Technical Note	LIGO-T020140-00	-W	02.09.24
Document	*Doc Number*	*Group*	*Date*

How To Use The knownpulsardemod DSO

Title

Gregory Mendell

Author

Contents

Chapter 1

Package: knownpulsardemod

1.1 Introduction

1.1.1 Overview

The knownpulsardemod dynamic shared object (DSO) was written to search for continuous gravitational waves from known pulsars or targeted sky positions for a single set of filter parameters. It is a contribution to LALwrapper and runs from within the wrapperAPI in standalone mode or the LDAS environment. The code can be checked out from the CVS repository: :pserver:username@gravity.phys.uwm.edu:/usr/local/cvs/lalwrapper and is located in the contrib/knownpulsardemod/ directories.

The code uses functions from the Ligo Algorithms Library (LAL) to process input data. The primary function used is the LALDemod function (see Berukoff and Papa[1], LAL Software Documentation, Pulsar Package, Chapter 4; available from http://www.lsc-group.phys.uwm.edu/lal/lsd.pdf). This function, and its dependent functions, coherently sums the data to extract the signal, correcting for phase and amplitude modulation due to the motion of the detector and intrinsic frequency changes in the source. (It is assumed that the instantaneous frequency can be well modeled by a Taylor series, when viewed from the Solar System barycenter.) The LALDemod function returns a statistic, called the $\mathcal{F}$-statistic, for each frequency in the requested band. The $\mathcal{F}$-statistic is related to the log of the likelihood function, and represents a measure of the ratio of signal power to the mean power of the noise. It is defined in Jaranowski, Krolak, and Schutz[2] (gr-qc/9804014); see this paper for a complete discussion of the theory. Note that it should not be confused with the F test or F distribtion discussed in statistic and data analysis books.

The computational scheme used to compute the $\mathcal{F}$-statistic is discussed by Schutz and Papa[3] (gr-qc/9905018), Williams and Schutz[4] (gr-qc/9912029) and S. Berukoff and M. A. Papa[1]. This scheme involves two steps. Step 1 is to split the observation time into shorter segments and generate a Short-time Fourier Transform (SFT) for each segment using ordinary FFT routines. The time baseline used to compute SFTs must be chosen short enough so that the source appears monocromatic at the detector during this time. Schutz and Papa[3] give the following limit for the SFT time, for which doppler shifts induced by motion of the Earth introduces frequency shifts smaller than the SFT frequency resolution:

$$T_{\mathrm{SFT}} \leq 5.5 \times 10^3 \sqrt{300\mathrm{Hz}/f)}\ \mathrm{s}. \tag{1.1}$$

Step 2 is to input the SFT data for a narrow frequency band of interest from all the SFTs that cover the observation time and to send this data into the LALDemod function to calculate the $\mathcal{F}$-statistic. Note that Step 2 only requires the data from a narrow frequency band; thus the number of complex data points needed for Step 2, for an observation time T and frequency band Δf is $T\Delta f$. Since Δf can be chosen very small, the data needed to coherently analyse a very long observation time T can fit into the memory of a single processor. Furthermore, Step 2 can be repeated for each band of interest in parallel; making this computational scheme ideal for a parallel cluster of computers. (Note that Step 1 only needs to be done once, and also can be done on a single processor, since the number of data points in T_{SFT} is $\sim 10^{6-7}$.)

Thus, the knownpulsardemod DSO serves two purposes. The first is to generate SFTs of the gravity wave channel, with appropriate meta data. The second is to send SFT data to the LALDemod function to stitch together the SFTs to calculate the $\mathcal{F}$-statistic, and to output results based on the returned values for $\mathcal{F}$. The $\mathcal{F}$ values are analyzed to give the signal-to-noise ratio, confidence, and other statistical measures, and the outcomes are stored in the directed signal periodic (SNGL_DPERIODIC) LDAS database table, and optionally in frame or ilwd format. Summary information goes into the SEARCH_SUMMARY and SEARCH_SUMVARS LDAS database tables.

Scripts written in tcl that drive the knownpulsardemod DSO are in CVS repository: :pserver:username@gravity.phys.uwm.edu:/usr/local/cvs/ldasmdc, under the knownpulsardemod/scripts directory. These scripts send dataPipeline commands the LDAS managerAPI to run the knownpulsardemod DSO.

The SFT generating part of this DSO was tested during the November 20001 Known Pulsar Demodulation Mock Data Challenge (MDC). Documentation and results for the MDC can be checked out of the knownpulsardemod directory from the CVS repository: :pserver:username@gravity.phys.uwm.edu:/usr/local/cvs/ldasmdc, under the knownpulsardemod/doc directory, and from the LIGO Document Control Center (DCC) as LIGO-T020014-00-W. A complete specification for the SFTs is also in the MDC document and DCC document LIGO-T020043-00-W.

1.1.2 $\mathcal{F}$-statistic, SFTs, and LALDemod

The following is a brief description of the $\mathcal{F}$-statistic and its computation from SFTs via LALDemod. (This section is based on Berukoff and Papa[1], and Jaranowski, Krolak, and Schutz[2]. See these references and references therein for further details.)

To track frequency and amplitude modulation of a signal embedded in a time series x_j, we must have a model of the phase Φ_j and calculate the detector response. (The subscript j is a discrete time index.) The latter is done via two two functions a_j and b_j, which are given in terms of the beam-pattern response functions F_+ and $F_\times$ by,

$$a = (F_+ \cos 2\psi - F_\times \sin 2\psi)/\sin\zeta, \tag{1.2}$$

$$b = (F_+ \sin 2\psi + F_\times \cos 2\psi)/\sin\zeta, \tag{1.3}$$

where ψ is the polarization angle of signal, and ζ is the angle between the interferometer arms. Even though ψ appears in these equations, the quantities a_j and b_j are independent of this angle. Since, a priori, the polarization angle is not known, the quantities a_j and b_j are the convenient response functions to use in the analysis. The phase depends on the instristic frequency evolution of the source, but also on its sky position, as do the detector response functions. The dimensionless $\mathcal{F}$-statistic defined in Eq. (110) of Jaranowski, Krolak, and Schutz[2] (gr-qc/9804014), but for a component signal, is given by

$$\mathcal{F} = \frac{4}{S_h(f_0)T_0} \frac{B|F_a|^2 + A|F_b|^2 - 2C\Re(F_a F_b^*)}{D}, \tag{1.4}$$

where

$$F_a = \sum_{j=0}^{NM-1} x_j a_j e^{-2\pi i \Phi_j} \Delta t, \tag{1.5}$$

$$F_b = \sum_{j=0}^{NM-1} x_j b_j e^{-2\pi i \Phi_j} \Delta t. \tag{1.6}$$

Note that these equations represent demodulated Fourier transforms of the data for the two polarizations of the gravitational waves. Furthermore, in the above equations, Δt is the time step size, T_0 is the observation time, S_h is the noise power spectral density, and A, B, C, and D are constants related to the the mean square values of a_j and b_j and their dot product:

$$A = \frac{2}{MN} \sum_{j=0}^{NM-1} a_j a_j, \tag{1.7}$$

$$B = \frac{2}{MN} \sum_{j=0}^{NM-1} b_j b_j, \tag{1.8}$$

$$C = \frac{2}{MN} \sum_{j=0}^{NM-1} a_j b_j, \tag{1.9}$$

$$D = AB - C^2. \tag{1.10}$$

Thus, the $\mathcal{F}$-statistic is the absolute square of a weighted sum of F_a and F_b; the weights were chosen by Jaranowski, Krolak, and Schutz[2] to maximizethe value of $\mathcal{F}$ when a signal is present.

In writing the previous equation we have assumed that there is a total of $M \cdot N$ data samples. This is to indicate that the time series x_j can be divided into M segments, each of N samples. If we introduce a short-time index $0 \leq n < N-1$ and a segment index $0 \leq \alpha < M-1$, so that $j = N\alpha + n$, we can rewrite the above sum for F_a as

$$F_a = \sum_{\alpha=0}^{M-1} \sum_{n=0}^{N-1} x_{\alpha n} a_{\alpha n} e^{-2\pi i \Phi_{\alpha n}} \Delta t. \tag{1.11}$$

Now, if $X_{\alpha k}^{\mathrm{SFT}}$ is the *un-normalized* Discrete Fourier Transform (DFT) of segment α, the SFTs are defined by

$$X_{\alpha k}^{\mathrm{SFT}} = \sum_{n=0}^{N-1} x_{\alpha n} e^{-2\pi i \frac{nk}{N}}. \tag{1.12}$$

The inverse DFT of $X_{\alpha k}$ gives $x_{\alpha n}$:

$$x_{\alpha n} = \frac{1}{N} \sum_{k=0}^{N-1} X_{\alpha k}^{\mathrm{SFT}} e^{2\pi i \frac{nk}{N}}. \tag{1.13}$$

Furthermore, note that $a_{\alpha n}$ is a periodic function with period equal to one sidereal day. If we choose the segment time baseline (which will be the SFT time baseline) to be much shorter than that then $a_{\alpha n}$ does not change significantly during one segment. Thus it can be taken outside the summation over n, and evaluated at the midpoint of each segment. Making the appropriate substitutions, Eq.1.5 becomes

$$F_a = \sum_{\alpha=0}^{M-1} a_\alpha Q_\alpha \sum_{k=0}^{N-1} X_{\alpha k}^{\mathrm{SFT}} P_{\alpha k} \Delta t. \tag{1.14}$$

Thus, the constants in Eq. (1.4) can be computed more simply by

$$A = \frac{2}{M} \sum_{\alpha=0}^{M-1} a_\alpha a_\alpha, \tag{1.15}$$

$$B = \frac{2}{M} \sum_{\alpha=0}^{M-1} b_\alpha b_\alpha, \tag{1.16}$$

$$C = \frac{2}{M} \sum_{\alpha=0}^{M-1} a_\alpha b_\alpha, \tag{1.17}$$

$$D = AB - C^2. \tag{1.18}$$

If the phase evolution can be described as linear in t during each SFT time, such that the phase can be Taylor expanded about the midpoint of each segment

$$\Phi_{\alpha n} = \Phi_{\alpha 1/2} + f_{\alpha 1/2}(t_{\alpha n} - t_{\alpha 1/2}). \tag{1.19}$$

then, Q_α and $P_{\alpha k}$ can be expressed in closed form,

$$P_{\alpha k} = \frac{\sin u_{\alpha k}}{u_{\alpha k}} - i \frac{1 - \cos u_{\alpha k}}{u_{\alpha k}}, \tag{1.20}$$

$$Q_\alpha = e^{i v_\alpha}, \tag{1.21}$$

$$u_{\alpha k} = 2\pi \left(\frac{T}{M} f_{\alpha 1/2} - k \right), \tag{1.22}$$

$$v_\alpha = -2\pi \left(\Phi_{\alpha 1/2} - \frac{T}{2M} f_{\alpha 1/2} \right). \tag{1.23}$$

$$\tag{1.24}$$

Note that $P_{\alpha k}$, is strongly peaked near the fudicial frequency $k^*_\alpha = \frac{T}{M} f_{\alpha 1/2}$, and only terms near the peak need to be summed, greatly reducing the complexity of the computation. Thus, Eq. (1.14) can be rewritten as just a sum over k near k^*_α. Furthermore, we can also absorb some of the normalization factors from Eq. (1.4) into F_a. Note that $T_o = MT_{\mathrm{SFT}}$, and that S_h realistically is not constant, but potentially different for every segment and frequency, and thus should be given as $S_{\alpha k}$. Thus, we define a dimensionless version of F_a, written as $\tilde{F}_a$ and given by

$$\tilde{F}_a = \sum_{\alpha=0}^{M-1} a_\alpha Q_\alpha \sum_{k=k^*_\alpha-\Delta k}^{k=k^*_\alpha+\Delta k} \left(\frac{\Delta t X^{\mathrm{SFT}}_{\alpha k}/\sqrt{T_{\mathrm{SFT}}}}{\sqrt{S_{\alpha k}}} \right) P_{\alpha k}. \tag{1.25}$$

According to Berukoff and Papa[1], if Δk is 8 the power loss due to this approximation is less than $\sim 5\%$. The expression for $\tilde{F}_b$ is analogous to that for $\tilde{F}_a$. Given the dimensionless SFT input

$$\left(\frac{\Delta t X^{\mathrm{SFT}}_{\alpha k}/\sqrt{T_{\mathrm{SFT}}}}{\sqrt{S_{\alpha k}}} \right) \tag{1.26}$$

the LALDemod function computes Eq. (1.4) via this equation:

$$\mathcal{F} = \frac{4}{M} \frac{B|\tilde{F}_a|^2 + A|\tilde{F}_b|^2 - 2C\Re(\tilde{F}_a \tilde{F}^*_b)}{D}. \tag{1.27}$$

Note that since k is a frequency index, we only need to input SFT data ($X^{\mathrm{SFT}}_{\alpha k}$) for a narrow band of frequencies, greatly reducing the memory needs to compute the $\mathcal{F}$-statistic; and since the sum over k is only over this narrow band the complexity of the computation is greatly reduced too. These are the advantages of using the SFTs. This does not mean that SFTs are the most computationally efficient way to computer the $\mathcal{F}$-statistic. One could compute F_a and F_b via FFTs of a demodulated time series (see Sec. 3.4 of Jaranowski, Krolak, and Schutz[2]). This is more computationally efficient than using SFTs, if one want to look for signals at all frequencies, for a given source position, by roughly a factor of $2\Delta k M/\log_2(MN)$. However, the SFT algorithm is easier to run on a parallel cluster than FFTs of demodulated time series; thus in practice it is the most efficient algorithm we currently have for coherent searches for continuous gravitational waves.

1.2 Header KnownPulsarDemod.h

1.2.1 Structures

struct DemodSearchParams

This structure to hold all parameters and data that need to be initialized in LALInitSearch and LALConditionData and preserved between calls to LALApplySearch in LALWrapper.

```
typedef struct
tagDemodSearchParams
{
  /* Basic beowulf node descriptors */
  BOOLEAN searchMaster;           /* TRUE on the search master */
  UINT4   rank;                   /* Rank of this slave */
  UINT4   numSlaves;              /* Total number of slaves */
  UINT4   curSlaves;              /* Current number of slaves still working */
  UINT4   numNodes;               /* Should equal number of slaves + 1 for search master */

  UINT4  numDBOutput;             /* Number of rows written to the database */

  gpsTimeInterval times;          /* The GPS start and end time of the data */

  /* next are parameters passed as arguments */
  INT2    doDemod;                /* 0 = compute SFTs, 1 compute SFTs and Demod; > 1 input SFTs */
  INT4    mObsCoh;                /* Number of DeFTs to compute from input observation time */
  UINT4   gpsStartTime;           /* GPS start time of data requested */
  REAL8   deltaT;                 /* time step in seconds = 1/sample rate */
  UINT4   inputN;                 /* number of input data points */
  REAL8   tSFT;                   /* duration in seconds used to compute SFTs */
  INT4    nSFT;                   /* number of time domain data points used to compute one SFTs */
  UINT4   mCohSFTPerCall;         /* number of SFTs to compute during each call to ApplySearch */
  REAL8   templateRA;             /* Right Ascension of template */
  REAL8   templateDec;            /* Declination of the template */
  REAL8   templatePhase0;         /* Initial phase of template (not used)*/
  REAL8   templateFreq0;          /* frequency of template */
  REAL8   templateFreq0Band;      /* Band around f0 to compute DeFT */
  REAL8   templateFreqDeriv0;     /* first deriv of freq of template */
  REAL8   templateFreqSecondDeriv0;  /* second deriv of freq of template */
  REAL8   templateFreqThirdDeriv0;   /* third deriv of freq of template */
  REAL8   templateFreqFourthDeriv0;  /* fourth deriv of freq of template */
  REAL8   templateFreqFifthDeriv0;   /* fifth deriv of freq of template */
  CHAR    generateTestData;   /* Generate test data: y = yes, n = no. */
  UINT2   testType;           /* 0,3 = signal only; 1,4 = with noise; 2 = add test data to input data */
  REAL8   testRA;             /* Right Ascension of test */
  REAL8   testDec;            /* Declination of the test */
  REAL8   testAmplitude;      /* Amplitude of the test */
  REAL8   testPhase0;         /* phase of test */
  REAL8   testFreq0;          /* frequency of test */
  REAL8   testFreqDeriv0;            /* first deriv of freq of test */
  REAL8   testFreqSecondDeriv0;      /* second deriv of freq of test */
  REAL8   testFreqThirdDeriv0;       /* third deriv of freq of test */
  REAL8   testFreqFourthDeriv0;      /* fourth deriv of freq of test */
  REAL8   testFreqFifthDeriv0;       /* fifth deriv freq of test */
  REAL8   testPolarization;          /* Polarization of test; NOT IMPLEMENTED */
  INT4    testDeletions;      /* If deletions > 1 then create gaps between tSFT segments of test data */
  CHAR    *ifoNickName;       /* IFO nickname is H2, H1, or L1;  */
  CHAR    *IFO;               /* IFO is the really the site = LHO, LLO, GEO, etc.... */
  CHAR    *targetName;        /* A name that identifies the target of the search (e.g., Crab Pulsar)*/
  CHAR    channel[dbNameLimit];  /* Channel Name (e.g., H2:LSC-AS_Q)*/
  INT4    inputSFTm;          /* Number of input SFTs */
  INT4    inputSFTn;          /* Number of data points in the input SFT */
  REAL8   inputSFTf0;         /* Start frequency of the input SFTs */
  REAL8   inputSFTband;       /* Band width of the input SFTs. */
  INT2    outputDeFT;         /* if > 0 then will output DeFT into frame or ilwd file */

  /* next are containers for storing data */
  REAL4Vector* vecInputData;  /* data received from the wrapper master */
  LALUnit     unitSFT;        /* Unit SFTs are stored in (e.g., ADC counts per root Hertz */

  /* next are containers for storing output results */
  REAL8 freqPWF0;         /* freq detected with power pwF0 */
  REAL8 pwMax;            /* power = absolute square of the amplitude at this freq */
  REAL8 phaseForPWF0;     /* phase that goes with amplitude at F0 power */
  REAL8 pwMean;           /* mean power detected (mean computed less power in F0 bin) */
  REAL8 pwF0OverpwMean;   /* ratio of powMax to pwMean */
  REAL8 pwF0;             /* Power in the bin closest to templateFreq0 */
  REAL8 pwStdDev;         /* Standard deviation in power (less pwF0) */

  CHAR  *dsoName;         /* Name of this DSO */

  /* Percent of input time series to produce each SFT was padded to fill in gaps */
  REAL4  percentPad;

  /* parameters for keeping track of which SFTs to do per call to ApplySearch */
  UINT4 firstK;           /* Index of the first SFT to compute this call to ApplySearch */
  UINT4 lastK;            /* Index of the last SFT to compute this call to ApplySearch */
  BOOLEAN finishedSFTs;   /* Set equal to true when all SFTS for this job have been computed */

  /* Parameters used to compute SFTs */

  RealFFTPlan  *pfwdReal;      /* Plan for computing FFTs from real input. */

  REAL8 tCoh;      /* Duration of DeFTs.  For now, this is taken as the time input into this code */
  REAL8 dfSFT;     /* Freq resolution of SFT = 1/duration of SFTs  */
  INT4 mCohSFT;    /* Number of SFTs in one DeFT. */
  INT4 mObsSFT;    /* Total number of SFTs in the observation time. */
  INT4 nDeltaF;    /* Number of output data points in one SFT */

  INT4 ifMin;      /* Index of minimum frequency in SFT band */
  REAL8 fMin;      /* min freq in SFT band */
  INT4 ifMax;      /* Index of maximum frequency in SFT band */
  INT4 if0Min;     /* Min search freq index in coarse-grained frequency resolution */
  REAL8 f0Min;     /* Min freq in coarse-grained freq resolution */
  INT4 if0Max;     /* Max search freq index in coarse-grained frequency resolution */
  REAL8 f0Max;     /* Max freq in coarse-grained freq resolution */

  INT4 nDeFT;     /* Number of bins in a DeFT */
  FFT **SFTData;   /* Container for SFTs = input to LALDemod  */

  DemodPowerStats **SFTStats; /* Container for statistics about each SFT */

  REAL4Vector *tvec;       /* Temporary holder in input when computing each SFT */
  COMPLEX8Vector *fvec;    /* Temporary holder of output when computing each SFT */
  LIGOTimeGPS *timeStamps; /* Container for SFT time stamps */
  gpsTimeInterval *timeIntervals;   /* Array of SFT time intervals */

  void (*funcName)(REAL8, REAL8, REAL8, REAL8 *, REAL8 *);
  /* Test function that converts local t = GPS time to T = barycenter time and dT/dt) */

  /* pointer to ephemeris data for DemodSearchParams */
  EphemerisData *edat;

  INT4 *sftPerCoh;  /* Number of SFTs to use in a coherence time */

}
DemodSearchParams;
```

struct DemodPowerStats

This structure to hold statistics about a power series.

```
typedef struct
tagDemodPowerStats
{
REAL8 pwMax;      /* max power */
REAL8 freqPWMax;  /* frequency associated with pwMax */
REAL8 pwMean;     /* mean power */
REAL8 pwStdDev;   /* std dev of power */
}
DemodPowerStats;
```

1.2.2 Error Codes

<*name*>	code	description
NULL	1	"Null pointer"
MOBSCOH	2	"mObsCoh = number of DeFTs must be 1 in current code"
GPSTINT	3	"Unexpected GPS time interval"
DELTAT	4	"Invalid deltaT"
INPUTN	5	"Unexpected number of input data points"
TSFT	6	"Invalid tSFT"
RA	7	"Invalid right ascension"
DEC	8	"Invalid declination"
PHASE	9	"Invalid phase"
FREQ	10	"Invalid frequency +/- 0.5*band (could be negative, outside LIGO band, or to high for sample rate)"
FREQDERIV	11	"One of the frequency derivatives is possibly too large; frequency will evolve outside allowed band"
GENTEST	12	"generateTestData must be y or n"
TESTTYPE	13	"Invalid test data type"
ALOC	14	"Memory allocation error"
NODATA	15	"No input data was found"
NDATA	16	"Invalid number of input data points"
TIMESTEP	17	"Incorrect input data time step"
STARTTIME	18	"Incorrect input data start time"
STOPTIME	19	"Incorrect input data stop time"
NSFT	20	"Invalid nSFT"
NTSFT	21	"nSFT and tSFT are inconsistent with deltaT"
MCOHSFTPERCALL	22	"Invalid mCohSFTPerCall"
DCONDESC	23	"Invalid or null ifoNickName"
IFO	24	"Invalid or null IFO"
TARGETNAME	25	"Invalid or null Target Name"
DATATESTTWO	26	"Invalid input data or inputN for test type 2"
CHANNEL	27	"Invalid or null channel"
ISTARTTIME	28	"Requested GPS start time resulted in invalid index to input data."
MISSINGSFTDATA	29	"Some requested input SFT data is missing"

The status codes in the table above are stored in the constants KNOWNPULSARDEMODH_E<*name*>, and the status descriptions in KNOWNPULSARDEMODH_MSGE<*name*>. The source code with these messages is in KnownPulsarDemod.h on line l.120.

1.3 Module KnownPulsarDemod.c

All the source code for the knownpulsardemod DSO currently resides in a single module: KnownPularDemod.c.

1.3.1 Dependencies

The following include files are needed to build the knownpulsardemod DSO (some are already included in LALDemod.h):

```
#include <stdio.h>
#include <math.h>
#include <lal/LALStdlib.h>
#include <lal/AVFactories.h>
```

```
#include <lal/RealFFT.h>
#include <lal/LALConstants.h>
#include <lal/LALDemod.h>
#include <lal/DetResponse.h>
#include <lal/DetectorSite.h>
#include <lal/LALComputeAM.h>
#include <lal/ComputeSky.h>
#include <lal/LALBarycenter.h>
#include <lal/Units.h>
#include <KnownPulsarDemod.h>
```

1.3.2 Static Functions

Besides the standard LALInitSearch, LALConditionData, LALApplySearch, and LALFinalizeSearch The KnownPularDemod.c module also contains static functions for unpacking ephemeris data, handling time stamps for SFTs, and building database and multidim data output (output by LDAS in ilwd or frame format).

1.3.3 Flags

The following flags can be uncommented/commented to turn on/off certain features of the code, mostly for debugging.

```
#define INCLUDE_DEMOD_CODE
#define INCLUDE_TEST_DATA_CODE

/* #define DEBUG_KNOWNPULSARDEMOD */
/* #define DEBUG_SFT_CODE */
/* #define DEBUG_OUTPUT_SFT_FILES */
/* #define DEBUG_DEMOD_CODE */
/* #define DEBUG_KNOWNPULSARDEMODINPUTFILES */
/* #define DEBUG_KNOWNPULSARDEMODOUTPUTFILES */
/* #define DEBUG_BUILDOUTPUT_CODE */
/* #define DEBUG_BUILDFRAMEOUTPUT_CODE */
/* #define DEBUG_BUILDDEMODFRAMEOUTPUT_CODE */
/* #define DEBUG_TEST_CODE */
/* #define DEBUG_FRACREMAINING_CODE */
/* #define DEBUG_TIMEDELAY_CODE */
/* #define DEBUG_QUALITYCHANNEL_CODE */
/* #define DEBUG_INPUTSFT_CODE */
/* #define DEBUG_LALWRAPPERINITBARYCENTER */

/* UNCOMMENT ONLY ONE OF INCLUDE_LALINITBARYCENTER OR INCLUDE_LALWRAPPERINITBARYCENTER AT A TIME */
/* #define INCLUDE_LALINITBARYCENTER */
#define INCLUDE_LALWRAPPERINITBARYCENTER
```

1.3.4 LALWRAPPERInitBarycenter.c

The function in LALWRAPPERInitBarycenter.c transfers ephemeris data from LDAS structures into LAL structures. It is necessary to use this function when inputing ephemeris data from LDAS using the -responsefiles option, for example like this:

```
-responsefiles { file:/ldas_outgoing/jobs/responsefiles/earth01.ilwd,pass
file:/ldas_outgoing/jobs/responsefiles/sun01.ilwd,pass }
```

Note that for now that this function is hard coded into KnownPularDemod.c.

In stand-alone mode it is possible to input ephemeris data from LAL using the LALInitBarycenter function, by uncommenting the INCLUDE_LALINITBARYCENTER flag and commenting the INCLUDE_LALWRAPPERINITBARYCENTER flag.

1.4 Filter Parameters

The knownpulsardemod DSO performs a coherent analysis of time series data for continuous gravitational waves for a single template of the signal in two steps. Step 1 is to split the total time series to be analyzed into M segments with N data points each. Standard FFT routines in LAL are used to create Discrete Fourier Transforms of the segments; the output of the FFTs are called SFTs (for Short-time Fourier Transforms.) Step 2 is to send SFT data to the LALDemod function to coherently stitch together the SFTs and output statistics that indicate the likelihood of signal detection. Step 1 need to be done just once; Step 2 is repeated for each source to be investigated. A more complete explanation is given above in the introductory section and references cited therein.

Below are the parameters that must be defined by the user and passed through to the knownpulsardemod DSO via the -filterparams option. This option is included in a dataPipeline command that is sent to LDAS, or in a schema file used with the wrapperAPI in stand-alone mode. All parameters must be given, even if unused, in the given order, except unused parameters can be left off at the end of the list.

1.4.1 The Mode Parameter

`INT2    doDemod:`

A flag that determines the mode the knownpulsardemod DSO will run in.

Case 0: Input is time series data; DSO just computes SFTs; output is SFTs.

Case 1: Input is time series data; DSO computes SFTs and runs LALDemod on them; output is results based on the $\mathcal{F}$-statistic.

Case >1: Input is SFTs; DSO run LALDemod on them; output is results based on the $\mathcal{F}$-statistic.

1.4.2 Time Series Parameters

The following parameters define the input time series when SFTs are generated, or the time series that was used to generate SFTs when SFT data is input.

`INT4    mObsCoh:`

Number of coherent searches to perform within the observation time. Currently this must be set equal to 1. (The reason for this parameter is that LALDemod allows an observation time to be split into several coherence times. However, for now, the knownpulsardemod DSO assumes the entire input observation time is to be used in the coherent search.)

`UINT4   gpsStartTime:`

The GPS start time from which to begin the calculation (a whole number of seconds).

`REAL8   deltaT:`

The time step in seconds = 1/sample rate.

`UINT4   inputN:`

If doDemod < 2 this is the total number of input data points in the input time series. If doDemod ≥ 2 this is the total number of input data points in the input time series that were used to generate the SFTs. For example if M SFTS are input, then this parameter is M*nSFT (nSFT is defined below).

`REAL8 tSFT:`

Duration in seconds of time series data used to generate 1 SFT.

`INT4 nSFT:`

The number of time domain data points used to compute one SFTs (must equal tSFT/deltaT).

`UINT4 mCohSFTPerCall:`

When computing SFTs, number of SFTs to compute during each call to LALApplySearch. Normally this should be set to inputN/nSFT. However, if mCohSFTPerCall is less than inputN/nSFT then the DSO will exist from LALApplySearch and report partial progress in the generation of SFTs; the wrapperAPI will call LALApplySearch repeatedly until all SFTs are generated.

1.4.3 Template Parameters

The Template Parameters describe a single source. If just SFTs are being computed, i.e., if doDemod $= 0$, then the values of the template parameters do not matter (but values must be given as place holders).

Otherwise, if doDemod > 0 the source to use with LALDemod and its auxilary functions must be defined. A source is defined by a single sky position, its frequency parameters. These are the initial frequency and the first five derivatives of the frequency, all measured at the Solar System barycenter at the start of the observation time.

`REAL8 templateRA:`

Right Ascension of the source.

`REAL8 templateDec:`

Declination of the source.

`REAL8 templatePhase0:`

Initial phase of source. This is not used with LALDemod and can always be set to 0.

`REAL8 templateFreq0:`

Frequency of the source at the start of the observation time, called f_0

`REAL8 templateFreq0Band:`

Band around f_0 to compute the $\mathcal{F}$-statistic.

`REAL8 templateFreqDeriv0:`

First derivivative of frequency at the start of the observation time.

`REAL8 templateFreqSecondDeriv0:`

Second derivivative of frequency at the start of the observation time.

`REAL8 templateFreqThirdDeriv0:`

Third derivative of frequency at the start of the observation time.

`REAL8 templateFreqFourthDeriv0:`

Fourth derivative of frequency at the start of the observation time.

`REAL8 templateFreqFifthDeriv0:`

Fifth derivative of frequency at the start of the observation time.

1.4.4 Test Parameters

Test Parameters are used to generate test data within the knownpulsardemod DSO. The values of the Test Parameters are used only if generateTestData is 'y'. Otherwise these parameters do not matter (but values must be given as place holders).

`CHAR generateTestData:`

Generate test data: y = yes, n = no.

`UINT2 testType:`

The type of test data to generate:

0 = sinusoidal test signal only

1 = 0 + gaussian noise (mean = 0, std dev = 1)

2 = add test signal of type 0 to input data

3 = same as 0 but without phase or amplitude modulation due to Earth's motion

4 = same as 1 but without phase or amplitude modulation due to Earth's motion

Note that currently amplitude modulation is not included in any test data generated within the DSO.

`REAL8 testRA:`

Right Ascension of test source.

`REAL8 testDec:`

Declination of the test source.

`REAL8 testAmplitude:`

Amplitude of the test signal.

`REAL8 testPhase0:`

Initial phase of the test signal.

`REAL8 testFreq0:`

Initial frequency of test signal.

`REAL8 testFreqDeriv0:`

Initial first derivative of frequency of the test signal.

`REAL8 testFreqSecondDeriv0:`

Initial second derivative of frequency of the test signal.

`REAL8 testFreqThirdDeriv0:`

Initial third derivative of frequency of the test signal.

`REAL8 testFreqFourthDeriv0:`

Initial fourth derivative of frequency of the test signal.

`REAL8 testFreqFifthDeriv0:`

Initial fifth derivative of frequency of the test signal.

`REAL8 testPolarization:`

Polarization of the test signal. Currently this is not used.

`INT4 testDeletions:`

If testDeletions > 1 then gaps are put in the time stamps between SFTs to simulate times of lock loss in an IFO.

1.4.5 Interferometer and Source Designation Parameters

The following parameters designate which site, which interferometer, which channel, and which source are used in the analysis.

`CHAR *ifoNickName:`

This is a two character nickname of the interfereometer the data came from, e.g., H2, H1, or L1.

`CHAR *IFO:`

This is really the site of the interferometer, e.g., LHO, LLO, GEO, etc....

`CHAR *targetName:`

A name that identifies the target of the search (e.g., Crab Pulsar).

`CHAR channel[dbNameLimit]:`

The channel the input time series data came from. For example, for input time series data this should be e.g., H2:LSC-AS_Q, H1:LSC-AS_Q, or L1:LSC-AS_Q . When inputing SFT data it is best to set channel[dbNameLimit] = “NA”, for not applicable. The reason is that the DSO code thinks any data it finds with a name that matches channel[dbNameLimit] came from a time series. Input SFT data is handled differently by the DSO and is recognized by setting the paramters below.

1.4.6 Input SFT Parameters

These parameters are used when the input data to the DSO is SFT data.

`INT4 inputSFTm:`

Number of input SFTs. Normally this is the same as inputN/nSFT, but could be smaller if gaps are present in the data.

`INT4 inputSFTn:`

Number of data points in the input SFT in the input frequency band. This is the input band width inputSFTband (see below) divided by the frequency resolution of a SFT = 1/tSFT. Thus, inputSFTn = inputSFTband*tSFT.

`REAL8 inputSFTf0:`

Start frequency of the input SFT data.

`REAL8 inputSFTband:`

Band width of the input SFTs.

1.4.7 Output Option Parameter

`INT2 outputDeFT:`

This parameter applies only if doDemod > 0. If doDemod > 0 then the output statistical analysis for the template signal is always written to the database. Furthermore, if the above parameter, outputDeFT, is > 0 then the $\mathcal{F}$-statistic (defined in the introductory section) is also output, for the requested frequency band, in either ilwd or frame format (depending on the -mddapi option sent to LDAS). (Note on the name of this parameter: in the code, historically the output of LALDemod is called a DeFT, for Demodulated Fourier Transform. Currently this is really the $\mathcal{F}$-statistic, which is related to the DeFTs for the two polarizations of the signal.)

1.5 dataPipeline Commands

1.5.1 dataPipeline commands for creating SFTs

SFTs are generated by setting doDemod = 0 in the -filterparams list (see the Filter Parameteres section above).

To generate SFTs, first to pick the beginning of the observation, tObsStart, and the SFT time, tSFT, and make these mod 16 equal to zero. (Making these mod 16 equal to zero is not absolutely necessary, but it simplifies understanding which raw frame files LDAS will use to generate an SFT since each LIGO raw frame file contains 16 seconds of data.) Let M equal the number of SFTs to generate: $M = T_o/\text{tSFT}$, where T_o is the observation time we wish to analyze. The gpsStartTime filter parameter for the α'th SFT then is tObsStart + α*tSFT, for $\alpha = 0$ to $M - 1$. However, since SFTs are generated on resampled data which involves filtering, extra data is brought in on either side of the time series. Thus, in the -framequery option of the dataPipeline commands given below, the time interval requested is from gpsStartTime - 16 seconds to gpsStartTime + tSFT + 15 seconds. A slice action is used in the dataconAPI to slice out the desired time series after resampling, and the knownpulsardemod DSO checks to make sure that the final input time series starts at the time given by the gpsStartTime filter parameter and has the correct length.

To generate all the SFTs in an observation time, *the process of generating one SFT must be put into a loop* over the observation time with time step tSFT. Scripts that do this are described in the next section.

The following examples are three ways to generate an SFT using a dataPipeLine command. All three examples generate 1 SFT on the H2:LSC-AS_Q channel for gpsStartTime = 693601392 and tSFT = 2048 seconds. The input data is resampled to 4096 Hz.

Example 1, no quality channel

This is useful if the input data is test data or already known to be good and one does not need to bother with generating a quality channel. (However, note that the DSO in this case will write a -1 into a column it will label "percent_clean_lock" in the SEARCH_SUMVARS table, since it did not have information about the locked state of the IFO for this data.)

```
cmd = { dataPipeline
-returnprotocol file:/H-P_SFT_H2-693601392_2048.gwf
-subject { sftJob }
-mddapi frame
-database ldas_tst
-dynlib libldasknownpulsardemod.so.0.0.0
-np 2
-filterparams (0,1,693601392,2.44140625e-4,8388608,2048,8388608,1,0.0,0.0,
0.0,1,0.5,0.0,0.0,0.0,0.0,0.0,'n',3,0.0,0.0,2.0,0.0,1,0.0,0.0,0.0,0.0,0.0,
0.0,1,"H2","LHO","NA","H2:LSC-AS_Q")
-datacondtarget wrapper
-aliases {rgw=_ch0}
-algorithms { srgw = slice(rgw,65546,8388608,1);
output(srgw, _,_,srgw, slice rgw); }
-framequery { R H {} 693601376-693603455 Adc(H2:LSC-AS_Q!resample!4!) }
}
```

Example 2, quality channel generated on local disk

In this example, a quality channel sampled at 1 Hz for the SFT duration of 2048 seconds was generated and put in the frame file

`ascii2frame-P_SegToQC_clean_locks_H2_seg.txt-693601392-2048.gwf`

This was done using the script SegToQC.tclsh (see below) which uses an input of clean lock GPS segments and ascii2frame (from ligotools). Thus, this file has be created on the local disk before the dataPipeline command is sent to LDAS. Furthermore, the %FILE syntax in the -framequery option requires that the cmd string be sent to LDAS via LJrun from the ligotools LDAS job package, which handles getting the file to LDAS.

```
cmd = { dataPipeline
-returnprotocol file:/H-P_SFT_H2-693601392_2048.gwf
-subject { sftJob }
-mddapi frame
-database ldas_tst
-dynlib libldasknownpulsardemod.so.0.0.0
-np 2
-filterparams (0,1,693601392,2.44140625e-4,8388608,2048,8388608,1,0.0,0.0,
0.0,1,0.5,0.0,0.0,0.0,0.0,0.0,'n',3,0.0,0.0,2.0,0.0,1,0.0,0.0,0.0,0.0,0.0,
0.0,1,"H2","LHO","NA","H2:LSC-AS_Q")
-datacondtarget wrapper
-aliases {rgw=_ch0; qc1=_ch1}
-algorithms { sqc1 = slice(qc1,0,2048,1); output(sqc1,_,_,sqc1,slice qc1);
srgw = slice(rgw,65546,8388608,1); output(srgw, _,_,srgw, slice rgw); }
-framequery { { R H
%FILE(ascii2frame-P_SegToQC_clean_locks_H2_seg.txt-693601392-2048.gwf)
{} Adc(H2:qc1) }
{R H {} 693601376-693603455 Adc(H2:LSC-AS_Q!resample!4!) } }
}
```

Example 3, quality channel generated by LDAS

The dataPipeline command also has an option, -dbqualitychannel, to created a quality channel based on segments in the database. The example below shows this. (Note that the example below pushes the quality channel to the dataconAPI; the pass option can be used to send it directly to the wrapperAPI.) Long term, either this option should be used, or driver script shoulds get and handle this information from the database.

Note that the example below has NOT been used or tested with LDAS release > 0.1.0. See the LDAS documentation for futher information.

```
cmd = { dataPipeline
-returnprotocol file:/H-P_SFT_H2-693601392_2048.gwf
-subject { sftJob }
-mddapi frame
-database ldas_tst
-dynlib libldasknownpulsardemod.so.0.0.0
-np 2
-filterparams (0,1,693601392,2.44140625e-4,8388608,2048,8388608,1,0.0,0.0,
0.0,1,0.5,0.0,0.0,0.0,0.0,0.0,'n',3,0.0,0.0,2.0,0.0,1,0.0,0.0,0.0,0.0,0.0,
0.0,1,"H2","LHO","NA","H2:LSC-AS_Q")
-dbqualitychannel {{ 1 693601392 0 693603440 0 {SELECT start_time as StartSec,
start_time_ns as StartNanoSec, end_time as StopSec, end_time_ns as
StopNanoSec FROM SEGMENT WHERE ( ( start_time >= 693601392 AND start_time <= 693603440 )
OR ( end_time <= 693603440 AND end_time >= 693601392 ) ) AND
( segment_group LIKE 'H2:Locked' ) } push qc1} }
-datacondtarget wrapper
-aliases {rgw=_ch0; qc1=_ch1}
-algorithms { sqc1 = slice(qc1,0,2048,1); output(sqc1,_,_,sqc1,slice qc1);
srgw = slice(rgw,65546,8388608,1); output(srgw, _,_,srgw, slice rgw); }
-framequery { R H {} 693601376-693603455 Adc(H2:LSC-AS_Q!resample!4!) }
}
```

1.5.2 dataPipeline commands for creating $\mathcal{F}$-statistic

The $\mathcal{F}$-statistic is generated by setting doDemod > 0 in the -filterparams list (see the Filter Parameteres section above).

This dataPipeline command request data from 3 SFTs and runs the knownpulardemod DSO to generate the $\mathcal{F}$-statistic in a 0.0004 Hz band centered on 59.5 Hz. The time interval given in the -framequery option covers the observation time to get SFT data from. The Proc(0!59!1!) syntax in this option means that LDAS should read in the 59 to 60 Hz band from each SFT. Note that in general that the -aliases section must be expanded to include an alias for each input SFT and the -algorithms section must be expanded to include an output action for each SFT.

```
cmd = { dataPipeline
-returnprotocol http://sftRequestTest.out
-subject { sftRequestTest }
-mddapi ligolw
-database ldas_tst
-dynlib libldasknownpulsardemod.so.0.0.0
-np 2
-filterparams (2,1,693601392,2.44140625e-4,25165824,2048,8388608,
3,0.0,0.0,0.0,59.5,0.0004,0.0,0.0,0.0,0.0,0.0,'n',3,0.0,0.0,2.0,
0.0,1,0.0,0.0,0.0,0.0,0.0,0.0,1,"H2","LHO","NA",input,3,2048,59,
1,0)
-datacondtarget wrapper
-responsefiles { file:/ldas_outgoing/jobs/responsefiles/earth01.ilwd,pass
file:/ldas_outgoing/jobs/responsefiles/sun01.ilwd,pass }
-aliases {sft1=_ch0; sft2=_ch1; sft3=\_ch2;}
-algorithms { output(sft1,_,_,sft1:primary,sft1 data);
output(sft2,_,_,sft2,sft2 data); output(sft3,_,_,sft3,sft3 data); }
-framequery { P_LDAS_KPD_SFT_H2 H {} {693601392-693607535} Proc(0!59!1!) }
}
```

1.6 Scripts

Scripts that drive the knownpulsardemod DSO and/or that perform auxilary actions are in CVS repository: :pserver:username@gravity.phys.uwm.edu:/usr/local/cvs/ldasmdc, under the knownpulsardemod/scripts directory.

1.6.1 Driver Scripts

This script can be used to run a single LDAS job, or loop over time through jobs. The script requires tcl and that the ligotools job package be installed. The job to be run is defined in a separate job file that is sourced. The job file must set the cmd variable which hold the command to send to LDAS.

```
DriveKPDJobs.tclsh

Syntax: ./DriveKPDJobs.tclsh [-v] [-s <site>] [-ft <frametype>]
 [-ifo <interferometer>] [-ch <channel>]
 [-db <database>] [-seg <seggroup>]
 [-segfile <segfile> -scol <num> -ecol <num> -qch <chname> -rmqc ]
 [-st <startTime>] [-dt <deltaT>] [-et <endTime>] [-e <emailaddress>]
 -log <logfile> [-alog] [-blog] [-rec <recordfile>] [-arec] [-brec]
 [-run] -job <jobfile>

-v        Print output to stdout.
-s        LDAS site to run job.  Allowed sites are lho, llo, mit, uwm, sw, dev, and test.
-ft       Frame type name to substitute for FRAME_TYPE in job file (e.g., R).
-ifo      Interferometer to substitute for INTERFEROMETER in job file (e.g., H, L, or LH).
-ch       Channel name to substitute for CHANNEL_NAME in job file (e.g., H2:LSC-AS_Q).
-db       LDAS database to use, e.g., lho_test, llo_test. (Set -database DATABASE in job file.)
-seg      Database segment group to use, e.g., with -dbqualitychannel. (Sets SEGMENT_GROUP in job file.)
-segfile  File with GPS segments (intervals) to use with SegToQC.tclsh to generate quality channel.
          If this option is given, output will be ascii2frame-P_SegToQC_segfile-gpsTime-deltaT.gwf.
          To load this file into a job use \%FILE(QC_FILE) in the job file framequery.
-scol     Column in segfile with GPS segment start time.
-ecol     Column in segfile with GPS segment end time.
-qch     Channel name to give to the quality channel generated by SetToQC.
-rmqc     Remove the file, ascii2frame-P_SegToQC_segfile-gpsTime-deltaT.gwf, after the job finishes
-st       GPS time to start the analysis. Set to "now" to start now (not yet implemented).
-dt       Duration in seconds to run each job on, and next job has start time st = st + dt.
-et       GPS time to end the analysis. Ends when time >= this time.
          Set to "endless" to run continuously. Do not set to run job once.
-e        Email address for messages from this script.
-log      Print output to specified log file.
-alog     Append to an existing log file.  (Otherwise if logfile exists it is overwritten.)
-blog     Create a backup copy the logfile upon exit. Backup name is logfile.timestamp.
-rec      Print 1 line record for each job run to specified record file.
-arec     Append to an existing record file.  (Otherwise if logfile exists it is overwritten.)
-brec     Create a backup copy the record file upon exit. Backup name is recordfile.timestamp.
-run      Run a command as LDAS job.
-job      File that sets up the command to run.

To see the command without running it, leave out the -run option.
If the run option is not given, the command is written to the log file,
but the job is not started or recorded.  This is noted in the log file.

To stop the script run the command: "touch stopjobdriver" from the script's pwd.
The script will stop after the next job finishes.

To restart the script from where it left off after the last record in a record file, use this syntax:

Syntax: ./DriveKPDJobs.tclsh [-v] [-run] -restart <recordfile> [-e <emailaddress>]
```

```
 -log <logfile> [-alog] [-blog] [-rec <recordfile>] [-arec] [-brec]

The site, frametype, interferometer, channel, database, seggroup, jobfile, deltaT, and endTime
fields will be read from the last record in the file given following the -restart option.
The startTime will be set to start time of last record + deltaT. Any of these field are
overwritten if you give the command line option for that field after the -restart option.

To redo jobs in a record file that finished with an error, use this syntax:

Syntax: ./DriveKPDJobs.tclsh [-v] [-run] -redo <recordfile> [-e <emailaddress>]
 -log <logfile> [-alog] [-blog] [-rec <recordfile>] [-arec] [-brec]

All jobs in the record file given after the redo option that finished with an error will be rerun.
```

Example of how to run the script to generate SFTs (note that the sftLocalQCFile.job and clean_locks_H2_seg.txt files and SegToQC.tclsh are also needed):

```
nohup ./DriveKPDJobs.tclsh -s lho -ft R -ifo H -ch H2:LSC-AS_Q
-db lho_beta -segfile clean_locks_H2_seg.txt
-scol 1 -ecol 2 -qch H2:qc1 -rmqc -st 693601392 -dt 2048 -et 695048835
-log e7h2.log -alog -rec e7h2.rec -arec -job sftLocalQCFile.job >& h2.out &
```

Example of how to run the script to generate the $\mathcal{F}$-statistic (note the requestSFTtest.job file is also needed):

```
./DriveKPDJobs.tclsh -v -s lho -ft P_LDAS_KPD_SFT_H2 -ifo H
-ch input -db ldas_tst -st 693601392 -dt 6144 -log testsftreq.log -alog
-rec testsftreq.rec -arec -job requestSFTtest.job
```

1.6.2 SegToQC script

```
The SegToQC.tclsh script creates a quality channel of 1's and 0's from a list of clean locked GPS
segments for an IFO; it uses ascii2frame to convert the quality channel from ascii into a frame file.

Syntax: ./SegToQC.tclsh [-v] -i <segmentinfile> -scol <colnum>
        -ecol <colnum> -o <qcoutfile> -ch <qcchannelname>
        -d <detector> -st <startTime> -dt <deltaT>

-v     Print information to stdout.
-i     File with GPS intervals of locked segments.
-scol  Column segment start time is in.
-ecol  Column segment end time is in.
-o     Name of the output frame file.
-ch    The quality channel name; must start with H1:, H2:, or L1:.
-d     The detector name to use with ascii2frame (e.g., LIGO_HANFORD_2K,).
-st    GPS time to start generating quality channel from.
-dt    Duration in seconds to generate quality channel.
```

Note that for now the quality channel is always generated with a sample rate of 1 Hz and is output in a framefile using ascii2frame. Depending on the ifo nickname given, the channel name in the output file is H1:qc1, H2:qc1, or L1:qc1.

The final number printed is the percent of zeros in the quality channel.

1.6.3 Auxilary scripts

These are other scripts available. Usually just running the script with no command line options displays a help screen, or the script is just a few lines long and hopefully easy to understand.

```
CreateDataRequestFiles.tcl (Convert files to startime endtime GPS intervals)
CheckTimeIntervals.tcl  (Check that intervals are sequential)
TotalTimeIntervals.tcl  (Find total time in all intervals)
MvSftToDir.tclsh        (Move SFTs from job directories to one directory)
SegToQCtxt.tclsh  (Like SegToQC.tclsh, but outputs an ascii quality channel)
ExtractXMLdata.tclsh    (Extract data in XML file; sent to std out in ascii)
getsftdata.tclsh        (Get SFT from LDAS; return in iwld format.)
ExtractILWDdata.tclsh   (Extract data in ilwd file; sent to std out in ascii)
FitHist2.m              (Matlab script to fit histograms of Rayleigh type)
FitHist1.m              (Matlab script to fit histograms of Chi-2 type)
```

1.7 Schema Files

Below is a basic schema file used for testing the knownpulsardemod DSO in stand-alone mode. The -filterparams section can be modfied as described in the Filter Parameters sections above as long as the necessary data is in the file given after the -inputFile option.

```
#
# lam boot schema for knownpulsardemod shared object
#
h -np 2 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)"
-nodelist="(1-1)"
-dynlib="/home/gmendell/searchcode/lib/lalwrapper/libldasknownpulsardemod.so"
-dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)"
-filterparams="(1,1,62348734,9.765625e-4,4617,1.126953125,1154,4,
0.0,0.0,0.0,400,0.5,-2.0e-10,0,0,0,0,n,0,0.0,0.0,2.0,0.0,400,
-2.0e-10,0,0,0,0,0,1,NA,LHO,NA)" -realTimeRatio=0.9
-doLoadBalance=FALSE -dataDistributor=W -jobID=8
-inputFile="/home/gmendell/searchcode/share/lalwrapper/wrapper.ilwd"
```

1.8 References

[1] S. Berukoff and M. A. Papa, LAL Software Documentation, Pulsar Package, Chapter 4; available from http://www.lsc-group.phys.uwm.edu/lal/lsd.pdf

[2] P. Jaranowski, A. Krolak, and B. F. Schutz, gr-qc/9804014.

[3] B. F. Schutz and M. A. Papa, gr-qc/9905018.

[4] P. R. Williams and B. Schutz, gr-qc/9912029.

Author: Mendell, Greg A.
$Id: KnownPulsarDemod.c,v 1.15 2002/09/18 22:08:49 gmendell Exp $

Index