDataTemp[2].im = cData[2].data->data[wTemp].im;
       LALCoherentInspiralEstimatePsiEpsilonCoaPhase(
status->statusPtr, caseID, params->segNorm, theta, phi, cDataTemp,
&inclination, &polarization, &coaPhase );
       thisEvent->inclination = inclination;
       thisEvent->polarization = polarization;
       thisEvent->coa_phase = coaPhase;
       LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
       thisEvent->eff_distance = distanceEstimate;
       thisEvent->ligo_axis_ra = phi;
       thisEvent->ligo_axis_dec = theta;

       tempTime = 0.0;
       fracpart = 0.0;
       intpart = 0.0;
       fflush( stdout );

     } /* done creating a new event */
     else if (params->maximizeOverChirp && k <=
(eventStartIdx + deltaEventIndex) && cohSNR > thisEvent->snr ) {
       /* if this is the same event, update the maximum */

       tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
       fracpart = modf( tempTime, &intpart );
       thisEvent->end_time.gpsSeconds = (INT4) intpart;
       thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
       thisEvent->snr = cohSNR;
       strcpy(thisEvent->ifos, caseStr);
       thisEvent->mass1 = input->tmplt->mass1;
       thisEvent->mass2 = input->tmplt->mass2;
       thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
       thisEvent->eta = input->tmplt->eta;

       inclination = 0.0;
       polarization = 0.0;
       cDataTemp[0].re = cData[0].data->data[k].re;
       cDataTemp[0].im = cData[0].data->data[k].im;
       cDataTemp[1].re = cData[1].data->data[qTemp].re;
       cDataTemp[1].im = cData[1].data->data[qTemp].im;
       cDataTemp[2].re = cData[2].data->data[wTemp].re;
       cDataTemp[2].im = cData[2].data->data[wTemp].im;
       LALCoherentInspiralEstimatePsiEpsilonCoaPhase(
status->statusPtr, caseID, params->segNorm, theta, phi, cDataTemp, &inclination,
&polarization, &coaPhase );
       thisEvent->inclination = inclination;
       thisEvent->polarization = polarization;
       thisEvent->coa_phase = coaPhase;
       LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
```

```
       thisEvent->eff_distance = distanceEstimate;
       thisEvent->ligo_axis_ra = phi;
       thisEvent->ligo_axis_dec = theta;

       tempTime = 0.0;
       fracpart = 0.0;
       intpart = 0.0;
       fflush( stdout );

     }
     else if ( k > (eventStartIdx + deltaEventIndex) ||
!(params->maximizeOverChirp) ) {
       /* clean up this event */
       MultiInspiralTable      *lastEvent = NULL;

       /* allocate memory for the newEvent */
       lastEvent = thisEvent;

       lastEvent->next = thisEvent = (MultiInspiralTable *)
         LALCalloc( 1, sizeof(MultiInspiralTable) );
       if ( !(lastEvent->next) )
         {
   ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
         }

       /* stick minimal data into the event */

       tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
       fracpart = modf( tempTime, &intpart );
       thisEvent->end_time.gpsSeconds = (INT4) intpart;
       thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
       thisEvent->snr = cohSNR;
       strcpy(thisEvent->ifos,caseStr);
       thisEvent->mass1 = input->tmplt->mass1;
       thisEvent->mass2 = input->tmplt->mass2;
       thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
       thisEvent->eta = input->tmplt->eta;

       inclination = 0.0;
       polarization = 0.0;
       cDataTemp[0].re = cData[0].data->data[k].re;
       cDataTemp[0].im = cData[0].data->data[k].im;
       cDataTemp[1].re = cData[1].data->data[qTemp].re;
       cDataTemp[1].im = cData[1].data->data[qTemp].im;
       cDataTemp[2].re = cData[2].data->data[wTemp].re;
       cDataTemp[2].im = cData[2].data->data[wTemp].im;
       LALCoherentInspiralEstimatePsiEpsilonCoaPhase(
status->statusPtr, caseID, params->segNorm, theta, phi, cDataTemp, &inclination,
&polarization, &coaPhase );
       thisEvent->inclination = inclination;
       thisEvent->polarization = polarization;
       thisEvent->coa_phase = coaPhase;
       LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
       thisEvent->eff_distance = distanceEstimate;
```

```
        thisEvent->ligo_axis_ra = phi;
        thisEvent->ligo_axis_dec = theta;

        /* Need to initialize the event start index to new value */
        if( k > (eventStartIdx + deltaEventIndex) )
          {
    eventStartIdx = k;
          }
        tempTime = 0.0;
        fracpart= 0.0;
        intpart = 0.0;

      }
     } /* matches if (cohSNR > cohSNRThresh) */
   }
       }
   }/* outer loop end */
       } /* else statement end */
     break;
   case 4: /* Network: 4 detectors including both H1 and H2 */
     if(caseID[0] && caseID[5])
       {
/*start search looping */

for (l=0;l < (INT4) numBeamPoints; l++)
  {
    theta = 0.0;
    phi = 0.0;
    theta = beamVec->detBeamArray[0].thetaPhiVs[l].data->data[0] * (REAL4) LAL_PI_180;
            phi   = beamVec->detBeamArray[0].thetaPhiVs[l].data->data[1] * (REAL4) LAL_PI_180;
    /* position vector to source relative to first detector */
    nHatVect[0] = cos(theta) * sin(phi);
    nHatVect[1] = sin(theta) * sin(phi);
    nHatVect[2] = cos(phi);

    /* Now calculate the distance (in meters) projected along sky-position */
    distance[1] = cartesianInnerProduct(s[1],nHatVect);
    distance[2] = cartesianInnerProduct(s[2],nHatVect);
    timeDelay[1] = distance[1]/LAL_C_SI;
    timeDelay[2] = distance[2]/LAL_C_SI;
    slidePoints[1] = rint( (fabs(timeDelay[1])/deltaT) + 1.0 );
    slidePoints[2] = rint( (fabs(timeDelay[2])/deltaT) + 1.0 );

    k = 0;
    q = 0;
    w = 0;
    m = 0;

    for(k=0;k<(INT4)numPoints;k++)
      {
REAL4 cohSNR = 0.0;
mTemp = 0;
qTemp = 0;
wTemp = 0;
for(m=k-buffer;m<k+buffer;m++)
  {
    if(m >= 0 && m < (INT4) numPoints)
```

```
      {
for (q = m-slidePoints[1]-buffer;q < m+slidePoints[1]+buffer;q++)
  {
    if(q >= 0 && q < (INT4) numPoints)
      {
for (w = q-slidePoints[2]-buffer; w < q+slidePoints[2]+buffer;w++)
  {
    if (w >= 0 && w < (INT4) numPoints)
      {
      cohSNRLocal = sqrt( (
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3]) *
                                        ((cData[0].data->data[k].re + cData[3].data->data[m].re)
                                        *(cData[0].data->data[k].re + cData[3].data->data[m].re)
                                        +(cData[0].data->data[k].im + cData[3].data->data[m].im)
                                        *(cData[0].data->data[k].im + cData[3].data->data[m].im))
                                        +(beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3]) *
                                        ( cData[1].data->data[q].re*cData[1].data->data[q].re +
                                        cData[1].data->data[q].im*cData[1].data->data[q].im ) +
                                        ( beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2]  +
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3] ) *
                                        ( cData[2].data->data[w].re*cData[2].data->data[w].re +
                                        cData[2].data->data[w].im*cData[2].data->data[w].im ) +
                                        2*(beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3]) *
                                        (cData[0].data->data[k].re*cData[1].data->data[q].re +
                                        cData[0].data->data[k].im*cData[1].data->data[q].im +
                                        cData[3].data->data[m].re*cData[1].data->data[q].re +
                                        cData[3].data->data[m].im*cData[1].data->data[q].im) +
                                        2*(beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3]) *
                                        (cData[0].data->data[k].re*cData[2].data->data[w].re +
                                        cData[0].data->data[k].im*cData[2].data->data[w].im +
                                        cData[3].data->data[m].re*cData[2].data->data[w].re +
                                        cData[3].data->data[m].im*cData[2].data->data[w].im) +
                                        2*(beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3]) *
                                        (cData[1].data->data[q].re*cData[2].data->data[w].re +
                                        cData[1].data->data[q].im*cData[2].data->data[w].im));

if(cohSNRLocal > cohSNR)
  {
    cohSNR = cohSNRLocal;
    mTemp = m;
```

```
    qTemp = q;
    wTemp = w;
  }
      }

  }

      }
  }
      }
    if( cohSNROut ) params->cohSNRVec->data->data[k] = cohSNR;

    if ( cohSNR > cohSNRThresh ) {
      if ( !*eventList ) {
/* store the start of the crossing */
eventStartIdx = k;

        /* if this is the first event, start the list */

thisEvent = *eventList = (MultiInspiralTable *)
  LALCalloc( 1, sizeof(MultiInspiralTable) );

if ( !thisEvent )
  {
    ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
  }

/* record the data that we need for the clustering algorithm */
tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
        fracpart = modf( tempTime, &intpart );
        thisEvent->end_time.gpsSeconds = (INT4) intpart;
        thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
        thisEvent->snr = cohSNR;
        strcpy(thisEvent->ifos,caseStr);
        thisEvent->mass1 = input->tmplt->mass1;
        thisEvent->mass2 = input->tmplt->mass2;
thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
thisEvent->eta = input->tmplt->eta;

        inclination = 0.0;
        polarization = 0.0;
        cDataTemp[0].re = cData[0].data->data[k].re;
        cDataTemp[0].im = cData[0].data->data[k].im;
        cDataTemp[1].re = cData[1].data->data[mTemp].re;
        cDataTemp[1].im = cData[1].data->data[mTemp].im;
        cDataTemp[2].re = cData[2].data->data[qTemp].re;
        cDataTemp[2].im = cData[2].data->data[qTemp].im;
        cDataTemp[3].re = cData[3].data->data[wTemp].re;
        cDataTemp[3].im = cData[3].data->data[wTemp].im;
        LALCoherentInspiralEstimatePsiEpsilonCoaPhase(
status->statusPtr, caseID, params->segNorm, theta, phi, cDataTemp, &inclination,
&polarization, &coaPhase );
        thisEvent->inclination = inclination;
        thisEvent->polarization = polarization;
thisEvent->coa_phase = coaPhase;
```

```
        LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
        thisEvent->eff_distance = distanceEstimate;
        thisEvent->ligo_axis_ra = phi;
        thisEvent->ligo_axis_dec = theta;

        tempTime = 0.0;
        fracpart = 0.0;
        intpart = 0.0;
        fflush( stdout );

      } /* done creating a new event */
      else if (params->maximizeOverChirp && k <=
(eventStartIdx + deltaEventIndex) && cohSNR > thisEvent->snr ) {
        /* if this is the same event, update the maximum */

        tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
        fracpart = modf( tempTime, &intpart );
        thisEvent->end_time.gpsSeconds = (INT4) intpart;
        thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
        thisEvent->snr = cohSNR;
        strcpy(thisEvent->ifos, caseStr);
        thisEvent->mass1 = input->tmplt->mass1;
        thisEvent->mass2 = input->tmplt->mass2;
thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
thisEvent->eta = input->tmplt->eta;

        inclination = 0.0;
        polarization = 0.0;
        cDataTemp[0].re = cData[0].data->data[k].re;
        cDataTemp[0].im = cData[0].data->data[k].im;
        cDataTemp[1].re = cData[1].data->data[mTemp].re;
        cDataTemp[1].im = cData[1].data->data[mTemp].im;
        cDataTemp[2].re = cData[2].data->data[qTemp].re;
        cDataTemp[2].im = cData[2].data->data[qTemp].im;
        cDataTemp[3].re = cData[3].data->data[wTemp].re;
        cDataTemp[3].im = cData[3].data->data[wTemp].im;
        LALCoherentInspiralEstimatePsiEpsilonCoaPhase(
status->statusPtr, caseID, params->segNorm, theta, phi, cDataTemp, &inclination,
&polarization, &coaPhase );
        thisEvent->inclination = inclination;
        thisEvent->polarization = polarization;
thisEvent->coa_phase = coaPhase;
        LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
        thisEvent->eff_distance = distanceEstimate;
        thisEvent->ligo_axis_ra = phi;
        thisEvent->ligo_axis_dec = theta;

        tempTime = 0.0;
        fracpart = 0.0;
        intpart = 0.0;
        fflush( stdout );

      }
```

```
      else if ( k > (eventStartIdx + deltaEventIndex) ||
!(params->maximizeOverChirp) ) {
        /* clean up this event */
        MultiInspiralTable      *lastEvent = NULL;

        /* allocate memory for the newEvent */
        lastEvent = thisEvent;

        lastEvent->next = thisEvent = (MultiInspiralTable *)
          LALCalloc( 1, sizeof(MultiInspiralTable) );
        if ( !(lastEvent->next) )
          {
    ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
          }

        /* stick minimal data into the event */

        tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
        fracpart = modf( tempTime, &intpart );
        thisEvent->end_time.gpsSeconds = (INT4) intpart;
        thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
        thisEvent->snr = cohSNR;
        strcpy(thisEvent->ifos,caseStr);
        thisEvent->mass1 = input->tmplt->mass1;
        thisEvent->mass2 = input->tmplt->mass2;
thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
thisEvent->eta = input->tmplt->eta;

        inclination = 0.0;
        polarization = 0.0;
        cDataTemp[0].re = cData[0].data->data[k].re;
        cDataTemp[0].im = cData[0].data->data[k].im;
        cDataTemp[1].re = cData[1].data->data[mTemp].re;
        cDataTemp[1].im = cData[1].data->data[mTemp].im;
        cDataTemp[2].re = cData[2].data->data[qTemp].re;
        cDataTemp[2].im = cData[2].data->data[qTemp].im;
        cDataTemp[3].re = cData[3].data->data[wTemp].re;
        cDataTemp[3].im = cData[3].data->data[wTemp].im;
        LALCoherentInspiralEstimatePsiEpsilonCoaPhase(
status->statusPtr, caseID, params->segNorm, theta, phi, cDataTemp, &inclination,
&polarization, &coaPhase );
        thisEvent->inclination = inclination;
        thisEvent->polarization = polarization;
thisEvent->coa_phase = coaPhase;
        LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
        thisEvent->eff_distance = distanceEstimate;
        thisEvent->ligo_axis_ra = phi;
        thisEvent->ligo_axis_dec = theta;

        /* Need to initialize the event start index to new value */
        if( k > (eventStartIdx + deltaEventIndex) )
          {
    eventStartIdx = k;
          }
```

```
        tempTime = 0.0;
        fracpart= 0.0;
        intpart = 0.0;

      }
    } /* matches if (cohSNR > cohSNRThresh) */
  }
      }
  } /* end for statement prior to computing distances*/
      } /* end outer if statement in case4*/
    else
      { /* Network: 4 detectors excluding either H1, H2, or both H1 and H2 */
/* there will be one extra loop over the last case since there are 3 nonzero
time delays*/

/*start search looping */

for (l=0;l < (INT4) numBeamPoints; l++)
  {
    theta = 0.0;
    phi = 0.0;
    theta = beamVec->detBeamArray[0].thetaPhiVs[l].data->data[0] * (REAL4) LAL_PI_180;
            phi   = beamVec->detBeamArray[0].thetaPhiVs[l].data->data[1] * (REAL4) LAL_PI_180;
    /* position vector to source relative to first detector */
    nHatVect[0] = cos(theta) * sin(phi);
    nHatVect[1] = sin(theta) * sin(phi);
    nHatVect[2] = cos(phi);

    /* Now calculate the distance (in meters) projected along sky-position */
    distance[1] = cartesianInnerProduct(s[1],nHatVect);
    distance[2] = cartesianInnerProduct(s[2],nHatVect);
    distance[3] = cartesianInnerProduct(s[3],nHatVect);
    timeDelay[1] = distance[1]/LAL_C_SI;
    timeDelay[2] = distance[2]/LAL_C_SI;
    timeDelay[3] = distance[3]/LAL_C_SI;
    slidePoints[1] = rint( (fabs(timeDelay[1])/deltaT) + 1.0 );
    slidePoints[2] = rint( (fabs(timeDelay[2])/deltaT) + 1.0 );
    slidePoints[3] = rint( (fabs(timeDelay[3])/deltaT) + 1.0 );

    k = 0;
    q = 0;
    w = 0;
    j = 0;

    for(k=0;k<(INT4)numPoints;k++)
      {
REAL4 cohSNR = 0.0;
qTemp = 0;
wTemp = 0;
jTemp = 0;
for (q = k-slidePoints[1]-buffer;q < k+slidePoints[1]+buffer;q++)
  {
    if(q >= 0 && q < (INT4) numPoints)
      {
for (w = q-slidePoints[2]-buffer; w < q+slidePoints[2]+buffer;w++)
  {
    if (w >= 0 && w < (INT4) numPoints)
```

```
      {
for(j = w-slidePoints[3]-buffer; j < w+slidePoints[3]+buffer; j++)
  {
    if(j >=0 && j < (INT4) numPoints)
      {
      cohSNRLocal = sqrt( (
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3]) *
                                        (cData[0].data->data[k].re*cData[0].data->data[k].re +
                                        cData[0].data->data[k].im*cData[0].data->data[k].im ) +
                                        (beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3]) *
                                        (cData[1].data->data[q].re*cData[1].data->data[q].re +
                                        cData[1].data->data[q].im*cData[1].data->data[q].im ) +
                                        (beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3]) *
                                        (cData[2].data->data[w].re*cData[2].data->data[w].re +
                                        cData[2].data->data[w].im*cData[2].data->data[w].im ) +
                                        (beamVec->detBeamArray[3].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[3].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[3].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[3].thetaPhiVs[l].data->data[3]) *
                                        (cData[3].data->data[j].re*cData[3].data->data[j].re +
                                        cData[3].data->data[j].im*cData[3].data->data[j].im ) +
                                        2*(beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3]) *
                                        (cData[0].data->data[k].re * cData[1].data->data[q].re +
                                        cData[0].data->data[k].im * cData[1].data->data[q].im) +
                                        2*(beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3]) *
                                        (cData[0].data->data[k].re*cData[2].data->data[w].re +
                                        cData[0].data->data[k].im * cData[2].data->data[w].im) +
                                        2*(beamVec->detBeamArray[0].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[3].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[0].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[3].thetaPhiVs[l].data->data[3]) *
                                        (cData[0].data->data[k].re*cData[3].data->data[j].re +
                                        cData[0].data->data[k].im*cData[3].data->data[j].im) +
                                        2*(beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3]) *
                                        (cData[1].data->data[q].re * cData[2].data->data[w].re +
                                        cData[1].data->data[q].im*cData[2].data->data[w].im) +
                                        2*(beamVec->detBeamArray[1].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[3].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[1].thetaPhiVs[l].data->data[3] *
```

```
                                        beamVec->detBeamArray[3].thetaPhiVs[l].data->data[3]) *
                                        (cData[1].data->data[q].re*cData[3].data->data[j].re +
                                        cData[1].data->data[q].im*cData[3].data->data[j].im) +
                                        2*(beamVec->detBeamArray[2].thetaPhiVs[l].data->data[2] *
                                        beamVec->detBeamArray[3].thetaPhiVs[l].data->data[2] +
                                        beamVec->detBeamArray[2].thetaPhiVs[l].data->data[3] *
                                        beamVec->detBeamArray[3].thetaPhiVs[l].data->data[3]) *
                                        (cData[2].data->data[w].re*cData[3].data->data[j].re +
                                        cData[2].data->data[w].im * cData[3].data->data[j].im));

if(cohSNRLocal > cohSNR)
  {
    cohSNR = cohSNRLocal;
    qTemp = q;
    wTemp = w;
    jTemp = j;
  }
      }
  }
      }
  }
      }
    if( cohSNROut ) params->cohSNRVec->data->data[k] = cohSNR;

    if ( cohSNR > cohSNRThresh ) {
      if ( !*eventList ) {
/* store the start of the crossing */
eventStartIdx = k;

/* if this is the first event, start the list */

thisEvent = *eventList = (MultiInspiralTable *)
  LALCalloc( 1, sizeof(MultiInspiralTable) );

if ( !thisEvent )
  {
    ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
  }

/* record the data that we need for the clustering algorithm */
tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
        fracpart = modf( tempTime, &intpart );
        thisEvent->end_time.gpsSeconds = (INT4) intpart;
        thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
        thisEvent->snr = cohSNR;
        strcpy(thisEvent->ifos,caseStr);
        thisEvent->mass1 = input->tmplt->mass1;
        thisEvent->mass2 = input->tmplt->mass2;
thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
thisEvent->eta = input->tmplt->eta;

        inclination = 0.0;
        polarization = 0.0;
        cDataTemp[0].re = cData[0].data->data[k].re;
        cDataTemp[0].im = cData[0].data->data[k].im;
```

```
        cDataTemp[1].re = cData[1].data->data[qTemp].re;
        cDataTemp[1].im = cData[1].data->data[qTemp].im;
        cDataTemp[2].re = cData[2].data->data[wTemp].re;
        cDataTemp[2].im = cData[2].data->data[wTemp].im;
        cDataTemp[3].re = cData[3].data->data[jTemp].re;
        cDataTemp[3].im = cData[3].data->data[jTemp].im;
        LALCoherentInspiralEstimatePsiEpsilonCoaPhase(
status->statusPtr, caseID, params->segNorm, theta, phi, cDataTemp, &inclination,
&polarization, &coaPhase );
        thisEvent->inclination = inclination;
        thisEvent->polarization = polarization;
thisEvent->coa_phase = coaPhase;
        LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
        thisEvent->eff_distance = distanceEstimate;
        thisEvent->ligo_axis_ra = phi;
        thisEvent->ligo_axis_dec = theta;

        tempTime = 0.0;
        fracpart = 0.0;
        intpart = 0.0;
        fflush( stdout );

      } /* done creating a new event */
      else if (params->maximizeOverChirp && k <= (eventStartIdx +
deltaEventIndex) && cohSNR > thisEvent->snr ) {
        /* if this is the same event, update the maximum */

        tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
        fracpart = modf( tempTime, &intpart );
        thisEvent->end_time.gpsSeconds = (INT4) intpart;
        thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
        thisEvent->snr = cohSNR;
        strcpy(thisEvent->ifos, caseStr);
        thisEvent->mass1 = input->tmplt->mass1;
        thisEvent->mass2 = input->tmplt->mass2;
thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
thisEvent->eta = input->tmplt->eta;

        inclination = 0.0;
        polarization = 0.0;
        cDataTemp[0].re = cData[0].data->data[k].re;
        cDataTemp[0].im = cData[0].data->data[k].im;
        cDataTemp[1].re = cData[1].data->data[qTemp].re;
        cDataTemp[1].im = cData[1].data->data[qTemp].im;
        cDataTemp[2].re = cData[2].data->data[wTemp].re;
        cDataTemp[2].im = cData[2].data->data[wTemp].im;
        cDataTemp[3].re = cData[3].data->data[jTemp].re;
        cDataTemp[3].im = cData[3].data->data[jTemp].im;
        LALCoherentInspiralEstimatePsiEpsilonCoaPhase(
status->statusPtr, caseID, params->segNorm, theta, phi, cDataTemp, &inclination,
&polarization, &coaPhase );
        thisEvent->inclination = inclination;
        thisEvent->polarization = polarization;
thisEvent->coa_phase = coaPhase;
```

```
        LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
        thisEvent->eff_distance = distanceEstimate;
        thisEvent->ligo_axis_ra = phi;
        thisEvent->ligo_axis_dec = theta;

        tempTime = 0.0;
        fracpart = 0.0;
        intpart = 0.0;
        fflush( stdout );

      }
      else if ( k > (eventStartIdx + deltaEventIndex) ||
!(params->maximizeOverChirp) ) {
        /* clean up this event */
        MultiInspiralTable      *lastEvent = NULL;

        /* allocate memory for the newEvent */
        lastEvent = thisEvent;

        lastEvent->next = thisEvent = (MultiInspiralTable *)
          LALCalloc( 1, sizeof(MultiInspiralTable) );
        if ( !(lastEvent->next) )
          {
    ABORT( status, FINDCHIRPH_EALOC, FINDCHIRPH_MSGEALOC );
          }

        /* stick minimal data into the event */

        tempTime = cData[0].epoch.gpsSeconds + 1.0e-9 *
cData[0].epoch.gpsNanoSeconds + k * deltaT;
        fracpart = modf( tempTime, &intpart );
        thisEvent->end_time.gpsSeconds = (INT4) intpart;
        thisEvent->end_time.gpsNanoSeconds = (INT4) ( 1.0e9*fracpart );
        thisEvent->snr = cohSNR;
        strcpy(thisEvent->ifos,caseStr);
        thisEvent->mass1 = input->tmplt->mass1;
        thisEvent->mass2 = input->tmplt->mass2;
thisEvent->mchirp = input->tmplt->totalMass *
pow( input->tmplt->eta, 3.0/5.0 );
thisEvent->eta = input->tmplt->eta;

        inclination = 0.0;
        polarization = 0.0;
        cDataTemp[0].re = cData[0].data->data[k].re;
        cDataTemp[0].im = cData[0].data->data[k].im;
        cDataTemp[1].re = cData[1].data->data[qTemp].re;
        cDataTemp[1].im = cData[1].data->data[qTemp].im;
        cDataTemp[2].re = cData[2].data->data[wTemp].re;
        cDataTemp[2].im = cData[2].data->data[wTemp].im;
        cDataTemp[3].re = cData[3].data->data[jTemp].re;
        cDataTemp[3].im = cData[3].data->data[jTemp].im;
        LALCoherentInspiralEstimatePsiEpsilonCoaPhase(
status->statusPtr, caseID, params->segNorm, theta, phi, cDataTemp, &inclination,
&polarization, &coaPhase );
        thisEvent->inclination = inclination;
        thisEvent->polarization = polarization;
```

```
thisEvent->coa_phase = coaPhase;
        LALCoherentInspiralEstimateDistance( status->statusPtr,
params->segNorm, params->templateNorm, deltaT, segmentLength, cohSNR, &distanceEstimate );
        thisEvent->eff_distance = distanceEstimate;
        thisEvent->ligo_axis_ra = phi;
        thisEvent->ligo_axis_dec = theta;

        /* Need to initialize the event start index to new value */
        if( k > (eventStartIdx + deltaEventIndex) )
          {
    eventStartIdx = k;
          }
        tempTime = 0.0;
        fracpart= 0.0;
        intpart = 0.0;

      }
    } /* matches if (cohSNR > cohSNRThresh) */
  }
      }
  } /*end the outermost for statement prior to computing distances*/
      } /*end else statement */
  } /* end case statement */

  /* normal exit */
  DETATCHSTATUSPTR( status );
  RETURN( status );
}
```

Appendix F

coherent_inspiral.c

```
/*-----------------------------------------------------------------------
 *
 * File Name: coherent_inspiral.c
 *
 * Author: Bose, S. and Seader, S. and Rogan, A.
 *
 * Revision: $Id: coherent_inspiral.c,v 1.32 2005/10/16 15:58:36 sseader Exp $
 *
 *-----------------------------------------------------------------------
 */

#include <config.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>

#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <regex.h>
#include <time.h>
#include <math.h>

#include <FrameL.h>
#include <lalapps.h>
#include <series.h>
#include <processtable.h>
#include <lalappsfrutils.h>

#include <lal/LALRCSID.h>
#include <lal/LALConfig.h>
#include <lal/LALStdio.h>
#include <lal/LALStdlib.h>
```

```
#include <lal/LALError.h>
#include <lal/LALDatatypes.h>
#include <lal/AVFactories.h>
#include <lal/LALConstants.h>
#include <lal/FrameStream.h>
#include <lal/DataBuffer.h>
#include <lal/LIGOMetadataTables.h>
#include <lal/LIGOMetadataUtils.h>
#include <lal/LIGOLwXML.h>
#include <lal/LIGOLwXMLRead.h>
#include <lal/Date.h>
#include <lal/Units.h>
#include <lal/FindChirp.h>
#include <lal/FindChirpSP.h>
#include <lal/FindChirpBCV.h>
#include <lal/FindChirpBCVSpin.h>
#include <lal/FindChirpChisq.h>
#include <lal/StochasticCrossCorrelation.h>
#include <lal/DetectorSite.h>
#include <lal/Random.h>
#include <lal/LALInspiral.h>
#include <lal/CoherentInspiral.h>
#include <lal/LALStatusMacros.h>
#include <lal/SkyCoordinates.h>

RCSID( "$Id: coherent_inspiral.c,v 1.32 2005/10/16 15:58:36 sseader Exp $" );

#define CVS_ID_STRING "$Id: coherent_inspiral.c,v 1.32 2005/10/16 15:58:36 sseader Exp $"
#define CVS_REVISION "$Revision: 1.32 $"
#define CVS_SOURCE "$Source: /usr/local/cvs/lscsoft/lalapps/src/inspiral/coherent_inspiral.c,v $"
#define CVS_DATE "$Date: 2005/10/16 15:58:36 $"
#define PROGRAM_NAME "coherent_inspiral"
#define CVS_NAME_STRING "$Name:  $"



static void
TestStatus (LALStatus *status, const char *expectedCodes, int exitCode);

static void
ClearStatus (LALStatus *status);

int arg_parse_check( int argc, char *argv[], MetadataTable procparams );

#define rint(x) (floor((x)+0.5))


/*
 *
 * variables that control program behaviour
 *
 */


/* input data parameters */

INT4   sampleRate           = -1;       /* sample rate of filter data   */
```

```
INT4   numPointsSeg         = -1;/* set to segment-length used in inspiral.c */
REAL4  fLow                 = -1;       /* low frequency cutoff         */
REAL4  dynRangeExponent     = -1;/* set to same value used in inspiral.c */

/*Coherent code specific inputs*/

char   H1filename[256];
char   Lfilename[256];
char   GEOfilename[256];
char   VIRGOfilename[256];
char   TAMAfilename[256];
char   H2filename[256];

INT4 H1file = 0;
INT4 H2file = 0;
INT4 Lfile = 0;
INT4 GEOfile = 0;
INT4 TAMAfile = 0;
INT4 VIRGOfile = 0;

CHAR   bankFileName[FILENAME_MAX]; /* name of input template bank  */
UINT4  cohSNROut            = 0;    /* default is not to write frame */
UINT4  eventsOut            = 0;    /* default is not to write events */
REAL4  cohSNRThresh         = -1;
INT4   maximizeOverChirp    = 0;    /* default is no clustering */
INT4   verbose              = 0;
CHAR   outputPath[FILENAME_MAX];

INT8  gpsStartTimeNS   = 0;         /* input data GPS start time ns */
LIGOTimeGPS gpsStartTime;           /* input data GPS start time    */
INT8  gpsEndTimeNS     = 0;         /* input data GPS end time ns   */
LIGOTimeGPS gpsEndTime;             /* input data GPS end time      */

LALStatus             status;
LALLeapSecAccuracy    accuracy = LALLEAPSEC_LOOSE;

CHAR  *userTag          = NULL;         /* string the user can tag with */
CHAR  *ifoTag           = NULL;         /* string to tag parent IFOs    */
INT4  globFrameData     = 0;            /* glob to get frame data */

/* Params to convert from geocentric to equatorial */

ConvertSkyParams             convertParams;
SkyPosition      tempSky;
MultiInspiralTable           *thisEventTemp = NULL;

int main( int argc, char *argv[] )
{
  FrChanIn      frChan;

  /* frame output data */

  struct FrFile *frOutFile  = NULL;
  struct FrameH *outFrame   = NULL;
  FrStream     *frStream = NULL;

  /* output */
```

```
MetadataTable         proctable;
MetadataTable         procparams;
ProcessParamsTable   *this_proc_param;
LIGOLwXMLStream       results;

FILE *filePtr[4];

CHAR   fileName[FILENAME_MAX];
CHAR   framename[FILENAME_MAX];
CHAR   xmlname[FILENAME_MAX];
CHAR   cohdataStr[LALNameLength];
CHAR   caseIDChars[4][LIGOMETA_IFOS_MAX] = {"0","0","0","0"};

INT4   numTmplts        = 0; /* number of templates */
INT4   cohSegLength     = 4; /* This should match hardcoded value in inspiral.c */
INT4   numPoints        = 0;
INT4   startTemplate    = -1;
INT4   stopTemplate     = -1;
INT4   numChannels      = 0;
INT4   frEnd            = 0;
UINT4  numSegments      = 1;       /* number of segments */
UINT4  numBeamPoints    = 3721;       /* number of sky position templates */
UINT4  nCohDataFr       = 0;
UINT8  eventID          = 0;

REAL4  m1               = 0.0;
REAL4  m2               = 0.0;
REAL4  dynRange         = 0.0;

/* counters and other variables */
INT4   i,j,k,l,w,p;
REAL4  theta,phi,vPlus,vMinus;
UINT4  numDetectors     = 0;
REAL8  tempTime[6]      = {0.0,0.0,0.0,0.0,0.0,0.0};
INT4   timeptDiff[5]    = {0,0,0,0,0};
UINT2  caseID[6]        = {0,0,0,0,0,0}; /* H1 L V G T H2 */
INT4   h1ChanNum        = 0;
INT4   h2ChanNum        = 0;
INT4   lChanNum         = 0;
INT4   geoChanNum       = 0;
INT4   virgoChanNum     = 0;
INT4   tamaChanNum      = 0;
INT4   chanNumArray[6]  = {-1, -1, -1, -1, -1, -1};

FrCache      *frGlobCache = NULL;
FrCache      *frInCache   = NULL;
FrCache      *tempCache   = NULL;
FrCacheSieve  sieve;
FrCacheSieve  tempSieve;

CoherentInspiralInitParams   *cohInspInitParams = NULL;
CoherentInspiralFilterParams *cohInspFilterParams = NULL;
CoherentInspiralFilterInput  *cohInspFilterInput = NULL;
CoherentInspiralBeamVector   *cohInspBeamVec = NULL;
CoherentInspiralCVector      *cohInspCVec = NULL;
MultiInspiralTable           *thisEvent = NULL;
```

```
MultiInspiralTable           *tempTable = NULL;
MetadataTable                savedEvents;
InspiralTemplate             *bankHead = NULL;
InspiralTemplate             *bankTemp = NULL;
InspiralTemplate             *bankTemp2 = NULL;
COMPLEX8TimeSeries            tempSnippet;
REAL4FrequencySeries         *segNormVector = NULL;

char namearray[6][256]  = {"0","0","0","0","0","0"}; /* input frame files */
char namearray2[6][256] = {"0","0","0","0","0","0"}; /* beam files */
char namearray3[6][256] = {"0","0","0","0","0","0"}; /* cData chan names */
char namearray4[6][256] = {"0","0","0","0","0","0"}; /* segNorm chan names */

set_debug_level( "1" ); /* change with parse option */

/* create the process and process params tables */
proctable.processTable = (ProcessTable *) calloc( 1, sizeof(ProcessTable) );
LAL_CALL( LALGPSTimeNow ( &status, &(proctable.processTable->start_time),
      &accuracy ), &status );
LAL_CALL( populate_process_table( &status, proctable.processTable,
      PROGRAM_NAME, CVS_REVISION, CVS_SOURCE, CVS_DATE ), &status );
this_proc_param = procparams.processParamsTable = (ProcessParamsTable *)
  calloc( 1, sizeof(ProcessParamsTable) );

arg_parse_check( argc, argv, procparams );
if (verbose)  fprintf(stdout, "called parse options..\n");

/* wind to the end of the process params table */
for ( this_proc_param = procparams.processParamsTable; this_proc_param->next;
    this_proc_param = this_proc_param->next );

fprintf(stdout,"Reading templates from %s\n",bankFileName);

/* read in the template bank from a ligo lw xml file */
  numTmplts = InspiralTmpltBankFromLIGOLw( &bankHead, bankFileName,
    startTemplate, stopTemplate );
if ( numTmplts < 0 )
{
  fprintf( stderr, "error: unable to read templates from %s\n",
      bankFileName );
  goto cleanexit;
}
else if ( numTmplts == 0 )
{
  /* if there are no tmplts, exit */
    fprintf( stderr, "no templates found in template bank file: %s\n"
      "exiting without searching for events.\n", bankFileName );
  goto cleanexit;
}

if ( verbose ) fprintf( stdout, "parsed %d templates from %s\n",
    numTmplts, bankFileName );

/* Now glob for frame data based on the gps start and duration in the */
/* thinca input file name */

if( globFrameData )
```

```
    {
      if ( verbose ) fprintf( stdout, "globbing for *.gwf frame files in current directory\n");
      LAL_CALL( LALFrCacheGenerate( &status, &frGlobCache, NULL, NULL ),
&status );
      /* check we globbed at least one frame file */
      if ( ! frGlobCache->numFrameFiles )
{
  fprintf( stderr, "error: no frame file files found\n");
  exit( 1 );
}

      /* sieve out the requested data type */
      memset( &sieve, 0, sizeof(FrCacheSieve) );
      sieve.srcRegEx = NULL;
      sieve.dscRegEx = NULL;
      sieve.urlRegEx = NULL;
      sieve.earliestTime = gpsStartTime.gpsSeconds;
      sieve.latestTime = gpsEndTime.gpsSeconds;
      LAL_CALL( LALFrCacheSieve( &status, &frInCache, frGlobCache, &sieve ),
&status );
      if( verbose ) fprintf(stdout,"num files after sieve: %d\n",frInCache->numFrameFiles);
      LAL_CALL( LALDestroyFrCache( &status, &frGlobCache ), &status );
      /* open the input data frame stream from the frame cache */
      /*LAL_CALL( LALFrCacheOpen( &status, &frStream, frInCache ), &status );*/

      /* set the mode of the frame stream to fail on gaps or time errors */
      /*frStream->mode = LAL_FR_DEFAULT_MODE;*/
    }

  /* Set the dynamic range */
  dynRange = pow( 2.0, dynRangeExponent );

  /* Loop over templates (or event id's) */

  numPoints = sampleRate * cohSegLength;
  bankTemp = bankHead;
  savedEvents.multiInspiralTable = NULL;
  k = 0;

  if( !globFrameData )
    {
      h1ChanNum        = -1;
      h2ChanNum        = -1;
      lChanNum         = -1;
      geoChanNum       = -1;
      virgoChanNum     = -1;
      tamaChanNum      = -1;
    }

  for( i=0; i<numTmplts; i++)
    {
      memcpy( caseIDChars[k], &bankTemp->ifo, sizeof(caseIDChars[k] - 1) );
      if( verbose ) fprintf(stdout,"caseIDChars = %s %s %s %s\n",caseIDChars[0],
caseIDChars[1],caseIDChars[2],caseIDChars[3] );
      eventID = bankTemp->event_id->id;
      if( verbose ) fprintf(stdout,"eventID = %Ld\n",eventID );
      if( i != numTmplts - 1 )
```

```
        {
          bankTemp2 = bankTemp;
          bankTemp = bankTemp->next;
        }
      if( i != 0 && ((bankTemp->event_id->id != eventID) || (i == numTmplts - 1)))
{
  if( k > 0 )
    {
      /* Here we should combine data for the last event */
      if( verbose ) fprintf(stdout,"combining data for the last event\n");
      for(j=0;j<4;j++)
{
  if( !strcmp( caseIDChars[j],"H1" ) )
    {
      caseID[0] = 1;
      if( !globFrameData ) h1ChanNum++;
    }
  else if( !strcmp( caseIDChars[j], "L1" ) )
    {
      caseID[1] = 1;
      if( !globFrameData ) lChanNum++;
    }
  else if( !strcmp( caseIDChars[j], "V1" ) )
    {
      caseID[2] = 1;
      if( !globFrameData ) virgoChanNum++;
    }
  else if( !strcmp( caseIDChars[j], "G1" ) )
    {
              caseID[3] = 1;
      if( !globFrameData ) geoChanNum++;
    }
  else if( !strcmp( caseIDChars[j], "T1" ) )
    {
      caseID[4] = 1;
      if( !globFrameData ) tamaChanNum++;
    }
  else if( !strcmp( caseIDChars[j], "H2" ) )
    {
      caseID[5] = 1;
      if( !globFrameData ) h2ChanNum++;
    }
}


      /* Now get the number of detectors */
      l = 0;
      for(j=0;j<6;j++)
        {
          if(caseID[j])
            {
              l++;
    }
}
      numDetectors = l;

      /* Now check that the number of detectors matches the number of frame files provided */
```

```
      l=0;
      if( H1file ) l++;
      if( H2file ) l++;
      if( Lfile ) l++;
      if( GEOfile ) l++;
      if( TAMAfile ) l++;
      if( VIRGOfile ) l++;

      if( ! globFrameData )
{
  if( (INT4)numDetectors != l )
    {
      fprintf( stderr, "You have events for %d detectors, but
specified frame files for %d detectors\n",numDetectors,l);
      if( (INT4)numDetectors > l )
{
  fprintf( stderr, "You must specify more frame files. Exiting...\n");
  exit(1);
}
      else
{
  if( verbose ) fprintf( stdout, "One or more of the frame files
specified will not be used for this event since the number of detectors is less than the number
of frame files you specified.\n");
}
    }
}

      l = 0;

      if( verbose ) fprintf(stdout,"numDetectors = %d\n", numDetectors);
      if( verbose ) fprintf(stdout,"caseID = %d %d %d %d %d %d (H1,L1,V1,G1,T1,H2)\n",
caseID[0], caseID[1], caseID[2], caseID[3], caseID[4], caseID[5]);


      /* Initialize the necessary structures */

      if( !(cohInspInitParams = (CoherentInspiralInitParams *)
calloc(1,sizeof(CoherentInspiralInitParams)) ))
{
  fprintf( stdout, "could not allocate memory for coherentInspiral init params\n" );
  goto cleanexit;
}

      cohInspInitParams->numDetectors            = numDetectors;
      cohInspInitParams->numSegments             = numSegments;
      cohInspInitParams->numPoints               = numPoints;
      cohInspInitParams->numBeamPoints           = numBeamPoints;
      cohInspInitParams->cohSNROut               = cohSNROut;


      /* create the data structures needed for coherentInspiral */

      if ( verbose ) fprintf( stdout, "initializing coherentInspiral...\n " );

      /* initialize coherentInspiral filter functions */
```

```
              LAL_CALL( LALCoherentInspiralFilterInputInit (&status, &cohInspFilterInput,
cohInspInitParams), &status );

      m1 = bankTemp2->mass1;
      m2 = bankTemp2->mass2;

      cohInspFilterInput->tmplt = (InspiralTemplate *)
 LALCalloc(1,sizeof(InspiralTemplate) );
      cohInspFilterInput->tmplt->mass1 = m1;
      cohInspFilterInput->tmplt->mass2 = m2;
      cohInspFilterInput->tmplt->totalMass = m1 + m2;
      cohInspFilterInput->tmplt->mu = m1 * m2 / (m1 + m2);
      cohInspFilterInput->tmplt->eta = (m1 * m2) / ((m1 + m2) * (m1 + m2 ));

      if (verbose)  fprintf( stdout, "m1:%f m2:%f totalmass:%f mu:%f eta%f\n",
m1, m2,cohInspFilterInput->tmplt->totalMass,cohInspFilterInput->tmplt->mu,
cohInspFilterInput->tmplt->eta);

      LAL_CALL( LALCoherentInspiralFilterParamsInit (&status, &cohInspFilterParams,
     cohInspInitParams),&status );

      cohInspFilterParams->deltaT                 = 1/((REAL4) sampleRate);
      cohInspFilterParams->cohSNRThresh           = cohSNRThresh;
      cohInspFilterParams->cohSNROut              = cohSNROut;
      cohInspFilterParams->numTmplts              = 1;
      cohInspFilterParams->fLow                   = fLow;
      cohInspFilterParams->maximizeOverChirp      = maximizeOverChirp;

      for( j=0; j<6; j++ )
{
  cohInspFilterParams->detIDVec->data[j] = caseID[j];
       }

      w=0;
      for ( j=0; j<6; j++ )
{
  if ( caseID[j] )
    {
      cohInspFilterParams->detectorVec->detector[w++] = lalCachedDetectors[j];
    }
}

      if (caseID[5])
{
  cohInspFilterParams->detectorVec->detector[numDetectors-1] =
lalCachedDetectors[0];
}


      /* Get the data that corresponds to this event from the glob cache */
              /* or from the specified frame files. */
              /* First, the file names and channel names must be set correctly */

      if( globFrameData )
{
  if( caseID[0] )
    {
```

```
      memset( &tempSieve, 0, sizeof(FrCacheSieve) );
      /*memset( &tempCache, 0, sizeof(FrCache) );*/
      tempSieve.srcRegEx = "H1";
      tempSieve.dscRegEx = "INSPIRAL";
      tempSieve.urlRegEx = NULL;
      tempSieve.earliestTime = bankTemp2->end_time.gpsSeconds;
      tempSieve.latestTime = bankTemp2->end_time.gpsSeconds + 1;
      LAL_CALL( LALFrCacheSieve( &status, &tempCache,
       frInCache, &tempSieve ), &status );
      if( tempCache->numFrameFiles != 1 )
{
  fprintf(stderr,"CacheSieve should only have returned 1
frame file. Exiting..\n");
  goto cleanexit;
}

      /* Need to strip the file name out of the url */
      char *tempName = NULL;
      char tempName2[256];
      tempName = strtok(tempCache->frameFiles->url,"//");
      tempName = strtok(NULL,"/");
      while( tempName != NULL)
{
  strcpy(tempName2,tempName);
  tempName = strtok(NULL,"/");
}
      tempName = NULL;
      if( !strcmp( tempName2, namearray[0] ) )
{
  /* if this file has been used, increment the chan # */
  h1ChanNum++;
}
      else
{
  h1ChanNum = 0;
  strcpy( namearray[0], tempName2 );
}
      strcpy(namearray2[0],"HanfordBeamCoeff.dat");
      LALSnprintf( namearray3[0], LALNameLength*sizeof(CHAR),
"H1:LSC-AS_Q_CData_%d", h1ChanNum );
      LALSnprintf( namearray4[0], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", h1ChanNum );
      LAL_CALL( LALDestroyFrCache(&status, &tempCache), &status );
      tempCache = NULL;
    }

  if( caseID[1] )
    {
      memset( &tempSieve, 0, sizeof(FrCacheSieve) );
      tempSieve.srcRegEx = "L1";
      tempSieve.dscRegEx = "INSPIRAL";
      tempSieve.urlRegEx = NULL;
      tempSieve.earliestTime = bankTemp2->end_time.gpsSeconds;
      tempSieve.latestTime = bankTemp2->end_time.gpsSeconds + 1;
      LAL_CALL( LALFrCacheSieve( &status, &tempCache,
       frInCache, &tempSieve ), &status );
      if( tempCache->numFrameFiles != 1 )
```

```
{
  fprintf(stderr,"CacheSieve should only have returned 1 frame file.
Exiting..\n");
  goto cleanexit;
}

      /* Need to strip the file name out of the url */
      char *tempName = NULL;
      char tempName2[256];
      tempName = strtok(tempCache->frameFiles->url,"//");
      tempName = strtok(NULL,"/");
      while( tempName != NULL)
{
  strcpy(tempName2,tempName);
  tempName = strtok(NULL,"/");
}
      tempName = NULL;
      if( !strcmp( tempName2, namearray[1] ) )
{
  /* if this file has been used, increment the chan # */
  lChanNum++;
}
      else
{
  lChanNum = 0;
  strcpy( namearray[1], tempName2 );
}
      strcpy(namearray2[1],"LivingstonBeamCoeff.dat");
      LALSnprintf( namearray3[1], LALNameLength*sizeof(CHAR),
"L1:LSC-AS_Q_CData_%d", lChanNum );
      LALSnprintf( namearray4[1], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", lChanNum );
      LAL_CALL( LALDestroyFrCache(&status, &tempCache), &status );
      tempCache = NULL;
    }

  if( caseID[2] )
    {
      memset( &tempSieve, 0, sizeof(FrCacheSieve) );
      tempSieve.srcRegEx = "V1";
      tempSieve.dscRegEx = "INSPIRAL";
      tempSieve.urlRegEx = NULL;
      tempSieve.earliestTime = bankTemp2->end_time.gpsSeconds;
      tempSieve.latestTime = bankTemp2->end_time.gpsSeconds + 1;
      LAL_CALL( LALFrCacheSieve( &status, &tempCache,
       frInCache, &tempSieve ), &status );
      if( tempCache->numFrameFiles != 1 )
{
  fprintf(stderr,"CacheSieve should only have returned 1 frame file.
Exiting..\n");
  goto cleanexit;
}
      /* Need to strip the file name out of the url */
      char *tempName = NULL;
      char tempName2[256];
      tempName = strtok(tempCache->frameFiles->url,"//");
      tempName = strtok(NULL,"/");
```

```
      while( tempName != NULL)
{
  strcpy(tempName2,tempName);
  tempName = strtok(NULL,"/");
}
      tempName = NULL;
      if( !strcmp( tempName2, namearray[2] ) )
{
  /* if this file has been used, increment the chan # */
  virgoChanNum++;
}
      else
{
  virgoChanNum = 0;
  strcpy( namearray[2], tempName2 );
}
      strcpy(namearray2[2],"VirgoBeamCoeff.dat");
      LALSnprintf( namearray3[2], LALNameLength*sizeof(CHAR),
"V1:LSC-AS_Q_CData_%d", virgoChanNum );
      LALSnprintf( namearray4[2], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", virgoChanNum );
      LAL_CALL( LALDestroyFrCache(&status, &tempCache), &status );
      tempCache = NULL;
    }

  if( caseID[3] )
    {
      memset( &tempSieve, 0, sizeof(FrCacheSieve) );
      tempSieve.srcRegEx = "G1";
      tempSieve.dscRegEx = "INSPIRAL";
      tempSieve.urlRegEx = NULL;
      tempSieve.earliestTime = bankTemp2->end_time.gpsSeconds;
      tempSieve.latestTime = bankTemp2->end_time.gpsSeconds + 1;
      LAL_CALL( LALFrCacheSieve( &status, &tempCache,
       frInCache, &tempSieve ), &status );
      if( tempCache->numFrameFiles != 1 )
{
  fprintf(stderr,"CacheSieve should only have returned 1 frame file.
Exiting..\n");
  goto cleanexit;
}
      /* Need to strip the file name out of the url */
      char *tempName = NULL;
      char tempName2[256];
      tempName = strtok(tempCache->frameFiles->url,"//");
      tempName = strtok(NULL,"/");
      while( tempName != NULL)
{
  strcpy(tempName2,tempName);
  tempName = strtok(NULL,"/");
}
      tempName = NULL;
      if( !strcmp(tempName2, namearray[3]) )
{
  /* if this file has been used, increment the chan # */
  geoChanNum++;
}
```

```
      else
{
  geoChanNum = 0;
  strcpy( namearray[3], tempName2 );
}
      strcpy(namearray2[3],"GeoBeamCoeff.dat");
      LALSnprintf( namearray3[3], LALNameLength*sizeof(CHAR),
"G1:DER_DATA_H_CData_%d", geoChanNum );
      LALSnprintf( namearray4[3], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", geoChanNum );
      LAL_CALL( LALDestroyFrCache(&status, &tempCache), &status );
      tempCache = NULL;
    }

  if( caseID[4] )
    {
      memset( &tempSieve, 0, sizeof(FrCacheSieve) );
      tempSieve.srcRegEx = "T1";
      tempSieve.dscRegEx = "INSPIRAL";
      tempSieve.urlRegEx = NULL;
      tempSieve.earliestTime = bankTemp2->end_time.gpsSeconds;
      tempSieve.latestTime = bankTemp2->end_time.gpsSeconds + 1;
      LAL_CALL( LALFrCacheSieve( &status, &tempCache,
       frInCache, &tempSieve ), &status );
      if( tempCache->numFrameFiles != 1 )
{
  fprintf(stderr,"CacheSieve should only have returned 1 frame file.
Exiting..\n");
  goto cleanexit;
}
      /* Need to strip the file name out of the url */
      char *tempName = NULL;
      char tempName2[256];
      tempName = strtok(tempCache->frameFiles->url,"//");
      tempName = strtok(NULL,"/");
      while( tempName != NULL)
{
  strcpy(tempName2,tempName);
  tempName = strtok(NULL,"/");
}
      tempName = NULL;
      if( !strcmp(tempName2, namearray[4]) )
{
  /* if this file has been used, increment the chan # */
  tamaChanNum++;
}
      else
{
  tamaChanNum = 0;
  strcpy( namearray[4], tempName2 );
}
      strcpy(namearray2[4],"TamaBeamCoeff.dat");
      LALSnprintf( namearray3[4], LALNameLength*sizeof(CHAR),
"T1:LSC-AS_Q_CData_%d", tamaChanNum );
      LALSnprintf( namearray4[4], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", tamaChanNum );
      LAL_CALL( LALDestroyFrCache(&status, &tempCache), &status );
```

```
      tempCache = NULL;
    }

  if( caseID[5] )
    {
      memset( &tempSieve, 0, sizeof(FrCacheSieve) );
      tempSieve.srcRegEx = "H2";
      tempSieve.dscRegEx = "INSPIRAL";
      tempSieve.urlRegEx = NULL;
      tempSieve.earliestTime = bankTemp2->end_time.gpsSeconds;
      tempSieve.latestTime = bankTemp2->end_time.gpsSeconds + 1;
      LAL_CALL( LALFrCacheSieve( &status, &tempCache,
       frInCache, &tempSieve ), &status );
      if( tempCache->numFrameFiles != 1 )
{
  fprintf(stderr,"CacheSieve should only have returned 1 frame file.
Exiting..\n");
  goto cleanexit;
}
      /* Need to strip the file name out of the url */
      char *tempName = NULL;
      char tempName2[256];
      tempName = strtok(tempCache->frameFiles->url,"//");
      tempName = strtok(NULL,"/");
      while( tempName != NULL)
{
  strcpy(tempName2,tempName);
  tempName = strtok(NULL,"/");
}
      tempName = NULL;
      if( !strcmp(tempName2, namearray[5]) )
{
  /* if this file has been used, increment the chan # */
  h2ChanNum++;
}
      else
{
  h2ChanNum = 0;
  strcpy( namearray[5], tempName2 );
}
      strcpy(namearray2[5],"HanfordBeamCoeff.dat");
      LALSnprintf( namearray3[5], LALNameLength*sizeof(CHAR),
"H2:LSC-AS_Q_CData_%d", h2ChanNum );
      LALSnprintf( namearray4[5], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", h2ChanNum );
      LAL_CALL( LALDestroyFrCache(&status, &tempCache), &status );
      tempCache = NULL;
    }

  chanNumArray[0] = h1ChanNum;
  chanNumArray[1] = lChanNum;
          chanNumArray[2] = virgoChanNum;
          chanNumArray[3] = geoChanNum;
          chanNumArray[4] = tamaChanNum;
          chanNumArray[5] = h2ChanNum;

}
```

```
      else
{
  /* If we arent globbing, names need to be set differently*/

  if(caseID[0])
    {
      strcpy(namearray[0],H1filename);
      strcpy(namearray2[0],"HanfordBeamCoeff.dat");
      LALSnprintf( namearray3[0], LALNameLength*sizeof(CHAR),
"H1:LSC-AS_Q_CData_%d", h1ChanNum );
      LALSnprintf( namearray4[0], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", h1ChanNum );
    }

                  if(caseID[1])
                    {
                      strcpy(namearray[1],Lfilename);
                      strcpy(namearray2[1],"LivingstonBeamCoeff.dat");
                      LALSnprintf( namearray3[1], LALNameLength*sizeof(CHAR),
"L1:LSC-AS_Q_CData_%d", lChanNum );
      LALSnprintf( namearray4[1], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", lChanNum );
                    }

                  if(caseID[2])
            {
      strcpy(namearray[2],VIRGOfilename);
      strcpy(namearray2[2],"VirgoBeamCoeff.dat");
      LALSnprintf(namearray3[2], LALNameLength*sizeof(CHAR),
"V1:LSC-AS_Q_CData_%d", virgoChanNum );
      LALSnprintf( namearray4[2], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", virgoChanNum );
    }

          if(caseID[3])
    {
      strcpy(namearray[3],GEOfilename);
      strcpy(namearray2[3],"GeoBeamCoeff.dat");
      LALSnprintf(namearray3[3], LALNameLength*sizeof(CHAR),
"G1:DER_DATA_H_CData_%d", geoChanNum );
      LALSnprintf( namearray4[3], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", geoChanNum );
    }

                  if(caseID[4])
                    {
                      strcpy(namearray[4],TAMAfilename);
                      strcpy(namearray2[4],"TamaBeamCoeff.dat");
                      LALSnprintf(namearray3[4], LALNameLength*sizeof(CHAR),
"T1:LSC-AS_Q_CData_%d", tamaChanNum );
      LALSnprintf( namearray4[4], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", tamaChanNum );
                    }

                  if(caseID[5])
                    {
                      strcpy(namearray[5],H2filename);
```

```
                    strcpy(namearray2[5],"HanfordBeamCoeff.dat");
                    LALSnprintf(namearray3[5], LALNameLength*sizeof(CHAR),
"H2:LSC-AS_Q_CData_%d", h2ChanNum );
      LALSnprintf( namearray4[5], LALNameLength*sizeof(CHAR),
"_SegNorm_%d", h2ChanNum );
    }

  chanNumArray[0] = h1ChanNum;
  chanNumArray[1] = lChanNum;
          chanNumArray[2] = virgoChanNum;
          chanNumArray[3] = geoChanNum;
          chanNumArray[4] = tamaChanNum;
          chanNumArray[5] = h2ChanNum;

}

      if(verbose) fprintf(stdout,"built namearrays\n");

              if(verbose) {
                  fprintf(stdout,"frame files: %s\n %s\n %s\n %s\n %s\n %s\n",
namearray[0],namearray[1],namearray[2],namearray[3],namearray[4],namearray[5]);
                  fprintf(stdout,"channels: %s\n %s\n %s\n %s\n %s\n %s\n",
namearray3[0],namearray3[1],namearray3[2],namearray3[3],namearray3[4],namearray3[5]);
  }

              /* get beam pattern coefficients if necessary */
      if ( (numDetectors == 3 && !( caseID[0] && caseID[5])) || numDetectors == 4 )
{
  cohInspBeamVec = cohInspFilterInput->beamVec;
  w=0;
  if( verbose ) fprintf(stdout, "This network requires beam-pattern
coefficients - reading them in...\n");
  for ( j=0; j<6; j++ ) {
    if ( caseID[j] ) {
      filePtr[w] = fopen(namearray2[j], "r");
      if(!filePtr[w])
{
  fprintf(stderr,"The file %s containing the coefficients could
not be found - exiting...\n",namearray2[j]);
  goto cleanexit;
}
      for ( l=0 ; l < (INT4) numBeamPoints ; l++)
{
  fscanf(filePtr[w],"%f %f %f %f",&theta,&phi,&vPlus,&vMinus);
  cohInspBeamVec->detBeamArray[w].thetaPhiVs[l].data->data[0] = theta;
  cohInspBeamVec->detBeamArray[w].thetaPhiVs[l].data->data[1] = phi;
  cohInspBeamVec->detBeamArray[w].thetaPhiVs[l].data->data[2] = vPlus;
  cohInspBeamVec->detBeamArray[w].thetaPhiVs[l].data->data[3] = vMinus;
}
      fclose(filePtr[w++]);
    }
  }
}
      cohInspCVec = cohInspFilterInput->multiCData;

              /* Read in the snippets associated with this event */
      l = 0;
```

```
      for( j=0; j<6; j++ )
{
  if( caseID[j] )
    {
      if( verbose ) fprintf(stdout, "getting the COMPLEX8TimeSeries\n");
                     LAL_CALL( LALFrOpen(&status,&frStream,NULL,namearray[j]), &status);
      if(!frStream)
{
  fprintf(stdout,"The file %s does not exist - exiting...\n", namearray[j]);
  goto cleanexit;
}

      /* Since there is a segnorm frame written at the front */
                     /* of the frame file for each c-data frame, I need to  */
                     /* determine the number of frames contained in the     */
                     /* frame file so that I can advance the stream past    */
                     /* the segNorm vectors                                 */

      p = 0;
      frEnd = 0;
      while( !frEnd )
{
  LAL_CALL( LALFrEnd( &status, &frEnd, frStream), &status);
  if( frEnd == 0 )
    {
      LAL_CALL( LALFrNext( &status,frStream ), &status);
      p++;
    }
  LAL_CALL( LALFrEnd( &status, &frEnd, frStream), &status);
}
      numChannels = p;
      LAL_CALL( LALFrRewind( &status, frStream ), &status );

      /* get segnorm */

for( w=0; w<chanNumArray[j]; w++)
  {
    LAL_CALL( LALFrNext( &status,frStream ), &status );
  }
frChan.name = namearray4[j];
segNormVector = (REAL4FrequencySeries *)
    LALCalloc( 1, sizeof(REAL4FrequencySeries) );
LAL_CALL( LALFrGetREAL4FrequencySeries( &status,
                            segNormVector, &frChan, frStream), &status);

REAL4 fFinal = 1.0 / (6.0 * sqrt(6.0) * LAL_PI *
cohInspFilterInput->tmplt->totalMass * LAL_MTSUN_SI);
REAL4 deltaF = 1.0 / ( cohInspFilterParams->deltaT * (REAL4) numPointsSeg );
                        INT4 kmax = fFinal / deltaF < numPointsSeg/2 ?
                                    fFinal / deltaF : numPointsSeg/2;
cohInspFilterParams->segNorm[l] = segNormVector->data->data[kmax];
LAL_CALL( LALDestroyVector( &status, &(segNormVector->data) ), &status );
LALFree( segNormVector );
segNormVector = NULL;
LAL_CALL( LALFrRewind( &status, frStream ), &status );
```

```
      /* Advance the stream to the appropriate frame */
      for( w=0; w<chanNumArray[j]+numChannels/2; w++)
{
  LAL_CALL( LALFrNext( &status,frStream ), &status );
}
      frChan.name = namearray3[j];
      LAL_CALL( LALFrGetCOMPLEX8TimeSeries( &status, &(cohInspCVec->cData[l]),
&frChan, frStream), &status);
      tempTime[l] = cohInspCVec->cData[l].epoch.gpsSeconds +
cohInspCVec->cData[l].epoch.gpsNanoSeconds * 1e-9;
      if( verbose ) fprintf(stdout,"tempTime = %f\n",tempTime[l]);
      LAL_CALL( LALFrClose( &status, &frStream ), &status );
      l++;
    }
}

      /* If we can estimate distance then compute templateNorm */
              /* At present, this is only good for frequency domain tmplts */
              /* Since each detectors data has been filtered with a templates*/
              /* that have the same mass pair, templateNorm is the same for */
              /* every detector and needs to be computed only once.         */

REAL4 cannonDist = 1.0; /* Mpc */
REAL4 m  = cohInspFilterInput->tmplt->totalMass;
REAL4 mu = cohInspFilterInput->tmplt->mu;
REAL4 deltaT = cohInspFilterParams->deltaT;
REAL4 distNorm = 2.0 *
                    LAL_MRSUN_SI / (cannonDist * 1e6 * LAL_PC_SI);
                REAL4 templateNorm = sqrt( (5.0*mu) / 96.0 ) *
                    pow( m / (LAL_PI*LAL_PI) , 1.0/3.0 ) *
                    pow( LAL_MTSUN_SI / deltaT, -1.0/6.0 );
distNorm *= dynRange;
templateNorm *= templateNorm;
templateNorm *= distNorm * distNorm;
cohInspFilterParams->templateNorm = templateNorm;
cohInspFilterParams->segmentLength = numPointsSeg;

      if (verbose) fprintf(stdout,"filtering the data..\n");
      if ( maximizeOverChirp && verbose )
{
  fprintf(stdout,"clustering events\n");
}

      /* Before the data gets filtered, I need to make the c-data snippets commensurate */
      for(j=0;j<(INT4)numDetectors - 1;j++)
{
  timeptDiff[j] = rint((tempTime[0] - tempTime[j+1]) * sampleRate);
  if( verbose ) fprintf(stdout,"timeptDiff = %d\n",timeptDiff[j]);
}

      /* Now allocate memory for a temporary storage vector */
      memset( &tempSnippet, 0, sizeof(COMPLEX8TimeSeries) );
      LAL_CALL( LALCCreateVector( &status, &(tempSnippet.data), numPoints ), &status );

      /* The following switch statement accomplishes the commensuration of the time series */
      switch( numDetectors )
{
```

```
          case 2:
    if( timeptDiff[0] < 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[1].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
tempSnippet.data->data[1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[1].data->data[j].re = 0;
    cohInspCVec->cData[1].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[1].data->data + abs(timeptDiff[0]),
tempSnippet.data->data, (numPoints - abs(timeptDiff[0])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])].re,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])].im,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])+1].re,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])+1].im);
cohInspCVec->cData[1].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[1].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
}
    else if( timeptDiff[0] > 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[1].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[abs(timeptDiff[0])].re,tempSnippet.data->data[abs(timeptDiff[0])].im,
tempSnippet.data->data[abs(timeptDiff[0])+1].re,tempSnippet.data->data[abs(timeptDiff[0])+1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[1].data->data[j].re = 0;
    cohInspCVec->cData[1].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[1].data->data, tempSnippet.data->data +
abs( timeptDiff[0] ), (numPoints - abs(timeptDiff[0])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[1].data->data[0].re, cohInspCVec->cData[1].data->data[0].im,
cohInspCVec->cData[1].data->data[1].re, cohInspCVec->cData[1].data->data[1].im);
cohInspCVec->cData[1].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[1].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else
      {
if( verbose ) fprintf(stdout,"Time series are commensurate...
directly combining them.\n");
      }
    break;
  case 3:
    if( timeptDiff[0] < 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[1].data->data,
```

```
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
tempSnippet.data->data[1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[1].data->data[j].re = 0;
    cohInspCVec->cData[1].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[1].data->data + abs(timeptDiff[0]),
tempSnippet.data->data, (numPoints - abs(timeptDiff[0])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])].re,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])].im,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])+1].re,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])+1].im);
cohInspCVec->cData[1].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[1].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else if( timeptDiff[0] > 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[1].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
tempSnippet.data->data[1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[1].data->data[j].re = 0;
    cohInspCVec->cData[1].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[1].data->data, tempSnippet.data->data +
abs( timeptDiff[0] ), (numPoints - abs(timeptDiff[0])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])].re,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])].im,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])+1].re,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])+1].im);
cohInspCVec->cData[1].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[1].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else
      {
if( verbose ) fprintf(stdout,"Time series are commensurate...
directly combining them.\n");
      }

    if( timeptDiff[1] < 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[2].data->data,
 numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
```

```
tempSnippet.data->data[1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[2].data->data[j].re = 0;
    cohInspCVec->cData[2].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[2].data->data + abs(timeptDiff[1]),
tempSnippet.data->data, (numPoints - abs(timeptDiff[1])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])].re,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])].im,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])+1].re,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])+1].im);
cohInspCVec->cData[2].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[2].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else if( timeptDiff[1] > 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[2].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
tempSnippet.data->data[1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[2].data->data[j].re = 0;
    cohInspCVec->cData[2].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[2].data->data, tempSnippet.data->data +
abs( timeptDiff[1] ), (numPoints - abs(timeptDiff[1])) * sizeof(COMPLEX8) );
if( verbose )fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])].re,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])].im,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])+1].re,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])+1].im);

cohInspCVec->cData[2].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[2].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else
      {
if( verbose ) fprintf(stdout,"Time series are commensurate...
directly combining them.\n");
      }
    break;
  case 4:
    if( timeptDiff[0] < 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[1].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
tempSnippet.data->data[1].im);
```

```
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[1].data->data[j].re = 0;
    cohInspCVec->cData[1].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[1].data->data + abs(timeptDiff[0]),
tempSnippet.data->data, (numPoints - abs(timeptDiff[0])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])].re,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])].im,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])+1].re,
cohInspCVec->cData[1].data->data[abs(timeptDiff[0])+1].im);
cohInspCVec->cData[1].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[1].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else if( timeptDiff[0] > 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[1].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[abs(timeptDiff[0])].re,
tempSnippet.data->data[abs(timeptDiff[0])].im,
tempSnippet.data->data[abs(timeptDiff[0])+1].re,
tempSnippet.data->data[abs(timeptDiff[0])+1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[1].data->data[j].re = 0;
    cohInspCVec->cData[1].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[1].data->data, tempSnippet.data->data +
abs( timeptDiff[0] ), (numPoints - abs(timeptDiff[0])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[1].data->data[0].re, cohInspCVec->cData[1].data->data[0].im,
cohInspCVec->cData[1].data->data[1].re, cohInspCVec->cData[1].data->data[1].im);
cohInspCVec->cData[1].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[1].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else
      {
if( verbose ) fprintf(stdout,"Time series are commensurate...
directly combining them.\n");
      }


    if( timeptDiff[1] < 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[2].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
tempSnippet.data->data[1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
```

```
    cohInspCVec->cData[2].data->data[j].re = 0;
    cohInspCVec->cData[2].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[2].data->data + abs(timeptDiff[1]),
tempSnippet.data->data, (numPoints - abs(timeptDiff[1])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])].re,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])].im,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])+1].re,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])+1].im);
cohInspCVec->cData[2].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[2].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else if( timeptDiff[1] > 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[2].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
tempSnippet.data->data[1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[2].data->data[j].re = 0;
    cohInspCVec->cData[2].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[2].data->data, tempSnippet.data->data +
abs( timeptDiff[1] ), (numPoints - abs(timeptDiff[1])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])].re,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])].im,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])+1].re,
cohInspCVec->cData[2].data->data[abs(timeptDiff[1])+1].im);
cohInspCVec->cData[2].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[2].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else
      {
if( verbose ) fprintf(stdout,"Time series are commensurate...
directly combining them.\n");
      }

    if( timeptDiff[2] < 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[3].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
tempSnippet.data->data[1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[3].data->data[j].re = 0;
    cohInspCVec->cData[3].data->data[j].im = 0;
  }
```

```
memcpy( cohInspCVec->cData[3].data->data + abs(timeptDiff[2]),
tempSnippet.data->data, (numPoints - abs(timeptDiff[2])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[3].data->data[abs(timeptDiff[2])].re,
cohInspCVec->cData[3].data->data[abs(timeptDiff[2])].im,
cohInspCVec->cData[3].data->data[abs(timeptDiff[2])+1].re,
cohInspCVec->cData[3].data->data[abs(timeptDiff[2])+1].im);
cohInspCVec->cData[3].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[3].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else if( timeptDiff[2] > 0 )
      {
memcpy( tempSnippet.data->data, cohInspCVec->cData[3].data->data,
numPoints * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"tempSnippet data: %f %f %f %f\n",
tempSnippet.data->data[0].re,tempSnippet.data->data[0].im,tempSnippet.data->data[1].re,
tempSnippet.data->data[1].im);
for(j=0;j<(INT4)numPoints;j++)
  {
    cohInspCVec->cData[3].data->data[j].re = 0;
    cohInspCVec->cData[3].data->data[j].im = 0;
  }
memcpy( cohInspCVec->cData[3].data->data, tempSnippet.data->data +
abs( timeptDiff[2] ), (numPoints - abs(timeptDiff[2])) * sizeof(COMPLEX8) );
if( verbose ) fprintf(stdout,"Some frame data: %f %f %f %f\n",
cohInspCVec->cData[3].data->data[abs(timeptDiff[2])].re,
cohInspCVec->cData[3].data->data[abs(timeptDiff[2])].im,
cohInspCVec->cData[3].data->data[abs(timeptDiff[2])+1].re,
cohInspCVec->cData[3].data->data[abs(timeptDiff[2])+1].im);

cohInspCVec->cData[3].epoch.gpsSeconds =
cohInspCVec->cData[0].epoch.gpsSeconds;
cohInspCVec->cData[3].epoch.gpsNanoSeconds =
cohInspCVec->cData[0].epoch.gpsNanoSeconds;
      }
    else
      {
if( verbose ) fprintf(stdout,"Time series are commensurate...
directly combining them.\n");
      }
    break;
}

      /* Now that the time series are commensurate, do the filtering... */
              cohInspFilterParams->cohSNRVec->epoch = cohInspCVec->cData[0].epoch;
      LALCoherentInspiralFilterSegment (&status, &thisEvent, cohInspFilterInput,
cohInspFilterParams);

 /* Now change from geocentric to equatorial */
      thisEventTemp =thisEvent;
      if ( (numDetectors == 3 && !( caseID[0] && caseID[5])) || numDetectors == 4 )
{
         while( thisEventTemp )
    {
      convertParams.system=COORDINATESYSTEM_EQUATORIAL;
```

```
      convertParams.zenith= NULL;
      convertParams.gpsTime=&(thisEventTemp->end_time);
      /*The next few lines gets the parameters ready for conversion.
Need to check to make sure that thisEvent->ligo_axis_dec is in proper notation, i.e. */

      tempSky.system=COORDINATESYSTEM_GEOGRAPHIC;
      tempSky.latitude=thisEventTemp->ligo_axis_dec;
      tempSky.longitude=thisEventTemp->ligo_axis_ra;
      LALConvertSkyCoordinates(&status,&tempSky, &tempSky,&convertParams);
      thisEventTemp->ligo_axis_dec=LAL_PI*0.5-tempSky.latitude;
      thisEventTemp->ligo_axis_ra=tempSky.longitude;

      thisEventTemp = thisEventTemp->next;
    }
}

      if ( cohSNROut )
{
  LALSnprintf( cohdataStr, LALNameLength*sizeof(CHAR),
       "SNR_%d", nCohDataFr++ );
  strcpy( cohInspFilterParams->cohSNRVec->name, "Coherent");
  outFrame = fr_add_proc_REAL4TimeSeries( outFrame,
       cohInspFilterParams->cohSNRVec, "none", cohdataStr );
}

      if ( !eventsOut )
{
  while( thisEvent )
    {
      MultiInspiralTable *tempEvent = thisEvent;
      thisEvent = thisEvent->next;
      LALFree( tempEvent );
    }
}


      if( thisEvent )
{
  if( verbose ) fprintf( stdout,"******> Dumping Events <******\n");
  if( !savedEvents.multiInspiralTable )
    {
      MultiInspiralTable *tempEvent = thisEvent;
      tempTable = (MultiInspiralTable *) LALCalloc( 1,
sizeof(MultiInspiralTable) );
      memcpy(tempTable,thisEvent,sizeof(MultiInspiralTable) );
      savedEvents.multiInspiralTable = tempTable;
      thisEvent = thisEvent->next;
      LALFree( tempEvent );
      tempEvent = NULL;
      if( thisEvent )
{
  while( thisEvent )
    {
      MultiInspiralTable *tempEvent = thisEvent;
      tempTable->next = (MultiInspiralTable *) LALCalloc( 1,
sizeof(MultiInspiralTable) );
      tempTable = tempTable->next;
```

```
      memcpy(tempTable, thisEvent, sizeof(MultiInspiralTable) );
      thisEvent = thisEvent->next;
      LALFree( tempEvent );
      tempEvent = NULL;
    }
}
    }
  else
    {
      while( thisEvent )
{
  MultiInspiralTable *tempEvent = thisEvent;
  tempTable->next = (MultiInspiralTable *) LALCalloc( 1,
sizeof(MultiInspiralTable) );
  tempTable = tempTable->next;
  memcpy(tempTable, thisEvent, sizeof(MultiInspiralTable) );
  thisEvent = thisEvent->next;
  LALFree( tempEvent );
  tempEvent = NULL;
}
    }




  while( thisEvent )
{
  MultiInspiralTable *tempEvent = thisEvent;
  thisEvent = thisEvent->next;
  LALFree( tempEvent );
  tempEvent = NULL;
}

}

      /* initialize */
      LAL_CALL( LALCDestroyVector( &status, &(tempSnippet.data) ), &status );
      LALFree( cohInspFilterInput->tmplt );
      cohInspFilterInput->tmplt = NULL;
      free( cohInspInitParams );

      /* Destroy params structure for coherent filter code */
      LAL_CALL( LALCoherentInspiralFilterParamsFinalize (&status,
   &cohInspFilterParams), &status );
              TestStatus (&status, "0", 1);
      ClearStatus (&status);

              /* Destroy input structure for coherent filter code */
      LAL_CALL( LALCoherentInspiralFilterInputFinalize (&status,
&cohInspFilterInput), &status);
              TestStatus (&status, "0", 1);
      ClearStatus (&status);

        k = -1;
      for(j=0;j<4;j++)
{
  memcpy(caseIDChars[j], "0",3);
```

```
}
      for(j=0;j<6;j++)
{
  caseID[j] = 0;
  if( !globFrameData )
    {
      chanNumArray[j] = -1;
      strcpy( namearray[j], "0" );
    }
  strcpy( namearray2[j], "0" );
  strcpy( namearray3[j], "0" );
}
      for(j=0;j<5;j++)
        {
  tempTime[j] = 0.0;
  timeptDiff[j] = 0;
}
      tempTime[5] = 0.0;

    }
  else
    {
      fprintf(stderr,"Error - there is only one detector associated with
event %Ld, aborting...\n", eventID );
      exit( 1 );
    }
}
      k++;
    }

  /* set the name of the output file(s) (without extension) */

  if ( userTag )
  {
    LALSnprintf( fileName, FILENAME_MAX, "%s-COHERENT_%s-%d-%d", ifoTag, userTag,
        gpsStartTime.gpsSeconds, gpsEndTime.gpsSeconds - gpsStartTime.gpsSeconds );
  }
  else
  {
    LALSnprintf( fileName, FILENAME_MAX, "%s-COHERENT-%d-%d", ifoTag,
        gpsStartTime.gpsSeconds, gpsEndTime.gpsSeconds - gpsStartTime.gpsSeconds );
  }


  if( cohSNROut )
    {
      if ( outputPath[0] )
{
  LALSnprintf( framename, FILENAME_MAX * sizeof(CHAR), "%s/%s.gwf",
outputPath, fileName);
}
      else
{
  LALSnprintf( framename, FILENAME_MAX * sizeof(CHAR), "%s.gwf", fileName );
}

      if ( verbose ) fprintf( stdout, "writing frame data to %s....", framename );
```

```
      frOutFile = FrFileONew( framename, 0);
      FrameWrite( outFrame, frOutFile);
      FrFileOEnd( frOutFile );
      if ( verbose ) fprintf(stdout, "done\n");
    }


  if ( eventsOut )
    {
      memset( &results, 0, sizeof(LIGOLwXMLStream) );
      if ( outputPath[0] )
{
  LALSnprintf( xmlname, FILENAME_MAX * sizeof(CHAR), "%s/%s.xml",
outputPath, fileName);
}
      else
{
  LALSnprintf( xmlname, FILENAME_MAX * sizeof(CHAR), "%s.xml", fileName );
}
      if ( verbose ) fprintf( stdout, "writing XML data to %s...\n", xmlname );
      LAL_CALL( LALOpenLIGOLwXMLFile( &status, &results, xmlname), &status );

      /* write the process table */
      if ( verbose ) fprintf( stdout, "  process table...\n" );
      /*      LALSnprintf( proctable.processTable->ifos, LIGOMETA_IFOS_MAX, "%s",
caseID );*/
      LAL_CALL( LALGPSTimeNow ( &status, &(proctable.processTable->end_time), &accuracy ),
&status );
      LAL_CALL( LALBeginLIGOLwXMLTable( &status, &results, process_table ), &status );
      LAL_CALL( LALWriteLIGOLwXMLTable( &status, &results, proctable, process_table ),
&status );
      LAL_CALL( LALEndLIGOLwXMLTable ( &status, &results ), &status );
      free( proctable.processTable );

      /* write the process params table */
      if ( verbose ) fprintf( stdout, "  process_params table...\n" );
      LAL_CALL( LALBeginLIGOLwXMLTable( &status, &results, process_params_table ),
&status );
      LAL_CALL( LALWriteLIGOLwXMLTable( &status, &results, procparams,
process_params_table ), &status );
      LAL_CALL( LALEndLIGOLwXMLTable ( &status, &results ), &status );
      while( procparams.processParamsTable )
{
  this_proc_param = procparams.processParamsTable;
  procparams.processParamsTable = this_proc_param->next;
  free( this_proc_param );
  }

      if( verbose ) fprintf(stdout,"  event params table\n ");

      LAL_CALL( LALBeginLIGOLwXMLTable( &status, &results, multi_inspiral_table )
, &status );
      LAL_CALL( LALWriteLIGOLwXMLTable( &status, &results, savedEvents,
multi_inspiral_table ), &status );
      LAL_CALL( LALEndLIGOLwXMLTable( &status, &results), &status );

      while( savedEvents.multiInspiralTable )
```

```
    {
      MultiInspiralTable *tempEvent = savedEvents.multiInspiralTable;
      savedEvents.multiInspiralTable = savedEvents.multiInspiralTable->next;
      LALFree( tempEvent );
      tempEvent = NULL;
    }
      /* close the output xml file */
      LAL_CALL( LALCloseLIGOLwXMLFile ( &status, &results ), &status );
      if ( verbose ) fprintf( stdout, "done. XML file closed\n" );

    }

  goto cleanexit;

  cleanexit:

  /* Free the template bank */
  while( bankHead )
    {
      InspiralTemplate *tempTmplt = bankHead;
      bankHead = bankHead->next;
      LALFree( tempTmplt->event_id );
      LALFree( tempTmplt );
      tempTmplt = NULL;
    }

  /* free the frame cache */
  if( frInCache ) LAL_CALL( LALDestroyFrCache( &status, &frInCache ), &status );

  if ( verbose ) fprintf( stdout, "checking memory leaks and exiting\n" );
  LALCheckMemoryLeaks();
  exit(0);

}/* main function end */


/* ---------------------------------------------------------------------- */



static void
TestStatus (
    LALStatus  *status,
    const char *ignored,
    int         exitcode
          )
{
  char  str[64];
  char *tok;

  if (verbose)
  {
    REPORTSTATUS (status);
  }

  if (strncpy (str, ignored, sizeof (str)))
  {
```

```
    if ((tok = strtok (str, " ")))
    {
      do
      {
        if (status->statusCode == atoi (tok))
        {
          return;
        }
      }
      while ((tok = strtok (NULL, " ")));
    }
    else
    {
      if (status->statusCode == atoi (tok))
      {
        return;
      }
    }
  }

  fprintf (stderr, "\nExiting to system with code %d\n", exitcode);
  exit (exitcode);
}


/*
 *
 * ClearStatus ()
 *
 * Recursively applies DETATCHSTATUSPTR() to status structure to destroy
 * linked list of statuses.
 *
 */
static void
ClearStatus (
    LALStatus   *status
            )
{
  if (status->statusPtr)
  {
    ClearStatus      (status->statusPtr);
    DETATCHSTATUSPTR (status);
  }
}

#define ADD_PROCESS_PARAM( pptype, format, ppvalue ) \
this_proc_param = this_proc_param->next = (ProcessParamsTable *) \
  calloc( 1, sizeof(ProcessParamsTable) ); \
  LALSnprintf( this_proc_param->program, LIGOMETA_PROGRAM_MAX, "%s", \
      PROGRAM_NAME ); \
      LALSnprintf( this_proc_param->param, LIGOMETA_PARAM_MAX, "--%s", \
          long_options[option_index].name ); \
          LALSnprintf( this_proc_param->type, LIGOMETA_TYPE_MAX, "%s", pptype ); \
          LALSnprintf( this_proc_param->value, LIGOMETA_VALUE_MAX, format, ppvalue );


#define USAGE1 \
```

```
"lalapps_inspiral [options]\n\n"\
"  --help                       display this message\n"\
"  --verbose                    print progress information\n"\
"  --version                    print version information and exit\n"\
"  --debug-level LEVEL          set the LAL debug level to LEVEL\n"\
"  --low-frequency-cutoff F     low f cutoff of previously filtered data\n"\
"  --ifo-tag STRING             set STRING to whatever the ifo-tag of \n"\
                                "the bank file(needed for file naming) \n"\
"  --user-tag STRING            set STRING to tag the file names\n"\
"\n"
#define USAGE2 \
"  --bank-file FILE             read template bank parameters from FILE\n"\
"  --sample-rate N              set data sample rate to N\n"\
"  --segment-length N           set N to same value used in inspiral.c\n"\
"  --dynamic-range-exponent N   set N to same value used in inspiral.c\n"\
"  --cohsnr-threshold RHO       set signal-to-noise threshold to RHO\n"\
"  --maximize-over-chirp        do clustering\n"\
"  --glob-frame-data            glob files in the pwd to obtain frame data\n"\
"  --gps-start-time SEC         GPS second of data start time (needed if globbing)\n"\
"  --gps-end-time SEC           GPS second of data end time (needed if globbing)\n"\
"\n"
#define USAGE3 \
"  --write-events               write events\n"\
"  --write-cohsnr               write cohsnr\n"\
"  --output-path                write files here\n"\
"  --H1-framefile               frame data for H1\n"\
"  --H2-framefile               frame data for H2\n"\
"  --L-framefile                frame data for L\n"\
"  --V-framefile                frame data for V\n"\
"  --G-framefile                frame data for G\n"\
"  --T-framefile                frame data for T\n"\
"\n"

int arg_parse_check( int argc, char *argv[], MetadataTable procparams )
{
   struct option long_options[] =
   {
     {"verbose",                  no_argument,       &verbose,          1 },
     {"help",                     no_argument,       0,                'h'},
     {"version",                  no_argument,       0,                'v'},
     {"debug-level",              required_argument, 0,                'd'},
     {"ifo-tag",                  required_argument, 0,                'I'},
     {"user-tag",                 required_argument, 0,                'B'},
     {"low-frequency-cutoff",     required_argument, 0,                'f'},
     {"bank-file",                required_argument, 0,                'u'},
     {"sample-rate",              required_argument, 0,                'r'},
     {"segment-length",           required_argument, 0,                'l'},
     {"dynamic-range-exponent",   required_argument, 0,                'e'},
     {"cohsnr-threshold",         required_argument, 0,                'p'},
     {"maximize-over-chirp",      no_argument,       &maximizeOverChirp, 1 },
     {"glob-frame-data",          no_argument,       &globFrameData,    1 },
     {"write-events",             no_argument,       &eventsOut,        1 },
     {"write-cohsnr",             no_argument,       &cohSNROut,        1 },
     {"gps-start-time",           required_argument, 0,                'a'},
     {"gps-end-time",             required_argument, 0,                'b'},
     {"output-path",              required_argument, 0,                'P'},
     {"H1-framefile",             required_argument, 0,                'A'},
```

```
    {"H2-framefile",            required_argument, 0,                'Z'},
    {"L-framefile",             required_argument, 0,                'L'},
    {"V-framefile",             required_argument, 0,                'V'},
    {"G-framefile",             required_argument, 0,                'G'},
    {"T-framefile",             required_argument, 0,                'T'},
    {0, 0, 0, 0}
  };

  int c;
  ProcessParamsTable *this_proc_param = procparams.processParamsTable;

  while (1)
    {
      /* getopt_long stores long options here */
      int option_index = 0;
      size_t optarg_len;

      c = getopt_long_only( argc, argv,
  "A:B:a:b:G:I:L:l:e:P:T:V:Z:d:f:h:p:r:u:v:",
  long_options, &option_index );

      if ( c == -1 )
{
  break;
}

      switch ( c )
{
case 0:
       /* if this option set a flag, do nothing else now */
  if ( long_options[option_index].flag != 0 )
    {
      break;      }
  else
    {
      fprintf( stderr, "error parsing option %s with argument %s\n",
long_options[option_index].name, optarg );
      exit( 1 );
    }
  break;

case 'A':
  strcpy(H1filename,optarg);
  H1file = 1;
  ADD_PROCESS_PARAM( "string", "%s", H1filename );
  break;

case 'G':
  strcpy(GEOfilename,optarg);
  GEOfile = 1;
  ADD_PROCESS_PARAM( "string", "%s", GEOfilename );
  break;

case 'L':
  strcpy(Lfilename,optarg);
  Lfile = 1;
  ADD_PROCESS_PARAM( "string", "%s", Lfilename );
```

```
  break;

case 'T':
  strcpy(TAMAfilename,optarg);
  TAMAfile = 1;
  ADD_PROCESS_PARAM( "string", "%s", TAMAfilename );
  break;

case 'V':
  strcpy(VIRGOfilename,optarg);
  VIRGOfile = 1;
  ADD_PROCESS_PARAM( "string", "%s", VIRGOfilename );
  break;

case 'Z':
  strcpy(H2filename,optarg);
  H2file = 1;
  ADD_PROCESS_PARAM( "string", "%s", H2filename );
  break;

        case 'B':
          /* create storaged for the ifo-tag */
          optarg_len = strlen( optarg ) + 1;
          userTag = (CHAR *) calloc( optarg_len, sizeof(CHAR) );
          memcpy( userTag, optarg, optarg_len );
          ADD_PROCESS_PARAM( "string", "%s", optarg );
          break;

        case 'I':
          /* create storaged for the ifo-tag */
          optarg_len = strlen( optarg ) + 1;
          ifoTag = (CHAR *) calloc( optarg_len, sizeof(CHAR) );
          memcpy( ifoTag, optarg, optarg_len );
          ADD_PROCESS_PARAM( "string", "%s", optarg );
          break;

case 'P':
  memset( outputPath, 0, FILENAME_MAX * sizeof(CHAR) );
  LALSnprintf( outputPath, FILENAME_MAX * sizeof(CHAR),"%s", optarg );
  ADD_PROCESS_PARAM( "string", "%s", outputPath );
  break;

case 'd': /* set debuglevel */
  set_debug_level( optarg );
  ADD_PROCESS_PARAM( "string", "%s", optarg );
  break;

case 'f': /* set fLow */
  fLow = (REAL4) atof (optarg);
  ADD_PROCESS_PARAM( "float", "%e", fLow );
  break;

case 'h':
  fprintf( stdout, USAGE1 );
  fprintf( stdout, USAGE2 );
  fprintf( stdout, USAGE3 );
  exit( 0 );
```

```
  break;

case 'p': /* set coherent SNR threshold */
  cohSNRThresh = atof (optarg);
  ADD_PROCESS_PARAM( "float", "%e", cohSNRThresh );
  break;

case 'l':
  numPointsSeg = atof(optarg);
  ADD_PROCESS_PARAM("float", "%e", numPointsSeg );
  break;

case 'e':
  dynRangeExponent = atof(optarg);
  ADD_PROCESS_PARAM("float", "%e", dynRangeExponent );
  break;

case 'r':
  sampleRate = atof(optarg);
  ADD_PROCESS_PARAM("float", "%e", sampleRate );
  break;

case 'u':
         /* create storage for the bank filename */
  /*optarg_len = strlen( optarg ) + 1;
  bankFileName = (CHAR *) calloc( optarg_len, sizeof(CHAR));
  memcpy( bankFileName, optarg, optarg_len );*/
  strcpy(bankFileName, optarg);
  ADD_PROCESS_PARAM( "string", "%s", bankFileName );
  break;

case 'v':
  /* print version information and exit */
         fprintf( stdout, "LIGO/LSC Multi-Detecter Search Code\n"
               "Bose/Seader <sukanta@wsu.edu> <sseader@wsu.edu>\n"
               "CVS Version: " CVS_ID_STRING "\n"
"CVS Tag: " CVS_NAME_STRING "\n" );
  exit( 0 );
  break;

case 'a':
  {
    long int gstartt = atol( optarg );
    if ( gstartt < 441417609 )
      {
              fprintf( stderr, "invalid argument to --%s:\n"
                  "GPS start time is prior to "
                  "Jan 01, 1994  00:00:00 UTC:\n"
                  "(%ld specified)\n",
                  long_options[option_index].name, gstartt );
     exit( 1 );
     }
           if ( gstartt > 999999999 )
           {
             fprintf( stderr, "invalid argument to --%s:\n"
                  "GPS start time is after "
                  "Sep 14, 2011  01:46:26 UTC:\n"
```

```
                "(%ld specified)\n",
                long_options[option_index].name, gstartt );
            exit( 1 );
          }
    gpsStartTimeNS += (INT8) gstartt * 1000000000LL;
          ADD_PROCESS_PARAM( "int", "%ld", gstartt );
        }
        break;

      case 'b':
        {
          long int gendt = atol( optarg );
          if ( gendt > 999999999 )
          {
            fprintf( stderr, "invalid argument to --%s:\n"
                "GPS end time is after "
                "Sep 14, 2011  01:46:26 UTC:\n"
                "(%ld specified)\n",
                long_options[option_index].name, gendt );
            exit( 1 );
          }
          else if ( gendt < 441417609 )
          {
            fprintf( stderr, "invalid argument to --%s:\n"
                "GPS end time is prior to "
                "Jan 01, 1994  00:00:00 UTC:\n"
                "(%ld specified)\n",
                long_options[option_index].name, gendt );
            exit( 1 );
          }
    gpsEndTimeNS += (INT8) gendt * 1000000000LL;
          ADD_PROCESS_PARAM( "int", "%ld", gendt );
        }
        break;

case '?':
  exit( 1 );
  break;

default:
  fprintf( stderr, "unknown error while parsing options\n" );
  exit( 1 );

}

    }

  if (optind < argc)
    {
      fprintf( stderr, "extraneous command line arguments:\n" );
      while ( optind < argc )
{
  fprintf ( stderr, "%s\n", argv[optind++] );
}
      exit( 1 );
    }
```

```
  if ( globFrameData && (H1file || H2file || Lfile || GEOfile || VIRGOfile || TAMAfile) )
    {
      fprintf( stderr, "Specify frame files or to glob for frames - not both\n" );
      exit( 1 );
    }

  /* check validity of input data time if globbing */
  /* the times should be that spanned by the bank(trigger) file */
  if ( ! gpsStartTimeNS )
    {
      fprintf( stderr, "--gps-start-time must be specified\n" );
      exit( 1 );
    }
  LAL_CALL( LALINT8toGPS( &status, &gpsStartTime, &gpsStartTimeNS ),
    &status );
  if ( ! gpsEndTimeNS )
    {
      fprintf( stderr, "--gps-end-time must be specified\n" );
      exit( 1 );
    }
  LAL_CALL( LALINT8toGPS( &status, &gpsEndTime, &gpsEndTimeNS ),
    &status );
  if ( gpsEndTimeNS <= gpsStartTimeNS )
    {
      fprintf( stderr, "invalid gps time range: "
          "start time: %d, end time %d\n",
          gpsStartTime.gpsSeconds, gpsEndTime.gpsSeconds );
      exit( 1 );
    }

  /* check sample rate has been given */
  if ( sampleRate < 0 )
    {
      fprintf( stderr, "--sample-rate must be specified\n" );
      exit( 1 );
    }

  if ( numPointsSeg < 0 )
    {
      fprintf( stderr, "--segment-length must be specified.\n" );
      fprintf( stderr,"It must be set to the same value as was used in inspiral.c when the
C-data was generated.\n");
      exit( 1 );
    }

  if ( dynRangeExponent < 0 )
    {
      fprintf( stderr, "--dynamic-range-exponent must be specified.\n" );
      fprintf( stderr,"It must be set to the same value as was used in inspiral.c
when the C-data was generated.\n");
      exit( 1 );
    }

  if ( ! bankFileName )
    {
      fprintf( stderr, "--bank-file must be specified\n" );
      exit( 1 );
```

```
    }

  if ( fLow < 0 )
    {
      fprintf( stderr, "--low-frequency-cutoff must be specified\n" );
      exit( 1 );
    }

  if ( cohSNRThresh < 0 )
    {
      fprintf( stderr, "--cohsnr-threshold must be specified\n" );
      exit( 1 );
    }

  if( !ifoTag )
    {
      fprintf(stderr, "--ifo-tag must be specified for file naming\n" );
      exit( 1 );
    }

  /* record the glob frame data option in the process params */
  if ( globFrameData )
    {
      this_proc_param = this_proc_param->next = (ProcessParamsTable *)
 calloc( 1, sizeof(ProcessParamsTable) );
      LALSnprintf( this_proc_param->program, LIGOMETA_PROGRAM_MAX,
        "%s", PROGRAM_NAME );
      LALSnprintf( this_proc_param->param, LIGOMETA_PARAM_MAX,
        "--glob-frame-data" );
      LALSnprintf( this_proc_param->type, LIGOMETA_TYPE_MAX, "string" );
      LALSnprintf( this_proc_param->value, LIGOMETA_TYPE_MAX, " " );
    }

  return 0;
}
#undef ADD_PROCESS_PARAM
```

Appendix G

CoherentBank.c

```
/*-----------------------------------------------------------------------
 *
 * File Name: coherent_bank.c
 *
 * Author: Fairhust, S. and Seader, S.E.
 *
 * Revision: $Id: coherentbank.c,v 1.4 2005/10/12 00:07:51 sfairhur Exp $
 *
 *-----------------------------------------------------------------------
 */


#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <sys/types.h>
#include <lal/LALStdio.h>
#include <lal/LALStdlib.h>
#include <lal/Date.h>
#include <lal/LIGOLwXML.h>
#include <lal/LIGOLwXMLRead.h>
#include <lal/LIGOMetadataUtils.h>
#include <lalapps.h>
#include <processtable.h>

RCSID("$Id: coherentbank.c,v 1.4 2005/10/12 00:07:51 sfairhur Exp $");

#define CVS_ID_STRING "$Id: coherentbank.c,v 1.4 2005/10/12 00:07:51 sfairhur Exp $"
#define CVS_NAME_STRING "$Name:  $"
#define CVS_REVISION "$Revision: 1.4 $"
#define CVS_SOURCE "$Source: /usr/local/cvs/lscsoft/lalapps/src/inspiral/coherentbank.c,v $"
#define CVS_DATE "$Date: 2005/10/12 00:07:51 $"
#define PROGRAM_NAME "lalapps_coherentbank"

#define ADD_PROCESS_PARAM( pptype, format, ppvalue ) \
  this_proc_param = this_proc_param->next = (ProcessParamsTable *) \
```

```
calloc( 1, sizeof(ProcessParamsTable) ); \
LALSnprintf( this_proc_param->program, LIGOMETA_PROGRAM_MAX, "%s", \
    PROGRAM_NAME ); \
LALSnprintf( this_proc_param->param, LIGOMETA_PARAM_MAX, "--%s", \
    long_options[option_index].name ); \
LALSnprintf( this_proc_param->type, LIGOMETA_TYPE_MAX, "%s", pptype ); \
LALSnprintf( this_proc_param->value, LIGOMETA_VALUE_MAX, format, ppvalue );

extern int vrbflg;
int allIFO = -1;

/*
 *
 * USAGE
 *
 */

static void print_usage(char *program)
{
  fprintf(stderr,
      "Usage: %s [options] [LIGOLW XML input files]\n"\
      "The following options are recognized.  Options not surrounded in []\n"\
      "are required.\n", program );
  fprintf(stderr,
      " [--help]                      display this message\n"\
      " [--verbose]                   print progress information\n"\
      " [--version]                   print version information and exit\n"\
      " [--debug-level]   level       set the LAL debug level to LEVEL\n"\
      " [--user-tag]      usertag     set the process_params usertag\n"\
      " [--comment]       string      set the process table comment\n"\
      "  --bank-file      file        the input trigger file.\n"\
      "  --ifos           ifos        list of ifos for which we have data\n"\
      "  --gps-start-time start_time  start time of the job\n"\
      "  --gps-end-time   end_time    end time of the job\n"\
      "  --enable-all-ifo             generate bank with templates for all ifos\n"\
      "  --disable-all-ifo            only generate bank for triggers in coinc\n"\
      "\n");
}


int main( int argc, char *argv[] )
{
  static LALStatus      status;
  LALLeapSecAccuracy    accuracy = LALLEAPSEC_LOOSE;


  INT4 numTriggers = 0;
  INT4 numCoincs = 0;
  INT4 numTmplts = 0;

  INT4        startTime = -1;
  LIGOTimeGPS startTimeGPS = {0,0};
  INT4        endTime = -1;
  LIGOTimeGPS endTimeGPS = {0,0};

  CHAR  ifos[LIGOMETA_IFOS_MAX];
  CHAR *inputFileName = NULL;
```

```
CHAR  comment[LIGOMETA_COMMENT_MAX];
CHAR *userTag = NULL;

CHAR  fileName[FILENAME_MAX];

SnglInspiralTable    *inspiralEventList=NULL;
SnglInspiralTable    *currentTrigger = NULL;
SnglInspiralTable    *newEventList = NULL;

CoincInspiralTable   *coincHead = NULL;
CoincInspiralTable   *thisCoinc = NULL;

SearchSummvarsTable  *inputFiles = NULL;
SearchSummvarsTable  *thisInputFile = NULL;

SearchSummaryTable   *searchSummList = NULL;
SearchSummaryTable   *thisSearchSumm = NULL;

MetadataTable         proctable;
MetadataTable         processParamsTable;
MetadataTable         searchsumm;
MetadataTable         searchSummvarsTable;
MetadataTable         inspiralTable;
ProcessParamsTable   *this_proc_param = NULL;
LIGOLwXMLStream       xmlStream;


/* getopt arguments */
struct option long_options[] =
{
  {"verbose",               no_argument,     &vrbflg,                1 },
  {"enable-all-ifo",        no_argument,     &allIFO,                1 },
  {"disable-all-ifo",       no_argument,     &allIFO,                0 },
  {"comment",               required_argument,     0,               'x'},
  {"user-tag",              required_argument,     0,               'Z'},
  {"help",                  no_argument,           0,               'h'},
  {"debug-level",           required_argument,     0,               'z'},
  {"version",               no_argument,           0,               'V'},
  {"bank-file",             required_argument,     0,               'b'},
  {"gps-start-time",        required_argument,     0,               's'},
  {"gps-end-time",          required_argument,     0,               't'},
  {"ifos",                  required_argument,     0,               'i'},
  {0, 0, 0, 0}
};
int c;

/*
 *
 * initialize things
 *
 */

lal_errhandler = LAL_ERR_EXIT;
set_debug_level( "1" );
/*  setvbuf( stdout, NULL, _IONBF, 0 );*/
```

```
/* create the process and process params tables */
proctable.processTable = (ProcessTable *) calloc( 1, sizeof(ProcessTable) );
LAL_CALL( LALGPSTimeNow ( &status, &(proctable.processTable->start_time),
      &accuracy ), &status );
LAL_CALL( populate_process_table( &status, proctable.processTable,
      PROGRAM_NAME, CVS_REVISION, CVS_SOURCE, CVS_DATE ), &status );
this_proc_param = processParamsTable.processParamsTable =
  (ProcessParamsTable *) calloc( 1, sizeof(ProcessParamsTable) );

/* initialize variables */
memset( comment, 0, LIGOMETA_COMMENT_MAX * sizeof(CHAR) );
memset( ifos, 0, LIGOMETA_IFOS_MAX * sizeof(CHAR) );

/* create the search summary and zero out the summvars table */
searchsumm.searchSummaryTable = (SearchSummaryTable *)
  calloc( 1, sizeof(SearchSummaryTable) );


/* parse the arguments */
while ( 1 )
{
  /* getopt_long stores long option here */
  int option_index = 0;
  size_t optarg_len;
  long int gpstime;

  c = getopt_long_only( argc, argv,
      "b:hi:s:t:x:z:VZ:", long_options,
      &option_index );

  /* detect the end of the options */
  if ( c == -1 )
  {
    break;
  }

  switch ( c )
  {
    case 0:
      /* if this option set a flag, do nothing else now */
      if ( long_options[option_index].flag != 0 )
      {
        break;
      }
      else
      {
        fprintf( stderr, "Error parsing option %s with argument %s\n",
            long_options[option_index].name, optarg );
        exit( 1 );
      }
      break;

    case 'b':
      /* set bank file */
      optarg_len = strlen( optarg ) + 1;
      inputFileName = (CHAR *) calloc( optarg_len, sizeof(CHAR));
      memcpy( inputFileName, optarg, optarg_len );
```

```
  ADD_PROCESS_PARAM( "string", "%s", optarg );
  break;

case 'i':
  /* set ifos */
  strncpy( ifos, optarg, LIGOMETA_IFOS_MAX * sizeof(CHAR) );
  ADD_PROCESS_PARAM( "string", "%s", optarg );
  break;

case 's':
  /* start time coincidence window */
  gpstime = atol( optarg );
  if ( gpstime < 441417609 )
  {
    fprintf( stderr, "invalid argument to --%s:\n"
        "GPS start time is prior to "
        "Jan 01, 1994  00:00:00 UTC:\n"
        "(%ld specified)\n",
        long_options[option_index].name, gpstime );
    exit( 1 );
  }
  if ( gpstime > 999999999 )
  {
    fprintf( stderr, "invalid argument to --%s:\n"
        "GPS start time is after "
        "Sep 14, 2011  01:46:26 UTC:\n"
        "(%ld specified)\n",
        long_options[option_index].name, gpstime );
    exit( 1 );
  }
  startTime = (INT4) gpstime;
  startTimeGPS.gpsSeconds = startTime;
  ADD_PROCESS_PARAM( "int", "%ld", startTime );
  break;

case 't':
  /* end time coincidence window */
  gpstime = atol( optarg );
  if ( gpstime < 441417609 )
  {
    fprintf( stderr, "invalid argument to --%s:\n"
        "GPS start time is prior to "
        "Jan 01, 1994  00:00:00 UTC:\n"
        "(%ld specified)\n",
        long_options[option_index].name, gpstime );
    exit( 1 );
  }
  if ( gpstime > 999999999 )
  {
    fprintf( stderr, "invalid argument to --%s:\n"
        "GPS start time is after "
        "Sep 14, 2011  01:46:26 UTC:\n"
        "(%ld specified)\n",
        long_options[option_index].name, gpstime );
    exit( 1 );
  }
  endTime = (INT4) gpstime;
```

```
  endTimeGPS.gpsSeconds = endTime;
  ADD_PROCESS_PARAM( "int", "%ld", endTime );
  break;

case 'x':
  if ( strlen( optarg ) > LIGOMETA_COMMENT_MAX - 1 )
  {
    fprintf( stderr, "invalid argument to --%s:\n"
        "comment must be less than %d characters\n",
        long_options[option_index].name, LIGOMETA_COMMENT_MAX );
    exit( 1 );
  }
  else
  {
    LALSnprintf( comment, LIGOMETA_COMMENT_MAX, "%s", optarg);
  }
  break;

case 'h':
  /* help message */
  print_usage(argv[0]);
  exit( 1 );
  break;

case 'z':
  set_debug_level( optarg );
  ADD_PROCESS_PARAM( "string", "%s", optarg );
  break;

case 'Z':
  /* create storage for the usertag */
  optarg_len = strlen(optarg) + 1;
  userTag = (CHAR *) calloc( optarg_len, sizeof(CHAR) );
  memcpy( userTag, optarg, optarg_len );

  this_proc_param = this_proc_param->next = (ProcessParamsTable *)
    calloc( 1, sizeof(ProcessParamsTable) );
  LALSnprintf( this_proc_param->program, LIGOMETA_PROGRAM_MAX, "%s",
      PROGRAM_NAME );
  LALSnprintf( this_proc_param->param, LIGOMETA_PARAM_MAX, "-userTag" );
  LALSnprintf( this_proc_param->type, LIGOMETA_TYPE_MAX, "string" );
  LALSnprintf( this_proc_param->value, LIGOMETA_VALUE_MAX, "%s",
      optarg );
  break;

case 'V':
  /* print version information and exit */
  fprintf( stdout, "Coherent Bank Generator\n"
      "Steve Fairhurst and Shawn Seader\n"
      "CVS Version: " CVS_ID_STRING "\n"
      "CVS Tag: " CVS_NAME_STRING "\n" );
  exit( 0 );
  break;

case '?':
  print_usage(argv[0]);
  exit( 1 );
```

```
      break;

    default:
      fprintf( stderr, "Error: Unknown error while parsing options\n" );
      print_usage(argv[0]);
      exit( 1 );
  }
}

if (optind < argc)
{
  fprintf( stderr, "extraneous command line arguments:\n" );
  while ( optind < argc )
  {
    fprintf ( stderr, "%s\n", argv[optind++] );
  }
  exit( 1 );
}

/* fill the comment, if a user has specified one, or leave it blank */
if ( ! *comment )
{
  LALSnprintf( proctable.processTable->comment, LIGOMETA_COMMENT_MAX, " " );
  LALSnprintf( searchsumm.searchSummaryTable->comment, LIGOMETA_COMMENT_MAX,
      " " );
}
else
{
  LALSnprintf( proctable.processTable->comment, LIGOMETA_COMMENT_MAX,
      "%s", comment );
  LALSnprintf( searchsumm.searchSummaryTable->comment, LIGOMETA_COMMENT_MAX,
      "%s", comment );
}

/* enable/disable-all-ifo is stored in the first process param row */
if ( allIFO == 1 )
{
  LALSnprintf( processParamsTable.processParamsTable->program,
      LIGOMETA_PROGRAM_MAX, "%s", PROGRAM_NAME );
  LALSnprintf( processParamsTable.processParamsTable->param,
      LIGOMETA_PARAM_MAX, "--enable-all-ifo" );
  LALSnprintf( processParamsTable.processParamsTable->type,
      LIGOMETA_TYPE_MAX, "string" );
  LALSnprintf( processParamsTable.processParamsTable->value,
      LIGOMETA_TYPE_MAX, " " );
}
else if ( allIFO == 0 )
{
  LALSnprintf( processParamsTable.processParamsTable->program,
      LIGOMETA_PROGRAM_MAX, "%s", PROGRAM_NAME );
  LALSnprintf( processParamsTable.processParamsTable->param,
      LIGOMETA_PARAM_MAX, "--disable-all-ifo" );
  LALSnprintf( processParamsTable.processParamsTable->type,
      LIGOMETA_TYPE_MAX, "string" );
  LALSnprintf( processParamsTable.processParamsTable->value,
      LIGOMETA_TYPE_MAX, " " );
}
```

```
else
{
  fprintf( stderr, "--enable-all-ifo or --disable-all-ifo "
      "argument must be specified\n" );
  exit( 1 );
}

/*
 *
 * check the values of the arguments
 *
 */

if ( startTime < 0 )
{
  fprintf( stderr, "Error: --gps-start-time must be specified\n" );
  exit( 1 );
}

if ( endTime < 0 )
{
  fprintf( stderr, "Error: --gps-end-time must be specified\n" );
  exit( 1 );
}

if( !inputFileName )
{
  fprintf(stderr,"You must specify a bank file. Exiting.\n");
  exit(1);
}

if ( !strlen(ifos) )
{
  fprintf(stderr,"You must specify a list of ifos with --ifos. Exiting.\n");
  exit(1);
}



/* read in the triggers */
numTriggers = XLALReadInspiralTriggerFile( &inspiralEventList,
    &currentTrigger, &searchSummList, &inputFiles, inputFileName );

if ( numTriggers < 0 )
{
  fprintf(stderr, "Error reading triggers from file %s", inputFileName);
  exit( 1 );
}
else if ( numTriggers == 0 )
{
  if( vrbflg )
  {
    fprintf( stdout,
       "%s contains no triggers - the coherent bank will be empty\n",
       inputFileName );
  }
}
```

```
else
{
  if( vrbflg )
  {
    fprintf( stdout,
        "Read in %d triggers from the file %s\n", numTriggers,
        inputFileName );
  }

  /* reconstruct the coincs */
  numCoincs = XLALRecreateCoincFromSngls( &coincHead, inspiralEventList );
  if( numCoincs < 0 )
  {
    fprintf(stderr, "Unable to reconstruct coincs from single ifo triggers");
    exit( 1 );
  }
  else if ( vrbflg )
  {
    fprintf( stdout,
        "Recreated %d coincs from the %d triggers\n", numCoincs,
        numTriggers );
  }

  /*
   *
   *  Create the coherent bank
   *
   */

  if ( allIFO )
  {
    numTmplts = XLALGenerateCoherentBank( &newEventList, coincHead, ifos );
  }
  else
  {
    numTmplts = XLALGenerateCoherentBank( &newEventList, coincHead, NULL );
  }

  if ( numTmplts < 0 )
  {
    fprintf(stderr, "Unable to generate coherent bank\n");
    exit( 1 );
  }
  else if ( vrbflg )
  {
    fprintf(stdout, "Generated a coherent bank with %d templates\n",
        numTmplts);
  }
}
/*
 *
 * write the output xml file
 *
 */

/* search summary entries: */
searchsumm.searchSummaryTable->in_start_time = startTimeGPS;
```

```
searchsumm.searchSummaryTable->in_end_time = endTimeGPS;
searchsumm.searchSummaryTable->out_start_time = startTimeGPS;
searchsumm.searchSummaryTable->out_end_time = endTimeGPS;
searchsumm.searchSummaryTable->nevents = numTmplts;


if ( vrbflg ) fprintf( stdout, "writing output file... " );

/* set the file name correctly */
if ( userTag )
{
  LALSnprintf( fileName, FILENAME_MAX, "%s-COHBANK_%s-%d-%d.xml",
      ifos, userTag, startTime, endTime - startTime );
}
else
{
  LALSnprintf( fileName, FILENAME_MAX, "%s-COHBANK-%d-%d.xml",
      ifos, startTime, endTime - startTime );
}
memset( &xmlStream, 0, sizeof(LIGOLwXMLStream) );
LAL_CALL( LALOpenLIGOLwXMLFile( &status , &xmlStream, fileName),
    &status );

/* write process table */
LAL_CALL( LALGPSTimeNow ( &status, &(proctable.processTable->end_time),
      &accuracy ), &status );
LAL_CALL( LALBeginLIGOLwXMLTable( &status, &xmlStream, process_table ),
    &status );
LAL_CALL( LALWriteLIGOLwXMLTable( &status, &xmlStream, proctable,
      process_table ), &status );
LAL_CALL( LALEndLIGOLwXMLTable ( &status, &xmlStream ), &status );

/* write process_params table */
LAL_CALL( LALBeginLIGOLwXMLTable( &status, &xmlStream,
      process_params_table ), &status );
LAL_CALL( LALWriteLIGOLwXMLTable( &status, &xmlStream, processParamsTable,
      process_params_table ), &status );
LAL_CALL( LALEndLIGOLwXMLTable ( &status, &xmlStream ), &status );

/* write search_summary table */
LAL_CALL( LALBeginLIGOLwXMLTable( &status, &xmlStream,
      search_summary_table ), &status );
LAL_CALL( LALWriteLIGOLwXMLTable( &status, &xmlStream, searchsumm,
      search_summary_table ), &status );
LAL_CALL( LALEndLIGOLwXMLTable ( &status, &xmlStream ), &status );

/* write the search_summvars tabls */
/* XXX not necessary as bank file specified in arguments XXX
LAL_CALL( LALBeginLIGOLwXMLTable( &status ,&xmlStream,
      search_summvars_table), &status );
searchSummvarsTable.searchSummvarsTable = inputFiles;
LAL_CALL( LALWriteLIGOLwXMLTable( &status, &xmlStream, searchSummvarsTable,
      search_summvars_table), &status );
LAL_CALL( LALEndLIGOLwXMLTable( &status, &xmlStream), &status ); */

/* write the sngl_inspiral table if we have one*/
if ( newEventList )
```

```
{
  LAL_CALL( LALBeginLIGOLwXMLTable( &status ,&xmlStream,
        sngl_inspiral_table), &status );
  inspiralTable.snglInspiralTable = newEventList;
  LAL_CALL( LALWriteLIGOLwXMLTable( &status, &xmlStream, inspiralTable,
        sngl_inspiral_table), &status );
  LAL_CALL( LALEndLIGOLwXMLTable( &status, &xmlStream), &status );
}

LAL_CALL( LALCloseLIGOLwXMLFile( &status, &xmlStream), &status );


if ( vrbflg ) fprintf( stdout, "done\n" );


/*
 *
 * clean up the memory that has been allocated
 *
 */


if ( vrbflg ) fprintf( stdout, "freeing memory... " );

free( proctable.processTable );
free( searchsumm.searchSummaryTable );

while ( processParamsTable.processParamsTable )
{
  this_proc_param = processParamsTable.processParamsTable;
  processParamsTable.processParamsTable = this_proc_param->next;
  free( this_proc_param );
}

while ( inputFiles )
{
  thisInputFile = inputFiles;
  inputFiles = thisInputFile->next;
  LALFree( thisInputFile );
}

while ( searchSummList )
{
  thisSearchSumm = searchSummList;
  searchSummList = searchSummList->next;
  LALFree( thisSearchSumm );
}


while ( inspiralEventList )
{
  currentTrigger = inspiralEventList;
  inspiralEventList = inspiralEventList->next;
  LAL_CALL( LALFreeSnglInspiral( &status, &currentTrigger ), &status );
}

while ( newEventList )
```

```
  {
    currentTrigger = newEventList;
    newEventList = newEventList->next;
    LAL_CALL( LALFreeSnglInspiral( &status, &currentTrigger ), &status );
  }

  while ( coincHead )
  {
    thisCoinc = coincHead;
    coincHead = coincHead->next;
    LALFree( thisCoinc );
  }

  if ( userTag ) free( userTag );

  if ( vrbflg ) fprintf( stdout, "done\n" );

  LALCheckMemoryLeaks();

  exit( 0 );
}
```

Bibliography

[1] A. Pai, S. Dhurandhar and S. Bose, Phys. Rev. D **64**, 042004 (2001)

[2] W. G. Anderson, P. R. Brady, J. D. E. Creighton and E. E. Flanagan, Phys. Rev. D **63**, 042003 (2001) [arXiv:gr-qc/0008066].

[3] J. Sylvestre, Phys. Rev. D **68**, 102005 (2003) [arXiv:gr-qc/0308062].

[4] B. F. Schutz, *A First Course in General Relativity* (Cambridge University Press, Cambridge, 1985).

[5] C. W. Misner, K. S. Thorne, and J. A. Wheeler, *Gravitation* (W. H. Freeman and Company, New York, 1973)

[6] C. W. Helstrom, *Statistical Theory of Signal Detection* (Pergamon Press, London, 1968).

[7] B. J. Owen, Phys. Rev. D **53**, 6749 (1996). (gr-qc/9511032)

[8] H. Goldstein, Chapter 4, *Classical Mechanics*, 2nd edition (Addison-Wesley Publishing Company, Inc., USA, 1980).

[9] S. V. Dhurandhar and M. Tinto, Mon. Not. R. Astro. Soc. **234**, 663 (1988).

[10] S. Bose, A. Pai and S. V. Dhurandhar, Int. J. Mod. Phys. D **9**, 325 (2000)

[11] B. Allen *et al.*, *GRASP: a Data Analysis Package for Gravitational Wave Detection*, version 1.9.8. Manual and package at http://www.lsc-group.phys.uwm.edu/.

[12] S. Bose, S. V. Dhurandhar, and A. Pai, Pramana –J. Phys. **53**, 1125 (1999). (gr-qc/9906064)

[13] S. V. Dhurandhar and M. Tinto,Mon.Not.R.Astron.Soc.234, 663 (1988).

[14] S. D. Mohanty and S. V. Dhurandhar, Phys. Rev. D **54**, 7108 (1996).

[15] B. S. Sathyaprakash and S. V. Dhurandhar, Phys. Rev. D **44**, 3819 (1991); B. S. Sathyaprakash, Phys. Rev. D **50**, 7111 (1994).

[16] B. J. Owen and B. S. Sathyaprakash, Phys. Rev. D **60**, 022002 (1999) [arXiv:gr-qc/9808076].

[17] S. D. Mohanty, Phys. Rev. D **57**, 630 (1998) [arXiv:gr-qc/9703081].

[18] A. Vicere, Phys. Rev. D **66**, 062002 (2002) [arXiv:gr-qc/0112013].

[19] N. Arnaud *et al.*, Phys. Rev. D **68**, 102001 (2003) [arXiv:gr-qc/0307100].

[20] E. E. Flanagan and S. A. Hughes, Phys. Rev. D **57**, 4566 (1998) [arXiv:gr-qc/9710129].

[21] A. Pai, S. Dhurandhar and S. Bose, Phys. Rev. D **64**, 042004 (2001) [arXiv:gr-qc/0009078].

[22] S. Bose, Class. Quant. Grav. **19**, 1437 (2002).

[23] S. Bose, and S. Seader, work in progress.

[24] B. Abbott *et al.* [LIGO Scientific Collaboration], Phys. Rev. D **72**, 082002 (2005) [arXiv:gr-qc/0505042].