LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Document Type	**LIGO-T010114-E**	4-10 September 2001

Burst/Stochastic Mock Data Challenge

Warren Anderson, Maria Barnes, Keith Bayer, Kent Blackburn, Patrick Brady, Sukanta Bose, Philip Charlton, Edward Daw, Phil Ehrens, Lee Samuel Finn, Ik Siong Heng, Alexander Ivanov, Erik Katsavounidis, Albert Lazzarini, Mary Lei, Szabolcs Marka, Ed Maros, Martin McHugh, Joseph Romano, Isaac Salzman, Peter Saulson, Antony Searle, Peter Shawhan, Daniel Sigg, Julien Sylvestre, Hareem Tariq, Alan Weinstein, John Whelan, John Zweizig

Distribution of this draft:

Burst & Stochastic Sources Upper Limit Working Groups

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS 20B-145
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

WWW: http://www.ligo.caltech.edu/

Processed with LaTeX on 2001/12/10

Contents

Somebody Killed the FrameAPI
(to the melody of “Video Killed the Radio Star”)

I logged in on the wireless back at MIT
To write some scripts and run them for the MDC
I couldn’t log in to the damned reposit’ry
oh-oh-ah-oh

Spent half the night just TEXing up the tests I’d do
Gave up and went to bed on page 692
I’ve got two hours left to run my test scripts through

oh-ah-oh (it’s a disaster)
oh-ah-oh (make it run faster)
Somebody killed the frameAPI
Somebody killed the frameAPI
I thought they said it wouldn’t die
oh-ah-ah-ah oh

And now there’s Caltech up there on the webcam screen
But who’s that guy from Argonne whom we’ve never seen?
Oh no I lost my changes when I typed ’make clean’

oh-ah-oh (it’s not the first time)
oh-ah-oh (won’t be the last time)
Somebody killed the frameAPI
Somebody killed the frameAPI
We don’t know what we’re gonna try
Just throw your hands up with a sigh

Oh-ah-aho-oh
Oh-ah-aho-oh

Somebody killed the frameAPI
Somebody killed the frameAPI
It won’t read frames, we don’t know why
They tried to start it on the fly
The startup scripts could not comply
Just download them and then retry,
Oh no, LIGO’s been compromised!

Somebody killed the frameAPI
Somebody killed the frameAPI
(fade)

1 Test Definition

1.1 Overview

The purpose of the Burst/Stochastic MDC is to verify:

- the burst analysis pipeline;
- the stochastic analysis pipeline;
- the DMT and LDAS systems at LIGO/MIT;
- that LDAS hardware and software can support at least two parallel analyses;
- the LDAS remote-support model.

The burst and stochastic analysis pipeline components tested in this MDC include:

- the DMT and its ability to place triggers in the metadata database;
- the various communication models used to receive data from external sources and their ability to place triggers in the metadata database;
- the datacondAPI's ability to retrieve and pass-on triggers from the metadata database;
- the datacondAPI's ability to do simple line removal (using regression), power spectrum estimation, quadrature amplitude modulation, linear filtering, and resampling;
- the burst and stochastic shared objects' ability to correctly receive and parse data from the datacondAPI, and to return analysis results to the metadata database and to proc frames.

1.2 Hardware components to be tested

The DMT and LDAS systems at LIGO/MIT.

1.3 Software components to be tested

- Integration of LDAS system elements providing data flow from frame files through the frameAPI, datacondAPI, wrapperAPI, eventmonAPI, and metadataAPI, with results written to the metadata database and proc frames.
- Integration of the DMT and the metadata database.
- LAL and LALWrapper shared objects for the burst and stochastic searches.
- LDAS Metadata/Event Database structure at MIT, with special emphasis on the new External Triggers table.
- Communication models and code used to transfer alarms/events from external data sources.

1.4 Required input data

- Simulated data (i.e., whitened gravitational wave strain together with response functions and noise power spectra).
- Real frame data taken by the LHO and LLO interferometers during the E5 engineering run.
- Real and simulated data for the external trigger tests. This includes all GRB alarms received from the GCN network and a full day's events recorded by ALLEGRO in the year 2000.
- Data will be expected in either frame format suitable for ingestion by the frameAPI, or in ILWD files suitable for ingestion by the wrapperAPI for stand-alone tests of the various shared objects.
- Details of the noise character and the simulated signals will be chosen for the purpose of verifying and diagnosing the analysis pipeline.

1.5 Output data to be generated

- Bursts search output is expected to conform to the sngl_burst table definition of the LDAS Metadata/Event Database structure.
- Stochastic search output will consist of frequency series corresponding to the integrand of the optimally-filtered cross-correlation statistic. This data will be written to both the metadata database (summ_spectrum table) and to proc frames.
- The external trigger parsers will produce metadata database compatible output in the form of XML files, which will be readily ingestible through the putMeta LDAS command.
- Other diagnostic information will be extracted from the LDAS logs; `stderr` and `stdout` will be captured and logged for later analysis where necessary.

1.6 Required Hardware

- LDAS Beowulf at MIT: The system used during the MDC will be a 16-node Beowulf system with a dual processor Beowulf master node which can communicate with other machines outside the Beowulf's private network. A SUN E450 system with a 0.5TB raid system will be running as dataserver and metaserver, while a quad processor Linux system will be the datacondAPI server.
- DMT system at LIGO/MIT: A SUN Blade 1000 with limited disk capacity ($\approx$ 30GB) will be the dedicated DMT machine.
- A Linux box at LIGO/MIT dedicated to running standalone tests of the shared objects. This machine, called `ldas-pcdev1.mit.edu`, is an AMD-Athlon having a 1.1 GHz CPU and 1 Gbyte RAM, running RedHat Linux 7.1. /ldcg is NFS mounted from the LIGO/MIT Beowulf onto the `ldas-pcdev1.mit.edu` system whenever new versions of LAL and LALWrapper are pushed to the sites; a stand-alone version of wrapperAPI is built whenever changes to the relevant ldas components are made. The `ldasmdc` CVS repository is checked out by user `ldas_mdc`; *only* output files generated during the standalone tests should be committed from this machine.
- In addition, the MDC team will use Linux boxes (e.g., personal laptops) for code development and testing.

1.7 Required Software

Required third party software packages for LDAS are listed on the "LDAS Installation Notes" web page: http://www.ldas-dev.ligo.caltech.edu/doc/INSTALL.html. The functionality expected from each of the LDAS APIs is described below.

1.7.1 DMT requirements

Read multiple channels from archived frames (to be supplied), process them with TBD monitors, and deposit triggers in the database.

1.7.2 Frame API requirements

Ingest frame files corresponding to data taken at LHO and LLO and transmit the requested data channels to the datacondAPI. The ability to retrieve and simultaneously transmit channels from frame files originating at two different sites is specifically required.

1.7.3 Data conditioning API requirements

- Receive data from the frameAPI and metadataAPI and pass results to the wrapperAPI.
- Remove lines (using regression) from the gravitational wave strain channel.
- Resample the gravitational wave strain channel to produce a time-series of length ~ 100 s at a sampling rate of $\Delta t = 1/1024$ s.
- Linear filter the data; e.g., a high pass filter with $f_{\rm low} \sim 50$ Hz.
- Whiten the data.
- Remove the mean of a time-series.
- Estimate power spectral and cross-spectral densities with Nyquist frequencies and frequency sample intervals: $f_{\rm Nyquist} = 512$ Hz, $\Delta f = 1/16$ Hz, $N = 2 f_{\rm Nyquist}/\Delta f = 16384$ points. The variance of the spectral estimate should be $\sim 1/8$, which requires 64 averages.
- Mix the gravitational wave strain channel with a local oscillator at the reference frequency of the ALLEGRO lock-in (907 Hz) and resample to the sample rate (250 Hz) of the ALLEGRO strain signal.
- Write results to the metadata database.

1.7.4 Wrapper API requirements

Ingest ILWDs from the datacondAPI, parse them into the wrapperAPI generic data types, run shared object burst and stochastic search codes from LALwrapper, and send results through a data socket to the eventmonAPI.

1.7.5 MPI API requirements

Dispatch and manage multiple instantiations of the wrapper API.

1.7.6 Event monitor API requirements

Ingest results from the wrapper API through a data socket, determine the intended destination of the data, and then send it to either the metadataAPI, frameAPI, or ligolwAPI (through a data socket) to produce metadata, proc frames, or LIGO_LW data, as requested.

1.7.7 Metadata API requirements

Ingest event information or results originating in the DMT, datacondAPI, and search filters, and insert it into the appropriate metadata database table. Retrieve information from the metadata database and provide it to the datacondAPI.

1.8 Required Personnel

Execution of the MDC will require the presence of the LAL and LALWrapper burst and stochastic search code developers, the support of the LDAS development team, the support of the LAL and LALWrapper development team, and support personnel at LIGO/MIT and LIGO/CIT to provide system administration services. In order to test the remote support model a significant part of the LDAS development team must, for this MDC, be resident at LIGO/CIT and *not* at LIGO/MIT.

2 Test Results

Each test that forms a part of the MDC is fully described in a test checklist (see Appendix A). Each step in any particular test is described on the appropriate checklist. The successful or unsuccessful completion of each step is recorded on the checklist together with any responses. If a given test is unsuccessful, a recommendation to proceed with subsequent tests without resolving the failure (and providing a fix) may be made if the failure is not critical. All such failures will be reported in the problem tracking system should they go unaddressed. The checklists will be fully completed and form part of the official record of this MDC.

3 Criteria for evaluation

3.1 Hardware

The performance of the master node on the prototype Beowulf will be monitored to insure that it is not overloaded with communication and management tasks for multiple searches.

3.2 Software

- Correctness of results: Output data generated by the search codes should be correctly parsed by the wrapperAPI and written to the metadata database and/or proc frames. Differences between the generated output and the expected results are considered a failure.
- Error handling: All errors should be correctly handled by each software layer.
- Throughput: “Continuous” operation during the MDC with no evidence of memory leaks.
- Capacity: Provide job control sufficient to perform the tasks required by the burst and stochastic searches.

- Multi-filter capability: Show that several burst search filters can run simultaneously. Provide a quantitative measure of cpu and memory usage for each of the filters. Establish “processing time” required by each filter.

4 Execution Plan

4.1 MDC prep test

In preparation for the MDC, an official release version of LDAS, LAL, and LALWrapper shall be built and installed on the LDAS development and test systems at CIT, and mirrored to the Hanford, Livingston and MIT systems without errors. In addition, the contents of the ldasmdc CVS repository, currently stored at gravity.phys.uwm.edu:/usr/local/cvs/ldasmdc, as well as simulated and real frame data from both the LIGO Hanford and LIGO Livingston Observations taken during the E5 engineering run will be made available for use during the MDC. Worksheet MDCPREP in Sec. A.1.1.

4.2 Documentation tests

Documentation for each component of LDAS, LAL, and LALwrapper should be complete. Worksheets LDAS User Commands, LALDOC, and LALWRAPPERDOC in Sec. A.2.1, Sec. A.2.2, and Sec. A.2.3, respectively.

4.3 Data conditioning tests

- Verify that the datacondAPI can auto-regress a channel. Worksheet REGRESSION01 in Sec. A.3.1.
- Verify that the datacondAPI can regress a channel against another channel. Worksheet REGRESSION02 in Sec. A.3.2.
- Verify that the datacondAPI can regress a channel against another channel only touching selected spectral bands. Worksheet REGRESSION04 in Sec. A.3.3.
- Verify that the datacondAPI can resample 90 seconds of a channel from 16384 Hz to 1024 Hz. Worksheet RESAMPLE01 in Sec. A.3.4.
- Verify that the datacondAPI can apply a high pass linear filter to the strain channel. Worksheet LINEARFILTER01 in Sec. A.3.5.
- Verify that the datacondAPI can apply a high pass linear filter to a PEM channel. Worksheet LINEARFILTER02 in Sec. A.3.6.
- Verify that the datacondAPI can estimate a Power Spectral Density with required properties. Worksheet PSDESTIMATE01 in Sec. A.3.7.
- Verify that the datacondAPI can estimate a Cross Spectral Density with required properties. Worksheet CSDESTIMATE01 in Sec. A.3.8.
- Verify that the datacondAPI can heterodyne the strain channel from 16384 Hz to 907 Hz. Worksheet HETERODYNE01 in Sec. A.3.9.
- Verify that the datacondAPI can read time metadata from frames. Worksheet TIMESERIESMD01 in Sec. A.3.10.

- Verify that the datacondAPI can correctly set metadata after mixing. Worksheet MIXERMD01 in Sec. A.3.11.
- Verify that the datacondAPI can correctly set metadata after continued mixing. Worksheet MIXERMD02 in Sec. A.3.12.
- Verify that the datacondAPI can correctly set metadata after slicing. Worksheet SLICEMD01 in Sec. A.3.13.
- Verify that the datacondAPI can correctly set metadata after resampling. Worksheet RESAMPLEMD01 in Sec. A.3.14.
- Verify that the datacondAPI can correctly set metadata for a PSD. Worksheet PSDMD01 in Sec. A.3.15.
- Verify that the datacondAPI can correctly perform database insertions and retrievals for PSDs. Worksheet PSDDB01 in Sec. A.3.16.
- Verify that the datacondAPI can correctly perform database insertions and retrievals for CSDs. Worksheet CSDDB01 in Sec. A.3.17.

4.4 External triggers tests

- Verify that the XML output of the ALLEGRO and GCN event handlers conforms to the external_triggers table definition and that trivial entries can be inserted into and retrieved from the External Triggers table of the LDAS Metadata/Event DB structure without loss or degradation. Worksheet ALGNCEXTDBINS in Sec. A.4.1.
- Validate the full pipeline from reading the ALLEGRO event data table to inserting events into the External Triggers table. Worksheet ALFULLPIPE100 in Sec. A.4.2.
- Verify that the code can handle 10 days of ALLEGRO events and transmit them at once with no failures. This is a fairly convincing model of updating the database table after ten days of network outage. Worksheet ALFULLPIPE1000 in Sec. A.4.3.
- Verify that the code can handle ˜O(10000) ALLEGRO events and transmit them at once with no failures. This is a fairly convincing model of refilling the database table with 3 months of events after it was wiped. Worksheet ALFULLPIPE10K in Sec. A.4.4.
- Extreme stress test of the full system. Attempt to push ˜3 years (˜O(100K)) of ALLEGRO events through the pipeline and instruct LDAS to ingest them at once. Worksheet ALFULLPIPE100K in Sec. A.4.5.
- Validate the standard operating procedure for the TCP/IP model. Worksheet ALFULLPIPECONT in Sec. A.4.6.
- Verify the full parsing and event insertion sequence handling the GCN alerts (e-mail communication model) Worksheet GCNFULLPIPE100 in Sec. A.4.7.

4.5 Data Monitoring Tool tests

The DMT tests verify that the DMT can provide reliable triggers calculated by independent monitors based on live PEM data. The tests include:

- Test the status of installation of DMT Hardware and Software at MIT and verify the proper functioning of the DMT infrastructure processes. Worksheets Nameserver01 and Process01 in Secs. A.5.1 and A.5.4.
- Verify the proper functioning of monitors using offline Monte Carlo data. Worksheets glitchMon_sa_white and glitchMon_sa_glitch in Secs. A.5.2 and A.5.3.
- Verify that monitors can read online data and that multiple monitors can run simultaneously without interfering with one another. Worksheet TID0101 in Sec. A.5.5.
- Test writing to the database. Worksheet TID0201 in Sec. A.5.6.
- Cross check DMT with Burst Pipeline for same channels, algorithms and signatures. Worksheet TID0301 in Sec. A.5.7.
- Test online documentation and web pages used to establish the instrument status and whether the DMT is functioning. Worksheet DMTDOCTEST in Sec. A.5.8.

4.6 Excess power statistic tests

A set of tests designed to test the integrity of the LDAS pipelines for burst detection using the `power` search code in LALWrapper. The goal was to update and repeat tests from the MPI MDC, to determine the correctness of the code, and to examine the dependence of computational cost on input parameters.

- Test the integrity of the code by updating the tests in Sec. 07power of the MPI MDC and repeating them. Worksheet POWER00 in Sec. A.6.1.
- Verify correctness of data flow by filtering a frame file containing zeros through the LDAS pipeline. Verify data-preprocessing requirements are met, output database tables are generated correctly and loaded onto the DB. Worksheet POWER01 in Sec. A.6.2.
- Verify that bursts can be detected in Gaussian noise with the correct time and frequency information. Worksheet POWER02 in Sec. A.6.3.
- Exercise multiple choices of thresholds and verify that filter output (trigger) is in the right direction-qualitative validation of filter performance. Worksheet POWER03 in Sec. A.6.4.

4.7 Slope detection search tests

Tests of the slope detector analysis software during this MDC are intended to demonstrate that the existing slope detection software works as part of the LAL / LDAS analysis pipeline.

- Verify that the search algorithm can be applied to artificially generated data, correctly generating triggers in the LDAS database for each event detected: Worksheet SLOPE00 in Sec. A.7.1.
- Verify that the search algorithm correctly generates triggers when applied to machine data from a LIGO engineering run: Worksheet SLOPE01 in Sec. A.7.2.

4.8 Time-frequency clusters shared object test

- Test the integrity of the code by ingesting a timeseries of zeros with all triggers engaged. Worksheet TFCLUSTERS0101 in Sec. A.8.1 for data of type REAL4, and worksheet TFCLUSTERS0102 in Sec. A.8.2 for type INT2S.
- Verify the integrity of spectrogram generation and first power threshold by running on white noise. Worksheet TFCLUSTERS0201 in Sec. A.8.3.
- Verify the integrity of the first threshold of clustering analysis. Worksheet TFCLUSTERS0202 in Sec. A.8.4.
- Verify the integrity of the second threshold of clustering analysis. Worksheet TFCLUSTERS0203 in Sec. A.8.5.
- Verify the integrity of the generalized power threshold. Worksheet TFCLUSTERS0204 in Sec. A.8.6.
- Verify the integrity of event lists merging by comparing a merged and a monolitic list. Worksheet TFCLUSTERS0301 in Sec. A.8.7.
- Verify the integrity of threshold estimation by running on white noise. Worksheet TFCLUSTERS0401 in Sec. A.8.8.
- Verify the integrity of data ingestion from the datacondAPI with a timeseries of zeros. Worksheet TFCLUSTERS0501 in Sec. A.8.9.
- Verify the integrity of trigger ingestion by the database. Worksheet TFCLUSTERS0502 in Sec. A.8.10.
- Verify empirically the validity of the statistical test involving the first power threshold and clustering analysis by looking at the expected number of clusters. Worksheet TFCLUSTERS1001 in Sec. A.8.11.
- Verify empirically the validity of clusters generation by looking at the distribution of cluster sizes. Worksheet TFCLUSTERS1101 in Sec. A.8.12.
- Verify empirically the validity of the first threshold in clustering analysis. Worksheet TFCLUSTERS1201 in Sec. A.8.13.
- Verify empirically the validity of the first and second thresholds in clustering analysis. Worksheet TFCLUSTERS1202 in Sec. A.8.14.
- Verify that obvious bursts in white noise can be seen. Worksheets TFCLUSTERS1301 and TFCLUSTERS1302 in Secs. A.8.15 and A.8.16.
- Verify the correctness of the results by comparing the output of the LDAS pipeline to the output of the DMT monitor TID for a seismometer. Worksheet TFCLUSTERS1401 in Sec. A.8.17.

4.9 Stochastic IFO-IFO correlation shared object tests

Verify that the stochastic shared object for IFO-IFO correlations can be successfully incorporated into the wrapperAPI and produce expected results when run in stand-alone mode.

- Verify that the stochastic signal shared object for IFO-IFO correlations correctly rejects improperly formed search parameters, the failure is correctly logged, and the job is cleanly shutdown: Worksheet STOCHASTICIFOIFOSOERR in Sec. A.9.1.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of trivial input data when filtered assuming white noise, a delta-function response, and coïncident and coäligned detectors. Worksheet STOCHASTICIFOIFOSO00000 in Sec. A.9.2.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of trivial input data when filtered assuming white noise, a delta-function response, and LHO and LLO detector geometries. Worksheet STOCHASTICIFOIFOSO00001 in Sec. A.9.3.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of trivial input data when filtered assuming LIGO-1 noise, an idealized real response function which perfectly whitens this noise, and LHO and LLO detector geometries. Worksheet STOCHASTICIFOIFOSO00011 in Sec. A.9.4.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of trivial input data when filtered assuming LIGO-1 noise, the E2 response function, and LHO and LLO detector geometries. Worksheet STOCHASTICIFOIFOSO00021 in Sec. A.9.5.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of trivial input data when filtered assuming LIGO-1 noise, an idealized real response function which perfectly whitens this noise in one detector, the E2 response function in the other detector, and LHO and LLO detector geometries. Worksheet STOCHASTICIFOIFOSO00031 in Sec. A.9.6.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of input data consisting of white noise when filtered assuming white noise, a delta-function response, and coïncident and coäligned detectors. Worksheet STOCHASTICIFOIFOSO01000 in Sec. A.9.7.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of input data consisting of white noise when filtered assuming white noise, a delta-function response, and LHO and LLO detector geometries. Worksheet STOCHASTICIFOIFOSO01001 in Sec. A.9.8.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of input data consisting of white noise when filtered assuming LIGO-1 noise, an idealized real response function which perfectly whitens this noise, and LHO and LLO detector geometries. Worksheet STOCHASTICIFOIFOSO01011 in Sec. A.9.9.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of input data consisting of "off-white" noise (LIGO-1 noise filtered by the E2 response function) when filtered assuming LIGO-1 noise, the E2 response function, and LHO and LLO detector geometries. Worksheet STOCHASTICIFOIFOSO01021 in Sec. A.9.10.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 100 sec of input data consisting of "off-white" noise (LIGO-1 noise filtered by the E2 response function) plus a simulated stochastic background signal when filtered assuming LIGO-1 noise, the E2 response function, and LHO and LLO detector geometries. Worksheet STOCHASTICIFOIFOSO01121 in Sec. A.9.11.

- Verify that the stochastic signal shared object for IFO-IFO correlations generates correct results for ~ 15 min of input data consisting of "off-white" noise (LIGO-1 noise filtered by the E2 response function) plus a simulated stochastic background signal when filtered assuming LIGO-1 noise, the E2 re-

sponse function, and LHO and LLO detector geometries. Worksheet STOCHASTICIFOIFOSO11121 in Sec. A.9.12.

4.10 Stochastic IFO-bar correlation shared object tests

Verify that the stochastic shared object for IFO-bar correlations can be successfully incorporated into the wrapperAPI and produce expected results when run in stand-alone mode.

- Verify that the stochastic signal shared object for IFO-bar correlations generates correct results for trivial input data consisting of a single 1 with the rest of the elements 0, and coincident and coaligned detectors: Worksheet stochasticbar00000 in Sec. A.10.1.
- Verify that the stochastic signal shared object for IFO-bar correlations generates correct results for input data consisting of only white, stationary noise and coincident and coaligned detectors: Worksheet stochasticbar00001 in Sec. A.10.2.
- Verify that the stochastic signal shared object for IFO-bar correlations generates correct results for input data consisting of a sine wave and coincident and coaligned detectors: Worksheet stochasticbar00020 in Sec. A.10.3.
- Verify that the stochastic signal shared object for IFO-bar correlations generates correct results for input data consisting of white stationary noise, coincident and coaligned detectors, and a non-zero stochastic signal: Worksheet stochasticbar00011 in Sec. A.10.4.
- Verify that the stochastic signal shared object for IFO-bar correlations generates correct results for input data consisting of white Gaussian noise, "real" positions and orientations of the detectors, and no stochastic signal: Worksheet stochasticbar00101 in Sec. A.10.5.
- Verify that the stochastic signal shared object for IFO-bar correlations generates correct results for input data consisting of white Gaussian noise, "real" positions and orientations of the detectors, and a non-zero stochastic signal: Worksheet stochasticbar00111 in Sec. A.10.6.

4.11 Stochastic pipeline tests

Verify the behavior of the stochastic IFO-IFO and IFO-bar shared objects within the full LDAS pipeline.

- Verify that trivial data can pass unmolested through LDAS. Worksheet PIPELINE00 in Sec. A.11.1.
- Verify that the stochastic IFO-IFO shared object correctly rejects improper search parameters in the LDAS environment. Worksheet PIPESTOCHIFOIFOERR in Sec. A.11.2.
- Verify that the data from stand-alone test STOCHASTICIFOIFOSO00000 produces the correct output when run with synthetically generated frames and auxilliary ILWD files. Worksheet PIPESTOCHIFOIFO0000 in Sec. A.11.3.
- Verify that trivial data of duration ~ 15 minutes produces the correct output when run with synthetically generated frames and auxilliary ILWD files. Worksheet PIPESTOCHIFOIFO1000 in Sec. A.11.4.
- Verify that trivial data of duration ~ 15 minutes produces the correct output when run with synthetically generated frames, response function from an auxilliary ILWD file, and PSD calculated by the data conditioning API. Worksheet PIPESTOCHIFOIFO1010 in Sec. A.11.5.

- Verify that real frame data can be run through the LDAS pipeline for the stochastic ifo-ifo shared object, with appropriate data conditioning, and write results to proc frames and the metadata database without error. Worksheet PIPESTOCHIFOIFO1911 in Sec. A.11.6.
- Verify that the spectra written into frames by the stochastic ifo-ifo shared object in LDAS pipeline mode contain the correct metadata. Worksheet PIPESTOCHIFOIFOFRAMETA in Sec. A.11.7.

4.12 Stochastic long-term operation test

Verify that the stochastic IFO-IFO search can run "unattended" for $\sim$ 2 hours, acting on $\sim$ 6 hrs of real frame data. Worksheet LONGTERMSTOCHIFOIFO in Sec. A.12.1.

5 Conclusions and recommendations

5.1 Overview

The Burst/Stochastic MDC took place at LIGO/MIT from 4-10 September 2001. Overall, it was a success. Probably the most useful aspect of this, or any other, MDC is that it gathered together people from many different areas of LIGO data analysis (e.g., the LDAS development team, LAL and LALWrapper DSO writers from the burst and stochastic upper limit groups, DMT authors, etc.) where they could work together and learn from each other in a somewhat stressful, but highly effective, environment for exchanging information. We suspect that most people left the MDC knowing much more about e.g., schema files, tcl scripts, ldas user commands, etc. (and their role within the LIGO data analysis model) than they did when they first arrived. The scope and goals of the MDC could have been better defined in advance, although with such a large number of participants—two different upper limit groups, the DMT, five different shared objects—it was difficult to develop, in advance, as coherent a plan as we would have liked. In addition:

- The LDAS remote-site support model worked extremely well. A small part of the LDAS development team (K. Blackburn, P. Charlton, and P. Shawhan) was present at LIGO/MIT during the MDC, while the majority of the team (M. Barnes, P. Ehrens, A. Ivanov, A. Lazzarini, M. Lei, E. Maros, and I. Salzman) provided support remotely from LIGO/CIT. Teleconferences and video conferences were held twice daily to keep the LIGO/CIT contingent and off-site participants aware of daily plans, progress, problems, etc. For example, pushing changes made to the LDAS and LAL/LALWrapper code to the LIGO/MIT Beowulf were coordinated during these meetings.
- LDAS was extremely stable over the course of the MDC, with only one or two minor bug fixes required during the week. Compared to past MDCs, the majority of the time during this MDC was spent on writing, running, and debugging *search-specific* ldas user commands and LALWrapper code (for the various burst filters and stochastic cross-correlations); not on testing and fixing the LDAS infrastructure.

5.2 Failures and open bugs

Each test executed during the MDC is fully described in a test checklist in Appendix A; failures were resolved as time permitted. At the end of the week, several failures remained outstanding, which are summarized in Table 5.2.

In addition, the stochastic group had originally planned to test the `SimulateSB` routines in the `inject` package of LAL, but were unable to perform these tests during the MDC. These routines simulate stochastic background signals (having a specified spectrum $\Omega_{\rm gw}(f)$) in two gravitational wave detectors (interferometers and/or bars) whitened by the instrument response function. We will eventually need to verify

Table 1: A list of failures encountered during the MDC.

Test Name	Result	Synopsis of Problem	Problem Report
LDAS User Commands	**Pass**	Parts of the documentation for the dataPipeline user command were not correct. Since this test only checks for existence and completeness of the documentation, the test passed; however, a problem report was filed about this.	ldas/1077
LALDOC	**Fail**	Non-existent or incomplete documentation for the burstsearch, tfclusters, and slopefilters LAL packages.	lal-inject/129, lal-burstsearch/130,131,132, lal-tfclusters/133,134,135, lal-slopefilters/136
LALWRAPPERDOC	**Fail**	Non-existent or incomplete documentation for the slopefilters and tfclusters shared objects and non-inclusion of the stochastic-bar documentation into the LALWrapper Software Documentation.	lalwrapper/137, lalwrapper-slope/138, lalwrapper-tfclusters/153
CSDDB01	**Fail**	Error when retrieving a complex spectral density (CSD) estimate from the metadata database—byte ordering was reversed.	data_conditioning_api/1065
ALFULLPIPE100K	**Fail**	Error ingesting on order 100K ALLEGRO events into the metadata database.	Recommended course of action was to increase the maximum database query heap size to 5000 Kbytes.
PIPESTOCHIFOIFOFRAMETA	**Fail**	Incorrect values for sampleRate and fShift were written to proc frames.	Recommended course of action was to contact Albert Lazzarini and Benoit Mours to discuss how best to store frequency series metadata in proc frames.

that these routines can produce both white and $\Omega_{\rm gw}(f) = \Omega_0 = {\rm const}$ stochastic background signals in the LHO and LLO (and LLO and ALLEGRO) detector geometries, whitened by non-trivial instrument response functions.

[*Editor's note: The* `SimulateSB` *routines have subsequently been modularized and restructured so that they now can be extended to handle heterodyned data and arbitrary pairs of detectors (involving bars, interferometers, or spheres).*]

Also, the stochastic IFO-bar correlation tests in Secs. A.10.1-A.10.6 were not run during the MDC due

to discrepancies that were found between the expected output for trivial data (generated using Matlab code) and the output from the stochastic-bar shared object running in standalone mode. It was not known at the time of the MDC whether the discrepancies observed were actually due to the LAL/LALWrapper code failing or if there were problems with the expected output and/or schema files. As such, we decided that these tests should be rerun at a later date, in order to verify the correctness of the stochastic IFO-bar code.

[*Editor's note: As it turned out, the above discrepancies were due to errors in the expected output data and schema files. All but one of the IFO-bar tests were subsequently successfully rerun in stand-alone mode. The one test that failed had a maximum relative error of* 10^{-6}*, which was just over the acceptable threshold of* 10^{-7}*. The materials for these subsequent tests are in the November 2001 "quasi-MDC" repository.[1]*]

5.3 Recommendations arising from the MDC

Several issues remain to be addressed after the MDC:

1. While the optimal filter for correlations between LLO and LHO at operating LIGO1 design sensitivity (with a noise spectrum with its minimum around 150 Hz) has negligible support beyond 300 Hz, thus justifying the plan to perform the correlation on data downsampled to 1024 Hz (Nyquist frequency 512 Hz), the noise spectrum for the engineering runs has had its minimum around 800 Hz; this means most of the support of the optimal filter is located at higher frequencies, as illustrated in Figure 1. This MDC tested the code assumung initial LIGO design sensitivity, with data for IFO-IFO correlations downsampled to 1024 Hz, but if the Upper Limits Run data still have a peak sensitivity significantly above 150 Hz, it may be necessary to analyze data downsampled only to 2048 Hz in order to obtain the best upper limit possible.

2. For the stochastic search, we would like to have an action `z=cohere(x,y)` in the datacondAPI, which returns the coherence of two time-series x and y. Such an action would be useful when looking for cross-correlations between PEM channels (e.g., voltage monitors) from the two sites. The problem with simply constructing the coherence as `z=norm(csd(x,y))/psd(x)psd(y)` in the datacondAPI is that z would be cast as a sequence with no metadata rather than a frequency series, which currently cannot be written to the metadata database.

3. The metadata build into the FrProcData structure defined in the frame specification (e.g., samplerate) are designed with time series data in mind and not directly applicable to frequency series data such as the spectra produced by the stochastic DSOs, and hence the values currently stored there by the frameAPI are neither strictly correct nor sufficient to reconstruct the complete frequency series.

   It is important for the stochastic search code, which produces the integrand of the optimally-filtered cross-correlation statistic as output, that a mechanism be developed to represent frequency series metadata in proc frames. Similar mechanisms for metadata related to other data types (e.g., time-frequency maps) will also be needed in the future.

   [*Editor's note: It was subsequently determined that the existing dX, startX and unitX fields of the FrVect structure were best suited to generic metadata and that some of the metadata fields in the FrProcData structure were best ignored for frequency series.*]

4. After running the longterm test for the stochastic IFO-IFO shared object, the resulting frames were uploaded by changing to the directory:
   `/.../ldasmdc/burst-stochastic/test/longterm/command`
   and executing the script:
   `fetchFrames.sh`
   which has the source

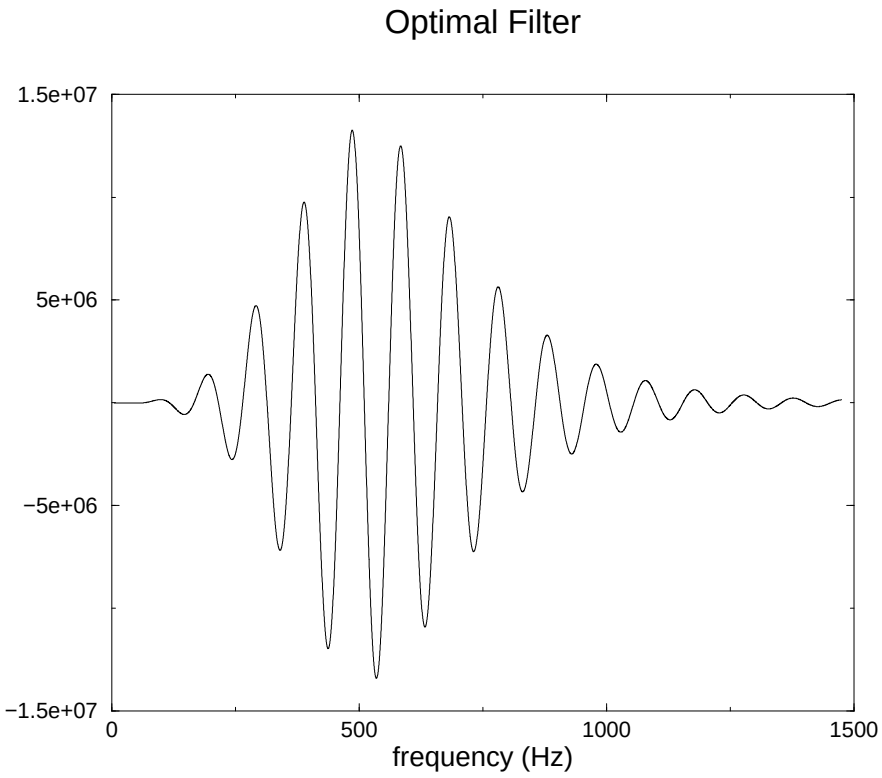

Figure 1: Optimal filter for cross-correlations between LIGO Livingston and LIGO Hanford, assuming an unwhitened noise power spectral density proportional to $\left(\frac{f}{800\,\mathrm{Hz}}\right)^{-4}+3+2\left(\frac{f}{800\,\mathrm{Hz}}\right)^{2}$, which is a reasonable "eyeball fit" to the shape of the E2 sensitivity curve available from [3].

```
#!/bin/bash
hostname=ldas.mit.edu
jids=‘cat ../output/LONGTERMSTOCHIFOIFO.txt |\
   awk ’/NORMAL*/ { OFS=" "; print substr($8,7) }’‘
for jobid in $jids ;
do
    lynx --source http://$hostname/ldas_outgoing/jobs/NORMAL$jobid/results.F > ../output/$jobid.F;
#    od -w1 -tc ../output/$jobid.F > ../output/$jobid.F.od;
done
```

The results are contained in the tar file
`/.../ldasmdc/burst-stochastic/test/longterm/output/LONGTERMSTOCHIFOIFO.tar.gz`
and can be extracted and converted to octal representations of the frame files by executing
`make octal`
from the `output` directory associated with this test. Each octal character is represented in ascii, and written once per line. It was expected that a `diff` comparison of all the frames would be identical except at octal characters in the range of 2000-2004, which are the characters which represent the least significant digits of the time stamp.

This was found not to be the case, and led to further investigation of the behavior of the stochastic

IFO-IFO shared object. It was discovered through a series of experiments that the cause for the discrepancies between the frame outputs is the *non-deterministic* behavior of FFTW's measured plans. Calculations which use the same plan *always* produce the same result. However, processes that measure their FFTW plans *can*, and sometimes will, produce non-identical results, even if all other factors involved in the calculations are identical. The 24 frames from the longterm contain three different sets of results (17 frames of one type, 5 of another, and 2 of a third). There is no obvious pattern in the ordering of these sets.

On the other hand, we found that with estimated plans, the results are identical given indentical inputs. Furthermore, we found that our code runs more quickly, since we do not do enough FFT's of sufficient length to realize the speed benefits of a measured plan. We therefore plan to change the stochastic IFO-IFO shared object to use estimated plans after this MDC. At that time, we will rerun the longterm test and verify that this produces deterministic output for the new shared object.

One final observation, which led to some confusion on our part. If data is trivial enough (we used a unit impulse for some tests), a measured plan seems to produce deterministic results. This is undoubtedly due to the fact that the vast majority of multiplications are by 0. In order to verify our results, we suggest the use of (at least) white Gaussian noise.

5. The code an communication, parsing and ingestion of external triggers is very robust for the tested communication models (e-mail, TCP-IP). The primary goal for the near future should be the implementation and ingestion of optical data from web accessible databases based on the tested software. It is also necessary to develop and test external trigger driven analysis algorithms and the necessary software for the various channels targeted (neutrino, GRB, bar-detectors and optical sources).

6. Manipulation of complex time-series in the datacondAPI was difficult at best. The action `"complex"` needs to output the same type as the input, in our case a time series. Right now, the output of `"complex"` is a sequence regardless of the input type. The action `"tseries"` was used to convert the sequence into a time series but it did not preserve the channel names of the time-series. It also sets the GPS time of the output to 0. Simple mathematical actions (`"add", "sub", "div",` etc) need to made available for complex time series.

   [*Editor's note: Changes were subsequently made to the datacondAPI, so that the action* `"complex"` *now outputs a complex time-series given two real time-series as input, and gives the resulting complex time-series a sensible name constructed from the names of the two input real time-series.*]

7. Several aspects of the bursts searches that will not be completed during the MDC are expected to be so immediately after. This includes, but is not limited to:

   - Burst filter performance for the MDC input data frames [2]: burst features extracted vs. burst features inserted.
   - Tuning of the filter(s) parameters for the burst search; efficiency and false dismissal rate of individual filters.
   - LDAS-DMT interaction through the DB and the full analysis pipeline.
   - Addressing aspects of the LIGO IFO-IFO coincidence analysis parameters

References

[1] http://fermat.utb.edu/cgi-bin/cvsweb.cgi/qmdc-200111

[2] http://www.ligo.caltech.edu/ ajw/bursts/mdc_data/mdc_data.html

[3] http://blue.ligo-wa.caltech.edu/engrun/E2/Results/Calibration/sensitivity_h.pdf

[4] LIGO Technical Document LIGO-T970130-D-E; VIRGO Technical Document VIRGO-SPE-LAP-5400-102.

[5] T. Damour, B. Iyer, and B. Sathyaprakash, *Phys. Rev. D* **63**, 044023 (2001).

[6] É.É. Flanagan, S.A. Hughes, *Phys. Rev. D* **57**, 4535-4565 (1998).

A Test checklists

The following pages contain detailed procedures that should be followed when testing the functionality and integration of the stochastic search code and LDAS.

A.1 MDC prep test

In preparation for the MDC, an official release version of LDAS shall be built and installed on the LDAS development and test systems at CIT, and mirrored to the Hanford, Livingston and MIT systems without errors. In addition, an official release version of each of LAL and LALWrapper shall be built, checked, installed and mirrored onto the same LDAS systems.

A.1.1 MDCPREP

Test Case: MDCPREP

Purpose: Prepare the environment for the Burst/Stochastic MDC.

Tester: Philip Charlton

Test machine: ldas.mit.edu

Date (mm/dd/yy): 9/8/01 **Time:** 13:43

ENVIRONMENT AND PREREQUISITES

Third party software needed for the MDC by LAL, LALWrapper, and LDAS shall be pre-installed and available to all LDAS computers and developer workstations through the LDAS /ldcg filesystem. In addition, the CVS repository for the MDC will be checked out and installed on the LDAS development and MIT systems. This will provide the necessary input data to be used by MDC tests in a subdirectory accessible to those LDAS computers and available through anonymous FTP and the web to the LIGO and LSC workstations. The required third party software consists of FFTW version 2.1.3, LAM version 6.5.4 (with LDAS specified patches), and FrameL version 4.22. The root directory for the location of input data shall be /ldas_outgoing/jobs which shall have the subdirectories listed in the PROCEDURE section below.

PROCEDURE

1. On both the LIGO/MIT LDAS system file servers, create the following directory hierarchy under the LDAS anonymous FTP area (/ldas_outgoing/jobs), which will hold the contents of the ldasmdc CVS repository currently stored at gravity.phys.uwm.edu:/usr/local/cvs/ldasmdc. This will create the shared directory /ldas_outgoing/jobs/ldasmdc/burst-stochastic, which will contain input data, test scripts built out of LDAS user commands, documentation explaining the usage for each of the tests executed during this MDC, and the results of running these test scripts. In addition, there will be made available to the LIGO/MIT LDAS system approximately 1200 seconds of frame data beginning at GPS starttime 680938500 from both the LIGO Hanford and LIGO Livingston Observatories during the E5 Engineering Run. These E5 frames will be stored in the /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/FULL_frames directory. Reduced frames will be stored in the /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/RDS_frames directory. Simulated frame data will also be stored in these directories.

   Pass

2. Verify correct installation of required software packages:

   (a) FFTW 2.1.3

   Pass

   (b) LAM 6.5.4(with LDAS patches)

   Pass

   (c) FrameL v4.22

   Pass

   (d) Approximately 1200 seconds of locked E5 Run frames for both LHO and LLO

   Pass

   (e) Mark the current, successfully built version of the LDAS S/W using the CVS tag `PRE05MDCYYMMDD`, where `YY` is the two digit year, `MM` the two digit month and `DD` the two digit day of the beginning

of this MDC.

Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: None

Notes:

TEST RESULT **Pass**

A.2 Documentation tests

The following tests are designed to verify the existence and completeness of documentation for LDAS User Commands, LAL packages, and LALWrapper shared objects relevant for this MDC.

A.2.1 LDAS User Commands

Test Case: LDAS User Commands

Purpose: Verify the existence and completeness of LDAS User Commands web-based documentation.

Tester: John Whelan

Test machine: hudson (my laptop)

Date (mm/dd/yy): 09/10/01 **Time:** 18:30 EDT

ENVIRONMENT AND PREREQUISITES

This test is executed from any LIGO user account with internet access and a web browser. It depends on the completion of MDCPREP and LDASBUILD.

PROCEDURE

1. The "LDAS User Commands" documentation can be accessed from the top level LDAS home page on the web server: www.ldas-dev.ligo.caltech.edu

   From the above web-page, click on the "LDAS User Commands" link (under the "Getting Started" section) and verify the existence of the "dataPipeline" link.

   Pass

2. Click on the "dataPipeline" link (near the bottom of the page) and verify the existence and completeness of the documentation for that command.

   Pass

3. Click on the "action syntax" link (in the -algorithms option section of the dataPipeline command) and verify the existence and completeness of the following datacondAPI actions:

   (a) value()

   Pass

   (b) rms()

   Pass

   (c) slice()

   Pass

   (d) HannWindow()

   Pass

   (e) KaiserWindow()

   Pass

   (f) RectangularWindow()

   Pass

   (g) psd()

   Pass

   (h) csd()

   Pass

   (i) linfilt()

   Pass

(j) resample()

Pass

(k) mix()

Pass

(l) sub()

Pass

(m) double()

Pass

(n) intermediate()

Pass

(o) arls()

Pass

(p) oelslr()

Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: ldas/1077

Notes: There were problems with the dataPipeline documentation which caused it to be incorrect rather than incomplete. They are documented in bug report ldas/1077 and pertain to the documentation of -filterparams describing a specialized syntax for inspiral searches and the lack of documentation for -ignoreintermediates. In addition, -outputformat was mislabelled as -returnformat, but this was fixed by the time the bug report was filed.

TEST RESULT **Pass**

A.2.2 LALDOC

Test Case: LALDOC

Purpose: Verify the existence and completeness of LAL documentation.

Tester: John Whelan

Test machine: ldas-pcdev1.mit.edu/oates.utb.edu

Date (mm/dd/yy): 09/14/01 **Time:** 16:00 CDT

ENVIRONMENT AND PREREQUISITES

This test is executed from any LIGO user account from an LDAS workstation which mounts the /ldcg filesystem. It depends on the completion of MDCPREP and LALBUILD.

PROCEDURE

1. Confirm the existence of the LAL documentation files /ldcg/doc/lal-0.9/lsd.pdf and /ldcg/doc/lal-0.9/lalspec.pdf.

   Pass

2. Using a portable document format (PDF) viewer verify that the LAL documentation is built and complete for the module `SimulateSB` in the package `inject` package.

   Pass

3. Using a portable document format (PDF) viewer verify that the LAL documentation is built and complete for package `stochastic`.

   Pass

4. Using a portable document format (PDF) viewer verify that the LAL documentation is built and complete for package `burstsearch`.

   Fail

5. Using a portable document format (PDF) viewer verify that the LAL documentation is built and complete for package `tfclusters`.

   Fail

6. Using a portable document format (PDF) viewer verify that the LAL documentation is built and complete for package `slopefilters`.

   Fail

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs: lal-inject/129; lal-burstsearch/130, 131, 132; lal-tfclusters/133, 134, 135; lal-slopefilters/136

Notes: Failures were caused by empty "Uses" sections in `burstsearch` (131) and `tfclusters` (134), a lack of documentation of the header `TFThresholds.h` (133), and a complete lack of documentation on `slopefilters` (136). The other bugs referred to incorrect rather than incomplete documentation,

namely bad cross-references in `inject` (129) and `burstsearch` (132), non-standard error code names in `TFThresholds.h` (130). and function prototypes explicitly entered rather than automatically extracted in `tfclusters` (135). Note also that the standard of completeness used for this test was stricter than that used for LALWRAPPERDOC.

TEST RESULT **Fail**

A.2.3 LALWRAPPERDOC

Test Case: LALWRAPPERDOC

Purpose: Verify the existence and completeness of LALWrapper documentation.

Tester: John Whelan

Test machine: ldas-pcdev1.mit.edu/oates.utb.edu

Date (mm/dd/yy): 09/14/01 **Time:** 18:00 CDT

ENVIRONMENT AND PREREQUISITES

This test is executed from any LIGO user account from an LDAS workstation which mounts the /ldcg filesystem. It depends on the completion of MDCPREP and LALWRAPPERBUILD.

PROCEDURE

The documentation for individual shared objects will be considered complete if there is a description of the functionality provided by the shared object, and if the filterparams and error codes are fully documented.

1. Confirm the existence of the LALWrapper documentation /ldcg/doc/lalwrapper-0.1/lalwrapper.pdf and /ldcg/doc/lalwrapper-0.1/ldas-lal.pdf, referred to hereafter as the LALWrapper documentation and the LALWrapper specification respectively.

   Pass

2. Using a portable document format (PDF) viewer verify that the LALWrapper documentation is built and contains the four parts: *How to get started*, *LDAS-LAL coding specification*, *The LAL-Wrapper Interface*, and *Contributed Shared Objects*.

   Pass

3. Using a portable document format (PDF) viewer verify that the LALWrapper documentation is built and complete for shared object `power`.

   Pass

4. Using a portable document format (PDF) viewer verify that the LALWrapper documentation is built and complete for shared object `stochastic`.

   Pass

5. Using a portable document format (PDF) viewer verify that the LALWrapper documentation is built and complete for shared object `stochasticbar`.

   Fail

6. Using a portable document format (PDF) viewer verify that the LALWrapper documentation is built and complete for shared object `tfclusters`.

   Fail

7. Using a portable document format (PDF) viewer verify that the LALWrapper documentation is built and complete for shared object `slopefilters`.

   Fail

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: lalwrapper/137; lalwrapper-slope/138; lalwrapper-tfclusters/153

Notes: Tests failed because the LALWrapper documentation does not include the `stochasticbar` and `slopefilters` documentation (137) and because the documentation for `tfclusters` is incomplete (the error codes are not documented) (153) and that for `slopefilters` is practically non-existent (138). Note that the stand-alone documentation for `stochasticbar` is complete enough that it would have passed this test if it had been included in the LALWrapper documentation.

TEST RESULT **Fail**

A.3 Data Conditioning tests

- *REGRESSION01* - Auto-regress the strain channel and check for reduced RMS.
- *REGRESSION02* - Regress the strain channel against a PEM channel and check for reduced RMS.
- *REGRESSION04* - Regress the strain channel against a PEM channel and compare before and after CSDs.
- *RESAMPLE01* - Resample the 16384 Hz strain channel to 1024 Hz for 90 seconds of data.
- *LINEARFILTER01* - Apply a 40 Hz high-pass filter to the strain channel.
- *LINEARFILTER02* - Apply a 40 Hz high-pass filter to a PEM channel.
- *PSDESTIMATE01* - Estimate a Power Spectral Density with Nyquist frequency 512 Hz, frequency delta 0.0625 and hence 16384 points, with a variance of 1/8 (thus 64 averages).
- *CSDESTIMATE01* - Estimate a Cross Spectral Density with Nyquist frequency 512 Hz, frequency delta 0.0625 and hence 16384 points, with a variance of 1/8 (thus 64 averages).
- *HETERODYNE01* - Heterodyne the 16384 Hz strain channel to 907 Hz.
- *TIMESERIESMD01* - Test that time-series meta-data is set correctly when reading from frames
- *MIXERMD01* - Test that time-series meta-data is set correctly by the mix action.
- *MIXERMD02* - Test that time-series meta-data is set correctly by two successive applications of the mix action.
- *SLICEMD01* - Test that time-series meta-data is set correctly by the slice action.
- *RESAMPLEMD01* - Test that time-series meta-data is set correctly by the resample action.
- *PSDMD01* - Test that PSD spectrum meta-data is set correctly by the psd action.
- *PSDDB01* - Verify database insertion and retrieval of a PSD.
- *CSDDB01* - Verify database insertion and retrieval of a CSD.

A.3.1 REGRESSION01

Test Case: REGRESSION01

Purpose: Auto-regress the strain channel and check for reduced RMS.

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 10:15

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/REGRESSION01.tclsh`

Input channels:

- `H2:LSC-AS_Q 680938500-1`

PROCEDURE

1. Execute the test script `datacondAPI/command/REGRESSION01.tclsh` to submit the test job to the system. Confirm that the script completed and that the `managerAPI` accepted the job.
   Job ID: 5298 **Pass**

```
#!/ldcg/bin/tclsh

## ******************************************************
## Name: REGRESSION01.tclsh
##
## Comment: Auto-regress the strain channel and check for
## reduced rms
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "REGRESSION01"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]

set times ${stime}-${etime}

set ofile "${test_name}_out.ilwd"
```

```
set subject "$test_name"
set resultname "rmsDecrease"
set resultcomment "rmsDecrease"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { y = _ch0; }
      -algorithms {
               yRMS = rms(y);
               bfloat = arls(y, 16);
               b = double(bfloat);
               w = linfilt(b, y);
               z = sub(y, w);
               zRMS = rms(z);
               rmsDecrease = sub(yRMS, zRMS);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Locate and open the test job output file `REGRESSION01_out.ilwd`. **Pass**
3. Check that the field `rmsDecrease` > 0. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.2 REGRESSION02

Test Case: REGRESSION02

Purpose: Regress the strain channel against a PEM channel and check for reduced RMS.

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 10:20

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/REGRESSION02.tclsh`

Input channels:

- `L1:LSC-AS_Q 680938500-628`
- `L0:PEM-EX_V1 680938500-628`

PROCEDURE

1. Execute the test script `datacondAPI/command/REGRESSION02.tcl` to submit the test job to the system. Confirm that the script completed and that the `managerAPI` accepted the job.
   Job ID: 5299 **Pass**

```
#!/ldcg/bin/tclsh

## ******************************************************
## Name: REGRESSION02.tclsh
##
## Comment: Regress the strain channel against a PEM
## channel and check for reduced rms
##
## Notes: Still needs email address and channel
## finalisation
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
```

```
}

# job specific values
set test_name "REGRESSION02"
set channel0 "L1:LSC-AS_Q"
set channel1 "L0:PEM-EX_V1"
set interferometers L
set stime 680938500
set duration 128
set etime [ expr $stime + $duration - 1 ]

# derived values
set times ${stime}-${etime}
set ch0alias [ mkchannel $channel0 $stime ]
set ch1alias [ mkchannel $channel1 $stime ]

set ofile "${test_name}_out.ilwd"
set subject "$test_name"

set resultname "rmsDecrease"
set resultcomment "rmsDecrease"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -framequery {
        { { L } {} $times Adc($channel0) }
        { { L } {} $times Adc($channel1) }
      }
      -aliases { yhigh = $ch0alias; u = $ch1alias; }
      -algorithms {
          y = resample(yhigh, 1, 8);
          w = oelslr(y, u, 60.0, 1.0, 3);
          yRMS = rms(y);
          z = sub(y, w);
          zRMS = rms(z);
          rmsDecrease = sub(yRMS, zRMS);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Locate and open the test job output file `REGRESSION02_out.ilwd`. **Pass**
3. Check that the field `rmsDecrease` > 0. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.3 REGRESSION04

Test Case: REGRESSION04

Purpose: Regress the strain channel against a PEM channel and compare the before and after CSDs.

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 10:20

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/REGRESSION04.tclsh`

Input channels:

- `L1:LSC-AS_Q 680938500-628`
- `L0:PEM-EX_V1 680938500-628`

PROCEDURE

1. Execute the test script `datacondAPI/command/REGRESSION04.tcl` to submit the test job to the system. Confirm that the script completed and that the `managerAPI` accepted the job.
   Job ID: 5300 **Pass**

```
#!/ldcg/bin/tclsh

## ********************************************************
## Name: REGRESSION04.tclsh
##
## Comment: Regress the strain channel against a PEM
## channel and check for reduced rms
##
## Notes: Still needs email address and channel
## finalisation
## ********************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
```

```
}

# job specific values
set test_name "REGRESSION04"
set channel0 "L1:LSC-AS_Q"
set channel1 "L0:PEM-EX_V1"
set interferometers L
set stime 680938500
set duration 128
set etime [ expr $stime + $duration - 1 ]

# derived values
set times ${stime}-${etime}
set ch0alias [ mkchannel $channel0 $stime ]
set ch1alias [ mkchannel $channel1 $stime ]

set ofile "${test_name}_out.ilwd"
set subject "$test_name"

set resultname "csd_post"
set resultcomment ""

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -framequery {
        { { L } {} $times Adc($channel0) }
        { { L } {} $times Adc($channel1) }
      }
      -aliases { yhigh = $ch0alias; u = $ch1alias; }
      -algorithms {
          y = resample(yhigh, 1, 8);
          psd(y, 2048);
          intermediate(,,psd_pre,);
          csd_pre = csd(y, u, 2048);
          abs(csd_pre);
          intermediate(,,csd_pre,);
          w = oelslr(y, u, 60.0, 5.0, 3);
          z = sub(y, w);
          psd(z, 2048);
          intermediate(,,psd_post,);
          csd_post = csd(z, u, 2048);
          abs(csd_post);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Locate and open the test job output file `REGRESSION04_out.ilwd`. **Pass**

3. Confirm that any (greater-than-epsilon) differences between `psd_pre` and `psd_post`, and between `csd_pre` and `csd_post` are confined to within $\pm$ 5 Hz of 60, 120, or 180 Hz, and that the differences are negative. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.4 RESAMPLE01

Test Case: RESAMPLE01

Purpose: Resample the 16384 Hz strain channel to 1024 Hz for 90 seconds of data.

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 10:33

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/RESAMPLE01.tclsh`

Input channels:

- `H2:LSC-AS_Q 680938500-90`

PROCEDURE

1. Execute the test script `datacondAPI/command/RESAMPLE01.tclsh` to submit the test job to the system. Confirm that the script completed and that the `managerAPI` accepted the job.
   Job ID: 5305 **Pass**

```
#!/ldcg/bin/tclsh

## ******************************************************
## Name: RESAMPLE01.tclsh
##
## Comment: Resample the 16384 Hz strain channel to 1024 Hz
## for 90 seconds of data.
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "RESAMPLE01"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 90
set etime [ expr $stime + $duration - 1 ]

set times ${stime}-${etime}
```

```
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "resampled"
set resultcomment "resampled"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { y = _ch0; }
      -algorithms {
              z = resample(y, 1, 16);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Locate and open the test job output file `RESAMPLE01_out.ilwd`. **Pass**
3. Check that the length (`dims`) of `resampled` is $1024 \times 90 = 92160$. **Pass**
4. Check that the sample rate (`SampleRate`) of `resampled` is 1024 Hz. **Pass**
5. Check that the base frequency (`BaseFrequency`) of `resampled` is 0 Hz. **Pass**
6. Check that the phase (`Phase`) of `resampled` is 0. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.5 LINEARFILTER01

Test Case: LINEARFILTER01

Purpose: Apply a 40 Hz high-pass filter to the strain channel.

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 10:36

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/LINEARFILTER01.tcl`
- `ldasmdc_data/burst-stochastic/hp-coefficents.ilwd`

Input channels:

- `H2:LSC-AS_Q 680938500-90`

PROCEDURE

1. Execute the test script `datacondAPI/command/LINEARFILTER01.tcl` to submit the test job to the system. Confirm that the script completed and that the `managerAPI` accepted the job.
   Job ID: 5307 **Pass**

```
#!/ldcg/bin/tclsh

## ******************************************************
## Name: LINEARFILTER01.tclsh
##
## Comment: Apply a 40 Hz high-pass filter to the data
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "LINEARFILTER01"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 90
set etime [ expr $stime + $duration - 1 ]

set times ${stime}-${etime}
```

```
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "psd_post"
set resultcomment "psd after high-pass filtering"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputprotocol
file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/hp-coefficents.ilwd
       -subject {
$subject }       -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { y = _ch0; a = coefficents:a; b = coefficents:b; }
      -algorithms {
               z = resample(y, 1, 16);
               psd(z, 1024);
               intermediate(,,psd_pre,{psd before high-pass filtering});
               x = linfilt(b, a, z);
               psd(x, 1024);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Locate and open the test job output file `LINEARFILTER01_out.ilwd`. **Pass**
3. Check that the `psd_post` differs from/is less that the `psd_pre` only in the 0-40 Hz bandwidth. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.6 LINEARFILTER02

Test Case: LINEARFILTER02

Purpose: Apply a 40 Hz high-pass filter to a PEM channel.

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 11:24

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/LINEARFILTER02.tcl`
- `ldasmdc_data/burst-stochastic/hp-coefficents.ilwd`

Input channels:

- `H0:PEM-EX_V1 680938500-90`

PROCEDURE

1. Execute the test script `datacondAPI/command/LINEARFILTER02.tcl` to submit the test job to the system. Confirm that the script completed and that the `managerAPI` accepted the job.
   Job ID: 5324 **Pass**

```
#!/ldcg/bin/tclsh

## ******************************************************
## Name: LINEARFILTER02.tclsh
##
## Comment: Apply a 40 Hz high-pass filter to the data
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "LINEARFILTER02"
set channel "H0:PEM-EX_V1"
set interferometers H
set stime 680938500
set duration 90
set etime [ expr $stime + $duration - 1 ]

set times ${stime}-${etime}
```

```
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "psd"
set resultcomment "psd after high-pass filtering"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputprotocol
file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/hp-coefficents.ilwd
       -subject {
$subject }       -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { y = _ch0; a = coefficents:a; b = coefficents:b; }
      -algorithms {
               z = resample(y, 1, 16);
               psd(z, 1024);
               intermediate(,,psd_pre, psd before high-pass filtering);
               x = linfilt(b, a, z);
               psd(x, 1024);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Locate and open the test job output file `LINEARFILTER02_out.ilwd`. **Pass**
3. Check that the `psd_post` differs from/is less that the `psd_pre` only in the 0-40 Hz bandwidth. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.7 PSDESTIMATE01

Test Case: PSDESTIMATE01

Purpose: Estimate a Power Spectral Density with Nyquist frequency 512 Hz, frequency delta 0.0625 and hence 16384 points, with a variance of 1/8 (thus 64 averages).

Tester: Antony C. Searle

Test machine: ligo.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 11:28

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/PSDESTIMATE01.tcl`

Input channels:

- `H2:LSC-AS_Q 680938500-9524`

PROCEDURE

1. Execute the test script `datacondAPI/command/PSDESTIMATE01.tclsh` to submit the test job to the system. Confirm that the script completed and that the `managerAPI` accepted the job.
   Job ID: 5326 **Pass**

```
#!/ldcg/bin/tclsh

## ******************************************************
## Name: PSDESTIMATE01.tclsh
##
## Comment: Estimate a Power Spectral Density with Nyquist
## frequency 512 Hz, frequency delta 0.0625 Hz and hence
## 16384 points, with a variance of 1/8 (thus 64 averages)
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "PSDESTIMATE01"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1024
```

```
set etime [ expr $stime + $duration - 1 ]

set times ${stime}-${etime}

set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "psd"
set resultcomment "power spectral density estimate"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { y = _ch0; }
      -algorithms {
                z = resample(y, 1, 16);
                x = psd(z, 16384);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Locate and open the test job output file `PSDESTIMATE01_out.ilwd`. **Pass**
3. Check that the length (`dims`) of `psd` is 16384. **Pass**
4. Check that the frequency base (`FrequencyBase`) of `psd` is -512 Hz. **Pass**
5. Check that the frequency delta(`FrequencyDelta`) of `psd` is 0.0625 Hz. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.8 CSDESTIMATE01

Test Case: CSDESTIMATE01

Purpose: Estimate a Cross Spectral Density with Nyquist frequency 512 Hz, frequency delta 0.0625 and hence 16384 points, with a variance of 1/8 (thus 64 averages).

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 11:32

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/CSDESTIMATE01.tclsh`

Input channels:

- `H2:LSC-AS_Q 680938500-9524`
- `L1:LSC-AS_Q 680938500-9524`

PROCEDURE

1. Execute the test script `datacondAPI/command/CSDESTIMATE01.tclsh` to submit the test job to the system. Confirm that the script completed and that the `managerAPI` accepted the job.
   Job ID: 5326 **Pass**

```
#!/ldcg/bin/tclsh

## ********************************************************
## Name: CSDESTIMATE01.tclsh
##
## Comment: Estimate a Cross Spectral Density with Nyquist
## frequency 512 Hz, frequency delta 0.0625 Hz and hence
## 16384 points, with a variance of 1/8 (thus 64 averages)
## ********************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
```

```
}

# job specific values
set test_name "CSDESTIMATE01"
set channel0 "H2:LSC-AS_Q"
set channel1 "L1:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1024
set etime [ expr $stime + $duration - 1 ]

set times ${stime}-${etime}
set ch0alias [ mkchannel $channel0 $stime ]
set ch1alias [ mkchannel $channel1 $stime ]

set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "csd"
set resultcomment "cross spectral density estimate"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -framequery {
        { { H } {} $times Adc($channel0) }
        { { L } {} $times Adc($channel1) }
      }
      -aliases { y = $ch0alias; z = $ch1alias; }
      -algorithms {
                ylow = resample(y, 1, 16);
                zlow = resample(z, 1, 16);
                x = csd(ylow, zlow, 16384);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Locate and open the test job output file `CSDESTIMATE01_out.ilwd`. **Pass**
3. Check that the length (`dims`) of `psd` is 16384. **Pass**
4. Check that the frequency base (`FrequencyBase`) of `csd` is -512 Hz. **Pass**
5. Check that the frequency delta(`FrequencyDelta`) of `csd` is 0.0625 Hz. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.9 HETERODYNE01

Test Case: HETERODYNE01

Purpose: Heterodyne the 16384 Hz strain channel to 907 Hz.

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 11:37

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/HETERODYNE01.tclsh`

Input channels:

- `H2:LSC-AS_Q 680938500-90`

PROCEDURE

1. Execute the test script `datacondAPI/command/HETERODYNE01.tclsh` to submit the test job to the system. Confirm that the script completed and that the `managerAPI` accepted the job.
   Job ID: 5322 **Pass**

```
#!/ldcg/bin/tclsh

## ******************************************************
## Name: HETERODYNE01.tclsh
##
## Comment: Heterodyne the 16384 Hz strain channel with 907
## Hz then resample to 250 Hz.
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "HETERODYNE01"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 90
set etime [ expr $stime + $duration - 1 ]

set times ${stime}-${etime}

set ofile "${test_name}_out.ilwd"
```

```
set subject "$test_name"
set resultname "heterodyned"
set resultcomment "heterodyned"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { y = _ch0; }
      -algorithms {
               x = mix(0, 0.1107177734375, y);
               z1 = resample(x, 5, 16);
               z2 = resample(z1, 5, 16);
               z3 = resample(z2, 5, 32);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Locate and open the test job output file `HETERODYNE01_out.ilwd`. **Pass**
3. Check that the length (`dims`) of `resampled` is $250 \times 90 = 22500$. **Pass**
4. Check that the sample rate (`SampleRate`) of `resampled` is 250 Hz. **Pass**
5. Check that the base frequency (`BaseFrequency`) of `resampled` is 907 Hz. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.10 TIMESERIESMD01

Test Case: TIMESERIESMD01

Purpose: To test that time-series meta-data is set correctly when reading from frames

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 11:46

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/TIMESERIESMD0101.tclsh`
- Frame files `H-680938500.F` to `H-680939699.F` (locked E5 data)

PROCEDURE

1. Execute the test script `datacondAPI/command/TIMESERIESMD0101.tclsh`. Confirm that the `managerAPI` accepted the job.
   Job ID: 5334 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: TIMESERIESMD0101.tclsh,v 1.4 2001/09/04 22:04:58 acsearle Exp $
;##
;## Test that initial time-series meta-data is correct
;##

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "TIMESERIESMD0101"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]

# derived values
set srate_exp 16384.0
set dims_exp [ expr int($duration*$srate_exp) ]
set freq_exp 0.0
set phi_exp 0.0
set gps_start_s_exp $stime
```

```
set gps_start_ns_exp 0

set offset [ expr ($duration*$srate_exp - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp $channel

set times ${stime}-${etime}

set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        value($dims_exp);
        intermediate(,,dims_exp, {Expected value of dims});

        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        value(gw);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

3. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/TIMESERIESMD0101_out.ilwd`

Pass

4. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
dims	16384	Yes
SampleRate	16384	Yes
BaseFrequency	0	Yes
Phase	0	Yes
GPSStartS	680938500	Yes
GPSStartNS	0	Yes
GPSEndS	680938500	Yes
GPSEndNS	99938964	Yes

Pass

5. Verify that the data in the final result is an array of `real_4` values.

Pass

6. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.11 MIXERMD01

Test Case: MIXERMD01

Purpose: To test that time-series meta-data is set correctly by the mix action

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 11:52

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/MIXERMD0101.tclsh`
- Frame files `H-680938500.F` to `H-680939699.F` (locked E5 data)

PROCEDURE

1. Execute the test script `datacondAPI/command/MIXERMD0101.tclsh`. Confirm that the `managerAPI` accepted the job.
   Job ID: 5335 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: MIXERMD0101.tclsh,v 1.4 2001/09/04 13:22:36 acsearle Exp $
;##
;## Test that the mix action sets time-series meta-data correctly
;##

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "MIXERMD0101"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]

# mix parameters
set phi 1.0
set freq [ expr 1.0/16.0 ]

#derived values
set srate_exp 16384.0
```

```
set dims_exp [ expr int($duration*$srate_exp) ]
set phi_exp $phi
set freq_exp [ expr $freq*$srate_exp/2.0 ]
set gps_start_s_exp $stime
set gps_start_ns_exp 0

set offset [ expr ($duration*$srate_exp - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "mix($phi, $freq, $channel)"

set times ${stime}-${etime}

set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        value($dims_exp);
        intermediate(,,dims_exp, {Expected value of dims});

        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        mix($phi, $freq, gw);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

3. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/MIXERMD0101_out.ilwd`

Pass

4. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Expected Value	Match Obtained?
`dims`	16384	Yes
`SampleRate`	16384 Hz	Yes
`BaseFrequency`	512 Hz	Yes
`Phase`	1.0	Yes
`GPSStartS`	680938500	Yes
`GPSStartNS`	0	Yes
`GPSEndS`	680938500	Yes
`GPSEndNS`	999938964	Yes

Pass

5. Verify that the data in the final result is an array of `complex_8` values.

Pass

6. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.12 MIXERMD02

Test Case: MIXERMD02

Purpose: To test that time-series meta-data is set correctly by two successive applications of the mix action

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 11:55

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/MIXERMD0201.tclsh`
- Frame files `H-680938500.F` to `H-680939699.F` (locked E5 data)

PROCEDURE

1. Execute the test script `datacondAPI/command/MIXERMD0201.tclsh`. Confirm that the `managerAPI` accepted the job.
   Job ID: 5337 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: MIXERMD0201.tclsh,v 1.3 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that sequential mix actions set time-series meta-data correctly
;##

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "MIXERMD0201"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]

# mix parameters
set phi 1.0
set freq [ expr 1.0/16.0 ]

set phi2 1.5
set freq2 [ expr 1.0/8.0 ]
```

```
#derived values
set srate_exp 16384.0
set dims_exp [ expr int($duration*$srate_exp) ]
set phi_exp [ expr $phi + $phi2 ]
set freq_exp [ expr ($freq + $freq2)*$srate_exp/2.0 ]
set gps_start_s_exp $stime
set gps_start_ns_exp 0

set offset [ expr ($duration*$srate_exp - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "mix($phi2, $freq2, mix($phi, $freq, $channel))"

set times ${stime}-${etime}

set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        value($dims_exp);
        intermediate(,,dims_exp, {Expected value of dims});

        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        m0 =  mix($phi, $freq, gw);
        m1 =  mix($phi2, $freq2, m0);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

2. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

3. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/MIXERMD0201_out.ilwd`

Pass

4. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the

expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
`dims`	16384	Yes
`SampleRate`	16384 Hz	Yes
`BaseFrequency`	1536 Hz	Yes
`Phase`	2.5	Yes
`GPSStartS`	680938500	Yes
`GPSStartNS`	0	Yes
`GPSEndS`	680938500	Yes
`GPSEndNS`	999938964	Yes

Pass

5. Verify that the data in the final result is an array of `complex_8` values.

Pass

6. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.13 SLICEMD01

Test Case: SLICEMD01

Purpose: To test that time-series meta-data is set correctly by the slice action

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 11:58

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/SLICEMD0101.tclsh`
- `datacondAPI/command/SLICEMD0102.tclsh`
- `datacondAPI/command/SLICEMD0103.tclsh`
- `datacondAPI/command/SLICEMD0104.tclsh`
- Frame files `H-680938500.F` to `H-680939699.F` (locked E5 data)

PROCEDURE

1. Execute the test script `datacondAPI/command/SLICEMD0101.tclsh`. Confirm that the `managerAPI` accepted the job.
   Job ID: 5338 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: SLICEMD0101.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the slice action sets time-series meta-data correctly
;##
;## Case 1: first element only

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "SLICEMD0101"
set channel "H2:LSC-AS_Q"
```

```
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameters
set phi 1.0
set freq [ expr 1.0/16.0 ]

# slice parameters
set start 0
set length 1
set stride 1

set PI 3.14159265358979

# derived values
set srate_exp [ expr $srate/$stride ]
set freq_exp [ expr $freq*($srate/2.0) ]
set phi_exp [ expr fmod($phi + $start*$PI*$freq, 2*$PI) ]

set offset [ expr $start/$srate ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_start_s_exp [ expr $stime + $offset_s ]
set gps_start_ns_exp [ expr 0 + $offset_ns ]

set offset [ expr ($length - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "slice(mix($phi, $freq, $channel), $start, $length, $stride)"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        m0 = mix($phi, $freq, gw);

        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        # Expected values should be checked against final result
        slice(m0, $start, $length, $stride);
      }
      -datacondtarget datacond
  }
"
sendCmd $cmd $host $port

exit
```

2. Verify that an email was received indicating where the results are to be found and that no errors were

reported.

Pass

3. Verify that an output file was created at
`http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/SLICEMD0101_out.ilwd`

Pass

4. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
`SampleRate`	16384	Yes
`BaseFrequency`	512	Yes
`Phase`	1	Yes
`GPSStartS`	680938500	Yes
`GPSStartNS`	0	Yes
`GPSEndS`	680938500	Yes
`GPSEndNS`	0	Yes

Pass

5. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.
Pass

6. Execute the test script `datacondAPI/command/SLICEMD0102.tclsh`. Confirm that the `managerAPI` accepted the job.
Job ID: 5339 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: SLICEMD0102.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the slice action sets time-series meta-data correctly
;##
;## Case 2: Last element only

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}
```

```
# job specific values
set test_name "SLICEMD0102"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameters
set phi 1.0
set freq [ expr 1.0/16.0 ]

# slice parameters
set start 16383
set length 1
set stride 1

set PI 3.14159265358979

# derived values
set srate_exp [ expr $srate/$stride ]
set freq_exp [ expr $freq*($srate/2.0) ]
set phi_exp [ expr fmod($phi + $start*$PI*$freq, 2*$PI) ]

set offset [ expr $start/$srate ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_start_s_exp [ expr $stime + $offset_s ]
set gps_start_ns_exp [ expr 0 + $offset_ns ]

set offset [ expr ($length - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "slice(mix($phi, $freq, $channel), $start, $length, $stride)"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        m0 = mix($phi, $freq, gw);

        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        # Expected values should be checked against final result
        slice(m0, $start, $length, $stride);
      }
      -datacondtarget datacond
  }
"
sendCmd $cmd $host $port

exit
```

7. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

8. Verify that an output file was created at
`http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/SLICEMD0102_out.ilwd`

Pass

9. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
`SampleRate`	16384	Yes
`BaseFrequency`	512	Yes
`Phase`	0.807	Yes
`GPSStartS`	680938500	Yes
`GPSStartNS`	99938964	Yes
`GPSEndS`	680938500	Yes
`GPSEndNS`	999938964	Yes

Pass

10. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.
Pass

11. Execute the test script `datacondAPI/command/SLICEMD0103.tclsh`. Confirm that the `managerAPI` accepted the job.
Job ID: 5341 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: SLICEMD0103.tclsh,v 1.2 2001/09/04 22:04:58 acsearle Exp $
;##
;## Test that the mix action sets time-series meta-data correctly
;##
;## Case 3: Middle elements of timeseries, no decimation

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
```

```
}

# job specific values
set test_name "SLICEMD0103"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameters
set phi 1.0
set freq [ expr 1.0/16.0 ]

# slice parameters
set start 8191
set length 8
set stride 1

set PI 3.14159265358979

# derived values
set srate_exp [ expr $srate/$stride ]
set freq_exp [ expr $freq*($srate/2.0) ]
set phi_exp [ expr fmod($phi + $start*$PI*$freq, 2*$PI) ]

set offset [ expr $start/$srate ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_start_s_exp [ expr $stime + $offset_s ]
set gps_start_ns_exp [ expr 0 + $offset_ns ]

set offset [ expr ($length - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "slice(mix($phi, $freq, $channel), $start, $length, $stride)"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        m0 = mix($phi, $freq, gw);

        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        # Expected values should be checked against final result
        slice(m0, $start, $length, $stride);
      }
      -datacondtarget datacond
  }
"
sendCmd $cmd $host $port
```

```
exit
```

12. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

13. Verify that an output file was created at
`http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/SLICEMD0103_out.ilwd`

Pass

14. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
SampleRate	16384	Yes
BaseFrequency	512	Yes
Phase	0.804	Yes
GPSStartS	680938500	Yes
GPSStartNS	99938964	Yes
GPSEndS	680938500	Yes
GPSEndNS	500366210	Yes

Pass

15. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.

Pass

16. Execute the test script `datacondAPI/command/SLICEMD0104.tclsh`. Confirm that the `managerAPI` accepted the job.
Job ID: 5342 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: SLICEMD0104.tclsh,v 1.2 2001/09/04 22:04:58 acsearle Exp $
;##
;## Test that the slice action sets time-series meta-data correctly
;##
;## Case 4: Middle elements of timeseries, with decimation

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err
```

```
;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "SLICEMD0104"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameters
set phi 1.0
set freq [ expr 1.0/16.0 ]

# slice parameters
set start 8191
set length 8
set stride 4

set PI 3.14159265358979

# derived values
set srate_exp [ expr $srate/$stride ]
set freq_exp [ expr $freq*($srate/2.0) ]
set phi_exp [ expr fmod($phi + $start*$PI*$freq, 2*$PI) ]

set offset [ expr $start/$srate ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_start_s_exp [ expr $stime + $offset_s ]
set gps_start_ns_exp [ expr 0 + $offset_ns ]

set offset [ expr ($length - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "slice(mix($phi, $freq, $channel), $start, $length, $stride)"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        m0 = mix($phi, $freq, gw);

        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        # Expected values should be checked against final result
        slice(m0, $start, $length, $stride);
      }
```

```
      -datacondtarget datacond
  }
"
sendCmd $cmd $host $port

exit
```

17. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

18. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/SLICEMD0104_out.ilwd`

Pass

19. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
`SampleRate`	4096	Yes
`BaseFrequency`	512	Yes
`Phase`	0.804	Yes
`GPSStartS`	680938500	Yes
`GPSStartNS`	499938964	Yes
`GPSEndS`	680938500	Yes
`GPSEndNS`	501647948	Yes

Pass

20. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.14 RESAMPLEMD01

Test Case: RESAMPLEMD01

Purpose: To test that time-series meta-data is set correctly by the re-sample action

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 12:11

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/RESAMPLEMD0101.tclsh`
- `datacondAPI/command/RESAMPLEMD0102.tclsh`
- `datacondAPI/command/RESAMPLEMD0103.tclsh`
- `datacondAPI/command/RESAMPLEMD0104.tclsh`
- `datacondAPI/command/RESAMPLEMD0105.tclsh`
- Frame files `H-680938500.F` to `H-680939699.F` (locked E5 data)

PROCEDURE

1. Execute the test script `datacondAPI/command/RESAMPLEMD0101.tclsh`. Confirm that the `managerAPI` accepted the job.
   Job ID: 5345 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: RESAMPLEMD0101.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the resample action sets time-series meta-data correctly
;##
;## Case 1: Downsample un-mixed time-series

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc max { a b } {
  if { $a >= $b } { return $a }
  return $b
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
```

```
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "RESAMPLEMD0101"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameters
set phi  0.0
set freq 0.0

# resample parameters
set p 5
set q 16
set n 10

# slice parameters
set start 0
set length 2
set stride 1

set PI 3.14159265358979

# derived values
set srate_exp [ expr $srate*$p/$q ]
set freq_exp [ expr $freq*($srate/2.0) ]
set delay [ expr double($n*[ max $p $q ]/$q) ]
set toffset [ expr $delay*$q/$p/$srate ]
set phi_exp [ expr fmod($phi - 2.0*$PI*$freq_exp*$toffset, 2*$PI) ]
if { $phi_exp < 0 } {
    set phi_exp [ expr $phi_exp + 2.0*$PI ]
}

# Since any realistic delay is going to be much smaller than 1 second,
# I'm doing a simple calculation of the new start time.
set gps_start_s_exp [ expr $stime - 1 ]
set gps_start_ns_exp [ expr int(pow(10,9) - $toffset*pow(10, 9)) ]

set offset [ expr ($length - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "slice(resample($channel, $p, $q), $start, $length, $stride)"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
```

```
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        r0 = resample(gw, $p, $q, $n);

        # Expected values should be checked against final result
        slice(r0, $start, $length, $stride);
      }
      -datacondtarget datacond
  }
"

sendCmd $cmd $host $port

exit
```

2. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

3. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/RESAMPLEMD0101_out.ilwd`

Pass

4. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
SampleRate	5120	Yes
BaseFrequency	0	Yes
Phase	0	Yes
GPSStartS	680938499	Yes
GPSStartNS	998046875	Yes
GPSEndS	680938499	Yes
GPSEndNS	99824189	Yes

Pass

5. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.

Pass

6. Execute the test script `datacondAPI/command/RESAMPLEMD0102.tclsh`. Confirm that the `managerAPI` accepted the job.

Job ID: 5356 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: RESAMPLEMD0102.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the resample action sets time-series meta-data correctly
;##
;## Case 2: Downsample time-series mixed with positive frequency

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
```

```
}

proc max { a b } {
  if { $a >= $b } { return $a }
  return $b
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "RESAMPLEMD0102"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameters
set phi 1.0
set freq [ expr 1.0/32.0 ]

# resample parameters
set p 5
set q 16
set n 10

# slice parameters
set start 0
set length 2
set stride 1

set PI 3.14159265358979

# derived values
set srate_exp [ expr $srate*$p/$q ]
set freq_exp [ expr $freq*($srate/2.0) ]
set delay [ expr double($n*[ max $p $q ]/$q) ]
set toffset [ expr $delay*$q/$p/$srate ]
set phi_exp [ expr fmod($phi - 2.0*$PI*$freq_exp*$toffset, 2*$PI) ]
if { $phi_exp < 0 } {
    set phi_exp [ expr $phi_exp + 2.0*$PI ]
}

# Since any realistic delay is going to be much smaller than 1 second,
# I'm doing a simple calculation of the new start time.
set gps_start_s_exp [ expr $stime - 1 ]
set gps_start_ns_exp [ expr int(pow(10,9) - $toffset*pow(10, 9)) ]

set offset [ expr ($length - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "slice(resample(mix($phi, $freq, $channel), $p, $q), $start, $length, $stride)"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
```

```
      -algorithms {
        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        # Mix so that we get non-default meta-data
        m0 = mix($phi, $freq, gw);
        r0 = resample(m0, $p, $q, $n);

        # Expected values should be checked against final result
        slice(r0, $start, $length, $stride);
      }
      -datacondtarget datacond
  }
"

sendCmd $cmd $host $port

exit
```

7. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

8. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/RESAMPLEMD0102_out.ilwd`

Pass

9. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
SampleRate	512	Yes
BaseFrequency	256	Yes
Phase	4.14	Yes
GPSStartS	680938499	Yes
GPSStartNS	998046875	Yes
GPSEndS	680938499	Yes
GPSEndNS	998242182	Yes

Pass

10. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.

Pass

11. Execute the test script `datacondAPI/command/RESAMPLEMD0103.tclsh`. Confirm that the `managerAPI` accepted the job.
Job ID: 5358 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: RESAMPLEMD0103.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the resample action sets time-series meta-data correctly
;##
;## Case 3: Downsample time-series mixed with negative frequency

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc max { a b } {
  if { $a >= $b } { return $a }
  return $b
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "RESAMPLEMD0103"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameters
set phi 1.0
set freq [ expr -1.0/32.0 ]

# resample parameters
set p 5
set q 16
set n 10

# slice parameters
set start 0
set length 2
set stride 1

set PI 3.14159265358979

# derived values
set srate_exp [ expr $srate*$p/$q ]
set freq_exp [ expr $freq*($srate/2.0) ]
set delay [ expr double($n*[ max $p $q ]/$q) ]
set toffset [ expr $delay*$q/$p/$srate ]
set phi_exp [ expr fmod($phi - 2.0*$PI*$freq_exp*$toffset, 2*$PI) ]
if { $phi_exp < 0 } {
    set phi_exp [ expr $phi_exp + 2.0*$PI ]
}

# Since any realistic delay is going to be much smaller than 1 second,
# I'm doing a simple calculation of the new start time.
set gps_start_s_exp [ expr $stime - 1 ]
set gps_start_ns_exp [ expr int(pow(10,9) - $toffset*pow(10, 9)) ]

set offset [ expr ($length - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
```

```
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "slice(resample(mix($phi, $freq, $channel), $p, $q), $start, $length, $stride)"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        # Mix so that we get non-default meta-data
        m0 = mix($phi, $freq, gw);
        r0 = resample(m0, $p, $q, $n);

        # Expected values should be checked against final result
        slice(r0, $start, $length, $stride);
      }
      -datacondtarget datacond
  }
"

sendCmd $cmd $host $port

exit
```

12. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

13. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/RESAMPLEMD0103_out.ilwd`

Pass

14. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
SampleRate	5120	Yes
BaseFrequency	-256	Yes
Phase	4.14	Yes
GPSStartS	680938499	Yes
GPSStartNS	998046975	Yes
GPSEndS	680938499	Yes
GPSEndNS	998242187	Yes

Pass

15. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.
Pass

16. Execute the test script `datacondAPI/command/RESAMPLEMD0104.tclsh`. Confirm that the `managerAPI` accepted the job.
Job ID: 5359 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: RESAMPLEMD0104.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the slice action sets time-series meta-data correctly
;##
;## Case 4: Upsample times-series mixed with positive frequency

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc max { a b } {
  if { $a >= $b } { return $a }
  return $b
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "RESAMPLEMD0104"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameters
set phi 1.0
set freq [ expr 1.0/32.0 ]

# resample parameters
set p 5
set q 2
set n 10
```

```
# slice parameters
set start 0
set length 2
set stride 1

set PI 3.14159265358979

# derived values
set srate_exp [ expr $srate*$p/$q ]
set freq_exp [ expr $freq*($srate/2.0) ]
set delay [ expr double($n*[ max $p $q ]/$q) ]
set toffset [ expr $delay*$q/$p/$srate ]
set phi_exp [ expr fmod($phi - 2.0*$PI*$freq_exp*$toffset, 2*$PI) ]
if { $phi_exp < 0 } {
    set phi_exp [ expr $phi_exp + 2.0*$PI ]
}

# Since any realistic delay is going to be much smaller than 1 second,
# I'm doing a simple calculation of the new start time.
set gps_start_s_exp [ expr $stime - 1 ]
set gps_start_ns_exp [ expr int(pow(10,9) - $toffset*pow(10, 9)) ]

set offset [ expr ($length - 1)/$srate_exp ]
set offset_s [ expr int($offset) ]
set offset_ns [ expr int(($offset - $offset_s)*pow(10,9)) ]
set gps_end_s_exp [ expr $gps_start_s_exp + $offset_s ]
set gps_end_ns_exp [ expr $gps_start_ns_exp + $offset_ns ]

set namemd_exp "slice(resample(mix($phi, $freq, $channel), $p, $q), $start, $length, $stride)"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        value($srate_exp);
        intermediate(,,srate_exp, {Expected value of SampleRate});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($phi_exp);
        intermediate(,,phi_exp, {Expected value of Phase});

        value($gps_start_s_exp);
        intermediate(,,gps_start_s_exp, {Expected value of GPSStartS});

        value($gps_start_ns_exp);
        intermediate(,,gps_start_ns_exp, {Expected value of GPSStartNS});

        value($gps_end_s_exp);
        intermediate(,,gps_end_s_exp, {Expected value of GPSEndS});

        value($gps_end_ns_exp);
        intermediate(,,gps_end_ns_exp, {Expected value of GPSEndNS});

        # Mix so that we get non-default meta-data
        m0 = mix($phi, $freq, gw);
        r0 = resample(m0, $p, $q, $n);

        # Expected values should be checked against final result
        slice(r0, $start, $length, $stride);
      }
      -datacondtarget datacond
  }
"

sendCmd $cmd $host $port

exit
```

17. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

18. Verify that an output file was created at
`http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/RESAMPLEMD0104_out.ilwd`

Pass

19. Open the output file and verify that it contains a list of expected values for the meta-data of the time-series and that the time-series and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
`SampleRate`	40960	Yes
`BaseFrequency`	256	Yes
`Phase`	0.018	Yes
`GPSStartS`	680938499	Yes
`GPSStartNS`	999389648	Yes
`GPSEndS`	680938499	Yes
`GPSEndNS`	999414062	Yes

Pass

20. Verify that the `NameMetaData` field of the final result matches the comment field of the final result.
Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.15 PSDMD01

Test Case: PSDMD01

Purpose: To test that PSD spectrum meta-data is set correctly by the psd action

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 14:10

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/PSDMD0101.tclsh`
- `datacondAPI/command/PSDMD0102.tclsh`
- Frame files `H-680938500.F` to `H-680939699.F` (locked E5 data)

PROCEDURE

1. Execute the test script `datacondAPI/command/PSDMD0101.tclsh`. Confirm that the `managerAPI` accepted the job.
   Job ID: 5361 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: PSDMD0101.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the psd action sets PSD spectrum meta-data correctly
;##
;## Case 1: PSD of an un-mixed time-series

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host  ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "PSDMD0101"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameter
```

```
set phi 0.0
set freq 0.0

# psd parameters
set fftlen 128
set overlap 64
set detrend 1
set winparam 0.0

# derived values
set freq_exp [ expr $freq*($srate/2.0) ]
set freq_base_exp [ expr -$srate/2.0 - $freq_exp ]
set freq_delta_exp [ expr $srate/$fftlen ]
set detrend_exp $detrend
set fftlen_exp $fftlen
set overlap_exp $overlap
set winparam_exp $winparam
set namemd_exp "psd($channel)"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        value($freq_base_exp);
        intermediate(,,freq_base_exp, {Expected value of FrequencyBase});

        value($freq_delta_exp);
        intermediate(,,freq_delta_exp, {Expected value of FrequencyDelta});

        value($detrend_exp);
        intermediate(,,detrend_exp, {Expected value of DetrendMethod});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($fftlen_exp);
        intermediate(,,fftlen_exp, {Expected value of FFTLength});

        value($overlap_exp);
        intermediate(,,overlap_exp, {Expected value of FFTOverlap});

        value($winparam_exp);
        intermediate(,,winparam_exp, {Expected value of WindowParameter});

        p0 = psd(gw, $fftlen, _, $overlap, $detrend);
      }
      -datacondtarget datacond
  }
"

sendCmd $cmd $host $port

exit
```

2. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

3. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/PSDMD0101_out.ilwd`

Pass

4. Open the output file and verify that it contains a list of expected values for the meta-data of the PSD spectrum and that the PSD spectrum and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
FrequencyBase	-8192	Yes
FrequencyDelta	128	Yes
DetrendMethod	1	Yes
BaseFrequency	0	Yes
FFTLength	128	Yes
FFTOverlap	64	Yes
WindowParameter	0	Yes

Pass

5. Verify that the NameMetaData field of the final result matches the comment field of the final result.
Pass

6. Execute the test script datacondAPI/command/PSDMD0102.tclsh. Confirm that the managerAPI accepted the job.
Job ID: 5363 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: PSDMD0102.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the psd action sets PSD spectrum meta-data correctly
;##
;## Case 2: PSD of a mixed time-series

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "PSDMD0102"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 1
set etime [ expr $stime + $duration - 1 ]
set srate 16384.0

# mix parameters
set phi 1.0
set freq [ expr 1.0/32.0 ]

# psd parameters
set fftlen 256
set overlap 128
set detrend 2
set winparam 0.1

# derived values
set freq_exp [ expr $freq*($srate/2.0) ]
set freq_base_exp [ expr -$srate/2.0 - $freq_exp ]
set freq_delta_exp [ expr $srate/$fftlen ]
```

```
set detrend_exp $detrend
set fftlen_exp $fftlen
set overlap_exp $overlap
set winparam_exp $winparam
set namemd_exp "psd(mix($phi, $freq, $channel))"

set times ${stime}-${etime}
set ofile "${test_name}_out.ilwd"
set subject "$test_name"
set resultname "$test_name result"
set resultcomment $namemd_exp

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -resultname { $resultname }
      -resultcomment { $resultcomment }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        value($freq_base_exp);
        intermediate(,,freq_base_exp, {Expected value of FrequencyBase});

        value($freq_delta_exp);
        intermediate(,,freq_delta_exp, {Expected value of FrequencyDelta});

        value($detrend_exp);
        intermediate(,,detrend_exp, {Expected value of DetrendMethod});

        value($freq_exp);
        intermediate(,,freq_exp, {Expected value of BaseFrequency});

        value($fftlen_exp);
        intermediate(,,fftlen_exp, {Expected value of FFTLength});

        value($overlap_exp);
        intermediate(,,overlap_exp, {Expected value of FFTOverlap});

        value($winparam_exp);
        intermediate(,,winparam_exp, {Expected value of WindowParameter});

        # Mix so that we get non-default meta-data
        m0 = mix($phi, $freq, gw);
        w0 = KaiserWindow(0.1);
        p0 = psd(m0, $fftlen, w0, $overlap, $detrend);
      }
      -datacondtarget datacond
  }
"

sendCmd $cmd $host $port

exit
```

7. Verify that an email was received indicating where the results are to be found and that no errors were reported.

   Pass

8. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/PSDMD0102_out.ilwd`

   Pass

9. Open the output file and verify that it contains a list of expected values for the meta-data of the PSD spectrum and that the PSD spectrum and its meta-data are contained in the final result. Verify that the expected values of the meta-data match the actual values contained in the final result:

Meta-data	Value	Matches expected?
FrequencyBase	-8448	Yes
FrequencyDelta	64	Yes
DetrendMethod	2	Yes
BaseFrequency	256	Yes
FFTLength	256	Yes
FFTOverlap	128	Yes
WindowParameter	0.1	Yes

Pass

10. Verify that the NameMetaData field of the final result matches the comment field of the final result.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.16 PSDDB01

Test Case: PSDDB01

Purpose: Verify database insertion and retrieval of a PSD

Tester: Antony C. Searle

Test machine: ldas.mit.edu

Date (mm/dd/yy): 09/07/01 **Time:** 14:20

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/PSDDB0101.tclsh`
- `datacondAPI/command/PSDDB0102.tclsh`
- `datacondAPI/command/PSDDB0103.tclsh`
- Frame files `H-680938500.F` to `H-680939699.F` (locked E5 data)

PROCEDURE

1. Execute the test script `datacondAPI/command/PSDDB0101.tclsh`. Confirm that the `managerAPI` accepted the job.
   Job ID: 5346 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: PSDDB0101.tclsh,v 1.3 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that a PSD can be inserted into the LDAS database
;##

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "PSDDB0101"
set channel "H2:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 128
set etime [ expr $stime + $duration - 1 ]

set times ${stime}-${etime}
```

```
set ofile "${test_name}_out.ilwd"
set subject "$test_name"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -interferometers { $interferometers }
      -times { $times }
      -framequery Adc($channel)
      -aliases { gw = _ch0; }
      -algorithms {
        gs = slice(gw, 0, 131072, 16);
        p = psd(gs, 16384, _, 8192);
        intermediate(ilwd, $ofile, psd, {Output of $test_name} );
      }
      -datacondtarget metadata
  }"

sendCmd $cmd $host $port

exit
```

2. Verify that an email was received stating that 1 row was inserted into the `summ_spectrum` table and no errors are reported. Note that the job may take several minutes to complete.

   Pass

3. Verify that an output file was created at
   `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/PSDDB0101_out.ilwd`

   Pass

4. Open the output file and verify that the following are correct:

Meta-data	Expected value	Matches?
`dims`	16384	Yes
`FrequencyBase`	−512.0	Yes
`FrequencyDelta`	0.0625	Yes
`DetrendMethod`	0	Yes
`Estimator`	Welch CSD Estimate	Yes
`BaseFrequency`	0.0	Yes
`FFTLength`	16384	Yes
`OverlapLength`	8192	Yes
`WindowName`	Hann Window	Yes
`WindowParameter`	0.0	Yes
`NameMetaData`	`psd(slice(H2:LSC-AS_Q, 0, 131072, 16))`	

   Note the first two values of the data 1. 7.21e-3 2. 5.57e-3
   and the last two values of the data 1. 3.33e-3 2. 5.57e-3

   Pass

5. Execute the test script `datacondAPI/command/PSDDB0102.tclsh`, entering the job ID from the first test when requested. Confirm that the `managerAPI` accepted the job.
   Job ID: 5366 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: PSDDB0102.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the result of a PSD action can be inserted into the LDAS
;## database
;##
```

```
proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "PSDDB0102"

set ofile "${test_name}_out.ilwd"
set subject "$test_name"

puts "$test_name: Retrieve PSD based on job ID"
puts -nonewline stdout "Enter job ID (eg. 12345): "
flush stdout
gets stdin jobid

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -dbquery { { select * from summ_spectrum where process_id in (select process_id from process where jobid=$jobid) } push p }
      -algorithms {
        value(p);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

6. Verify that an email was received indicating where the results are to be found and that no errors were reported.

   Pass

7. Verify that an output file was created at `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/PSDDB0102_out.ilwd`

   Pass

8. Open the output file and verify that the following are correct:

Meta-data	Expected value	Matches?
`dims`	16384	Yes
`FrequencyBase`	-512.0	Yes
`FrequencyDelta`	0.0625	Yes

   NOTE: not all PSD meta-data is stored in the database. Some fields in the ILWD file may be left blank or have default values. **Pass**

9. Verify that the first two and last two values of the data are the same as those recorded above.

   Pass

10. Execute the test script `datacondAPI/command/PSDDB0103.tclsh`, entering the job ID from the first test when requested. Confirm that the `managerAPI` accepted the job.
Job ID: 3367 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: PSDDB0103.tclsh,v 1.3 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the result of a PSD action can be retrieved from the LDAS
;## database
;##

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "PSDDB0103"

set ofile "${test_name}_out.ilwd"
set subject "$test_name"

puts "$test_name: Retrieve PSD based on job ID"
puts -nonewline stdout "Enter job ID (eg. 12345): "
flush stdout
gets stdin jobid

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    getMetaData
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -returnformat { ilwd ascii }
      -sqlquery { select * from summ_spectrum where process_id in (select process_id from process where jobid=$jobid) }
  }"

sendCmd $cmd $host $port

exit
```

11. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

12. Verify that an output file was created at
`http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/PSDDB0103_out.ilwd`
Pass

13. Open the output file and verify that the following are correct:

Meta-data	Expected value	Matches?
START_TIME	680938500	Yes
START_TIME_NS	0	Yes
END_TIME	680938627	Yes
END_TIME_NS	999023437	Yes
START_FREQUENCY	−512.0	Yes
DELTA_FREQUENCY	0.0625	Yes
CHANNEL	slice(H2:LSC-AS_Q, 0, 131072, 16)	Yes
SPECTRUM_TYPE	Welch	Yes
SPECTRUM_LENGTH	16384	Yes

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.3.17 CSDDB01

Test Case: CSDDB01

Purpose: Verify database insertion and retrieval of a CSD

Tester: Antony C. Searlel-

Test machine:

Date (mm/dd/yy): **Time:**

das.mit.edu09/07/0114:30

ENVIRONMENT AND PREREQUISITES

Input files:

- `datacondAPI/command/CSDDB0101.tclsh`
- `datacondAPI/command/CSDDB0102.tclsh`
- `datacondAPI/command/CSDDB0103.tclsh`
- Frame files `H-680938500.F` to `H-680939699.F` (locked LHO E5 data)
- Frame files `L-680938500.F` to `L-680939699.F` (locked LLO E5 data)

PROCEDURE

1. Execute the test script `datacondAPI/command/CSDDB0101.tclsh`. Confirm that the `managerAPI` accepted the job.
   Job ID: 5370 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: CSDDB0101.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that a CSD can be retrieved from the LDAS database
;##

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}
```

```
# job specific values
set test_name "CSDDB0101"
set channel0 "H2:LSC-AS_Q"
set channel1 "L1:LSC-AS_Q"
set interferometers H
set stime 680938500
set duration 128
set etime [ expr $stime + $duration - 1 ]

# derived values
set times ${stime}-${etime}
set ch0alias [ mkchannel $channel0 $stime ]
set ch1alias [ mkchannel $channel1 $stime ]

set ofile "${test_name}_out.ilwd"
set subject "$test_name"

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -inputformat { frame }
      -subject { $subject }
      -framequery {
        { { H } {} $times Adc($channel0) }
        { { L } {} $times Adc($channel1) }
      }
      -aliases { gw_lho = $ch0alias; gw_llo = $ch1alias; }
      -algorithms {
        gw_lho_s = slice(gw_lho, 0, 131072, 16);
        gw_llo_s = slice(gw_llo, 0, 131072, 16);
        p = csd(gw_lho_s, gw_llo_s, 16384, _, 8192);
        intermediate(ilwd, $ofile, csd, {Output of $test_name} );
      }
      -datacondtarget metadata
  }"

sendCmd $cmd $host $port

exit
```

2. Verify that an email was received stating that 1 row was inserted into the `summ_csd` table and no errors are reported. Note that the job may take several minutes to complete.

Pass

3. Verify that an output file was created at
`http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/CSDDB0101_out.ilwd`

Pass

4. Open the output file and verify that the following are correct:

Meta-data	Expected value	Matches?
`dims`	16384	Yes
`FrequencyBase`	-512.0	Yes
`FrequencyDelta`	0.0625	Yes
`DetrendMethod`	0	Yes
`Estimator`	Welch CSD Estimate	Yes
`BaseFrequency`	0.0	Yes
`FFTLength`	16384	Yes
`OverlapLength`	8192	Yes
`WindowName`	HannWindow	Yes
`WindowParameter`	0.0	Yes

Note the first two values of the data 1. -3.043e-4 2. 0.0
and the last two values of the data 1. 1.974e-3 2. -6.585e-4

Pass

5. Verify that the `NameMetaData` is
   `csd(slice(H2:LSC-AS_Q, 0, 131072, 16),slice(L1:LSC-AS_Q, 0, 131072, 16))`
   Pass

6. Execute the test script `datacondAPI/command/CSDDB0102.tclsh`, entering the job ID from the first test when requested. Confirm that the `managerAPI` accepted the job.
   Job ID: 5372 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: CSDDB0102.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the result of a CSD action can be inserted into the LDAS
;## database
;##

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "CSDDB0102"

set ofile "${test_name}_out.ilwd"
set subject "$test_name"

puts "$test_name: Retrieve CSD based on job ID"
puts -nonewline stdout "Enter job ID (eg. 12345): "
flush stdout
gets stdin jobid

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -outputformat { ilwd ascii }
      -dbquery { { select * from summ_csd where process_id in (select process_id from process where jobid=$jobid) } push p }
      -algorithms {
        value(p);
      }
      -datacondtarget datacond
  }"

sendCmd $cmd $host $port

exit
```

7. Verify that an email was received indicating where the results are to be found and that no errors were reported.
   Pass

8. Verify that an output file was created at
   `http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/CSDDB0102_out.ilwd`
   Pass

9. Open the output file and verify that the following are correct:

Meta-data	Expected value	Matches?
`dims`	16384	Yes
`FrequencyBase`	-512.0	Yes
`FrequencyDelta`	0.0625	Yes

NOTE: not all CSD meta-data is stored in the database. Some fields in the ILWD file may be left blank or have default values. **Pass**

10. Verify that the first two and last two values of the data are the same as those recorded above.

Fail

11. Execute the test script `datacondAPI/command/CSDDB0103.tclsh`, entering the job ID from the first test when requested. Confirm that the `managerAPI` accepted the job.
Job ID: 5375 **Pass**

```
#!/ldcg/bin/tclsh

;## $Id: CSDDB0103.tclsh,v 1.2 2001/09/04 22:04:57 acsearle Exp $
;##
;## Test that the result of a CSD action can be retrieved from the LDAS
;## database
;##

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

;## Default settings
set user ldas_mdc
set pwrd beowulf
set email mdc@gravity.phys.psu.edu
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "~/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

# job specific values
set test_name "CSDDB0103"

set ofile "${test_name}_out.ilwd"
set subject "$test_name"

puts "$test_name: Retrieve CSD based on job ID"
puts -nonewline stdout "Enter job ID (eg. 12345): "
flush stdout
gets stdin jobid

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    getMetaData
      -subject { $subject }
      -returnprotocol { file:/$ofile }
      -returnformat { ilwd ascii }
      -sqlquery { select * from summ_csd where process_id in (select process_id from process where jobid=$jobid) }
  }"

sendCmd $cmd $host $port

exit
```

12. Verify that an email was received indicating where the results are to be found and that no errors were reported.

Pass

13. Verify that an output file was created at
`http://<testmachine>/ldas_outgoing/jobs/NORMAL<jobid>/CSDDB0103_out.ilwd`

Pass

14. Open the output file and verify that the following are correct:

Meta-data	**Expected value**	**Matches?**
START_TIME	680938500	Yes
START_TIME_NS	0	Yes
END_TIME	680938627	Yes
END_TIME_NS	999023437	Yes
START_FREQUENCY	−512.0	Yes
DELTA_FREQUENCY	0.0625	Yes
CHANNEL1	slice(H2:LSC-AS_Q, 0, 131072, 16)	Yes
CHANNEL2	slice(L1:LSC-AS_Q, 0, 131072, 16)	Yes
SPECTRUM_TYPE	Welch	Yes
SPECTRUM_LENGTH	16384	Yes

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs: data_conditioning_api/1065

Notes:

TEST RESULT **Fail**

A.4 External trigger tests

External Trigger related tests, verifying pipelines and database access for TCP/IP socket(ALLEGRO) and e-mail(GCN) communication models.

Sec. A.4.1 *ALGNCEXTDBINS* - Verifies that the XML output of the ALLEGRO and GCN event handlers conforms to the external_triggers table definition and verifies that these trivial entries can be inserted into and retrieved from the External Triggers table of the LDAS Metadata/Event DB structure without loss or degradation. We utilize TCP/IP socket communication when obtaining IGEC standard events from ALLEGRO and we (presently) use automatically parsed e-mails to receive warnings from GCN. Both code packages should produce fully LDAS Metadata/Event DB compliant and parsable XML output. This test is designed to verify that the XML output data files of the ALLEGRO and GCN event handlers conform to the external_triggers table definition. This test also verifies that these trivial entries can be inserted into and retrieved from the External Triggers table of the LDAS Metadata/Event DB structure without loss or degradation. The complete success of this test is a prerequisite for every test during this MDC that are related to external triggers.

Sec. A.4.2 *ALFULLPIPE100* - Validates the full pipeline from reading the ALLEGRO event data table to inserting events into the External Triggers table. This test uses real ALLEGRO data from last year's (2000) run (with periodic marker/calibrator events inserted). The number of events transmitted during this test should be ˜100, which is roughly equivalent to 1 day of events transmitted to IGEC members. The data transmitted by the client is received by the server side parser which pads the static fields with ALLEGRO specific information and prepares the XML file for the External Triggers table of the LDAS Metadata/Event DB structure. After this the events are automatically ingested.

Sec. A.4.3 *ALFULLPIPE1000* - Verifies that the code can handle 10 days of ALLEGRO events and transmit them at once with no failures. This is a fairly convincing model of updating the database table after ten days of network outage. The test uses simulated data. The number of events transmitted during this test is 1000, which is equivalent with 10 day of events transmitted to IGEC members. This test is very similar to ALFULLPIPE100.

Sec. A.4.4 *ALFULLPIPE10K* - Verifies that the code can handle ˜O(10000) ALLEGRO events and transmit them at once with no failures. This is a fairly convincing model of refilling the database table with 3 months of events after it was wiped. The test uses simulated data. The number of events should be transmitted during this test is 10000, which is equivalent to 3 months of events transmitted to IGEC members. This test is very similar to ALFULLPIPE100(0).

Sec. A.4.5 *ALFULLPIPE100K* - Extreme stress test of the full system. It attempts to push ˜3 years (˜O(100K)) of ALLEGRO events through the pipeline and instruct LDAS to ingest them at once. Since the extreme nature of this test, it is expected that some part of the system will stop functioning due to built in limitations. The purpose of the test is to identify the system components fully functioning under this extreme load, the system components refusing to take the load, the reason for particular bottlenecks, quantify the limitations and prepare a recommendation on how to make the main code aware of these limits. Therefore this test is unique in the sense that a failure of any step will not necessarily lead to the ultimate failure of the full test. The test should only be considered a failure if either no particular limitations are found when the code breaks due to buggy code or the proper way to overcome identified limitations must result in a major revision of the base code. Any error messages and their resolution *must be noted* at the bottom of this test sheet, including the recommendation to resolve each particular limitation.

Sec. A.4.6 *ALFULLPIPECONT* - Validates the standard operating procedure for the TCP/IP channel. Uses real ALLEGRO data from last year's (2000) run. The number of events transmitted during this test should be ˜80-100, which is roughly equivalent to 1 day of events transmitted to IGEC members. This is a long test, it takes nearly a full day to complete. During the test the ALLEGRO data file is filled at a slow rate, one event at the time at uneven time intervals. This test fully simulates the real situation encountered during normal operation interrupted with very short intervals of transmission line (e.g. network) problems.

Sec. A.4.7 *GCNFULLPIPE100* - Verifies the full parsing and event insertion sequence into the External Triggers table of the LDAS Metadata/Event DB structure at MIT:mit_test. Various real GRB alerts are used for this test. We received these alerts from the GCN network during the course of the past year. The alerts were automatically pre formatted by the automatic e-mail parser, which created a backup copy, sliced off the headers and and computed the GPS start time based on the UTC time supplied with the alert. The test uses a simple unix script to insert all alerts into the database table one by one. This effectively simulates the highest rate of alerts arriving to be ingested.

NOTE: Since the test procedures for this section are somewhat different from other sections, some guidelines are provided here.

- Tests involving ALLEGRO data transmission require a remote client with the appropriate software installed (allegroClient)
- Each test requires input data and produces various output files. All of these are recorded in the CVS in appropriate sub-directories named after individual tests.
  - *input/<TESTNAME>* contains all input files associated with the test, which can include data streams, XML files and raw GCN notices.
  - *output/<TESTNAME>* contains the output of the test, which can include various log files, XML output and XML extract of events from database tables.
  - *command/<TESTNAME>* contains the compiled special code necessary for the individual tests. Symbolic links can be used in case of commands used for for multiple tests.
  - *doc* contains the the test documentation; note that there is a separate TEX file for each test.
  - *expected* is empty but created for compatibility
  - *src/<TESTNAME>* contains special code and compile/run instructions necessary for the individual tests. Symbolic links can be used in case of commands used for for multiple tests.
- The individual test documentation pages should contain all information required to run the tests.
- Each step of the test must be marked as **Pass**, **N/A** or **Fail**.
- A command example is given at several steps of the tests.
- The program, which provides the near-real time event list at ALLEGRO, were written by Ik Siong Heng of LSU. The communications and DB related software was written by Hareem Tariq of FIT and Szabi Márka.

A.4.1 ALGNCEXTDBINS

Test Case: ALGNCEXTDBINS

Purpose: Verifies that the XML output of the ALLEGRO and GCN event handlers conforms to the external_triggers table definition and verifies that these trivial entries can be inserted into and retrieved from the External Triggers table of the LDAS Metadata/Event DB structure without loss or degradation.

Tester: Szabolcs Marka

Test machine: LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/07/01 **Time:** 18:00:00

ENVIRONMENT AND PREREQUISITES

- LDAS putMeta command works on local machine
- External Trigger database table MIT:mit_test is accessible
- GUILD running on local machine
- Pre-prepared XML files from ALLEGRO and GCN parser for DB input

PROCEDURE

Verifies that the XML output of the ALLEGRO and GCN event handlers conforms to the external_triggers table definition and verifies that these trivial entries can be inserted into and retrieved from the External Triggers table of the LDAS Metadata/Event DB structure without loss or degradation.We utilize TCP/IP socket communication when obtaining IGEC standard events from ALLEGRO and we (presently) use automatically parsed e-mails to receive warnings from GCN. Both code packages should produce fully LDAS Metadata/Event DB compliant and parsable XML output. This test is designed to verify that the XML output data files of the ALLEGRO and GCN event handlers conform to the external_triggers table definition. This test also verifies that these trivial entries can be inserted into and retrieved from the External Triggers table of the LDAS Metadata/Event DB structure without loss or degradation. The complete success of this test is a prerequisite for every test during this MDC that are related to external triggers. The test procedure is very simple. One must use pre prepared XML output files made by the each of the communication packages (ALLEGRO/GCN) and insert it into the mit_test database using the putMeta LDAS command. Every command must be executed on local (MIT) machines and the packages, which produce the XML files, also must be compiled and run locally. All command examples assume that pwd is /home/<user>/<ldasmdc dir>/burst-stochastic/test/ExtTrigs/ .

1. LDAS putMeta command is available on local machine
   which putMeta

   Pass

2. GUILD is available on local machine
   guild &

   Pass

3. MIT:mit_test) External Trigger database table is accessible locally
   GUILD will show it...
   Pass

4. ALLEGRO event was ingested successfully
   putMeta -u smarka -f input/ALGNCEXTDBINS_ALL.xml -d MIT:mit_test
   Pass

5. ALLEGRO event was retrieved without problems
   Check with GUILD, export result into output/ALGNCEXTDBINS/ALGNCEXTDBINS_ALL.xml
   Pass

6. ALLEGRO input and output data is identical
   Compare XML input and output
   Pass

7. GCN event was ingested successfully
   putMeta -u smarka -f input/ALGNCEXTDBINS/ALGNCEXTDBINS_GCN.xml -d MIT:mit_test
   Pass

8. GCN event was retrieved without problems
   Check with GUILD, export result into output/ALGNCEXTDBINS/ALGNCEXTDBINS_GCN.xml
   Pass

9. GCN input and output data is identical
   Compare xml input and output to each other and to input/ALGNCEXTDBINS/ALGNCEXTDBINS_GCN.notice
   Pass

10. Is the External Trigger table comprehensive?
    Validate whether all desired information has columns allocated or they can be associated with columns allocated for other purposes
    Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: None

Notes: Opinions are needed to evaluate/validate the design/definition of the external_triggers table and its content.

TEST RESULT **Pass**

A.4.2 ALFULLPIPE100

Test Case: ALFULLPIPE100

Purpose: Validates the full pipeline from reading the ALLEGRO event data table to inserting events into the External Triggers table.

Tester: Szabolcs Marka, Ik Siong Heng

Test machine: SAM.PHYS.LSU.EDU, LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/07/01 **Time:** 18:00:00

ENVIRONMENT AND PREREQUISITES

Required communication software must be installed on both the remote client (allegroClient) and local server (MIT_Server). putMeta command and GUILD of LDAS should be functional. The client must be compiled on remote machine. The server on the local computer must be running.

PROCEDURE

Validates the full pipeline from reading the ALLEGRO event data table to inserting events into the External Triggers table.
Utilize sam.phys.lsu.edu as the remote client and use real ALLEGRO data from last year's (2000) run (with periodic marker/calibrator events inserted). The number of events transmitted during this test should be 100, which is roughly equivalent to 1 day of events transmitted to IGEC members. The data transmitted by the client is received by the server side parser which pads the static fields with ALLEGRO specific information and prepares the XML file for the External Triggers table of the LDAS Metadata/Event DB structure. After this the events are automatically ingested.

1. Start up the sever on local machine and check server status
   (MIT_Server >& ALFULLPIPE100.log & ; ps **Pass**
2. Reset client bookmark
   echo AL000000000 > allegroLog.dat
3. Transmit the 100 events from ALFULLPIPE100.remote.dat to LANCELOT (18.120.0.66)
   allegroClient 18.120.0.66 ALFULLPIPE100.remote.dat allegroLog.dat > ALFULLPIPE100.remote.log
   Pass
4. Check client and server side logs for error messages and compare the number of transmitted events
   Pass
5. Visually check database for the newly filed events and accuracy with GUILD
   guild & and use MIT mit_test database
   Pass
6. Extract these recently filed events with GUILD.
   Save them as ExtTrigs/output/ALFULLPIPE100_DB.xml
   Pass
7. Using standard unix tools, compare extracted values to input values
   ExtTrigs/output/ALFULLPIPE100_DB.xml vs.ExtTrigs/output/ALFULLPIPE100_in.xml

8. Were the ALLEGRO events ingested successfully?

Pass

9. Was the input transmitted and output data identical?

Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: None

Notes:

TEST RESULT **Pass**

A.4.3 ALFULLPIPE1000

Test Case: ALFULLPIPE1000

Purpose: Verifies that the code can handle 10 days of ALLEGRO events and transmit them at once with no failures. This is a fairly convincing model of updating the database table after ten days of network outage.

Tester: Szabolcs Marka

Test machine: SAM.PHYS.LSU.EDU, LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/08/01 **Time:** 18:17:00

ENVIRONMENT AND PREREQUISITES

Required communication software must be installed on both the remote client (allegroClient) and local server (MIT_Server). putMeta command and GUILD of LDAS should be functional. The client must be compiled on remote machine. The server on the local computer must be running.

PROCEDURE

Verifies that the code can handle 10 days of ALLEGRO events and transmit them at once with no failures. This is a fairly convincing model of updating the database table after ten days of network outage. Utilize sam.phys.lsu.edu as the remote client and use simulated data. The number of events transmitted during this test were 1000, which is equivalent with 10 day of events transmitted to IGEC members. The data, transmitted by the client, is received by the server side parser which pads the static fields with ALLEGRO specific information and prepares the XML file for the External Triggers table of the LDAS Metadata/Event DB structure. After this the events are automatically ingested. This test is very similar to ALFULLPIPE100.

1. Start up the sever on local machine and check server status
   (MIT_Server >& ALFULLPIPE1000.log & ; ps **Pass**
2. Reset client bookmark
   echo AL000000000 > allegroLog.dat
3. Transmit the 100 events from ALFULLPIPE1000.remote.dat
   allegroClient 18.120.0.66 ALFULLPIPE1000.remote.dat allegroLog.dat > ALFULLPIPE1000.remote.log
   Pass
4. Check client and server side logs for error messages and compare the number of transmitted events
   Pass
5. Visually check database for the newly filed events and accuracy with GUILD
   guild & and use MIT mit_test database
   Pass
6. Extract these recently filed events with guild.
   Save them as ExtTrigs/output/ALFULLPIPE1000_DB.xml
   Pass
7. Using standard unix tools, compare randomly chosen extracted values to input values
   ExtTrigs/output/ALFULLPIPE1000_DB.xml vs.ExtTrigs/output/ALFULLPIPE1000_in.xml

8. Were the ALLEGRO events ingested successfully?

Pass

9. Was the input transmitted and output data identical where examined?

Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: None

Notes: The server log file missed some reassuring messages which appeared on the screen...some solaris feature at work. Anyway, the events were smoothly transmitted and ingested. There are stray events in the GUILD output file, since I was did not restrict the search enough to only extract the newly ingested events.

TEST RESULT **Pass**

A.4.4 ALFULLPIPE10K

Test Case: ALFULLPIPE10K

Purpose: Verifies that the code can handle ˜O(10000) ALLEGRO events and transmit them at once with no failures. This is a fairly convincing model of refilling the database table with 3 months of events after it was wiped.

Tester: Szabolcs Marka

Test machine: SAM.PHYS.LSU.EDU, LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/09/01 **Time:** 18:17:00

ENVIRONMENT AND PREREQUISITES

Required communication software must be installed on both the remote client (allegroClient) and local server (MIT_Server). putMeta command and GUILD of LDAS should be functional. The client must be compiled on remote machine. The server on the local computer must be running.

PROCEDURE

Verifies that the code can handle ˜O(10000) ALLEGRO events and transmit them at once with no failures. This is a fairly convincing model of refilling the database table with 3 months of events after it was wiped.
Utilize sam.phys.lsu.edu as the remote client and use simulated data. The number of events should be transmitted during this test is 10000, which is equivalent to 3 months of events transmitted to IGEC members. The data transmitted by the client is received by the server side parser which pads the static fields with ALLEGRO specific information and prepares the XML file for the External Triggers table of the LDAS Metadata/Event DB structure. After this the events are automatically ingested. This test is very similar to ALFULLPIPE100(0).

1. Start up the sever on local machine and check server status
   (MIT_Server >& ALFULLPIPE10K.log & ; ps **Pass**
2. Reset client bookmark
   echo AL000000000 > allegroLog.dat
3. Transmit the 10K events from ALFULLPIPE10K.remote.dat
   allegroClient 18.120.0.66 ALFULLPIPE10K.remote.dat allegroLog.dat > ALFULLPIPE10K.remote.log
   Pass
4. Check client and server side logs for error messages and compare the number of transmitted events
   Pass
5. Visually check database for the newly filed events and accuracy with GUILD
   guild & and use MIT mit_test database
   Pass
6. Extract these recently filed events with guild.
   Save them as ExtTrigs/output/ALFULLPIPE10K_DB.xml
   Pass

7. Using standard unix tools, compare randomly chosen extracted values to input values ExtTrigs/output/ALFULLPIPE10K_DB.xml vs.ExtTrigs/output/ALFULLPIPE10K_in.xml

8. Were the ALLEGRO events ingested successfully?

Pass

9. Was the input transmitted and output data identical where examined?

Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: None

Notes:

TEST RESULT **Pass**

A.4.5 ALFULLPIPE100K

Test Case: ALFULLPIPE100K

Purpose: Extreme stress test of the full system. It attempts to push 3 years (O(100K)) of ALLEGRO events through the pipeline and instruct LDAS to ingest them at once.

Tester: Szabolcs Marka

Test machine: SAM.PHYS.LSU.EDU, LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/09/01 **Time:** 19:17:00

ENVIRONMENT AND PREREQUISITES

Required communication software must be installed on both the remote client (allegroClient) and local server (MIT_Server). putMeta command and GUILD of LDAS should be functional. The client must be compiled on remote machine. The server on the local computer must be running.

PROCEDURE

Extreme stress test of the full system. It attempts to push ˜3 years (˜O(100K)) of ALLEGRO events through the pipeline and instruct LDAS to ingest them at once. Since the extreme nature of this test, it is expected that some part of the system will stop functioning due to built in limitations. The purpose of the test is fourfold:

1. Identify the system components fully functioning under this extreme load
2. Identify the system components refusing to take the load (if any)
3. Identify the reason for particular bottlenecks (if any)
4. Quantify the limitations (if any) and prepare a recommendation on how to make the main code aware of these limits.

Therefore this test is unique in the sense that a failure of any step will not necessarily lead to the ultimate failure of the test. The test should only be considered a failure if either no limitations are found when the code breaks due to buggy code or the proper way to overcome identified limitations must result in a major revision of the base code. Any error messages and their resolution *must be noted* at the bottom of this test sheet, including the recommendation to resolve each particular limitation.

1. Start up the sever on local machine and check server status
   MIT_Server >& ALFULLPIPE100K.log & ; ps
2. Reset client bookmark
   echo AL000000000 > allegroLog.dat
3. Transmit the 100,000 events from ALFULLPIPE100K.remote.dat
   allegroClient 18.120.0.66 ALFULLPIPE100K.remote.dat allegroLog.dat > ALFULLPIPE100K.remote.log
4. Check client and server side logs for error messages and compare the number of transmitted events

5. Visually check database for the newly filed events and accuracy with GUILD
   guild & and use MIT mit_test database

6. Was the client side of the transmission successful?

   Pass

7. Was the server side of the transmission successful?

   Pass

8. Did the server produced perfect XML output?

   Pass

9. Were the received events identical to those transmitted?

   Pass

10. Were all events ingested successfully?

    Fail

11. Were the ingested events identical to those received?

    N/A

12. Were GUILD able to extract all events?

    N/A

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes: *Data Ingestion ERROR:* Issued command was : putMeta -u smarka -f ALFULLPIPE100K_in.xml -d MIT:mit_test lancelot> ./allegroServer > ALFULLPIPE100K.log Error: Subject: NORMAL5975 error! metadataAPI: putMetaData failed: parseURL port:_001c3f88_LdasContainer_p: getElementAttribute _00206a20_LdasContainer_p size: dbEasyStmt_Submit _272b620_dbEasyStmt_p: ***—¿err:'Start of Error Messages File: ../../../../../api/metadataAPI/so/src/query.cc Line: 196 SQLSTATE: 57011 Native ErrorCode: -973 [IBM][CLI Driver][DB2/SUN] SQL0973N Not enough storage is available in the "QUERY_HEAP" heap to process the statement. SQLSTATE=57011 * * * * − − − >sqlcmd: insert into external_trigger (event_id,notice_type,trigger_source, trigger_type,start_time,start_time_ns,duration,amplitude,frequency1, bandwidth1,frequency2, bandwidth2, signal_to_noise,comment,web,web1) values (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)

Problem resolution: The database query heap size is a configurable parameter in the database manager. Currently it is set to 4000 KB as max. The query heap is Maximum amount of memory (in pages) that can be allocated for the query heap. A query heap is used to store each query in the agent's private memory. (by Mary Lei)

Recommended course of action: The limit on the maximum number of events per XML file in the server code must be set to 5000.

TEST RESULT **Pass**

A.4.6 ALFULLPIPECONT

Test Case: ALFULLPIPECONT

Purpose: Validates the standard operating procedure for the TCP/IP channel.

Tester: Szabolcs Marka, Ik Siong Heng

Test machine: SAM.PHYS.LSU.EDU, LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/07/01 **Time:** 18:00:00

ENVIRONMENT AND PREREQUISITES

Required communication software must be installed on both the remote client (allegroClient) and local server (MIT_Server). putMeta command and GUILD of LDAS should be functional. The client must be compiled on remote machine. The server on the local computer must be running.

PROCEDURE

Utilize sam.phys.lsu.edu as the remote client and use real ALLEGRO data from last year's (2000) run (with periodic marker/calibrator events inserted). The number of events transmitted during this test should be 80-100, which is roughly equivalent to 1 day of events transmitted to IGEC members. his is a long test, it takes nearly a full day to complete. During the test the ALLEGRO data file is filled at a slow rate, one event at the time at uneven time intervals. The client should be invoked at random time at the beginning and at regular times at the end. The data transmitted by the client is received by the server side parser which pads the static fields with ALLEGRO specific information and prepares the XML file for the External Triggers table of the LDAS Metadata/Event DB structure. After this the events are automatically ingested. This test fully simulates the real situation encountered during normal operation interrupted with very short intervals of transmission line (e.g. network) problems. This test should be started late at the day. The first few events should be transmitted immediately then the events should be left to accumulate during the night, so a big chunk will be transmitted at the morning. Afterwards, the allegroClient code should be invoked by hand at random intervals followed by a regular transfer initiated by a standard cron or unix script.

1. Start up the ALLEGRO DAQ simulator to provide events at a low rate at the evening
   Pass
2. Start up the sever on local machine and check server status
   (MIT_Server >& ALFULLPIPE100.log & ; ps **Pass**
3. Reset client bookmark
   echo AL000000000 > allegroLog.dat
4. Wait 10 minutes and transmit the first couple of events already present in the data file.
   allegroClient 18.120.0.66 ¡DATA file name¿ allegroLog.dat > ALFULLPIPE100.remote.log
   Pass
5. Check client and server side logs for error messages and compare the number of transmitted events
   Pass
6. Visually check database for the newly filed events and accuracy with GUILD
   guild & and use MIT mit_test database
   Pass

7. leave the server running but do not transmit events until next morning.
8. Start the client by hand at random times with gaps from 1 minute to 15 minutes.
9. Check client and server side logs for error messages and compare the number of transmitted events
   Visually check database for the newly filed events and accuracy with GUILD
   guild & and use MIT mit_test database

   Pass
10. Start the client automatically at regular times with approximate gaps of 10 minutes
11. Check client and server side logs for error messages and compare the number of transmitted events
    Visually check database for the newly filed events and accuracy with GUILD
    guild & and use MIT mit_test database

    Pass
12. Using standard unix tools, compare the ingested values to input values
    Were all the events ingested successfully?

    Pass
13. Was the input transmitted and output data identical?

    Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: None

Notes:

TEST RESULT **Pass**

A.4.7 GCNFULLPIPE100

Test Case: GCNFULLPIPE100

Purpose: Verifies the full parsing and event insertion sequence into the External Triggers table of the LDAS Metadata/Event DB structure at MIT:mit_test

Tester: Szabolcs Marka

Test machine: LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/10/01 **Time:** 18:00:00

ENVIRONMENT AND PREREQUISITES

putMeta command and GUILD of LDAS should be functional. The client must be compiled locally. The server on the local computer must be running. Parser MIT_GCN must be compiled and running locally. Alert files must be locally accessible.

PROCEDURE

Verifies the full parsing and event insertion sequence into the External Triggers table of the LDAS Metadata/Event DB structure at MIT:mit_test.
Utilize LANCELOT.MIT.EDU as the parser and to access the database table. Use the pre-prepared warnings from the ../input/GCNFULLPIPE100/Alerts/Real/ directory. These are various GRB alerts we received from the GCN network during the course of the past year. These alerts were automatically pre formatted by the automatic e-mail parser, which created a backup copy, sliced off the headers and and computed the GPS start time based on the UTC time supplied with the alert. Use a simple unix script to insert all alerts into the database table one by one. This effectively simulates the highest rate of alerts arriving to be ingested.

1. Start up the script invoking the parser on local machine and check its status periodically.
   foreach F (‘ls ExtTrig/input/GCNFULLPIPE100/Alerts/Real/∗‘)
   echo ”Starting” F
   MIT_GCN GCN_lookup_table.txt F -v >>& GCNFULLPIPE100.log
   echo ”Ending” F
   end

   Pass

2. Check logs for error messages and compare the number of transmitted events

   Pass

3. Visually check database for the newly filed events and accuracy with GUILD
   guild & and use MIT mit_test database

   Pass

4. When all the events are ingested, extract these recently filed events with GUILD.
   Save them as ExtTrigs/output/GCNFULLPIPE100_DB.xml

   Pass

5. Using standard unix tools, compare extracted values to input values

6. Were the GCN events ingested successfully?

   Pass

7. Was the input transmitted and output data identical?

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.5 Data Monitoring Tool tests

This group of tests verify that the DMT and its monitors operate correctly in a stand-alone mode and as an element of the analysis pipeline. The dmt tests are divided into five stages:

1. Test DMT Services: Verify that the basic DMT infrastructure is installed correctly and performs the required services (A.5.1).
2. Stand-alone monitor tests: The DMT monitors are run individually to verify that they run without crashing, and give expected results for known input data (A.5.2, A.5.3).
3. DMT Subsystem tests: Verify that the DMT process manager works and that multiple Monitors run in parallel (A.5.4, A.5.5).
4. Test Trigger logging and overall reliability (A.5.6, A.5.7).
5. DMT documentation test: Verify that the DMT observer page works and correct (A.5.8).

The individual tests that comprise these phases are as follows:

- *Nameserver01* - Verify that the DMT message system and the name server are installed and working correctly.
- *glitchMon_sa_white* - Verifies that the glitch monitor (glitchMon) runs without errors, and does not produce glitch triggers when given stationary white noise.
- *glitchMon_sa_glitch* - Verifies that glitchMon generates triggers corresponding to the Monte-Carlo glitches and that the Trigger Manager correctly logs the triggers.
- *Process01* - Verify that the DMT process manager and the data distribution subsystems are working correctly.
- *TID0101* - Test that the data server runs correctly with the shared memory partition as the data source.
- *TID0201* - Test that the event catalog is working properly.
- *TID0301* - Test the integrity of the results by direct comparison with the output of the LDAS DSO tfclusters.
- *DMTDOCTEST* - Test of automatic DMT documentation process posting DMT status info and monitor documentation on the web.

NOTES:

- The stand-alone monitor tests and the DMT subsystem test use data generated by the “makeFrame” utility which is included in the DMT distribution. This utility program generates white noise with a specified bias and sigma for multiple channels. makeFrame optionally includes glitches consisting of a sinusoid multiplied by a Gaussian envelope.
- All glitchMon tests use configuration files specifying the following complement of channels and associated frequency bands.

Channel	Threshold	f-Low	f-High
H0:PEM-PSL2_MIC	9.0	200.0	250.0
H0:PEM-PSL2_ACCX	5.0	165.0	175.0
H0:PEM-LVEA_SEISX	5.0	5.0	15.0
H0:PEM-LVEA_SEISY	5.0	5.0	15.0
H0:PEM-LVEA_SEISZ	5.0	5.0	15.0
H0:PEM-MX_SEISX	5.0	5.0	15.0
H0:PEM-MX_SEISY	5.0	5.0	15.0
H0:PEM-MX_SEISZ	5.0	5.0	15.0
H0:PEM-MY_SEISX	5.0	5.0	15.0
H0:PEM-MY_SEISY	5.0	5.0	15.0
H0:PEM-MY_SEISZ	5.0	5.0	15.0
H0:PEM-EX_SEISX	5.0	5.0	15.0
H0:PEM-EX_SEISY	5.0	5.0	15.0
H0:PEM-EX_SEISZ	5.0	5.0	15.0
H0:PEM-EY_SEISX	5.0	5.0	15.0
H0:PEM-EY_SEISY	5.0	5.0	15.0
H0:PEM-EY_SEISZ	5.0	5.0	15.0
H0:PEM-COIL_MAGY	5.0	560.0	580.0
H0:PEM-COIL_MAGZ	5.0	560.0	580.0

- All tests involving the tid DMT monitor must run on lancelot.mit.edu, with the tid software from the LIGO-MIT CVS archive, release tid-0-1-0. The path has to be set to `/home/julien/tid:/home/julien/tid/client:/home/julien/tid/class:$PATH` and the linker path has to be set to `/export/dmt/pro/lib:/home/julien/root/lib`.
- The tests running the data viewer require that ROOT version 3.00 be installed on the system.

A.5.1 NameServer01

Test Case: NameServer01

Purpose: Test the DMT message system and name server

Tester: John Zweizig

Test machine: LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/10/01 **Time:** 18:00:00

This test verifies that the DMT messaging system and the name server are correctly installed. The name server is run on an alternate IP port and tested by adding, removing and replacing names.

ENVIRONMENT AND PREREQUISITES

- Installed DMT software version 1.3.1 or later.
- The procedure assumes that the standard DMT setup script has been run e.g. with:
  ```
  source /export/dmt/pro/bin/setup.
  ```

PROCEDURE

Run the the namserver test script as follows:

```
> command/dmtService_NameServer
```

The script runs over several tests and prints whether each test was successful. Tests check whether the name server can be run and whether names can be added, deleted and modified using the standard utility programs.

1. Check that all tests are followed by “SUCCESS!”

Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: None

Notes: None

TEST RESULT **Pass**

A.5.2 glitchMon_sa_white

Test Case: glitchMon_sa_white

Purpose: Stand-alone Test of glitchMon with White Noise

Tester: John Zweizig

Test machine: lancelot.mit.edu

Date (mm/dd/yy): 09/10/01 **Time:** 18:00:00

This test verifys that the stanndard glitch monitor (glitchMon) runs without failure and does not produce extraneous triggers during a 128 second run. The configuration file in this test uses a typical subset of channels, thresholds and frequency bands as described above. This test uses Monte-Carlo generated white noise with the standar deviation and bias measured for the specified channels during the E5 run.

ENVIRONMENT AND PREREQUISITES

- DMT libraries version 1.3.1 or later, root version 3.01-06 or later and glitchMon must be installed.
- The procedure assumes that the DMT environment has been set up by sourcing the dmt setup script.
- The test may be run from the cvs directory or any directory with a similar structure.
- If the test data frames (`input/T-667000000.F` and `input/T-667000064.F`) have not been created, they are prepared with "makeFrame" using the definition file in `glitchMon.white` as follows (starting from the dmt/test cvs directory).

```
> cd input
> makeFrame glitchMon.white
```

PROCEDURE

Run the test script from the test directory:

```
> command/standAlone_glitchMon white > output/glitchMon_sa_white.log
```

This script creates a local name server and trigger manager and runs glitchMon using the configuration file in `input/glichMon_sa_white.conf`. The glitchMon output is saved in `output/glitchmon_sa_white.out` and the list of generated triggers is saved in `output/glitchmon_sa_white.dat`.

1. Verify that the entire data sample was read by inspecting `output/glitchMon_sa_white.log`. The start and end times should be 667000000 and 667000128, respectively.

   Pass

2. Verify that no trigger were generated or recorded. The number of triggers generated by glitchMon and recorded by TrigMgr (listed in `output/glitchMon_sa_white.log`) should be 0.

   Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: None

Notes: None

TEST RESULT **Pass**

A.5.3 glitchMon_sa_glitch

Test Case: glitchMon_sa_glitch

Purpose: Standalone Test of glitchMon with White Noise

Tester: John Zweizig

Test machine: lancelot.mit.edu

Date (mm/dd/yy): 09/10/01 **Time:** 18:00:00

This test verifys that glitchMon finds Monte-Carlo generated glitches and produces valid triggers. The configuration file in this test uses the same set of channels and configuration as for the previous white-noise test. This test uses Monte-Carlo generated white noise with the sigma and bias measured for the specified channels during the E5 run.

ENVIRONMENT AND PREREQUISITES

- Installed processes and environmental sett-up is as for the previous test.
- If the test data frames (`input/T-667010000.F` and `input/T-667010064.F`) have not been created, they are prepared with "makeFrame" using the definition file in `glitchMon.glitch` as follows (starting from the dmt/test cvs directory).

```
> cd input
> makeFrame glitchMon.glitch
```

PROCEDURE

Run the test script from the test directory:

```
> cmd/standAlone_glitchMon glitch > output/glitchMon_sa_glitch.log.
```

This script creates a local name server and trigger manager and runs glitchmon using the configuration file in `input/glichMon_sa_glitch.conf`. The glitchmon output is saved in `output/glitchmon_sa_glitch.out` and the list of generated triggers is saved in `output/glitchmon_sa_glitch.dat`.

1. Check that glitchmon processed the entire data sample. The start and end times in `output/glitchMon_sa_glitch.log` should be 667010000 and 667010128, respectively.

   Pass

2. The numbers of triggers generated by glitchMon and recorded by TrigMgr are also recorded in `output/glitchMon_sa_glitch.log`. These should be equal and non-zero.

   Pass

SUMMARY

Known faults encountered – list bug IDs: None.

New faults submitted – list bug IDs: None.

Notes: None.

TEST RESULT **Pass**

A.5.4 Process01

Test Case: Process01

Purpose: Test the process manager and data distribution subsystems.

Tester: John Zweizig

Test machine: LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/10/01 **Time:** 18:00:00

This test verifies that the DMT process manager and data distribution subsystems are correctly installed and working. It also verifys that the Monitors run without interference.

ENVIRONMENT AND PREREQUISITES

- All tests must be run on `lancelot.mit.edu` or an equivalent DMT node.
- Revision 1.3.1 or later of the DMT software must be installed.
- Ligotools dataflow package version 3.0.1 or later.
- The procedure assumes that the following environment variables are set: `DMTHOME` (DMT top directory e.g. `/export/dmt`) and `DMTVERSION` (DMT version name, e.g. `pro`).
- The standard DMT environment is set with the following command: `source $DMTHOME/$DMTVERSION/bin/setup`.
- The MDC cvs repository is assumed to be in `$HOME/mdc/ldasmdc/burst-stochastic/`.

PROCEDURE

1. Verify that the desired RDS frame data are available in the `/aux/rds/e5/*` or an equivalent directory.

   Pass

2. Verify that the directories versions, etc. specified in `command/MIT_BurStoMDC_Initial` are correct.

   Pass

3. Start the process manager as follows:

   ```
   cd $HOME/mdc/ldasmdc/burst-stochastic/test/dmt/command/
   procmgt --file:./procmgt start
   ```

   The script runs a process manager which in turn starts the DMT environment processes. Verify with `ps` that the following processes are running:

   ```
   procmgt     NameServer     TrigMgr     DpushRT
   glitchMon   SegGener
   ```

   Pass

4. Verify that data are being read into the shared memory partition by typing

```
setenv LIGOSMPART MDC_Online
smdump
```

The numbers in the "count" column of the "Buffer Status" table should be non-zero and should increase with time is `smdump` is type repeatedly. The count fields in the "Consumer Status" table should be approximately equal to the sum of the counts in the buffer status table.

Pass

5. Check that frames are accessible by running FrTest for about 10 seconds (type ctrl/c after 10 seconds). FrTest should print

```
Frame reading terminated after 10 frames.
```

The number of frames may vary, but it should be > 0 and no messages starting with "Error:" should be printed.

Pass

6. Note that the process environment generated with this test is used by subsequent tests. The final step of this test should not be run until the other tests are completed.
   Stop the process manager and subordinate processes with

```
pkill procmgt
```

All process listed above should have stopped and the shared memory partition should be deleted (check with ps and smdump).

Pass

SUMMARY

Known faults encountered – list bug IDs: None

New faults submitted – list bug IDs: None

Notes: Perl is installed in a non-standard location on lancelot, requiring that the first line of procmgt be changed.
In revision 1.3.1 of the process manager, processes which are run under intermediate shells (i.e. TrigMgr, DpushRT and NameServer) are not stopped by the process manager. The processes must then be stopped with pkill for procmgt to finish.

TEST RESULT **Pass**

A.5.5 TID0101

Test Case: TID0101

Purpose: Test that the data server runs correctly with the shared memory partition as the data source

Tester: Julien Sylvestre

Test machine: lancelot.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 9:39

ENVIRONMENT AND PREREQUISITES

This test requires the executable `tidd` and the DMT environment.

PROCEDURE

The test should run on the RDS frame files from the E5 engineering run.

On the DMT machine (`lancelot.mit.edu`), issue the following command:

```
cd ldasmdc/burst-stochastic/test/dmt/command
cp TID0101 .tid
tidd
```

Let the executable run for one hour.

1. The executable was still running after one hour of operation. **Pass**
2. The time id number (the numbers that are appearing on the screen while `tidd` runs) was greater than 3600 after one full hour of operation. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.5.6 TID0201

Test Case: TID0201

Purpose: Test that the event catalog is working properly.

Tester: Julien Sylvestre

Test machine: lancelot.mit.edu

Date (mm/dd/yy): 09/10/01 **Time:** 16:32

ENVIRONMENT AND PREREQUISITES

NOTE: This test requires the relase tid-0-1-1 from the LIGO-MIT CVS archive. This test requires the executables `tidd` and `tidclass`, and the DMT environment.

PROCEDURE

The test should run on the RDS frame files from the E5 engineering run.

On the DMT machine (`lancelot.mit.edu`), issue the following command:

```
cd ldasmdc/burst-stochastic/test/dmt/command
cp TID0201 .tid
tid
```

In a different shell, issue the following: `cd ldasmdc/burst-stochastic/test/dmt/command`

```
cp TID0201CLASS .TidClass
tidclass
```

In a different shell, issue the following: `cd ldasmdc/burst-stochastic/test/dmt/command`

```
cp TID0201GUI .tidgui
Gui
```

Look for airplanes signatures in `H0:PEM-MX_SEISX`. Note the central GPS times of these signals.

Let all three executables run until `tidd` returns.

Open the file `ldasmdc/burst-stochastic/test/dmt/output/TID0201.html` with a web browser, and verify that `mxpump` is detected every 900 seconds, and that airplanes were detected where they were seen in the dat viewer.

1. `mxpump` was detected every 900s. **Pass**
2. The correct number of airplane signals was seen. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.5.7 TID0301

Test Case: TID0301

Purpose: Test the integrity of the results by a direct comparison with the output from the LDAS DSO tfclusters.

Tester: Julien Sylvestre

Test machine: lancelot.mit.edu

Date (mm/dd/yy): 09/10/01 **Time:** 11:53

ENVIRONMENT AND PREREQUISITES

This test requires the prior completion of TFCLUSTERS1401. This test requires the executables `tid`, and the DMT environment.

PROCEDURE

The test should run on the RDS frame files from the E5 engineering run, from GPS times 681144358 and 681147957.

On the DMT machine (`lancelot.mit.edu`), issue the following command:

```
cd ldasmdc/burst-stochastic/test/dmt/command
cp TID0301 .tid
tid
```

As soon as `tid` is running and displaying time id numbers, issue the following commands in a different shell: `cd ldasmdc/burst-stochastic/test/dmt/command`

```
cp TID0301GUI .tidgui
tidgui
```

When the data viewer appears:

1. Click on 'Edit', choose 'Measures'.
2. In the window that pops up, click on 'path', set the output to `ldasmdc/burst-stochastic/test/dmt/output/TID0301.bin`.
3. Set the 'Stop time' field to 3000.
4. Click on 'New Measure'. Click on the tab 'Measure #1'. Set the 'Measure' field to 'H0:PEM-MX_SEISX'.
5. Select the 'Measure Parameters': 'Min Time', 'Max Time', 'Min Frequency', 'Max Frequency', and 'Size'.
6. Click on the tab 'Control'. Click on 'Now', and then on 'Start'.

NOTE: You have 200 seconds to complete the steps above. For a longer period, the test must be restarted.

When `tid` is done, launch Matlab, and issue the following: `addpath('/home/julien/ldasmdc/burst-stochastic/test/dmt/src')`

```
tidrectangles('/home/julien/ldasmdc/burst-stochastic/test/dmt/output/TID0301.bin','
stochastic/test/tfclusters/output/TFCLUSTERS1401.txt')
```

1. The matlab script indicated that the two outputs were compatible. **Pass**

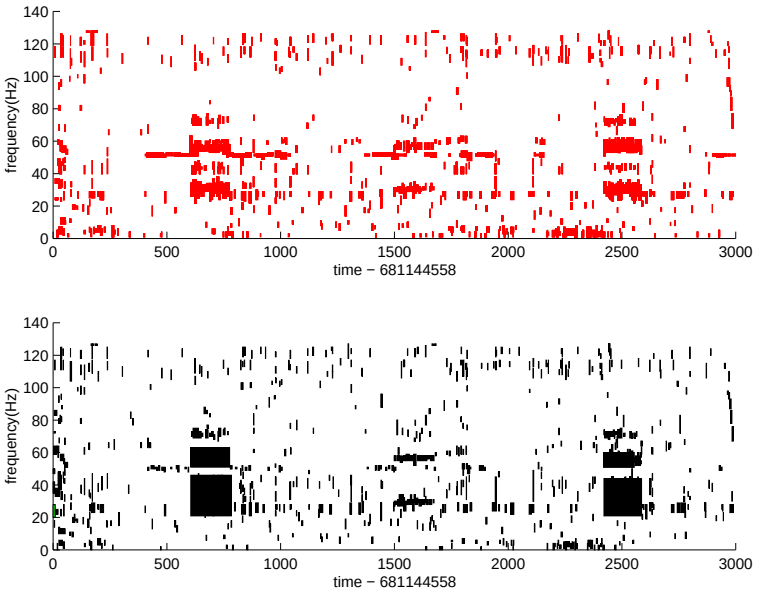

Figure 2: Comparison of output from the LDAS DSO tfclusters (green) and the DMT monitor TID (red).

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.5.8 DMTDOCTEST

Test Case: DMTDOCTEST

Purpose: Test of automatic DMT documentation process posting DMT status info and monitor documentation on the web.

Tester: Szabolcs Marka

Test machine: LANCELOT.MIT.EDU

Date (mm/dd/yy): 09/10/01 **Time:** 18:00:00

ENVIRONMENT AND PREREQUISITES

DMT should be up and available with monitors running. The DMT observer script should be up and installed.

PROCEDURE

Follow the steps below to check the performance of the automatic DMT observer system.

1. Does the LANCELOT and MIT link appear on the main DMT observer page?
   blue.ligo-wa.caltech.edu/gds/dmt/Monitors/spi.html

   Pass

2. Is the LANCELOT page fresh? (The date of last refresh should be less than 15 minutes)
   lancelot.mit.edu/gds/dmt/Monitors/LANCELOT_spi.html

   Pass

3. Does the LANCELOT page refreshes the screen once a minute?

   Pass

4. Do the links in the bottom of the header work and give the right information?
   Check for machine load (TOP) and trigger manager statistics/logs.

   Pass

5. Check whether the status of running monitors are correctly and dynamically displayed.
   Check for SegGener and glitchMon runnig under process manager
   SeqInsert should be listed as missing
   NamePushRT, NameServer and TrigMgr should be the active support processes.
   Start up BitTest and check whether it appears automatically as Monitors in testing phase

   Pass

6. Check whether the runtime properties of monitors are displayed, make sense and agree with the TOP data.

   Pass

7. Check whether the Description links point to the documentation of the right monitor (wherever they are present)

   Pass

8. Check whether the one line description appears after the monitor's name (Undocumented monitor label is OK for now.)

   Pass

9. Check whether the links on the monitor's name point to a valid directory. (This supposed to be the web output dir)

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes: Monitor developers should provide more detailed documentation and their monitors should have informative and frequently refreshed web output.

TEST RESULT **Pass**

A.6 Excess power statistic tests

These tests are designed to verify the excess power search shared object running under the LDAS pipeline. A general overview of the tests can be found in Sec. 4.6.

The input data for the tests are frame files containing 512 and 64 seconds of interferometer gravity wave channel `H2:LSC-AS_Q` sampled at 16384 Hz. The files should contain white gaussian noise of mean 0 and variance 1 with a Zwerger-Muller or 200Hz burst superimposed as designated by the file name and documented at `http://www.ligo.caltech.edu/ ajw/bursts/mdc_data/mdc_data.html`

1. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_64.F`
2. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_64.F`
3. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_64.F`
4. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_512.F`
5. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_512.F`
6. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZMbig_512.F`
7. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_512.F`

A.6.1 POWER00

Test Case: POWER00

Purpose: Verify the power shared object passes tests POWER01 and POWER02 from the MPI MDC in section 07power

Tester: Patrick Brady

Test machine: ldas.ligo.mit.edu

Date (mm/dd/yy): 09/04/01 **Time:** 15:30

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS. It depends on completion of the pipeline tests in subsection A.1.

PROCEDURE

Run the POWER01 and POWER02 LDAS user commands in sections 07power of the MPI MDC. These tests have been updated to use the `dataPipeline` command and the input data described therein. Ensure that all pass criteria are met.

1. POWER01 of 07power

Pass

2. POWER02 of 07power

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.6.2 POWER01

Test Case: POWER01

Purpose: Verify that the power shared object can filter zeros in the LDAS pipleline producing no ouput for the database.

Tester: Patrick Brady

Test machine: ldas.ligo.mit.edu

Date (mm/dd/yy): 09/04/01 **Time:** 15:30

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS. It depends on the successful completion all test checklists in subsections A.1 and A.6.1.

The following input data are required for all tests:

1. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_zeros_512.F`
2. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_512.F`
3. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd`

on the `ldas` system.

PROCEDURE

Run the LDAS user command scripts listed in the tests below. Inspect the job log files using the Control Monitor client and ensure that all APIs used in the job pipeline exited without errors.

No output should be generated by the shared object.

1. The pipeline command should read in the data (all zeros in this case) resample it to 1024 Hz, compute the spectrum, append the appropriate response function, and filter the data. The segment duty cycle is unity. The job should complete without errors and no events should be recorded.

`/.../ldasmdc/burst-stochastic/test/power/command/POWER0101.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email patrick@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,1,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.001,16,channel)
      -subject POWER0101
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER0101 }
      -resultname { POWER0101 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,1026,1);
         intermediate(,mpi,rgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_zeros_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"
```

```
set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

Pass

2. The pipeline command should read in the data (white noise containing Zwerger-Muller waveforms) resample it to 1024 Hz, compute the spectrum, append the appropriate response function, and filter the data. The segment duty cycle is 4. The job should complete without errors and one event should be inserted into the database.

   `/.../ldasmdc/burst-stochastic/test/power/command/POWER0102.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name patrick -password booze -email patrick@gravity.phys.uwm.edu }
  {
   dataPipeline
      -np 3
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,22,513,3,1,1,200.0,1.0,32,3.0,0.5,4,0.00001,1,channel)
      -subject POWER0102
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER0102 }
      -resultname { POWER0102 }
      -responsefiles {
           file:/home/patrick/iulmdc/ldasmdc/inspiral/test/13challenge/input/response.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_zeros_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.phys.uwm.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

Pass

SUMMARY

Known faults encountered – list bug IDs: The duration and frequency band information is incorrectly recorded in the database; closed 20 November 2001

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.6.3 POWER02

Test Case: POWER02

Purpose: Verify that the power shared object can filter white gaussian noise in the LDAS pipeline

Tester: Erik Katsavounidis

Test machine: ldas-dev

Date (mm/dd/yy): 09/04/01 **Time:** 09:00

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on MIT-LDAS. It depends on the successful completion all test checklists in this subsection.

The following input data are required for these tests:

1. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_64.F`
2. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_512.F`

Every test is devided into two parts:

I: execution and correctness of running

II: verification of the correctness of the database entry

PROCEDURE

Run the LDAS user command scripts listed in the tests below. Inspect the job log files using the Control Monitor client and ensure that all APIs used in the job pipeline exited without errors.

Study the filter output as a function of the threshold chosen.

Verify trigger insertion into the database using GUILD.

1. The pipeline command should read in the gaussian noise from a frame containing 64/512 seconds of data sampled at 16384 Hz, downsample the data at 1024 Hz, calculate a power spectrum of FFT, append the appropriate response function, filter this through a single power template with multiple data segments of length of 1024 points and overlap 512.

   (a) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1005.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.01,16,channel)
      -subject POWER1005
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1005 }
      -resultname { POWER1005 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
```

```
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6224

(b) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1006.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.001,16,channel)
      -subject POWER1006
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1006 }
      -resultname { POWER1006 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6225

(c) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1007.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.0001,16,channel)
      -subject POWER1007
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1007 }
      -resultname { POWER1007 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6226

(d) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1008.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.00001,16,channel)
      -subject POWER1008
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1008 }
      -resultname { POWER1008 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6228

(e) `/.../ldasmdc/burst-stochastic/test/power/command/POWER2006.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,1021,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.001,16,channel)
      -subject POWER2006
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER2006 }
      -resultname { POWER2006 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6255

(f) `/.../ldasmdc/burst-stochastic/test/power/command/POWER2007.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
```

```
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,1021,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.0001,16,channel)
      -subject POWER2007
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER2007 }
      -resultname { POWER2007 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6257

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes: A rather long processing (computer) time of up to 10 times the length of the time series was observed (running on 4 nodes). This could be related to non optimal use of the input parameters or due to the fact that the compilation had the debugger on. Even if the correctness of the database entry(entries) for each of the jobs has not been checked throughtout all the subtests, a passing grade is assigned to the tests as their execution, correctness of running and database filling was sucessfull.

TEST RESULT **Pass**

A.6.4 POWER03

Test Case: POWER03

Purpose: Verify that the power shared object can filter Zwerger-Muller-like and sinusodial-like signal superimposed to gaussian noise in the LDAS pipeline

Tester: Erik Katsavounidis

Test machine: ldas-dev

Date (mm/dd/yy): 09/04/01 **Time:** 09:00

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on MIT-LDAS. It depends on the successful completion all test checklists in this subsection.

The following input data are required for these tests:

1. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_64.F`
2. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_64.F`
3. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_512.F`
4. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZMbig_512.F`
5. `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_512.F`

Every test is devided into two parts:

I: execution and correctness of running

II: verification or the correctness of the database entry

PROCEDURE

Run the LDAS user command scripts listed in the tests below. Inspect the job log files using the Control Monitor client and ensure that all APIs used in the job pipeline exited without errors.

Study the filter output as a function of the threshold chosen.

Verify trigger insertion into the database using GUILD.

1. The pipeline command should read in the gaussian noise from a frame containing 64/512 seconds of data sampled at 16384 Hz, downsample the data at 1024 Hz, calculate a power spectrum of FFT, append the appropriate response function, filter this through a single power template with multiple data segments of length of 1024 points and overlap 512.

(a) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1009.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.01,16,channel)
      -subject POWER1009
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1009 }
      -resultname { POWER1009 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6229

(b) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1010.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.001,16,channel)
      -subject POWER1010
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1010 }
      -resultname { POWER1010 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6231

(c) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1011.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.0001,16,channel)
```

```
      -subject POWER1011
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1011 }
      -resultname { POWER1011 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6232

(d) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1012.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.00001,16,channel)
      -subject POWER1012
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1012 }
      -resultname { POWER1012 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6234

(e) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1017.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.01,16,channel)
      -subject POWER1017
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1017 }
      -resultname { POWER1017 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
```

```
        rgw = resample(gw, 1, 16);
        srgw = slice(rgw,0,65536,1);
        intermediate(,mpi,srgw,resampled gw timeseries);
        p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6250

(f) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1018.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.001,16,channel)
      -subject POWER1018
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1018 }
      -resultname { POWER1018 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6251

(g) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1019.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.0001,16,channel)
      -subject POWER1019
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1019 }
      -resultname { POWER1019 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
```

```
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6252

(h) `/.../ldasmdc/burst-stochastic/test/power/command/POWER1020.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,126,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.00001,16,channel)
      -subject POWER1020
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER1020 }
      -resultname { POWER1020 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,65536,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_64.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6254

(i) `/.../ldasmdc/burst-stochastic/test/power/command/POWER2010.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,1021,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.001,16,channel)
      -subject POWER2010
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER2010 }
      -resultname { POWER2010 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6258

(j) `/.../ldasmdc/burst-stochastic/test/power/command/POWER2011.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,1021,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.0001,16,channel)
      -subject POWER2011
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER2011 }
      -resultname { POWER2011 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZM_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6259

(k) `/.../ldasmdc/burst-stochastic/test/power/command/POWER2014.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,1021,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.001,16,channel)
      -subject POWER2014
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER2014 }
      -resultname { POWER2014 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZMbig_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6261

(l) `/.../ldasmdc/burst-stochastic/test/power/command/POWER2015.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,1021,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.0001,16,channel)
      -subject POWER2015
      -datacondtarget mpi
      -mddapi ligolw
```

```
      -state {}
      -resultcomment { POWER2015 }
      -resultname { POWER2015 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZMbig_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6262

(m) `/.../ldasmdc/burst-stochastic/test/power/command/POWER2018.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,1021,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.001,16,channel)
      -subject POWER2018
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER2018 }
      -resultname { POWER2018 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6272

(n) `/.../ldasmdc/burst-stochastic/test/power/command/POWER2019.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.uwm.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldaspower.so
      -filterparams (1026,1021,513,3,1,1,1.0,1.0,32,3.0,0.5,1,0.0001,16,channel)
      -subject POWER2019
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { POWER2019 }
      -resultname { POWER2019 }
      -responsefiles {
           file:/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/resp_power_test_EK.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
```

```
        p = psd( rgw, 1026 );
      }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_SIN_512.F
      -framequery Adc(H2:LSC-AS_Q)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

LDAS Job 6273

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes: A rather long processing (computer) time of up to 10 times the length of the time series was observed (running on 4 nodes). This could be related to non optimal use of the input parameters or due to the fact that the compilation had the debugger on. Even if the correctness of the database entry(entries) for each of the jobs has not been checked throughtout all the subtests, a passing grade is assigned to the tests as their execution, correctness of running and database filling was sucessfull.

TEST RESULT **Pass**

A.7 Slope DSO Tests

These tests are to establish that the slope detection shared object for bursts operates correctly as part of the LDAS / LAL pipeline, ingesting data, searching it for bursts using a simple time-domain filter and peak search algorithm, and sending a trigger to the LDAS database for each burst detected.

A.7.1 SLOPE00

Test Case: SLOPE00

Purpose: Verify that the slope shared object works in the LDAS/LAL/LALWRAPPER pipeline environment. In particular, passing of command-line arguments to the DSO running as part of the LDAS pipeline, and writing of events to the LDAS database by the DSO, are tested.

Tester: E. Daw

Test machine: `ldas.mit.edu`

Date (mm/dd/yy): 09/08/2001 **Time:** 13.35EDT

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS. It requires the presence of the frame data file `mdc_white_512.F`. A copy of this data file is included in the CVS repository for the burst-stochastic mock data challenge.

The data file contains 64 seconds of pseudorandom generated uncorrelated Gaussian distributed noise under the channel name `H2:LSC-AS_Q`. The sampling rate of this simulated data is 16384Hz, so a total of 2^{20} data points pass through the slope filter.

PROCEDURE

Run the `tcl` command script `/input/slope_00.tcl`. Monitor the progress of the job through the various MPIs in the pipeline using the LDAS web-based watchdog software. Verify using this software that events are successfully added to the database.

Pass

SUMMARY

Known faults encountered – list bug IDs: none

New faults submitted – list bug IDs: none

Notes: With regards to supplying the channel name as an input parameter to the search: note that the channel names are altered to reflect the history of the data during analysis by the LDAS pipeline, particularly any preprocessing carried out by the data conditioning API. With regards to the writing of DMT triggers, several modifications to the database outPut structure were made before the output was successful. In addition, an extra field was added to the sngl_burst database table, called search. This field contains a character array unique to each search code, so that it can be determined which search generated a given trigger in the database.

TEST RESULT **Pass**

A.7.2 SLOPE01

Test Case: SLOPE01

Purpose: Verify that a `tcl` script written by Phillip Charlton works in combination with the slope shared object, and allows the shared object to seek out arbitrary channels in arbitrary frames of engineering run data.

Tester: E. Daw

Test machine: `ligo.mit.edu`

Date (mm/dd/yy): 09/07/2001 **Time:** 14.00 EDT

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS. It requires the presence of the Hanford `E5` data on the LDAS disk system mounted on the Beowulf.

PROCEDURE

Run the `tcl` command script `/input/slope_01.tcl`. Monitor the progress of the job through the various MPIs in the pipeline using the LDAS web-based watchdog software. Verify using this software that events are successfully added to the database.

Pass

SUMMARY

Known faults encountered – list bug IDs: none

New faults submitted – list bug IDs: none

Notes: see test SLOPE00

TEST RESULT **Pass**

A.8 tfclusters tests

The tests in this section were designed to test various aspects of the tfclusters package, including the integrity of the data pipeline (ingestion of data from datacondAPI, ingestion of events by database), the integrity of the tfclusters code, and the statistical validity of the operations performed.

The lal/lalwrapper algorithms contain the following parts that will be tested independently as much as possible:

1. Data Ingestion/Casting to load time series of various types.
2. Threshold Estimation for non-white noise, from best-fitted Rice distribution to the power histogram.
3. Spectrogram Generation and First Power Threshold for the generation of thresholded ("black and white") spectrograms.
4. Clustering Analysis: First Threshold raw threshold on cluster size.
5. Clustering Analysis: Second Threshold formation of generalized clusters from close, small clusters.
6. Integrated Power Threshold last threshold on total power in cluster.
7. Event Lists Merging performed on the search master, combine outputs from the different nodes.
8. Event Generation for Database to generate a well-formatted file to be ingested in the metadata database.

NOTE: It is assumed that all stand-alone wrapperAPI jobs are ran from `burst-stochastic/test/tfclusters/results`.

A.8.1 TFCLUSTERS0101

Test Case: TFCLUSTERS0101

Purpose: Test the integrity of Data Ingestion by running on a vector of zeros.

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 8:40

ENVIRONMENT AND PREREQUISITES

This test is executed through the stand-alone wrapperAPI.

PROCEDURE

The test should run on the ilwd file `/.../ldasmdc/burst-stochastic/test/tfclusters/input/Z0101.ilwd` which contains $2^{17} + 2^{10}$ zeros in a REAL4 timeseries. There should be no error messages in running the commands below. There should be no files with names as `output_*.ildw` created.

```
lamboot -v
mpirun -v /.../ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0101.sche
```

```
#
# lam boot schema for trivial shared object
#
h -np 4 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-3)" -dynlib="/ldcg/lib/lalwrapper/libldastfclusters.so" -
dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(132096,33792,1,256,1024,0.1,-1e-3,1,0,128,5,0,0,0,0,0,0,2,3,4,4)" -
realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/Z0101.ilwd"
```

1. There was no error messages in running the commands above. **Pass**
2. No `output_*.ildw` file was created. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.2 TFCLUSTERS0102

Test Case: TFCLUSTERS0102

Purpose: Test the integrity of Data Casting by running on a vector of zeros of type INT2S.

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 8:45

ENVIRONMENT AND PREREQUISITES

This test is executed through the stand-alone wrapperAPI.

PROCEDURE

The test should run on the ilwd file `/.../ldasmdc/burst-stochastic/test/tfclusters/input/Z0102.ilwd` which contains $2^{17} + 2^{10}$ zeros in a INT2S timeseries. There should be no error messages in running the commands below. There should be no files with names as `output_*.ildw` created.

```
lamboot -v
mpirun -v /.../ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0102.sche
```

```
#
# lam boot schema for trivial shared object
#
h -np 4 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-3)" -dynlib="/ldcg/lib/lalwrapper/libldastfclusters.so" -
dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(132096,33792,1,256,1024,0.1,-1e-3,1,0,128,5,0,0,0,0,0,0,2,3,4,4)" -
realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/Z0102.ilwd"
```

1. There was no error messages in running the commands above. **Pass**
2. No `output_*.ildw` file was created. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.3 TFCLUSTERS0201

Test Case: TFCLUSTERS0201

Purpose: Verify the integrity of Spectrogram Generation and First Power Threshold by running on white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 8:50

ENVIRONMENT AND PREREQUISITES

This test is executed through the stand-alone wrapperAPI.

PROCEDURE

The test should run on the ilwd file `/.../ldasmdc/burst-stochastic/test/tfclusters/input/W0201.ilwd` which contains a REAL4 timeseries of length 2^{17}, sampled at 256 Hz. There should be no error messages in running the commands below. There should be one file named `output_1.ildw` created.

```
    lamboot -v
mpirun -v /.../ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0201.sche
```

```
#
# lam boot schema for trivial shared object
#
h -np 4 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-3)" -dynlib="/ldcg/lib/lalwrapper/libldastfclusters.so" -
dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(131072,66048,1,256,1024,0.01,-1e-3,1,0,128,1)" -realTimeRatio=0.9 -
doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/W0201.ilwd"
```

1. There was no error messages in running the commands above. **Pass**
2. File `output_1.ildw` was created. It contains 576 events [1]. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

[1] The number of events can be read from the second line in the file, in the field “size”.

A.8.4 TFCLUSTERS0202

Test Case: TFCLUSTERS0202

Purpose: Verify the integrity of the First Threshold of Clustering Analysis by running on white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 8:52

ENVIRONMENT AND PREREQUISITES

This test is executed through the stand-alone wrapperAPI.

PROCEDURE

The test should run on the ilwd file `/.../ldasmdc/burst-stochastic/test/tfclusters/input/W0201.ilwd` which contains a REAL4 timeseries of length 2^{17}, sampled at 256 Hz. There should be no error messages in running the commands below. There should be one file named `output_1.ildw` created.

```
lamboot -v
mpirun -v /.../ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0202.sche
```

```
#
# lam boot schema for trivial shared object
#
h -np 4 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-3)" -dynlib="/ldcg/lib/lalwrapper/libldastfclusters.so" -
dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(131072,66048,1,256,1024,0.1,-1e-3,1,0,128,5,0,0,0,0,0,0,0,0,0)" -
realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/W0201.ilwd"
```

1. There was no error messages in running the commands above. **Pass**
2. File `output_1.ildw` was created. It contains 24 events. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.5 TFCLUSTERS0203

Test Case: TFCLUSTERS0203

Purpose: Verify the integrity of the Second Threshold of Clustering Analysis by running on white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 8:53

ENVIRONMENT AND PREREQUISITES

This test is executed through the stand-alone wrapperAPI.

PROCEDURE

The test should run on the ilwd file `/.../ldasmdc/burst-stochastic/test/tfclusters/input/W0201.ilwd` which contains a REAL4 timeseries of length 2^{17}, sampled at 256 Hz. There should be no error messages in running the commands below. There should be one file named `output_1.ildw` created.

NOTE: Save the output file `output_1.ildw` under the name `output_TFCLUSTERS0203.ildw` in `/.../ldasmdc/burst-stochastic/test/tfclusters/output` as it will be needed by the test TFCLUSTERS0301.

```
lamboot -v
mpirun -v /.../ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0203.sche
cp output_1.ilwd /.../ldasmdc/burst-stochastic/test/tfclusters/output/output_TFCLUS
```

```
#
# lam boot schema for trivial shared object
#
h -np 4 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-3)" -dynlib="/ldcg/lib/lalwrapper/libldastfclusters.so" -
dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(131072,66048,1,256,1024,0.1,-1e-3,1,0,128,5,0,0,0,0,0,2,3,4,4)" -
realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/W0201.ilwd"
```

1. There was no error messages in running the commands above. **Pass**
2. File `output_1.ildw` was created. It contains 43 events. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.6 TFCLUSTERS0204

Test Case: TFCLUSTERS0204

Purpose: Verify the integrity of the Generalized Power Threshold by running on white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 8:53

ENVIRONMENT AND PREREQUISITES

This test is executed through the stand-alone wrapperAPI.

PROCEDURE

The test should run on the ilwd file `/.../ldasmdc/burst-stochastic/test/tfclusters/input/W0201.ilwd` which contains a REAL4 timeseries of length 2^{17}, sampled at 256 Hz. There should be no error messages in running the commands below. There should be one file named `output_1.ildw` created.

```
    lamboot -v
mpirun -v /.../ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0204.sche
```

```
#
# lam boot schema for trivial shared object
#
h -np 4 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-3)" -dynlib="/ldcg/lib/lalwrapper/libldastfclusters.so" -
dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(131072,66048,1,256,1024,0.01,-1e-3,0.5,0,128,1)" -realTimeRatio=0.9 -
doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/W0201.ilwd"
```

1. There was no error messages in running the commands above. **Pass**
2. File `output_1.ildw` was created. It contains 259 events. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.7 TFCLUSTERS0301

Test Case: TFCLUSTERS0301

Purpose: Verify the integrity of Event Lists Merging by running on white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 8:55

ENVIRONMENT AND PREREQUISITES

This test is executed through the stand-alone wrapperAPI. It uses the output of TFCLUSTERS0203.

PROCEDURE

The test should run on the ilwd file `/.../ldasmdc/burst-stochastic/test/tfclusters/input/W0201.ilwd` which contains a REAL4 timeseries of length 2^{17}, sampled at 256 Hz. There should be no error messages in running the commands below. There should be one file named `output_1.ildw` created.

```
   lamboot -v
mpirun -v /.../ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0301.sche
/.../ldasmdc/burst-stochastic/test/tfclusters/src/DumpOut output_1.ilwd
/.../ldasmdc/burst-stochastic/test/tfclusters/output/output_TFCLUSTERS0203.ildw
```

```
#
# lam boot schema for trivial shared object
#
h -np 4 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-3)" -dynlib="/ldcg/lib/lalwrapper/libldastfclusters.so" -
dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(131072,131072,1,256,0,0.1,-1e-3,1,0,128,5,0,0,0,0,0,0,2,3,4,4)" -
realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/W0201.ilwd"
```

1. There was no error messages in running the commands above. **Pass**
2. File `output_1.ildw` was created. It contains 43 events. **Pass**
3. The output of the `DumpOut` command was “The two event lists concord!”. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.8 TFCLUSTERS0401

Test Case: TFCLUSTERS0401

Purpose: Verify the integrity of Threshold Generation by running on white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 8:56

ENVIRONMENT AND PREREQUISITES

This test is executed through the stand-alone wrapperAPI.

PROCEDURE

The test should run on the ilwd file `/.../ldasmdc/burst-stochastic/test/tfclusters/input/W0201.ilwd` which contains a REAL4 timeseries of length 2^{17}, sampled at 256 Hz. There should be no error messages in running the commands below. There should be one file named `output_1.ildw` created.

```
lamboot -v
mpirun -v /.../ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0401.sche
```

```
#
# lam boot schema for trivial shared object
#
h -np 4 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-3)" -dynlib="/ldcg/lib/lalwrapper/libldastfclusters.so" -
dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(131072,66048,1,256,1024,0.1,1e-3,1,0,128,5,0,0,0,0,0,0,2,3,4,4)" -
realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/W0201.ilwd"
```

1. There was no error messages in running the commands above. **Pass**
2. File `output_1.ildw` was created. It contains 49 events. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.9 TFCLUSTERS0501

Test Case: TFCLUSTERS0501

Purpose: Verify the integrity of Data Ingestion from the datacondAPI with a timeseries of zeros

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 11:43

ENVIRONMENT AND PREREQUISITES

This test is executed through LDAS with the dataPipeline command.

PROCEDURE

The test should run on the frame file `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/Z683399414-1-512.F` which contains a REAL4 timeseries of length 2^{17}, sampled at 256 Hz, and containing only zeroes. There should be no error messages in running the commands below. There should be no event written in the database.

`/ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0501.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.psu.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldastfclusters.so
      -filterparams (131072,33536,1,256,1024,0.1,-1e-3,1,0,128,5,0,0,0,0,0,0,2,3,4,4)
      -subject TFCLUSTERS0501
      -datacondtarget mpi
      -resultname { TFCLUSTERS0501 result}
      -aliases x=_ch0
      -algorithms {m=mean(x);
                   y=sub(x,m);
                   }
      -frames { /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/Z683399414-1-512.F}
      -framequery Adc(0)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

1. There was no error messages in running the commands above. **Pass**
2. No event was written in the database. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.10 TFCLUSTERS0502

Test Case: TFCLUSTERS0502

Purpose: Verify the integrity of Trigger Ingestion by the database with white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 12:36

ENVIRONMENT AND PREREQUISITES

This test is executed through LDAS with the dataPipeline command.

PROCEDURE

The test should run on the frame file `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/W683400150-1-512.F` which contains a REAL4 timeseries of length 2^{17}, sampled at 256 Hz, and containing white noise of unit variance. There should be no error messages in running the commands below. There should be 49 events written in the database.

`/ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS0502.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.psu.edu }
  {
   dataPipeline
      -dynlib /ldcg/lib/lalwrapper/libldastfclusters.so
      -filterparams (131072,33536,1,256,1024,0.1,-1e-3,1,0,128,5,0,0,0,0,0,0,2,3,4,4)
      -subject TFCLUSTERS0502
      -datacondtarget mpi
      -resultname { TFCLUSTERS0502 result}
      -aliases x=_ch0
      -algorithms {m=mean(x);
                   y=sub(x,m);
                   }
      -frames { /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/W683400150-1-512.F}
      -framequery Adc(0)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

1. There was no error messages in running the commands above. **Pass**
2. 49 events were written in the database. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.11 TFCLUSTERS1001

Test Case: TFCLUSTERS1001

Purpose: Verify the validity of the statistical test involving the First Threshold in Clustering Analysis, with white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 12:42

ENVIRONMENT AND PREREQUISITES

This test is executed through the stand-alone wrapperAPI.

PROCEDURE

The test should run on the ilwd file `/.../ldasmdc/burst-stochastic/test/tfclusters/input/W1001.ilwd` which contains a REAL4 timeseries of unit variance white noise of length 2^{15}, sampled at 256 Hz. There should be no error messages in running the commands below. There should be one file named `output_1.ildw` created. The number of event in that file, N, should satisfy the following within the statistical errors:

$$\Lambda = \frac{N}{2^{14}} = 0.0657703 \pm 2 \cdot 10^{-7} \tag{1}$$

(Ziff et al., PRL, **79**, 18). The one sigma error on Λ is approximately $\sqrt{N}/2^{14}$.

```
    lamboot -v
mpirun -v /.../ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS1001.sche
```

```
#
# lam boot schema for trivial shared object
#
h -np 4 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-3)" -dynlib="/ldcg/lib/lalwrapper/libldastfclusters.so" -
dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(32768,32768,1,256,0,0.5,-1e-3,1,0,128,1)" -realTimeRatio=0.9 -
doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/W1001.ilwd"
```

1. There was no error messages in running the commands above. **Pass**
2. File `output_1.ildw` was created. **Pass**
3. The measured Λ is within 1σ from the accepted value. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.12 TFCLUSTERS1101

Test Case: TFCLUSTERS1101

Purpose: Verify the validity of clusters generation with white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 12:44

ENVIRONMENT AND PREREQUISITES

This test is executed through LDAS with the dataPipeline command. The output data are extracted from the database with Guild, and fed into a Matlab script.

PROCEDURE

The test should run on the frame file `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/W512_256s.F` which contains a REAL4 timeseries of length 2^{17}, sampled at 512 Hz, and containing white noise of unit variance. There should be no error messages in running the commands below. The histogram of the output cluster sizes should match the analytical predictions, as is to be verified using a matlab script that runs a χ^2 test at the 95% confidence level.

First run the following under LDAS; note the job number:
`/ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS1101.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.psu.edu }
  {
   dataPipeline
      -np 3
      -dynlib /ldcg/lib/lalwrapper/libldastfclusters.so
      -filterparams (131072,131072,1,512,0,0.1,-1,1,0,256,1)
      -subject TFCLUSTERS1101
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { TFCLUSTERS1101 }
      -resultname { TFCLUSTERS1101 }
      -aliases x=_ch0
      -algorithms {m=mean(x);
                   y=sub(x,m);
                   }
      -frames { /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/W512_256s.F}
      -framequery Adc(0)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

Then use Guild to generate a file with the cluster amplitudes:

- Select “LDAS metadata database” from the main menu, then “Single-ifo events”, then “Burst candidates”.
- In the window that pops up, in the file “Columns”, check “Selected”, and enter “AMPLITUDE” in the text field.
- In the field “Maximum number of records to fetch”, remove the default number, let the field blank.
- In the section “Qualifiers”, check the “LDAS job number” field, and enter the number of the job under LDAS, from above.

- Submit by clicking the “Refresh & Submit” button.
- In the window that pops up, click the “Save as...” button.
- In the window that pops up, select the check buttons “Save displayed data as text”, “Tab-delimited” in section “Text formatting”, “Do not include” in sections “Column names”, “Row numbers” and “Truncation marks”.
- Click the “Continue to filename selection” button, and save the file in `/ldasmdc/burst-stochastic/test/tfclusters/output` as `TFCLUSTERS1101.txt`

Finally, open Matlab, and do the following:

- “addpath(’/ldasmdc/burst-stochastic/test/tfclusters/src’)”. Make sure you give the full path, e.g. you may have to add “/home/julien” before the path above.
- “tfch(’/ldasmdc/burst-stochastic/test/tfclusters/output/TFCLUSTERS1101.txt’,131072,0.1,[0])”. Make sure you give the full path, e.g. you may have to add “/home/julien” before the path above.

1. There was no error messages in running the commands above. **Pass**
2. 5293 events were written in the database. **Pass**
3. The output of the matlab script was “Measured distribution matches expectations at the 95% level.”. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

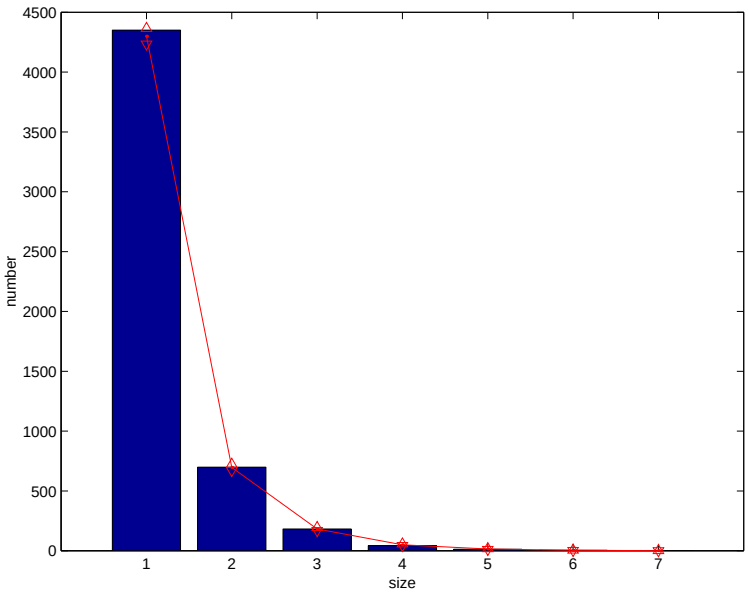

Figure 3: Histogram output from TFCLUSTERS1101. Blue are data, red are expectations. Red triangles are estimates of the $1-\sigma$ error.

A.8.13 TFCLUSTERS1201

Test Case: TFCLUSTERS1201

Purpose: Verify the validity of the statistical test involving the First Threshold in Clustering Analysis, with white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 12:54

ENVIRONMENT AND PREREQUISITES

This test is executed through LDAS with the dataPipeline command. The output data are extracted from the database with Guild, and fed into a Matlab script.

PROCEDURE

The test should run on the frame file `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/S683237311-1-8192.F` which contains a REAL4 timeseries of length 2^{21}, sampled at 256 Hz, and containing white noise of unit variance. There should be no error messages in running the commands below. The histogram of the output cluster sizes should match the analytical predictions, as is to be verified using a matlab script. Note that due to the fact that we are only looking at the tail on an histogram that is exponentially decaying, each bin is far from being gaussian. Hence, the standard χ^2 test is not applicable; the matlab script to be used below simply asks that $\chi^2/\mathrm{DOF} < 20$, where DOF is the number of bins with at least 30 events.

First run the following under LDAS; note the job number:
`/ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS1201.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.psu.edu }
  {
   dataPipeline
      -np 10
      -dynlib /ldcg/lib/lalwrapper/libldastfclusters.so
      -filterparams (2097152,263936,1,256,2048,0.1,-1,1,0,128,5,0,0,0,0,0,0,0,0,0,0)
      -subject TFCLUSTERS1201
      -datacondtarget mpi
      -mddapi ligolw
      -resultname { TFCLUSTERS1201 result }
      -aliases x=_ch0
      -algorithms {m=mean(x);
                   y=sub(x,m);
                   }
      -frames { /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/S683237311-1-8192.F}
      -framequery Adc(0)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

Then use Guild to generate a file with the cluster amplitudes:

- Select “LDAS metadata database” from the main menu, then “Single-ifo events”, then “Burst candidates”.
- In the window that pops up, in the file “Columns”, check “Selected”, and enter “AMPLITUDE” in the text field.
- In the field “Maximum number of records to fetch”, remove the default number, let the field blank.

- In the section “Qualifiers”, check the “LDAS job number” field, and enter the number of the job under LDAS, from above.
- Submit by clicking the “Refresh & Submit” button.
- In the window that pops up, click the “Save as...” button.
- In the window that pops up, select the check buttons “Save displayed data as text”, “Tab-delimited” in section “Text formatting”, “Do not include” in sections “Column names”, “Row numbers” and “Truncation marks”.
- Click the “Continue to filename selection” button, and save the file in `/ldasmdc/burst-stochastic/test/tfclusters/output` as `TFCLUSTERS1201.txt`

Finally, open Matlab, and do the following:

- “addpath(’/ldasmdc/burst-stochastic/test/tfclusters/src’)”. Make sure you give the full path, e.g. you may have to add “/home/julien” before the path above.
- “**tfcapp**(’/ldasmdc/burst-stochastic/test/tfclusters/output/TFCLUSTERS1201.txt’,2097152,0.1,[0,0,0,0,0,0,0,0,0,0])”. Make sure you give the full path, e.g. you may have to add “/home/julien” before the path above.

1. There was no error messages in running the commands above. **Pass**
2. 327 events were written in the database. **Pass**
3. The output of the matlab script was “Measured distribution matches expectations (reduced chi-square ¡ 20).”. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

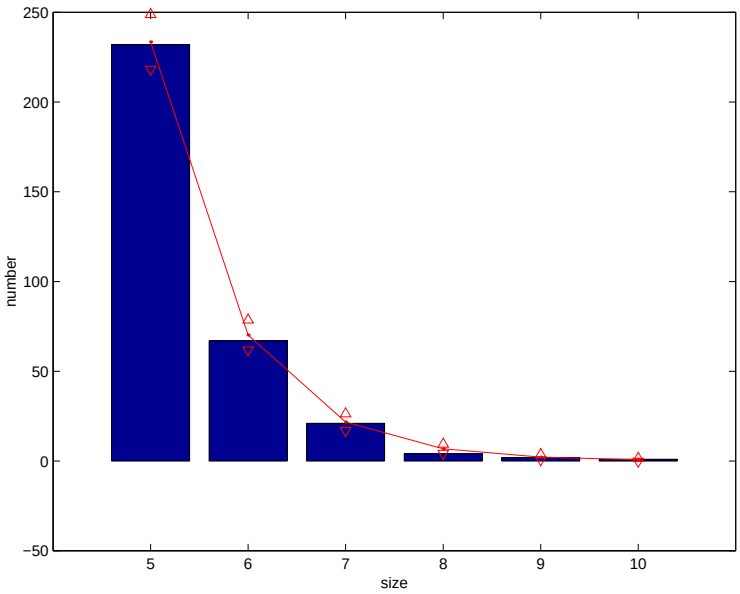

Figure 4: Histogram output from TFCLUSTERS1201. Blue are data, red are expectations. Red triangles are estimates of the $1-\sigma$ error.

A.8.14 TFCLUSTERS1202

Test Case: TFCLUSTERS1202

Purpose: Verify the validity of the statistical test involving the First and Second Thresholds in Clustering Analysis, with white noise

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 13:04

ENVIRONMENT AND PREREQUISITES

This test is executed through LDAS with the dataPipeline command. The output data are extracted from the database with Guild, and fed into a Matlab script.

PROCEDURE

The test should run on the frame file `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/W683403631-1-8192.F` which contains a REAL4 timeseries of length 2^{21}, sampled at 256 Hz, and containing white noise of unit variance. There should be no error messages in running the commands below. The histogram of the output cluster sizes should match the analytical predictions, as is to be verified using a matlab script. Note that due to the fact that we are only looking at the tail on an histogram that is exponentially decaying, each bin is far from being gaussian. Hence, the standard χ^2 test is not applicable; the matlab script to be used below simply asks that $\chi^2/\mathrm{DOF} < 20$, where DOF is the number of bins with at least 30 events.

First run the following under LDAS; note the job number:
`/ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS1202.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email mdc@gravity.phys.psu.edu }
  {
   dataPipeline
      -np 10
      -dynlib /ldcg/lib/lalwrapper/libldastfclusters.so
      -filterparams (2097152,263936,1,256,2048,0.1,-1,1,0,128,5,0,0,0,0,0,0,0,2,2,3)
      -subject TFCLUSTERS1202
      -datacondtarget mpi
      -mddapi ligolw
      -resultname { TFCLUSTERS1202 result }
      -aliases x=_ch0
      -algorithms {m=mean(x);
                   y=sub(x,m);
                   }
      -frames { /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/W683403631-1-8192.F}
      -framequery Adc(0)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

Then use Guild to generate a file with the cluster amplitudes:

- Select “LDAS metadata database” from the main menu, then “Single-ifo events”, then “Burst candidates”.
- In the window that pops up, in the file “Columns”, check “Selected”, and enter “AMPLITUDE” in the text field.

- In the field “Maximum number of records to fetch”, remove the default number, let the field blank.
- In the section “Qualifiers”, check the “LDAS job number” field, and enter the number of the job under LDAS, from above.
- Submit by clicking the “Refresh & Submit” button.
- In the window that pops up, click the “Save as...” button.
- In the window that pops up, select the check buttons “Save displayed data as text”, “Tab-delimited” in section “Text formatting”, “Do not include” in sections “Column names”, “Row numbers” and “Truncation marks”.
- Click the “Continue to filename selection” button, and save the file in `/ldasmdc/burst-stochastic/test/tfclusters/output` as `TFCLUSTERS1202.txt`

Finally, open Matlab, and do the following:

- “addpath(’/ldasmdc/burst-stochastic/test/tfclusters/src’)”. Make sure you give the full path, e.g. you may have to add “/home/julien” before the path above.
- “**tfcapp**(’/ldasmdc/burst-stochastic/test/tfclusters/output/TFCLUSTERS1202.txt’,2097152,0.1,[0,0,0,0,0,0,0,**2,2,3**])”. Make sure you give the full path, e.g. you may have to add “/home/julien” before the path above.

1. There was no error messages in running the commands above. **Pass**
2. 460 events were written in the database. **Pass**
3. The output of the matlab script was “Measured distribution matches expectations (reduced chi-square ¡ 20).”. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

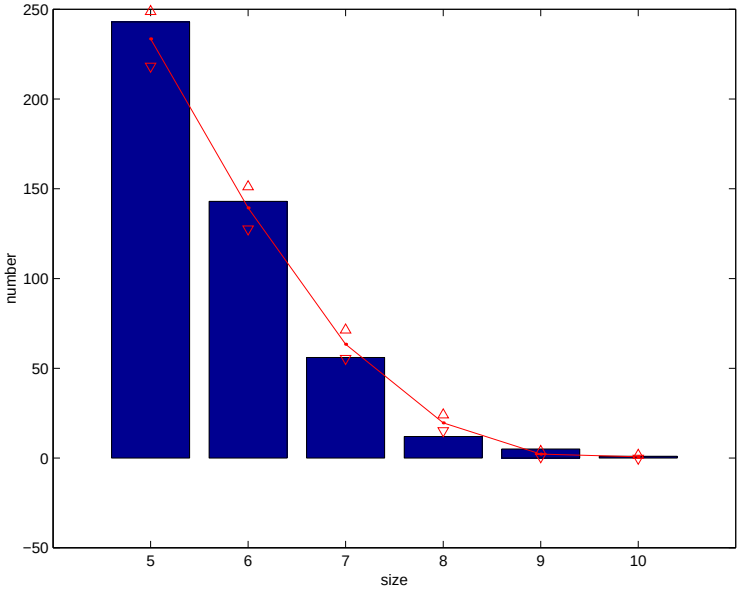

Figure 5: Histogram output from TFCLUSTERS1202. Blue are data, red are expectations. Red triangles are estimates of the $1-\sigma$ error.

A.8.15 TFCLUSTERS1301

Test Case: TFCLUSTERS1301

Purpose: Verify that the threshold to be used in TFCLUSTERS1302 is low enough that false alarms are unlikely

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 13:10

ENVIRONMENT AND PREREQUISITES

This test is executed through LDAS with the dataPipeline command.

PROCEDURE

The test should run on the frame file `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_512.F` which contains 512s of white noise of unit variance at 16384Hz. There should be no error messages in running the commands below. There shouldn't be any event written in the database (the probability to have at least one false alarm is 0.0718).

`/ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS1301.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email julien@ligo.mit.edu }
  {
   dataPipeline
      -np 10
      -dynlib /ldcg/lib/lalwrapper/libldastfclusters.so
      -filterparams (8388608,1277952,0.03125,16384,262144,0.0213,-1,1,0,8192,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
      -subject TFCLUSTERS1301
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { TFCLUSTERS1301 }
      -resultname { TFCLUSTERS1301 }
      -aliases x=_ch0
      -algorithms {m=mean(x);
                   y=sub(x,m);
                   }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_512.F
      -framequery Adc(0)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

1. There was no error messages in running the commands above. **Pass**
2. No event were written in the database. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.16 TFCLUSTERS1302

Test Case: TFCLUSTERS1302

Purpose: Verify that obvious bursts can be detected

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 13:20

ENVIRONMENT AND PREREQUISITES

This test is executed through LDAS with the dataPipeline command.

PROCEDURE

The test should run on the frame file `/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZMbig_512.F` which contains 512s of white noise of unit variance at 16384Hz, and a large, short burst. There should be no error messages in running the commands below. There should be a few events written in the database.

`/ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS1302.tclsh`

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email julien@ligo.mit.edu }
  {
   dataPipeline
      -np 10
      -dynlib /ldcg/lib/lalwrapper/libldastfclusters.so
      -filterparams (8388608,1277952,0.03125,16384,262144,0.0213,-1,1,0,8192,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
      -subject TFCLUSTERS1302
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { TFCLUSTERS1302 }
      -resultname { TFCLUSTERS1302 }
      -aliases x=_ch0
      -algorithms {m=mean(x);
                   y=sub(x,m);
                   }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_ZMbig_512.F
      -framequery Adc(0)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

1. There was no error messages in running the commands above. **Pass**
2. Some events were written in the database. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.8.17 TFCLUSTERS1401

Test Case: TFCLUSTERS1401

Purpose: Generate events from a seismometer channel for a comparison with the DMT monitor TID

Tester: Julien Sylvestre

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/09/01 **Time:** 13:24

ENVIRONMENT AND PREREQUISITES

This test is executed through LDAS with the dataPipeline command.

PROCEDURE

The test should run on the RDS frame files from the E5 engineering run, between times 681144558 and 681147558. There should be no error messages in running the commands below. There should be a events written in the database.

First, run the following script, taking note of the LDAS job number:
`/ldasmdc/burst-stochastic/test/tfclusters/command/TFCLUSTERS1401.tclsh`

```
#!/ldcg/bin/tclsh

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user "ldas_mdc"
set pwrd "beowulf"
set email "mdc@gravity.phys.psu.edu"
set host ldas.mit.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

#set stime 680938500  ;## E5 full frames, locked
set stime 681144558  ;## E5 reduced frames
set duration 3000
set channel H0:PEM-MX_SEISX

# Derived variables

set etime [ expr $stime + $duration - 1 ]
set times $stime-$etime
set ch0alias [ mkchannel $channel $stime ]

set cmd "
  ldasJob { -name $user -password $pwrd -email $email } {
    dataPipeline
      -np 10
      -dynlib /ldcg/lib/lalwrapper/libldastfclusters.so
      -filterparams (768000,124672,1,256,32768,0.1,1e-2,1,0,128,5,0,0,0,0,0,0,0,2,2,3)
```

```
      -subject TFCLUSTERS1301
      -datacondtarget mpi
      -mddapi ligolw
      -resultname { TFCLUSTERS1301 result }
      -aliases { x = $ch0alias; }
      -algorithms {
        m = mean(x);
        y = sub(x, m);
      }
      -framequery {
        { H {} $times Adc($channel) }
      }
  }"

sendCmd $cmd $host $port

exit
```

Then use Guild to generate a file with information about the clusters:

- Select “LDAS metadata database” from the main menu, then “Single-ifo ev ents”, then “Burst candidates”.
- In the window that pops up, in the file “Columns”, check “Selected”, a nd enter “START_TIME, DURATION, CENTRAL_FREQ, BANDWIDTH, AMPLITUDE” in the tex t field.
- In the field “Maximum number of records to fetch”, remove the default nu mber, let the field blank.
- In the section “Qualifiers”, check the “LDAS job number” field, and en ter the number of the job under LDAS, from above.
- Submit by clicking the “Refresh & Submit” button.
- In the window that pops up, click the “Save as...” button.
- In the window that pops up, select the check buttons “Save displayed data as text”, “Tab-delimited” in section “Text formatting”, “Do not include” in sections “Column names”, “Row numbers” and “Truncation marks”.
- Click the “Continue to filename selection” button, and save the file in `/ldasmdc/burst-stochastic/test/tfclusters/output` as `TFCLUSTERS1401.txt`

1. There was no error messages in running the commands above. **Pass**
2. Some events were written in the database. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9 Stochastic IFO-IFO correlation DSO standalone tests

The tests in this section use the standalone wrapper API to test LDAS shared object `libldasstochastic.so`, which calculates the frequency-domain integrand of the optimally-filtered cross-correlation statistic.

There are two types of tests that are performed: the first test that error checking for input parameters is correct, the second check that the shared object calculates the correct values for valid parameters.

The naming convention for error checking tests is `STOCHASTICIFOIFOSOERR??` where `??` is a two digit number. Error checking tests are ordered in the same way as the error checks themselves.

The naming convention for validation tests `STOCHASTICIFOIFOSO00000` *et seq.* is as follows:

- The first digit (0/1) indicates the length of the time series as well as the frequency series sampling parameters.
- The second digit (0/1) indicates whether the data are trivial (a 1 at the start of each segment followed by a string of 0s) or simulated colored noise with the appropriate power spectral density (PSD).
- The third digit (0/1) indicates whether or not the data contain a simulated stochastic background.
- The fourth digit (0/1/2/3) labels the choice of response function and noise PSD.
- The fifth digit (0/1) indicates whether the simulation and analysis are performed for coïncident and coäligned detectors, or for correlations between LIGO Hanford and LIGO Livingston.

The `STOCHASTICIFOIFOSO#####` tests run in the stand-alone wrapperAPI and require a complete installation of LAL, LALWrapper, and the standalone wrapperAPI.

They operate on ILWDs created from the following datasets (which were created by the matlab code in the `src` directory):

- `src/dmro_trivial.10x10238.1024.dat` Trivial time series data consisting of 10 identical segments of $10238 = (3^2 * 5^2 * 7 * 13 + 1)/2$ points each; each segment has a 1 in the first element, with 0s in the rest; the dataset contains 102380 points in all, representing 99 seconds, 980468750 nanoseconds of data sampled at 1024 Hz.
- `src/dmro1_white.10x10238.1024.dat` and `src/dmro2_white.10x10238.1024.dat` Time series data consisting of white gaussian noise; 102380 points, representing 99 seconds, 980468750 nanoseconds of data sampled at 1024 Hz. These are two independent random data streams with the same statistical properties.
- `src/dmro1_offwhite.10x10238.1024.dat` and `src/dmro2_offwhite.10x10238.1024.dat` Time series data consisting of colored gaussian noise, whose PSD is constructed by multiplying the analytic fit to the LIGO-I design sensitivity given in [5] by the square of the frequency-domain response function from the E2 run[2]. (This PSD is the same as the one contained in `src/spec_offwhite.-512-512.2.dat`, although it was sampled with a different frequency resolution when using it to construct the FFT of this time series data.) The dataset contains 102380 points, representing 99 seconds, 980468750 nanoseconds of data sampled at 1024 Hz. These are two independent random data streams with the same statistical properties.
- `src/dmro1_offplusomega.10x10238.1024.dat` and `src/dmro2_offplusomega.10x10238.1024.dat` Time series data consisting of the colored gaussian noise from `src/dmro1_offwhite.10x10238.1024.dat` and `src/dmro2_offwhite.10x10238.1024.dat` plus a simulated stochastic background signal

with a constant $\Omega_{\rm GW}(f) = 10^{-3}$, as recorded at LHO and LLO, convolved with the E2 response function. Each dataset contains 102380 points, representing 99 seconds, 980468750 nanoseconds of data sampled at 1024 Hz.

- `src/dmro1_offplusomega.10x92138.1024.dat` and `src/dmro2_offplusomega.10x92138.1024.dat` Time series consisting of colored gaussian noise plus stochastic signal as in `src/dmro1_offplusomega.10x10238.1024.dat` and `src/dmro2_offplusomega.10x10238.1024.dat` , but containing $921380 = 10 * (3^4 * 5^2 * 7^1 * 13 + 1)/2$ points representing 899 seconds, 785156250 nanoseconds sampled at 1024 Hz.

- `src/spec_white.-512-512.2.dat` A *two-sided* trivial power spectral density consisting of all 1s, containing 512 points with a start frequency of -512 Hz and a frequency spacing of 2 Hz.

- `src/spec_offwhite.-512-512.2.dat` A *two-sided* power spectral density describing the whitened output of a detector whose unwhitened PSD is given by the analytic approximation to the LIGO-I design sensitivity from [5] and whose response function is the one obtained from the E2 run [2]. The series contains 512 points with a start frequency of -512 Hz and a frequency spacing of 2 Hz.

- `src/spec_offwhite.-512-512.1_4.dat` A *two-sided* power spectral density describing the whitened output of a detector whose unwhitened PSD is given by the analytic approximation to the LIGO-I design sensitivity from [5] and whose response function is the one obtained from the E2 run[2]. The series contains 4096 points with a start frequency of -512 Hz and a frequency spacing of 0.25 Hz.

- `src/resp_ident.40-500.2.dat` A complex frequency series containing a trivial response function, with every element having a real part of 1 and an imaginary part of 0, containing 230 points with a start frequency pf 40 Hz and a frequency spacing of 2 Hz.

- `src/resp_ideal.40-500.2.dat` A complex frequency series containing an idealized response function whose value at every frequency is one over the square root of the PSD given by the analytic approximation to the LIGO1 design sensitivity from [5]. The series contains 230 points with a start frequency of 40 Hz and a frequency spacing of 2 Hz.

- `src/resp_cmplx.40-500.2.dat` A complex frequency series containing the response function for the E2 run as desecribed in [2] The series contains 230 points with a start frequency of 40 Hz and a frequency spacing of 2 Hz.

- `src/resp_cmplx.40-500.1_4.dat` A complex frequency series containing the response function for the E2 run as described in [2]. The series contains 1840 points with a start frequency of 40 Hz and a frequency spacing of 0.25 Hz.

The units are ADC counts for all `dmro` datasets, $(\texttt{ADC counts})^2/\texttt{Hz}$ for all `spec` datasets, and $\texttt{ADC counts}/(10^{-18}\,\texttt{strain})$ for all `resp` datasets.

Each `STOCHASTICIFOIFOSO#####` test is run by the script `command/runtest.tclsh` which has the following source:

```
#!/ldcg/bin/tclsh
# $Id: runtest.tclsh,v 1.1 2001/09/01 00:47:18 whelan Exp $
# Script to run pipeline tests

set test [lindex $argv 0]

exec mpirun -v $test.schema  >@ stdout

foreach i {1 2} {
```

```
    set newname $test
    append newname _process_$i
    file rename -force process_$i.ilwd ../output/$newname.ilwd
}

for {set i 1} {$i <= 10} {incr i} {
    set newname $test
    append newname _output_$i
    file rename -force output_$i.ilwd ../output/$newname.ilwd
}
```

The data output by the scripts are compared to the expected outputs by the script `command/datadiff.tclsh` which has the following source:

```
#!/ldcg/bin/tclsh
# $Id: datadiff.tclsh,v 1.11 2001/09/08 14:41:08 whelan Exp $
# Script to compare ILWD outputs to expected output

set test [lindex $argv 0]

set verbose [lindex $argv 1]

set ilwdfile [open ../output/$test.ilwd]

set ilwd [read $ilwdfile]

close $ilwdfile

regexp {\<complex_8[^\>]*\>([^\<]+)\<} $ilwd -> framedata
regexp {\<char_u[^\>]+\>((\\\d{3})+)?\</char_u\>} $ilwd -> binarydata
binary scan [subst $binarydata] f* blobdata

set framearray [split $framedata]
set blobarray [split $blobdata]

set expfile [open ../expected/$test.dat]

set exparrayreal [list]
set exparrayimag [list]

while {[gets $expfile line] >= 0} {
    if {[scan $line "%f %f" real imag] != 2} {
  puts "error reading"
return 1
    }
    lappend exparrayreal $real
    lappend exparrayimag $imag
}

close $expfile

if {[llength $framearray] != [llength $blobarray]} {
  puts "error reading"
return 1
}

set num [ expr [llength $framearray] / 2 ]

if {$num != [llength $exparrayreal]} {
  puts "error reading"
return 1
}

if {$num != [llength $exparrayimag]} {
  puts "error reading"
return 1
}

set realframediff -1
set imagframediff -1
set realblobdiff -1
set imagblobdiff -1

set realframetot 0
set imagframetot 0
set realblobtot 0
set imagblobtot 0
set realexptot 0
set imagexptot 0

for {set i 0} {$i < $num} {incr i} {
    set realframe [lindex $framearray [expr 2 * $i] ]
    set imagframe [lindex $framearray [expr 2 * $i + 1] ]
    set realblob [lindex $blobarray [expr 2 * $i] ]
    set imagblob [lindex $blobarray [expr 2 * $i + 1] ]
    set realexp [lindex $exparrayreal $i]
    set imagexp [lindex $exparrayimag $i]

    set realexptot [expr $realexptot + $realexp]
    set imagexptot [expr $imagexptot + $imagexp]
```

```
    set realblobtot [expr $realblobtot + $realblob]
    set imagblobtot [expr $imagblobtot + $imagblob]

    set realframetot [expr $realframetot + $realframe]
    set imagframetot [expr $imagframetot + $imagframe]

# Compare data in frame output to data in expected output
    set diff [expr abs($realframe - $realexp)]
    if {$diff > $realframediff} {
set realframediff $diff
set realframeind $i
    }
    if {$verbose != ""} {
puts "$i exp $realexp frame $realframe diff [expr $realframe - $realexp]"
    }
    set diff [expr abs($imagframe - $imagexp)]
    if {$diff > $imagframediff} {
set imagframediff $diff
set imagframeind $i
    }
# Compare data in blob output to data in expected output
    set diff [expr abs($realblob - $realexp)]
    if {$diff > $realblobdiff} {
set realblobdiff $diff
set realblobind $i
    }
    if {$verbose != ""} {
puts "$i exp $realexp blob $realblob diff [expr $realblob - $realexp]"
    }
    set diff [expr abs($imagblob - $imagexp)]
    if {$diff > $imagblobdiff} {
set imagblobdiff $diff
set imagblobind $i
    }

}
puts "Compared $num entries"
puts "Expected total $realexptot + i $imagexptot"
puts "Actual total in frame output is $realframetot + i $imagframetot"
puts "Maximum error in real frame output is $realframediff at $realframeind"
puts "Maximum error in imag frame output is $imagframediff at $imagframeind"
puts "Actual total in database output is $realblobtot + i $imagblobtot"
puts "Maximum error in real database output is $realblobdiff at $realblobind"
puts "Maximum error in imag database output is $imagblobdiff at $imagblobind"
puts "================================="
puts "Fractional error in database total is [expr ($realblobtot - $realexptot)/abs($realexptot)]\
+ i [expr ($imagblobtot - $imagexptot)/abs($realexptot)]"
puts "(max real database err) / (real total) = [expr $realblobdiff / abs($realexptot)]"
puts "(max imag database err) / (real total) = [expr $imagblobdiff / abs($realexptot)]"
puts "================================="
puts "Fractional error in frame total is [expr ($realframetot - $realexptot)/abs($realexptot)]\
+ i [expr ($imagframetot - $imagexptot)/abs($realexptot)]"
puts "(max real frame err) / (real total) = [expr $realframediff / abs($realexptot)]"
puts "(max imag frame err) / (real total) = [expr $imagframediff / abs($realexptot)]"
```

The two formats of data (summary spectra for the database and multidim data for a proc frame) output by the scripts are compared to one another by the script `command/datacomp.tclsh` which has the following source:

```
#!/ldcg/bin/tclsh
# $Id: datacomp.tclsh,v 1.2 2001/09/08 14:41:08 whelan Exp $
# Script to compare ILWD outputs to each other

set test [lindex $argv 0]

set verbose [lindex $argv 1]

set ilwdfile [open ../output/$test.ilwd]

set ilwd [read $ilwdfile]

close $ilwdfile

regexp {\<complex_8[^\>]*\>([^\<]+)\<} $ilwd -> framedata
regexp {\<char_u[^\>]+\>((\\\d{3})+)?\</char_u\>} $ilwd -> binarydata
binary scan [subst $binarydata] f* blobdata

set framearray [split $framedata]
set blobarray [split $blobdata]

set num [ expr [llength $framearray] / 2 ]

set realdiff -1
set imagdiff -1

set realframetot 0
set imagframetot 0
set realblobtot 0
set imagblobtot 0
```

```
for {set i 0} {$i < $num} {incr i} {
    set realframe [lindex $framearray [expr 2 * $i] ]
    set imagframe [lindex $framearray [expr 2 * $i + 1] ]
    set realblob [lindex $blobarray [expr 2 * $i] ]
    set imagblob [lindex $blobarray [expr 2 * $i + 1] ]

    set realblobtot [expr $realblobtot + $realblob]
    set imagblobtot [expr $imagblobtot + $imagblob]

    set realframetot [expr $realframetot + $realframe]
    set imagframetot [expr $imagframetot + $imagframe]

# Compare data in frame output to data in blob output
    set diff [expr abs($realframe - $realblob)]
    if {$diff > $realdiff} {
set realdiff $diff
set realind $i
    }
    if {$verbose != ""} {
puts "$i blob $realblob frame $realframe diff [expr $realframe - $realblob]"
    }
    set diff [expr abs($imagframe - $imagblob)]
    if {$diff > $imagdiff} {
set imagdiff $diff
set imagind $i
    }
}
puts "Compared $num entries"
puts "Actual total in frame output is $realframetot + i $imagframetot"
puts "Actual total in database output is $realblobtot + i $imagblobtot"
puts "Maximum discrepancy in real output is $realdiff at $realind"
puts "Maximum discrepancy in imag output is $imagdiff at $imagind"
puts "================================="
puts "Fractional discrepancy in total is [expr ($realframetot - $realblobtot)/abs($realframetot)]\
+ i [expr ($imagblobtot - $imagframetot)/abs($realframetot)]"
puts "(max real disc) / (real total) = [expr $realdiff / abs($realframetot)]"
puts "(max imag disc) / (real total) = [expr $imagdiff / abs($realframetot)]"
```

Note that both of these scripts decode the binary data destined for the database assuming it was generated on a machine with the same architecture (little-endian or big-endian) as the wrapperAPI job which generated them was originally run on, so it’s important to run these checks all on the same machine.

A.9.1 STOCHASTICIFOIFOSOERR

Test Case: STOCHASTICIFOIFOSOERR

Purpose: Test that libldasstochastic.so correctly rejects improper search parameters in the standalone wrapperAPI environment.

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 09:10 EDT

This test checks that improper input parameters to the stochastic IFO-IFO shared object raise signals that terminate execution properly and generate the correct error messages.

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on ldas-pcdev1.mit.edu as user ldas_mdc with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

This test is executed via schema files with a standalone wrapperAPI running the stochastic shared object. There are 13 sub-tests requiring the following 13 schema files:

```
STOCHASTICIFOIFOSOERR00.schema
STOCHASTICIFOIFOSOERR01.schema
STOCHASTICIFOIFOSOERR02.schema
STOCHASTICIFOIFOSOERR03.schema
STOCHASTICIFOIFOSOERR04.schema
STOCHASTICIFOIFOSOERR05.schema
STOCHASTICIFOIFOSOERR06.schema
STOCHASTICIFOIFOSOERR07.schema
STOCHASTICIFOIFOSOERR08.schema
STOCHASTICIFOIFOSOERR09.schema
STOCHASTICIFOIFOSOERR10.schema
STOCHASTICIFOIFOSOERR11.schema
STOCHASTICIFOIFOSOERR12.schema
```

A single input ilwd file is required, although the contents should be irrelevant since the shared object is not expected to carry out any analysis. For this purpose, all tests will use the input file:

```
STOCHASTICIFOIFOSOERR.ilwd
```

LAM must be running; if it's not issue the command

```
lamboot -v
```

The test are written to be performed in a bash shell.

This test depends on the successful completion all test checklists in Sec. A.1.

PROCEDURE

With LAM already running (lamboot -v), change directory to:

```
/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/
```

and perform the following tests:

1. Test that the shared object correctly rejects input with the wrong number of arguments by executing

the command:

```
mpirun -v STOCHASTICIFOIFOSOERR00.schema >& \
../output/STOCHASTICIFOIFOSOERR00.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR00.schema,v 1.3 2001/09/07 22:03:18 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1)"
```

The resulting output can be viewed with the command:
`less /../output/STOCHASTICIFOIFOSOERR00.txt`

The test will have passed if this file contains the phrase:
`ABORT: Wrong number of arguments`.

Pass

2. Test that the shared object correctly rejects input schema files that do not have “-filterparams” as the first argument by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR01.schema >& \
../output/STOCHASTICIFOIFOSOERR01.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR01.schema,v 1.3 2001/09/07 22:03:18 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
```

```
# argv[4]  = f0               Start frequency for optimal filter
# argv[5]  = deltaF           Frequency spacing for optimal filter
# argv[6]  = outputNumFreq    Number of frequencies in CC spectrum
# argv[7]  = outputF0         Start frequency for CC spectrum
# argv[8]  = outputDeltaF     Frequency spacing for CC spectrum
# argv[9]  = dmro1name        Name of first data channel
# argv[10] = dmro2name        Name of second data channel
# argv[11] = response1name    Name of first response fcn
# argv[12] = response2name    Name of second response fcn
# argv[13] = epochsMatch      Check that time streams are simultaneous
# argv[14] = siteID1          Index of first detector in lalCachedDetectors
# argv[15] = siteID2          Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name    Name of first PSD
# argv[17] = spectrum2name    Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparam="(32,32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparam="(32,32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:
`less /../output/STOCHASTICIFOIFOSOERR01.txt`

The test will have passed if this file contains the phrase:
`Unknown command line argument: -filterparam`.

Pass

3. Test that the shared object correctly rejects input schema files that do not specify a positive number of segments to process by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR02.schema >& \
../output/STOCHASTICIFOIFOSOERR02.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR02.schema,v 1.3 2001/09/07 22:03:18 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments      Number of data segments
# argv[2]  = numPoints        Number of points in a data segment
# argv[3]  = numFrequencies   Number of frequencies in optimal filter
# argv[4]  = f0               Start frequency for optimal filter
# argv[5]  = deltaF           Frequency spacing for optimal filter
# argv[6]  = outputNumFreq    Number of frequencies in CC spectrum
# argv[7]  = outputF0         Start frequency for CC spectrum
# argv[8]  = outputDeltaF     Frequency spacing for CC spectrum
# argv[9]  = dmro1name        Name of first data channel
# argv[10] = dmro2name        Name of second data channel
# argv[11] = response1name    Name of first response fcn
# argv[12] = response2name    Name of second response fcn
# argv[13] = epochsMatch      Check that time streams are simultaneous
# argv[14] = siteID1          Index of first detector in lalCachedDetectors
# argv[15] = siteID2          Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name    Name of first PSD
# argv[17] = spectrum2name    Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(-32,32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
```

```
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(-32,32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:

`less /../output/STOCHASTICIFOIFOSOERR02.txt`

The test will have passed if this file contains the phrase:

`ABORT: Number of data segments is zero or negative.`

Pass

4. Test that the shared object correctly rejects input schema files that do not specify a positive number of data points to analyze by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR03.schema >& \
../output/STOCHASTICIFOIFOSOERR03.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR03.schema,v 1.3 2001/09/07 22:03:18 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,-32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,-32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:

`less /../output/STOCHASTICIFOIFOSOERR03.txt`

The test will have passed if this file contains the phrase:

`ABORT: Number of points in a data segment is zero or negative.`

Pass

5. Test that the shared object correctly rejects input schema files that do not specify that the optimal filter be calculated for at least one frequency by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR04.schema >& \
../output/STOCHASTICIFOIFOSOERR04.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR04.schema,v 1.3 2001/09/07 22:03:18 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,32,0,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,32,0,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:
`less /../output/STOCHASTICIFOIFOSOERR04.txt`
The test will have passed if this file contains the phrase:
`ABORT: Number of frequencies is zero or negative`.

Pass

6. Test that the shared object correctly rejects input schema files specify a negative starting frequency for the optimal filter by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR05.schema >& \
../output/STOCHASTICIFOIFOSOERR05.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR05.schema,v 1.3 2001/09/07 22:03:19 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
```

```
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,32,1,-32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,32,7,-32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:
`less /../output/STOCHASTICIFOIFOSOERR05.txt`
The test will have passed if this file contains the phrase:
`ABORT: Start frequency is negative`.

Pass

7. Test that the shared object correctly rejects input schema files that do not specify a positive frequency bin size for the optimal filter by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR06.schema >& \
../output/STOCHASTICIFOIFOSOERR06.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR06.schema,v 1.3 2001/09/07 22:03:19 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,32,1,32,-32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,32,7,32,-32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:
`less /../output/STOCHASTICIFOIFOSOERR06.txt`

The test will have passed if this file contains the phrase:
`ABORT: Frequency spacing is zero or negative.`

Pass

8. Test that the shared object correctly rejects input schema files that do not specify a positive number of frequencies for the cross-correlation spectral densities by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR07.schema >& \
../output/STOCHASTICIFOIFOSOERR07.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR07.schema,v 1.3 2001/09/07 22:03:19 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,32,1,32,32,-3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,32,7,32,32,-3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:
`less /../output/STOCHASTICIFOIFOSOERR07.txt`
The test will have passed if this file contains the phrase:
`ABORT: Number of frequencies is zero or negative.`

Pass

9. Test that the shared object correctly rejects input schema files that specify a negative starting frequency for the cross-correlation spectral densities by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR08.schema >& \
../output/STOCHASTICIFOIFOSOERR08.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR08.schema,v 1.3 2001/09/07 22:03:19 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
```

```
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,32,1,32,32,3,-80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,32,7,32,32,3,-80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:
`less /../output/STOCHASTICIFOIFOSOERR08.txt`
The test will have passed if this file contains the phrase:
`ABORT: Start frequency is negative`.

Pass

10. Test that the shared object correctly rejects input schema files that do not specify a positive frequency bin size for the cross-correlation spectral densities by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR09.schema >& \
../output/STOCHASTICIFOIFOSOERR09.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR09.schema,v 1.3 2001/09/07 22:03:19 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,32,1,32,32,3,80,-64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
```

```
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,32,7,32,32,3,80,-64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:

`less /../output/STOCHASTICIFOIFOSOERR09.txt`

The test will have passed if this file contains the phrase:

`ABORT: Frequency spacing is zero or negative`.

Pass

11. Test that the shared object correctly rejects input schema files that do not specify a valid detector number for the first detector by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR10.schema >& \
../output/STOCHASTICIFOIFOSOERR10.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR10.schema,v 1.3 2001/09/07 22:03:19 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,32,1,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,250,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,250,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:

`less /../output/STOCHASTICIFOIFOSOERR10.txt`

The test will have passed if this file contains the phrase:

`ABORT: Detector Site ID out of range`.

Pass

12. Test that the shared object correctly rejects input schema files that do not specify a valid detector number for the second detector by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR11.schema >& \
../output/STOCHASTICIFOIFOSOERR11.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR11.schema,v 1.3 2001/09/07 22:03:19 warren Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSOERR.ilwd

h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,32,1,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,-1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)" -realTimeRatio=0.9
#  -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0
#  -inputFile="../input/STOCHASTICIFOIFOSOERR.ilwd"
#  -filterparams="(32,32,7,32,32,3,80,64,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,-1,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:
`less /../output/STOCHASTICIFOIFOSOERR11.txt`
The test will have passed if this file contains the phrase:
`ABORT: Detector Site ID out of range`.

Pass

13. Test that the shared object correctly rejects input schema files that request an optimal filter with more frequency bins than the number of data points allows by executing the command:

```
mpirun -v STOCHASTICIFOIFOSOERR12.schema >& \
../output/STOCHASTICIFOIFOSOERR12.txt
```

This schema file contains:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: STOCHASTICIFOIFOSOERR12.schema,v 1.2 2001/09/07 18:45:33 whelan Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
```

```
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(32,8,17,0,32,16,0,32,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

The resulting output can be viewed with the command:

`less /../output/STOCHASTICIFOIFOSOERR12.txt`

The test will have passed if this file contains the phrase:

`ABORT:  Number of frequencies in optimal filter is greater than number of data elements`.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.2 STOCHASTICIFOIFOSO00000

Test Case: STOCHASTICIFOIFOSO00000

Purpose: Test liblasstochastic.so behavior for ∼ 100 sec of trivial data, filtered assuming white noise, delta-function response, coïncident & coäligned detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 9:59 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/STOCHASTICIFOIFOSO00000.ilwd`, which was constructed by the script `src/STOCHASTICIFOIFOSO00000.ilwd.sh` using
`src/dmro_trivial.10x10238.1024.dat` for both IFO channels,
`src/spec_white.-512-512.2.dat` for both whitened PSDs, and `src/resp_ident.40-500.2.dat` for both response functions, and compares its output to
`expected/STOCHASTICIFOIFOSO00000_output_1.dat`; both of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/`
   and execute the script
   `./runtest.tclsh STOCHASTICIFOIFOSO00000`
   It will run an MPI job with the following schema, from the file `STOCHASTICIFOIFOSO00000.schema` in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO00000
#
# $Id: STOCHASTICIFOIFOSO00000.schema,v 1.3 2001/09/07 18:45:33 whelan Exp $
#
# Invokes libstochastic.so with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = H2\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
# argv[13] = 1                Check that time streams are simultaneous
```

```
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 0                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
# argv[17] = spectrum2        Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO00000.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO00000.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO00000_`.

2. Examine the output files as follows:

   (a) Verify that the files all contain the same data and metadata aside from the start and end times and possibly the number of frames used by performing diffs among them and verifying that the only lines which differ are those containing the fields listed in the caption of table 2:

   ```
   diff ../output/STOCHASTICIFOIFOSO00000_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00000_output_2.ilwd
   diff ../output/STOCHASTICIFOIFOSO00000_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00000_output_3.ilwd
   ...
   diff ../output/STOCHASTICIFOIFOSO00000_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00000_output_10.ilwd
   ```

   Pass

   (b) Now verify that the correct metadata are stored for the output spectrum in `../output/STOCHASTICIFOIFOSO00000_output_1.ilwd \`:

   i. `start_frequency:summ_spectrum:dataBase` and `start_freq` are 40=4e+1 Hz
   ii. `stop_freq` is 498=4.98e+2 Hz
   iii. `delta_frequency:summ_spectrum:dataBase` and `freq:step_size` are 2 Hz
   iv. The size of the binary large object in the database entry `spectrum:summ_spectrum:dataBase` is $230*8 = 1840$ bytes (230 8-byte complex numbers).
   v. There are 230 points in the output frequency series: `filter_output::sequence`
   vi. The units of the container labelled `data` in the `filter_output::sequence` are $\mathrm{s}/\mathrm{Hz} = \mathrm{s}^2$
   vii. Check that the metadata in the files `../output/STOCHASTICIFOIFOSO00000_output_1.ilwd \` through `../output/STOCHASTICIFOIFOSO00000_output_10.ilwd \` agree with those listed in table 2.

   Pass

   (c) Next, verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another

Segment	Start sec	Start ns	Stop sec	Stop ns	Frames used
1	488811610	0	488811619	998046875	10
2	488811619	998046875	488811629	996093750	11
3	488811629	996093750	488811639	994140625	11
4	488811639	994140625	488811649	992187500	11
5	488811649	992187500	488811659	990234375	11
6	488811659	990234375	488811669	988281250	11
7	488811669	988281250	488811679	986328125	11
8	488811679	986328125	488811689	984375000	11
9	488811689	984375000	488811699	982421875	11
10	488811699	982421875	488811709	980468750	11

Table 2: Start and stop times for `../output/STOCHASTICIFOIFOSO00000_output_#.ilwd`; the fields to check are `start_time:summ_spectrum:dataBase`, `start_time_ns:summ_spectrum:dataBase`, `end_time:summ_spectrum:dataBase`, `end_time_ns:summ_spectrum:dataBase`, and `frames_used:summ_spectrum:dataBase` in the `database` section, and `gps_sec:start_time`, `gps_nan:start_time`, `gps_sec:stop_time`, and `gps_nan:stop_time` in the `filter_output::sequence` section.

Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
4.4e-1 + i 0.0	1.1e-9 + i 0.0	1.1e-9	0.0

Table 3: Discrepancies in `STOCHASTICIFOIFOSO00000_output_1`; fill in the columns to two significant figures.

by running the script
`./datacomp.tclsh STOCHASTICIFOIFOSO00000_output_1`
and filling out table 3; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) should be less than 10^{-6}.

Pass

(d) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
by running the script
`./datadiff.tclsh STOCHASTICIFOIFOSO00000_output_1`
and filling out table 4; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) should be less than 10^{-4}.

Pass

SUMMARY

Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
4.4e-1 + i 0.0	1.8e-7 + i 0.0	1.8e-8	0.0

Table 4: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO00000_output_1`; fill in the columns to two significant figures.

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.3 STOCHASTICIFOIFOSO00001

Test Case: STOCHASTICIFOIFOSO00001

Purpose: Test `libldasstochastic.so` behavior for $\sim$ 100 sec of trivial data, filtered assuming white noise, delta-function response, LHO & LLO detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 10:53 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test *also* uses the file `input/STOCHASTICIFOIFOSO00000.ilwd` (there is no file `input/STOCHASTICIFOIFOSO00001.ilwd`, since the data are the same for both tests), and compares its output to
`expected/STOCHASTICIFOIFOSO00001_output_1.dat`; both of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/`
   and execute the script
   `./runtest.tclsh STOCHASTICIFOIFOSO00001`
   It will run an MPI job with the following schema, from the file `STOCHASTICIFOIFOSO00001.schema` in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO00001
#
# $Id: STOCHASTICIFOIFOSO00001.schema,v 1.3 2001/09/07 18:45:33 whelan Exp $
#
# Invokes libstochastic.so with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = H2\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
# argv[13] = 1                Check that time streams are simultaneous
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 1                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
```

Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
-7.3e-1 + i 0.0	-6.9e-9 + i 0.0	6.0e-9	0.0

Table 5: Discrepancies in `STOCHASTICIFOIFOSO00001_output_1`; fill in the columns to two significant figures.

```
# argv[17] = spectrum2         Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO00000.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO00000.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
## The following commented-out lines would be used if we were bothering
## to duplicate input data with only the channel names changed
##### h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
##### -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
##### -dynlib-dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
##### -dynlib-realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W
##### -jobID=8 -dynlib-uniqueID=9.0
##### -inputFile="../input/STOCHASTICIFOIFOSO00001.ilwd"
##### -dynlib-filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO00001_`.

2. Examine the output files as follows:

   (a) Verify that the files all contain the same data and metadata aside from the start and end times and possibly the number of frames used by performing diffs among them and verifying that the only lines which differ are those containing the fields listed in the caption of table 2:

   ```
   diff ../output/STOCHASTICIFOIFOSO00001_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00001_output_2.ilwd
   diff ../output/STOCHASTICIFOIFOSO00001_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00001_output_3.ilwd
   ...
   diff ../output/STOCHASTICIFOIFOSO00001_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00001_output_10.ilwd
   ```

   Pass

   (b) Next, verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another
   by running the script
   `./datacomp.tclsh STOCHASTICIFOIFOSO00001_output_1`
   and filling out table 5; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) should be less than 10^{-6}.

   Pass

   (c) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in

Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
-7.3e-1 + i 0.0	-3.8e-7 + i 0.0	2.1e-7	0.0

Table 6: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO00001_output_1`; fill in the columns to two significant figures.

`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
by running the script
`./datadiff.tclsh STOCHASTICIFOIFOSO00001_output_1`
and filling out table 6; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) should be less than 10^{-4}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.4 STOCHASTICIFOIFOSO00011

Test Case: STOCHASTICIFOIFOSO00011

Purpose: Test libldasstochastic.so behavior for ∼ 100 sec of trivial data, filtered assuming LIGO-I noise, ideal response, LHO & LLO detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 11:00 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on ldas-pcdev1.mit.edu as user ldas_mdc with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
lamboot -v.

This test uses the file input/STOCHASTICIFOIFOSO00011.ilwd, which was constructed by the script src/STOCHASTICIFOIFOSO00011.ilwd.sh using
src/dmro_trivial.10x10238.1024.dat for both IFO channels,
src/spec_white.-512-512.2.dat for both whitened PSDs, and src/resp_ideal.40-500.2.dat for both response functions, and compares its output to
expected/STOCHASTICIFOIFOSO00011_output_1.dat; both of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
make data
from the directory
/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/.

PROCEDURE

1. Set the current directory to
   /.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/
   and execute the script
   ./runtest.tclsh STOCHASTICIFOIFOSO00011
   It will run an MPI job with the following schema, from the file STOCHASTICIFOIFOSO00011.schema in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO00011
#
# $Id: STOCHASTICIFOIFOSO00011.schema,v 1.2 2001/09/07 18:45:33 whelan Exp $
#
# Invokes libstochastic.so with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = L1\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
# argv[13] = 1                Check that time streams are simultaneous
```

Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
1.8e-9 + i 0.0	-8.0e-8 + i 0.0	2.6e-8	0.0

Table 7: Discrepancies in STOCHASTICIFOIFOSO00011_output_1; fill in the columns to two significant figures.

```
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 1                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
# argv[17] = spectrum2        Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO00011.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO00011.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO00011_`.

2. Examine the output files as follows:

   (a) Verify that the files all contain the same data and metadata aside from the start and end times and possibly the number of frames used by performing diffs among them and verifying that the only lines which differ are those containing the fields listed in the caption of table 2:

   ```
   diff ../output/STOCHASTICIFOIFOSO00011_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00011_output_2.ilwd
   diff ../output/STOCHASTICIFOIFOSO00011_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00011_output_3.ilwd
   ...
   diff ../output/STOCHASTICIFOIFOSO00011_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00011_output_10.ilwd
   ```

   Pass

   (b) Next, verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another
   by running the script
   `./datacomp.tclsh STOCHASTICIFOIFOSO00011_output_1`
   and filling out table 7; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) should be less than 10^{-6}.

   Pass

   (c) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
   by running the script
   `./datadiff.tclsh STOCHASTICIFOIFOSO00011_output_1`

Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
1.8e-9 + i 0.0	2.6e-5 + i 0.0	1.0 e-6	0.0

Table 8: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO00011_output_1`; fill in the columns to two significant figures.

and filling out table 8; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) should be less than 10^{-4}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.5 STOCHASTICIFOIFOSO00021

Test Case: STOCHASTICIFOIFOSO00021

Purpose: Test `libldasstochastic.so` behavior for $\sim$ 100 sec of trivial data, filtered assuming LIGO-I noise, E2 response, LHO & LLO detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 11:10 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/STOCHASTICIFOIFOSO00021.ilwd`, which was constructed by the script `src/STOCHASTICIFOIFOSO00021.ilwd.sh` using
`src/dmro_trivial.10x10238.1024.dat` for both IFO channels,
`src/spec_offwhite.-512-512.2.dat` for both whitened PSDs, and
`src/resp_cmplx.40-500.2.dat` for both response functions, and compares its output to
`expected/STOCHASTICIFOIFOSO00021_output_1.dat`; both of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/`
   and execute the script
   `./runtest.tclsh STOCHASTICIFOIFOSO00021`
   It will run an MPI job with the following schema, from the file `STOCHASTICIFOIFOSO00021.schema` in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO00021
#
# $Id: STOCHASTICIFOIFOSO00021.schema,v 1.2 2001/09/07 18:45:33 whelan Exp $
#
# Invokes libstochastic.so with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = L1\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
# argv[13] = 1                Check that time streams are simultaneous
```

Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
1.2e-7 + i 0.0	-6.9e-9 + i 0.0	4.2e-9	0.0

Table 9: Discrepancies in `STOCHASTICIFOIFOSO00021_output_1`; fill in the columns to two significant figures.

```
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 1                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
# argv[17] = spectrum2        Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO00021.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO00021.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO00021_`.

2. Examine the output files as follows:

   (a) Verify that the files all contain the same data and metadata aside from the start and end times and possibly the number of frames used by performing diffs among them and verifying that the only lines which differ are those containing the fields listed in the caption of table 2:

   ```
   diff ../output/STOCHASTICIFOIFOSO00021_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00021_output_2.ilwd
   diff ../output/STOCHASTICIFOIFOSO00021_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00021_output_3.ilwd
   ...
   diff ../output/STOCHASTICIFOIFOSO00021_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00021_output_10.ilwd
   ```

   Pass

   (b) Next, verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another
   by running the script
   `./datacomp.tclsh STOCHASTICIFOIFOSO00021_output_1`
   and filling out table 9; to pass, all the fractional errors must be less than 10^{-6}.

   Pass

   (c) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
   by running the script
   `./datadiff.tclsh STOCHASTICIFOIFOSO00021_output_1`
   and filling out table 10; to pass, all the fractional errors must be less than 10^{-4}.

   Pass

Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
1.2e-7 + i 1.6e-24	3.0e-6 - i 1.3e-17	2.3e-7	5.7e-18

Table 10: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO00021_output_1`; fill in the columns to two significant figures.

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.6 STOCHASTICIFOIFOSO00031

Test Case: STOCHASTICIFOIFOSO00031

Purpose: Test libldasstochastic.so behavior for $\sim$ 100 sec of trivial data, filtered assuming LIGO-I noise, ideal response in one detector & E2 response in the other, LHO & LLO detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 11:25 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on ldas-pcdev1.mit.edu as user ldas_mdc with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
lamboot -v.

This test uses the file input/STOCHASTICIFOIFOSO00031.ilwd, which was constructed by the script src/STOCHASTICIFOIFOSO00031.ilwd.sh using
src/dmro_trivial.10x10238.1024.dat for both IFO channels,
src/spec_white.-512-512.2.dat for the first whitened PSD,
src/spec_offwhite.-512-512.2.dat for the second, src/resp_ideal.40-500.2.dat for the first response function,
and src/resp_cmplx.40-500.2.dat for the second, and compares its output to
expected/STOCHASTICIFOIFOSO00031_output_1.dat; both of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
make data
from the directory
/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/.

PROCEDURE

1. Set the current directory to
   /.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/
   and execute the script
   ./runtest.tclsh STOCHASTICIFOIFOSO00031
   It will run an MPI job with the following schema, from the file STOCHASTICIFOIFOSO00031.schema in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO00031
#
# $Id: STOCHASTICIFOIFOSO00031.schema,v 1.2 2001/09/07 18:45:33 whelan Exp $
#
# Invokes libstochastic.so with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
```

Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
-1.3e-8 + i 7.6e-8	-1.8e-8 + i 6.3e-9	3.9e-9	3.8e-9

Table 11: Discrepancies in STOCHASTICIFOIFOSO00031_output_1; fill in the columns to two significant figures.

```
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = L1\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
# argv[13] = 1                Check that time streams are simultaneous
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 1                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
# argv[17] = spectrum2        Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO00031.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO00031.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO00031_`.

2. Examine the output files as follows:

   (a) Verify that the files all contain the same data and metadata aside from the start and end times and possibly the number of frames used by performing diffs among them and verifying that the only lines which differ are those containing the fields listed in the caption of table 2:

   ```
   diff ../output/STOCHASTICIFOIFOSO00031_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00031_output_2.ilwd
   diff ../output/STOCHASTICIFOIFOSO00031_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00031_output_3.ilwd
   ...
   diff ../output/STOCHASTICIFOIFOSO00031_output_1.ilwd \
     ../output/STOCHASTICIFOIFOSO00031_output_10.ilwd
   ```

   Pass

   (b) Next, verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another
   by running the script
   `./datacomp.tclsh STOCHASTICIFOIFOSO00031_output_1`
   and filling out table 11; to pass, all the fractional errors must be less than 10^{-6}.

   Pass

   (c) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`

Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
-1.3e-8 + i 7.6e-8	-2.5e-6 + i 7.1e-6	2.5e-7	4.6e-7

Table 12: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO00031_output_1`; fill in the columns to two significant figures.

by running the script
`./datadiff.tclsh STOCHASTICIFOIFOSO00031_output_1`
and filling out table 12; to pass, all the fractional errors must be less than 10^{-4}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.7 STOCHASTICIFOIFOSO01000

Test Case: STOCHASTICIFOIFOSO01000

Purpose: Test `libldasstochastic.so` behavior for $\sim$ 100 sec of white noise, filtered assuming white noise, delta-function response, coïncident & coäligned detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 11:50 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/STOCHASTICIFOIFOSO01000.ilwd`, which was constructed by the script `src/STOCHASTICIFOIFOSO01000.ilwd.sh` using
`src/dmro1_white.10x10238.1024.dat` and
`src/dmro2_white.10x10238.1024.dat` for the IFO channels,
`src/spec_white.-512-512.2.dat` for both whitened PSDs, and `src/resp_ident.40-500.2.dat` for both response functions, and compares its output to
`expected/STOCHASTICIFOIFOSO01000_output_1.dat`,
`expected/STOCHASTICIFOIFOSO01000_output_2.dat`, ...,
`expected/STOCHASTICIFOIFOSO01000_output_10.dat`; all of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/`
   and execute the script
   `./runtest.tclsh STOCHASTICIFOIFOSO01000`
   It will run an MPI job with the following schema, from the file `STOCHASTICIFOIFOSO01000.schema` in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO01000
#
# $Id: STOCHASTICIFOIFOSO01000.schema,v 1.2 2001/09/07 18:45:33 whelan Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
```

Segment	Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
1	-1.5e+2 - i 1.8e+1	-1.0e-8 - i 4.9e-9	3.4e-9	3.3e-9
2	-3.2e+1 + i 1.0e+2	1.6e-8 - i 5.4e-8	1.5e-8	1.5e-8
3	2.3e+1+ i 2.3e+1	7.5e-8 + i 2.3e-8	2.2e-8	5.4e-8
4	-2.7e+2 + i 2.0e+2	3.2e-10 + i 1.4e-9	1.8e-9	1.7e-9
5	-1.4e+1 - i 3.4e+1	2.4e-7 + i 9.4e-8	3.0e-7	3.2e-8
6	-1.4e+2 - i 2.7e+1	-1.9e-8 + i 5.0e-9	2.6e-8	3.3e-9
7	-5.1e+1 - i 1.7e+1	1.2e-8 - i 3.4e-8	9.2e-9	8.1e-9
8	1.7e+2 + i 4.4e+1	-8.4e-9 + i 1.3e-9	6.2e-9	2.9e-9
9	-8.9e0 + i 1.6e+1	1.3e-7 - i 4.3e-10	5.0e-8	5.2e-8
10	-2.7e+2 + i 3.1e+1	3.3e-9 - i 2.3e-9	1.8e-9	1.7e-9

Table 13: Discrepancies in `STOCHASTICIFOIFOSO01000_output_1`; fill in the columns to two significant figures.

```
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = H2\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
# argv[13] = 1                Check that time streams are simultaneous
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 0                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
# argv[17] = spectrum2        Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO01000.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO01000.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,0,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO01000_`.

2. Examine the output files as follows:

   (a) Verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another by running the script `./datacomp.tclsh` for each of the ten segments:

   ```
   ./datacomp.tclsh STOCHASTICIFOIFOSO01000_output_1
   ./datacomp.tclsh STOCHASTICIFOIFOSO01000_output_2
   ...
   ./datacomp.tclsh STOCHASTICIFOIFOSO01000_output_10
   ```

   and filling out table 13; to pass, all the fractional errors must be less than 10^{-6}.

   Pass

   (b) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in

Segment	Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
1	-1.5e+2 - i 1.8e+1	-3.0e-7 + i 3.7e-8	1.6e-7	1.3e-7
2	-3.2e+1 + i 1.0e+2	-5.6e-7 + i 1.3e-6	6.5e-7	1.0e-6
3	2.3e+1 + i 2.3e+2	3.9e-7 - i 9.9e-7	7.4e-7	8.7e-7
4	-2.7e+2+ i 2.0e+2	-2.1e-7 + i 2.4e-7	1.6e-7	2.1e-7
5	-1.4e+1 - i 3.4e+1	-3.1e-6 - i 9.5e-7	1.2e-6	1.9e-6
6	-1.4e+2 - i 2.7e+1	-5.4e-7 + i 3.3e-7	2.1e-7	2.2e-7
7	-5.1e+1 - i 1.7e+1	-1.2e-7 + i 1.4e-7	6.0e-7	6.4e-7
8	1.7e+2 + i 4.4e+1	6.3e-7 - i 6.3e-8	4.7e-7	1.4e-7
9	-8.9e0 + i 1.6e+1	-3.3e-6 - i 1.6e-6	2.5e-6	3.1e-6
10	-2.7e+2 + i 3.1e+1	-4.0e-7 - i 1.7e-7	1.2e-7	6.0e-8

Table 14: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO01000_output_1`; fill in the columns to two significant figures.

`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
by running the script `./datadiff.tclsh` for each of the ten segments:

```
./datadiff.tclsh STOCHASTICIFOIFOSO01000_output_1
./datadiff.tclsh STOCHASTICIFOIFOSO01000_output_2
...
./datadiff.tclsh STOCHASTICIFOIFOSO01000_output_10
```

and filling out table 14; to pass, all the fractional errors must be less than 10^{-4}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.8 STOCHASTICIFOIFOSO01001

Test Case: STOCHASTICIFOIFOSO01001

Purpose: Test `libldasstochastic.so` behavior for $\sim$ 100 sec of white noise, filtered assuming white noise, delta-function response, LHO & LLO detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 13:42 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test *also* uses the file `input/STOCHASTICIFOIFOSO01000.ilwd` (there is no file `input/STOCHASTICIFOIFOSO01001.ilwd`, since the data are the same for both tests), and compares its output to
`expected/STOCHASTICIFOIFOSO01001_output_1.dat`,
`expected/STOCHASTICIFOIFOSO01001_output_2.dat`, ...,
`expected/STOCHASTICIFOIFOSO01001_output_10.dat`; all of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/`
   and execute the script
   `./runtest.tclsh STOCHASTICIFOIFOSO01001`
   It will run an MPI job with the following schema, from the file `STOCHASTICIFOIFOSO01001.schema` in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO01001
#
# $Id: STOCHASTICIFOIFOSO01001.schema,v 1.2 2001/09/07 18:45:33 whelan Exp $
#
#
# Invokes libstochastic.so with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = H2\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
```

Segment	Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
1	6.5e+2 + i 9.5e+1	6.0e-9 + i 6.5e-9	7.2e-9	6.4e-9
2	-3.9e+2 - i 4.2e+2	-1.2e-8 + i 7.8e-9	1.3e-8	7.0e-9
3	8.3e+1 + i 1.5e+2	-5.9e-8 - i 8.9e-9	3.3e-8	2.6e-8
4	1.0e+3 - i 9.0e+2	-1.3e-9 + i 7.7e-9	3.7e-9	4.8e-9
5	-5.1e+2 + i 2.3e+2	-2.8e-8 + i 5.4e-9	9.5e-9	6.1e-9
6	2.7e+2 - i 4.4e+1	6.8e-9 + i 4.8e-9	1.6e-8	1.7e-8
7	8.1e+1 + i 3.0e0	-1.3e-7 + i 1.2e-8	5.0e-8	1.2e-8
8	-6.7e+2 - i 2.3e+2	-4.3e-9 - i 4.2e-9	2.6e-9	7.4e-9
9	-6.4e+1 + i 1.0e+2	-1.1e-7 - i 1.5e-8	6.2e-8	7.0e-8
10	9.2e+2 - i 2.3e+2	-2.4e-9 - i 3.4e-10	5.0e-9	3.1e-9

Table 15: Discrepancies in `STOCHASTICIFOIFOSO01001_output_1`; fill in the columns to two significant figures.

```
# argv[13] = 1                Check that time streams are simultaneous
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 1                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
# argv[17] = spectrum2        Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO01000.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO01000.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,H2\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
## The following commented-out lines would be used if we were bothering
## to duplicate input data with only the channel names changed
##### h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
##### -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
##### -dynlib-dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
##### -dynlib-realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W
##### -jobID=8 -dynlib-uniqueID=9.0
##### -inputFile="../input/STOCHASTICIFOIFOSO01001.ilwd"
##### -dynlib-filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO01001_`.

2. Examine the output files as follows:

   (a) Verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another by running the script `./datacomp.tclsh` for each of the ten segments:

   ```
   ./datacomp.tclsh STOCHASTICIFOIFOSO01001_output_1
   ./datacomp.tclsh STOCHASTICIFOIFOSO01001_output_2
   ...
   ./datacomp.tclsh STOCHASTICIFOIFOSO01001_output_10
   ```

   and filling out table 15; to pass, all the fractional errors must be less than 10^{-6}.

Pass

Segment	Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
1	6.5e+2 + i 9.5e+1	6.3e-7 + i 1.9e-7	5.4e-7	3.7e-7
2	-3.9e+2 - i 4.2e+2	-1.0e-6 - i 1.1e-6	4.3e-7	8.4e-7
3	8.3e+1 + i 1.5e+2	7.0e-7 + i 4.1e-6	2.9e-6	4.4e-6
4	1.0e+3 - i 9.0e+2	7.7e-7 - i 7.7e-7	3.7e-7	3.9e-7
5	-5.1e+2 + i 2.3e+2	-7.5e-7 + i 1.8e-7	5.3e-7	1.9e-7
6	2.7e+2 - i 4.4e+1	8.7e-7 - i 9.3e-7	1.2e-6	6.9e-7
7	8.1e+1 + i 3.0e0	2.7e-7 - i 1.4e-6	1.8e-6	3.7e-6
8	-6.7e+2 - i 2.3e+2	-1.3e-6 - i 2.2e-7	1.2e-6	3.9e-7
9	-6.4e+1 + i 1.0e+2	1.7e-6 + i 1.6e-6	3.3e-6	3.0e-6
10	9.2e+2 - i 2.3e+2	8.6e-7 - i 2.3e-7	3.5e-7	2.5e-7

Table 16: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO01001_output_1`; fill in the columns to two significant figures.

(b) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
by running the script `./datadiff.tclsh` for each of the ten segments:
```
./datadiff.tclsh STOCHASTICIFOIFOSO01001_output_1
./datadiff.tclsh STOCHASTICIFOIFOSO01001_output_2
...
./datadiff.tclsh STOCHASTICIFOIFOSO01001_output_10
```
and filling out table 16; to pass, all the fractional errors must be less than 10^{-4}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.9 STOCHASTICIFOIFOSO01011

Test Case: STOCHASTICIFOIFOSO01011

Purpose: Test `libldasstochastic.so` behavior for $\sim$ 100 sec of white noise, filtered assuming LIGO-I noise, ideal response, LHO & LLO detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 14:07 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/STOCHASTICIFOIFOSO01011.ilwd`, which was constructed by the script `src/STOCHASTICIFOIFOSO01011.ilwd.sh` using
`src/dmro1_white.10x10238.1024.dat` and
`src/dmro2_white.10x10238.1024.dat` for the IFO channels,
`src/spec_white.-512-512.2.dat` for both whitened PSDs, and `src/resp_ideal.40-500.2.dat` for both response functions, and compares its output to
`expected/STOCHASTICIFOIFOSO01011_output_1.dat`,
`expected/STOCHASTICIFOIFOSO01011_output_2.dat`, ...,
`expected/STOCHASTICIFOIFOSO01011_output_10.dat`; all of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/`
   and execute the script
   `./runtest.tclsh STOCHASTICIFOIFOSO01011`
   It will run an MPI job with the following schema, from the file `STOCHASTICIFOIFOSO01011.schema` in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO01011
#
# $Id: STOCHASTICIFOIFOSO01011.schema,v 1.2 2001/09/07 18:45:33 whelan Exp $
#
# Invokes libstochastic.so with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
```

Segment	Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
1	2.9e-6 + i 7.3e-6	-3.7e-8 + i 5.5e-8	1.7e-8	1.7e-8
2	-8.9e-6 - i 5.0e-6	4.4e-9 + i 2.4e-8	5.5e-9	5.4e-9
3	2.2e-5 - i 1.8e-6	9.6e-9 - i 2.8e-9	2.3e-9	2.2e-9
4	2.9e-5 - i 3.4e-5	3.6e-10 + i 6.4e-9	1.7e-9	1.7e-9
5	-1.7e-5 + i 1.6e-5	5.3e-9 + i 9.6e-9	2.7e-9	2.6e-9
6	5.9e-6 - i 1.4e-5	2.6e-8 + i 7.3e-9	8.4e-9	7.9e-9
7	-2.8e-6 - i 1.3e-5	2.2e-8 - i 3.1e-8	1.7e-8	1.7e-8
8	-8.1e-6 - i 7.1e-6	2.2e-8 - i 1.5e-8	6.0e-9	6.1e-9
9	6.3e-6 + i 4.0e-6	1.6e-8 - i 5.5e-8	7.7e-9	7.4e-9
10	1.3e-5 - i 9.9e-6	5.1e-9 + i 7.6e-9	3.6e-9	3.6e-9

Table 17: Discrepancies in `STOCHASTICIFOIFOSO01011_output_1`; fill in the columns to two significant figures.

```
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = L1\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
# argv[13] = 1                Check that time streams are simultaneous
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 1                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
# argv[17] = spectrum2        Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO01011.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO01011.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO01011_`.

2. Examine the output files as follows:

   (a) Verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another by running the script `./datacomp.tclsh` for each of the ten segments:

   ```
   ./datacomp.tclsh STOCHASTICIFOIFOSO01011_output_1
   ./datacomp.tclsh STOCHASTICIFOIFOSO01011_output_2
   ...
   ./datacomp.tclsh STOCHASTICIFOIFOSO01011_output_10
   ```

   and filling out table 17; to pass, all the fractional errors must be less than 10^{-6}.

   Pass

   (b) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in

Segment	Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
1	2.9e-6 + i 7.3e-6	-5.5e-6 + i 1.2e-6	2.6e-6	2.3e-6
2	-8.9e-6 - i 5.0e-6	-1.1e-6 - i 8.2e-7	7.2e-7	6.5e-7
3	2.2e-5 - i 1.8e-6	6.2e-7 + i 2.4e-7	4.8e-7	2.4e-7
4	2.9e-5 - i 3.4e-5	-1.4e-7 - i 6.8e-8	2.5e-7	2.9e-7
5	-1.7e-5 + i 1.6e-5	-9.9e-7 + i 2.3e-7	4.2e-7	2.6e-7
6	5.9e-6 - i 1.4e-5	-3.0e-6 - i 4.4e-6	8.2e-7	8.9e-7
7	-2.8e-6 - i 1.3e-5	-1.0e-6 - i 6.4e-6	1.5e-6	1.7e-6
8	-8.1e-6 - i 7.1e-6	4.6e-7 - i 4.9e-7	7.2e-7	7.0e-7
9	6.3e-6 + i 4.0e-6	2.0e-6 + i 1.0e-6	8.5e-7	5.1e-7
10	1.3e-5 - i 9.9e-6	-3.8e-7 - i 2.9e-7	4.1e-7	3.4e-7

Table 18: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO01011_output_1`; fill in the columns to two significant figures.

`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
by running the script `./datadiff.tclsh` for each of the ten segments:

```
./datadiff.tclsh STOCHASTICIFOIFOSO01011_output_1
./datadiff.tclsh STOCHASTICIFOIFOSO01011_output_2
...
./datadiff.tclsh STOCHASTICIFOIFOSO01011_output_10
```

and filling out table 18; to pass, all the fractional errors must be less than 10^{-4}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.10 STOCHASTICIFOIFOSO01021

Test Case: STOCHASTICIFOIFOSO01021

Purpose: Test `libldasstochastic.so` behavior for $\sim$ 100 sec of "off-white" noise, filtered assuming LIGO-I noise, E2 response, LHO & LLO detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 14:27 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/STOCHASTICIFOIFOSO01021.ilwd`, which was constructed by the script `src/STOCHASTICIFOIFOSO01021.ilwd.sh` using
`src/dmro1_offwhite.10x10238.1024.dat` and
`src/dmro2_offwhite.10x10238.1024.dat` for the IFO channels,
`src/spec_offwhite.-512-512.2.dat` for both whitened PSDs, and
`src/resp_ident.40-500.2.dat` for both response functions, and compares its output to
`expected/STOCHASTICIFOIFOSO01021_output_1.dat`,
`expected/STOCHASTICIFOIFOSO01021_output_2.dat`, ...,
`expected/STOCHASTICIFOIFOSO01021_output_10.dat`; all of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/`
   and execute the script
   `./runtest.tclsh STOCHASTICIFOIFOSO01021`
   It will run an MPI job with the following schema, from the file `STOCHASTICIFOIFOSO01021.schema` in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO01021
#
# $Id: STOCHASTICIFOIFOSO01021.schema,v 1.2 2001/09/07 18:45:33 whelan Exp $
#
# Invokes libstochastic.so with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
```

Segment	Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
1	-1.0e-2 + i 1.0e-2	3.9e-9 + i 7.3e-9	4.7e-9	4.3e-9
2	9.8e-3 - i 1.9e-2	-1.8e-10 + i 5.7e-9	4.7e-9	5.0e-9
3	2.1e-2 + i 5.4e-3	3.8e-10 - i 3.1e-9	2.3e-9	2.2e-9
4	-2.7e-3 - i 8.7e-3	-4.0e-8 + i 1.0e-8	1.7e-8	1.7e-8
5	4.0e-4 - i 1.0e-3	3.3e-7 - i 1.6e-7	1.0e-7	1.0e-7
6	5.3e-3 - i 4.7e-3	-2.4e-8 + i 4.2e-8	9.4e-9	9.5e-9
7	-1.3e-2 + i 5.8e-3	-2.9e-9 - i 2.6e-9	3.8e-9	3.8e-9
8	2.4e-3 - i 1.4e-2	-5.5e-8 - i 3.1e-8	2.0e-8	2.0e-8
9	9.6e-3 - i 3.6e-3	-8.0e-9 + i 5.3e-9	4.3e-9	5.0e-9
10	2.8e-3 - i 4.8e-3	7.9e-9 - i 4.8e-8	1.5e-8	1.7e-8

Table 19: Discrepancies in `STOCHASTICIFOIFOSO01021_output_1`; fill in the columns to two significant figures.

```
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = L1\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
# argv[13] = 1                Check that time streams are simultaneous
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 1                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
# argv[17] = spectrum2        Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO01021.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO01021.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO01021_`.

2. Examine the output files as follows:

   (a) Verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another by running the script `./datacomp.tclsh` for each of the ten segments:

   ```
   ./datacomp.tclsh STOCHASTICIFOIFOSO01021_output_1
   ./datacomp.tclsh STOCHASTICIFOIFOSO01021_output_2
   ...
   ./datacomp.tclsh STOCHASTICIFOIFOSO01021_output_10
   ```

   and filling out table 19; to pass, all the fractional errors must be less than 10^{-6}.

   Pass

   (b) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in

Segment	Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
1	-1.0e-2 + i 1.0e-2	-1.1e-7 - i 1.7e-7	3.4e-7	2.4e-7
2	9.8e-3 - i 1.9e-2	4.3e-7 - i 2.4e-7	2.4e-7	4.3e-7
3	2.1e-2 + i 5.4e-3	7.8e-8 + i 6.4e-8	1.1e-7	1.3e-7
4	-2.7e-3 - i 8.7e-3	1.0e-6 + i 3.6e-6	9.3e-7	1.1e-6
5	4.0e-4 - i 1.0e-3	5.6e-6 - i 6.7e-6	3.8e-6	6.8e-6
6	5.3e-3 - i 4.7e-3	-2.5e-7 + i 2.3e-6	4.7e-7	5.1e-7
7	-1.2e-2 + i 5.8e-3	-2.5e-7 + i 1.4e-7	2.9e-7	3.3e-7
8	2.4e-3 - i 1.4e-2	-4.4e-6 + i 1.4e-7	1.3e-6	1.1e-6
9	9.6e-3 - i 3.6e-3	6.9e-7 + i 1.0e-6	2.5e-7	3.6e-7
10	2.8e-3 - i 4.8e-3	5.1e-7 + i 7.9e-7	8.3e-7	6.5e-7

Table 20: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO01021_output_1`; fill in the columns to two significant figures.

`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
by running the script `./datadiff.tclsh` for each of the ten segments:

```
./datadiff.tclsh STOCHASTICIFOIFOSO01021_output_1
./datadiff.tclsh STOCHASTICIFOIFOSO01021_output_2
...
./datadiff.tclsh STOCHASTICIFOIFOSO01021_output_10
```

and filling out table 20; to pass, all the fractional errors must be less than 10^{-4}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.11 STOCHASTICIFOIFOSO01121

Test Case: STOCHASTICIFOIFOSO01121

Purpose: Test `libldasstochastic.so` behavior for $\sim$ 100 sec of "off-white" noise plus a simulated stochastic background signal, filtered assuming LIGO-I noise, E2 response, LHO & LLO detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 15:03 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/STOCHASTICIFOIFOSO01121.ilwd`, which was constructed by the script `src/STOCHASTICIFOIFOSO01121.ilwd.sh` using
`src/dmro1_offplusomega.10x10238.1024.dat` and
`src/dmro2_offplusomega.10x10238.1024.dat` for the IFO channels,
`src/spec_offwhite.-512-512.2.dat` for both whitened PSDs, and
`src/resp_ident.40-500.2.dat` for both response functions, and compares its output to
`expected/STOCHASTICIFOIFOSO01121_output_1.dat`,
`expected/STOCHASTICIFOIFOSO01121_output_2.dat`, ...,
`expected/STOCHASTICIFOIFOSO01121_output_10.dat`; all of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/`
   and execute the script
   `./runtest.tclsh STOCHASTICIFOIFOSO01121`
   It will run an MPI job with the following schema, from the file `STOCHASTICIFOIFOSO01121.schema` in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO01121
#
# $Id: STOCHASTICIFOIFOSO01121.schema,v 1.2 2001/09/07 18:45:33 whelan Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = 10               Number of data segments
# argv[2]  = 10238            Number of points in a data segment
# argv[3]  = 230              Number of frequencies in optimal filter
# argv[4]  = 40               Start frequency for optimal filter
# argv[5]  = 2                Frequency spacing for optimal filter
```

Segment	Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
1	-1.0e-2 + i 1.0e-2	1.5e-9 - i 5.1e-9	3.9e-9	4.8e-9
2	9.9e-3 - i 1.9e-2	-2.1e-9 - i 5.6e-10	4.6e-9	4.9e-9
3	2.2e-2 + i 5.6e-3	-6.3e-9 - i 4.8e-9	2.2e-9	2.1e-9
4	-2.9e-3 - i 8.9e-3	4.7e-8 - i 1.2e-8	1.6e-8	1.5e-8
5	4.0e-4 - i 9.4e-4	-4.6e-7 - i 1.7e-7	1.2e-7	1.2e-7
6	5.3e-3 - i 4.6e-3	6.8e-9 - i 3.1e-8	8.9e-9	9.1e-9
7	-1.3e-2 + i 5.7e-3	-8.4e-9 + i 9.7e-9	3.7e-9	3.7e-9
8	2.1e-3 - i 1.4e-2	-1.4e-8 + i 2.6e-8	2.3e-8	2.3e-8
9	9.5e-3 - i 3.8e-3	1.5e-9 + i 1.3e-9	5.0e-9	4.6e-9
10	2.4e-3 - i 4.9e-3	-2.0e-8 + i 2.8e-8	1.7e-8	2.0e-8

Table 21: Discrepancies in `STOCHASTICIFOIFOSO01121_output_1`; fill in the columns to two significant figures.

```
# argv[6]  = 230              Number of frequencies in CC spectrum
# argv[7]  = 40               Start frequency for CC spectrum
# argv[8]  = 2                Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q     Name of first data channel
# argv[10] = L1\:LSC-AS_Q     Name of second data channel
# argv[11] = response1        Name of first response fcn
# argv[12] = response2        Name of second response fcn
# argv[13] = 1                Check that time streams are simultaneous
# argv[14] = 0                Index of first detector in lalCachedDetectors
# argv[15] = 1                Index of second detector in lalCachedDetectors
#                                 NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1        Name of first PSD
# argv[17] = spectrum2        Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO01121.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO01121.ilwd"
#  -filterparams="(10,10238,230,40,2,230,40,2,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO01121_`.

2. Examine the output files as follows:

   (a) Verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another by running the script `./datacomp.tclsh` for each of the ten segments:

   ```
   ./datacomp.tclsh STOCHASTICIFOIFOSO01121_output_1
   ./datacomp.tclsh STOCHASTICIFOIFOSO01121_output_2
   ...
   ./datacomp.tclsh STOCHASTICIFOIFOSO01121_output_10
   ```

   and filling out table 21; to pass, all the fractional errors must be less than 10^{-6}.

   Pass

   (b) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in

Segment	Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
1	-1.0e-2 + i 1.0e-2	-5.0e-8 - i 1.5e-7	2.5e-7	2.7e-7
2	9.9e-3 - i 1.9e-2	4.3e-7 - i 1.6e-7	2.4e-7	3.5e-7
3	2.1e-2 + i 5.6e-3	1.2e-8 + i 1.0e-8	1.1e-7	1.5e-7
4	-2.9e-3 - i 8.9e-3	7.9e-7 + i 3.7e-6	9.3e-7	1.2e-6
5	4.0e-4 - i 9.3e-4	8.1e-6 - i 1.3e-5	4.2e-6	5.5e-6
6	5.3e-3 - i 4.6e-3	-2.3e-7 + i 2.8e-6	4.0e-7	5.3e-7
7	-1.3e-2 + i 5.7e-3	-5.4e-7 + i 7.0e-8	2.2e-7	3.2e-7
8	2.1e-3 - i 1.4e-2	-3.7e-6 - i 1.5e-7	1.4e-6	1.6e-6
9	9.5e-3 - i 3.8e-3	6.3e-7 + i 6.9e-7	3.6e-7	3.5e-7
10	2.4e-3 - i 4.9e-3	1.4e-7 - i 4.4e-7	1.0e-6	7.4e-7

Table 22: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO01121_output_1`; fill in the columns to two significant figures.

`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
by running the script `./datadiff.tclsh` for each of the ten segments:

```
./datadiff.tclsh STOCHASTICIFOIFOSO01121_output_1
./datadiff.tclsh STOCHASTICIFOIFOSO01121_output_2
...
./datadiff.tclsh STOCHASTICIFOIFOSO01121_output_10
```

and filling out table 22; to pass, all the fractional errors must be less than 10^{-4}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.9.12 STOCHASTICIFOIFOSO11121

Test Case: STOCHASTICIFOIFOSO11121

Purpose: Test liblasstochastic.so behavior for ∼ 15 min of "off-white" noise plus a simulated stochastic background signal, filtered assuming LIGO-I noise, E2 response, LHO & LLO detectors

Tester: Joseph D. Romano

Test machine: ldas-pcdev1.mit.edu

Date (mm/dd/yy): 09/08/01 **Time:** 16:40 EDT

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/STOCHASTICIFOIFOSO11121.ilwd`, which was constructed by the script `src/STOCHASTICIFOIFOSO11121.ilwd.sh` using
`src/dmro1_offplusomega.10x92138.1024.dat` and
`src/dmro2_offplusomega.10x92138.1024.dat` for the IFO channels,
`src/spec_offwhite.-512-512.1_4.dat` for both whitened PSDs, and
`src/resp_ident.40-500.1_4.dat` for both response functions, and compares its output to
`expected/STOCHASTICIFOIFOSO11121_output_1.dat`,
`expected/STOCHASTICIFOIFOSO11121_output_2.dat`, ...,
`expected/STOCHASTICIFOIFOSO11121_output_10.dat`; all of these files are gzip-compressed in the repository, but they must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-IFO tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/command/`
   and execute the script
   `./runtest.tclsh STOCHASTICIFOIFOSO11121`
   It will run an MPI job with the following schema, from the file `STOCHASTICIFOIFOSO11121.schema` in the same directory:

```
#
# lam boot schema to run test STOCHASTICIFOIFOSO11121
#
# $Id: STOCHASTICIFOIFOSO11121.schema,v 1.3 2001/09/07 18:45:33 whelan Exp $
#
# libstochastic.so should be invoked with filter parameters
#
# argv[1]  = 10              Number of data segments
# argv[2]  = 92138           Number of points in a data segment
# argv[3]  = 1840            Number of frequencies in optimal filter
# argv[4]  = 40              Start frequency for optimal filter
# argv[5]  = 0.25            Frequency spacing for optimal filter
```

```
# argv[6]  = 1840              Number of frequencies in CC spectrum
# argv[7]  = 40                Start frequency for CC spectrum
# argv[8]  = 0.25              Frequency spacing for CC spectrum
# argv[9]  = H1\:LSC-AS_Q      Name of first data channel
# argv[10] = L1\:LSC-AS_Q      Name of second data channel
# argv[11] = response1         Name of first response fcn
# argv[12] = response2         Name of second response fcn
# argv[13] = 1                 Check that time streams are simultaneous
# argv[14] = 0                 Index of first detector in lalCachedDetectors
# argv[15] = 1                 Index of second detector in lalCachedDetectors
#                                  NOTE: 0 = LHO ; 1 = LLO
# argv[16] = spectrum1         Name of first PSD
# argv[17] = spectrum2         Name of second PSD
#
# And input file ../input/STOCHASTICIFOIFOSO11121.ilwd
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so" -dataAPI="(datahost,5678)" -
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/STOCHASTICIF
filterparams="(10,92138,1840,40,0.25,1840,40,0.25,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
#
# Broken up, the preceding line reads
#
#  h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)"
#  -dynlib="/ldcg/lib/lalwrapper/libldasstochastic.so"
#  -dataAPI="(datahost,5678)" -resultAPI"=(eventmon,1002)"
#  -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
#  -uniqueID=9.0 -inputFile="../input/STOCHASTICIFOIFOSO11121.ilwd"
#  -filterparams="(10,92138,1840,40,0.25,1840,40,0.25,H1\:LSC-AS_Q,L1\:LSC-AS_Q,response1,response2,1,0,1,spectrum1,spectrum2)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/output/`, prepending to each filename the prefix `STOCHASTICIFOIFOSO11121_`.

2. Examine the output files as follows:

   (a) Verify that the correct metadata are stored for the output spectrum in `../output/STOCHASTICIFOIFOSO11121_output_1.ilwd \`:

   i. `start_frequency:summ_spectrum:dataBase` and `start_freq` are 40=4e+1 Hz

   ii. `stop_freq` is 499.75=4.9975e+2 Hz

   iii. `delta_frequency:summ_spectrum:dataBase` and `freq:step_size` are 0.25=2.5e-1 Hz

   iv. The size of the binary large object in the database entry `src/spectrum:summ_spectrum:dataBase` is $1840 * 8 = 14720$ bytes (1840 8-byte complex numbers).

   v. There are 1840 points in the output frequency series: `filter_output::sequence`

   vi. The units of the container labelled `data` in the `filter_output::sequence` are $\mathrm{s}/\mathrm{Hz} = \mathrm{s}^2$

   vii. Check that the metadata in the files `../output/STOCHASTICIFOIFOSO11121_output_1.ilwd \` through `../output/STOCHASTICIFOIFOSO11121_output_10.ilwd \` agree with those listed in table 23.

   Pass

   (b) Verify that the values in the multidimdata and database sections of the of the ILWDs agree with one another by running the script `./datacomp.tclsh` for each of the ten segments:

```
./datacomp.tclsh STOCHASTICIFOIFOSO11121_output_1
./datacomp.tclsh STOCHASTICIFOIFOSO11121_output_2
...
./datacomp.tclsh STOCHASTICIFOIFOSO11121_output_10
```

   and filling out table 24; to pass, all the fractional errors must be less than 10^{-6}.

   Pass

Segment	Start sec	Start ns	Stop sec	Stop ns	Frames used
1	488811610	0	488811699	978515625	90
2	488811699	978515625	488811789	957031250	91
3	488811789	957031250	488811879	935546875	91
4	488811879	935546875	488811969	914062500	91
5	488811969	914062500	488812059	892578125	91
6	488812059	892578125	488812149	871093750	91
7	488812149	871093750	488812239	849609375	91
8	488812239	849609375	488812329	828125000	91
9	488812329	828125000	488812419	806640625	91
10	488812419	806640625	488812509	785156250	91

Table 23: Start and stop times for `../output/STOCHASTICIFOIFOSO11121_output_#.ilwd`; the fields to check are `start_time:summ_spectrum:dataBase`, `start_time_ns:summ_spectrum:dataBase`, `end_time:summ_spectrum:dataBase`, `end_time_ns:summ_spectrum:dataBase`, and `frames_used:summ_spectrum:dataBase` in the `database` section, and `gps_sec:start_time`, `gps_nan:start_time`, `gps_sec:stop_time`, and `gps_nan:stop_time` in the `filter_output::sequence` section.

Segment	Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
1	2.3e-2 - i 1.8e-2	-1.5e-9 - i 6.2e-9	2.1e-9	2.1e-9
2	-3.4e-2 - i 2.7e-2	9.0e-9 - i 2.4e-9	1.4e-9	1.4e-9
3	-2.9e-2 - i 3.0e-4	1.1e-8 - i 1.7e-8	1.7e-9	1.7e-9
4	-2.7e-2 + i 1.4e-2	-5.3e-9 + i 1.4e-8	1.8e-9	1.8e-9
5	4.4e-3 - i 1.5e-2	9.9e-8 + i 5.5e-8	1.1e-8	1.1e-8
6	2.2e-2 + i 1.5e-2	1.8e-8 + i 2.3e-8	2.3e-9	2.3e-9
7	2.7e-2 + i 1.4e-2	1.1e-8 + i 4.4e-10	1.8e-9	1.8e-9
8	-8.4e-3 - i 1.3e-2	-1.9e-8 + i 4.7e-9	6.0e-9	5.9e-9
9	-4.7e-3 + i 4.7e-2	-5.1e-11 + i 4.8e-8	1.0e-8	1.1e-8
10	2.6e-2 - i 2.3e-2	-4.7e-9 + i 1.8e-10	1.9e-9	1.9e-9

Table 24: Discrepancies in `STOCHASTICIFOIFOSO11121_output_1`; fill in the columns to two significant figures.

(c) Finally, verify that the values in the multidimdata (frame output) section of the ILWDs agree with those in
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/expected/`
by running the script `./datadiff.tclsh` for each of the ten segments:
```
./datadiff.tclsh STOCHASTICIFOIFOSO11121_output_1
./datadiff.tclsh STOCHASTICIFOIFOSO11121_output_2
...
./datadiff.tclsh STOCHASTICIFOIFOSO11121_output_10
```
and filling out table 25; to pass, all the fractional errors must be less than 10^{-4}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

Segment	Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
1	2.3e-2 - i 1.8e-2	8.8e-7 - i 1.0e-6	1.3e-5	1.1e-5
2	-3.4e-2 - i 2.7e-2	2.0e-6 + i 7.0e-7	8.8e-6	6.8e-6
3	-2.9e-2 - i 3.0e-4	7.1e-7 - i 1.1e-6	7.5e-6	1.2e-5
4	-2.7e-2 + i 1.4e-2	1.1e-6 - i 4.7e-7	8.6e-6	1.3e-5
5	4.4e-3 - i 1.5e-2	-1.5e-7 + i 1.8e-5	6.2e-5	5.7e-5
6	2.2e-2 + i 1.5e-2	-6.1e-7 + i 4.8e-7	1.1e-5	1.1e-5
7	2.7e-2 + i 1.4e-2	6.6e-7 + i 7.7e-7	1.4e-5	7.1e-6
8	-8.4e-3 - i 1.3e-2	1.3e-6 - i 1.1e-7	2.3e-5	7.2e-5
9	-4.7e-3 + i 4.7e-2	7.6e-7 + i 4.5e-6	7.3e-5	5.4e-5
10	2.6e-2 - i 2.3e-3	6.1e-7 + i 4.7e-7	2.6e-5	9.1e-6

Table 25: Errors in frame multidimdata section of `STOCHASTICIFOIFOSO11121_output_1`; fill in the columns to two significant figures.

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.10 Stochastic IFO-bar correlation DSO standalone tests

The tests in this section use the standalone wrapper API to test LDAS shared object `libldasstochasticbar.so`, which calculates the frequency-domain integrand of the optimally-filtered cross-correlation statistic.

There are two types of tests that are performed: the first test that error checking for input parameters is correct, the second check that the shared object calculates the correct values for valid parameters.

The naming convention for validation tests `stochasticbar00000` *et seq.* is as follows:

- The first two digits are extra numbers that are not used at this point in time. They are always set to 0.
- The third digit (0/1) indicates whether the simulation and analysis are performed for coincident and coaligned detectors, or if the orientation information for LIGO Livingston and ALLEGRO (for 1996-2000) is used.
- The fourth digit (0/1/2) indicates whether there is no signal, a stochastic-background signal or a sine-wave signal present.
- The fifth digit (0/1/2) indicates the whether there is, no noise, white noise or coloured noise present.

The `stochasticbar#####` tests run in the stand-alone wrapperAPI and require a complete installation of LAL, LALWrapper, and the standalone wrapperAPI.

The units are ADC counts for all `dmro` datasets, $(\texttt{ADC counts})^2/\texttt{Hz}$ for all `spec` datasets, and $\texttt{ADC counts}/(10^{-18}\,\texttt{strain})$ for all `resp` datasets.

Each `stochasticbar#####` test is run by the script `command/runtest.tclsh` which has the following source:

```
#!/ldcg/bin/tclsh
# $Id: runtest.tclsh,v 1.1 2001/09/06 22:23:21 heng Exp $
# Script to run pipeline tests

set test [lindex $argv 0]

exec mpirun -v $test.schema  >@ stdout

foreach i {1 2} {
    set newname $test
    append newname _process_$i
    file rename -force process_$i.ilwd ../output/$newname.ilwd
}

for {set i 1} {$i <= 10} {incr i} {
    set newname $test
    append newname _output_$i
    file rename -force output_$i.ilwd ../output/$newname.ilwd
}
```

A.10.1 `stochasticbar00000`

Test Case: `stochasticbar00000`

Purpose: Test `libldasstochasticbar.so` behavior for $\sim$ 15 minutes of data, where the real component of the first sample is 1 and everything else is 0, filtered assuming white noise, flat response response, coincident & coaligned detectors

Tester:

Test machine:

Date (mm/dd/yy): **Time:**

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed. It will also require Matlab and the `"ilwdread"` matlab function distributed as part of the `ligotools` package for the data comparison tests.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/stochasticbar00000.ilwd` which should be stored in the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/input/`

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/command/`
   and execute the script
   `./runtest.tclsh stochasticbar00000`
   It will run an MPI job with the following schema, from the file `stochasticbar00000.schema` in the same directory:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: stochasticbar00000.schema,v 1.1 2001/09/06 22:23:20 heng Exp $
#
# libstochasticbar.so should be invoked with the following arguments
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
# argv[18] = allegroLatitude   Latitude of ALLEGRO
# argv[19] = allegroLongitude  Longitude of ALLEGRO
# argv[20] = allegroAzimuth    Azimuth of ALLEGRO
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochasticbar.so" -dataAPI="(datahost,5678)"
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/stochasticba
filterparams="(10,22500,24900,782.5,0.01,120,784,2,L1\:LSC-AS_Q,AL\:GW_CHAN,response1,response2,1,0,0,spectrum1,spectrum2,0,0,0)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/output/`, prepending to each filename the prefix `stochasticbar00000_`.

2. Examine the output files as follows:

   (a) Verify that the files all contain the same data and metadata aside from the start end end times and possibly the number of frames used by performing diffs among them and verifying that the only lines which differ are those containing the fields listed in the caption of table 2:

```
diff ../output/stochasticbar00000_output_1.ilwd \
  ../output/stochasticbar00000_output_2.ilwd
diff ../output/stochasticbar00000_output_1.ilwd \
  ../output/stochasticbar00000_output_3.ilwd
...
diff ../output/stochasticbar00000_output_1.ilwd \
  ../output/stochasticbar00000_output_10.ilwd
```

   Pass / Fail

   (b) Go to the ouput data directory at `../output/stochasticbar00000_output_1.ilwd \`:

   i. `start_frequency:summ_spectrum:dataBase` and `start_freq` are 782.5=7.825e+2 Hz
   ii. `stop_frequency:summ_spectrum:dataBase` and `stop_freq` are 1031.49=1.03149e+3 Hz
   iii. `delta_frequency:summ_spectrum:dataBase` and `freq:step_size` are 0.01 Hz
   iv. The type of the output spectrum is `complex_8`.
   v. There are 24900 points in the output frequency series.
   vi. The units of the container labelled `data` in the `filter_output::sequence` are $\mathrm{s}/\mathrm{Hz} = \mathrm{s}^2$

   Pass / Fail

   (c) i. To compare the output data with the expected output, go to `/.../ldasmdc/burst-stochastic/test/expected/` You should see a series of Matlab files. If these files are gzipped, type `make` and all the Matlab files will be unzipped.

   ii. Start up matlab by typing `matlab` and load the file `stochasticbar00000Expected.mat`.

   iii. Load the first output file by typing from within Matlab
   `p = ilwdread('../output/stochasticbar00000_output_1.ilwd')` and extract the spectra by typing
   `output_spectra=p.m1.m4.m9.complex_8;`
   Also, create a corresponding frequency vector by doing
   `f1=[782.5:0.01:1031.49];`

   iv. Calculate the absolute differences in the real and imaginary components like so
```
real_diff=abs(real(CCspec(:,1))-real(output_spectra));
imag_diff=abs(imag(CCspec(:,1))-imag(output_spectra));
```
   and find the maximum
```
max_real_diff=max(real_diff);
max_imag_diff=max(imag_diff);
```

   v. Now calculate the sum of the real and imaginary components expected output
```
real_tot_expected=sum(real(CCspec(:,1)));
imag_tot_expected=sum(imag(CCspec(:,1)));
```

and likewise for the output files

```
real_tot_database=sum(real(output_spectra));
imag_tot_database=sum(imag(output_spectra));
```

vi. Finally, calculate the fractional errors

```
magn_tot_expected = sqrt (real_tot_expectedˆ2 + imag_tot_expectedˆ2);
max_real_diff/magn_tot_expected
max_imag_diff/magn_tot_expected
```

vii. For this test, the imaginary errors must be zero and the fractional error in the real parts should be less than 10^{-7}.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

_____

New faults submitted – list bug IDs: _____

_____

Notes: _____

_____

TEST RESULT **Pass / Fail**

A.10.2 `stochasticbar00001`

Test Case: `stochasticbar00001`

Purpose: Test `libldasstochasticbar.so` behavior for $\sim$ 900 sec of white noise, filtered assuming white PSD, flat response and coincident & coaligned detectors

Tester:

Test machine:

Date (mm/dd/yy): **Time:**

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed. It will also require Matlab and the `"ilwdread"` matlab function distributed as part of the `ligotools` package for the data comparison tests.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test *also* uses the file `input/stochasticbar00001.ilwd`

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/command/`
   and execute the script
   `./runtest.tclsh stochasticbar00001`
   It will run an MPI job with the following schema, from the file `stochasticbar00001.schema` in the same directory:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: stochasticbar00001.schema,v 1.1 2001/09/06 22:23:20 heng Exp $
#
# libstochasticbar.so should be invoked with the following arguments
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
# argv[18] = allegroLatitude   Latitude of ALLEGRO
# argv[19] = allegroLongitude  Longitude of ALLEGRO
# argv[20] = allegroAzimuth    Azimuth of ALLEGRO
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochasticbar.so" -dataAPI="(datahost,5678)"
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/stochasticba
filterparams="(10,22500,24900,782.5,0.01,120,784,2,L1\:LSC-AS_Q,AL\:GW_CHAN,response1,response2,1,0,0,spectrum1,spectrum2,0,0,0)"
```

   and then move the output and process ILWDs into the directory
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/output/`, prepending to each filename the prefix `stochasticbar00001_`.

2. Examine the output files as follows:

   (a) Go to the ouput data directory at `../output/stochasticbar00001_output_1.ilwd`

      i. `start_frequency:summ_spectrum:dataBase` and `start_freq` are 782.5=7.825e+2 Hz
      ii. `stop_frequency:summ_spectrum:dataBase` and `stop_freq` are 1031.49=1.03149e+3 Hz
      iii. `delta_frequency:summ_spectrum:dataBase` and `freq:step_size` are 0.01 Hz
      iv. The type of the output spectrum is `complex_8`.
      v. There are 24900 points in the output frequency series.
      vi. The units of the container labelled `data` in the `filter_output::sequence` are $\mathrm{s}/\mathrm{Hz} = \mathrm{s}^2$

      Pass / Fail

   (b) i. To compare the output data with the expected output, go to `/.../ldasmdc/burst-stochastic/test/expected/` You should see a series of Matlab files. If these files are gzipped, type `make` and all the Matlab files will be unzipped.

      ii. Start up matlab by typing `matlab` and load the file `stochasticbar00001Expected.mat`.

      iii. Load the first output file by typing from within Matlab
      `p = ilwdread('../output/stochasticbar00001_output_1.ilwd');` and extract the spectra by typing
      `output_spectra=p.m1.m4.m9.complex_8;`
      Also, create a corresponding frequency vector
      `f1=[782.5:0.01:1031.49];`

      iv. Calculate the absolute differences in the real and imaginary components like so
      `real_diff=abs(real(CCspec(:,1))-real(output_spectra));`
      `imag_diff=abs(imag(CCspec(:,1))-imag(output_spectra));`
      and find the maximum
      `max_real_diff=max(real_diff);`
      `max_imag_diff=max(imag_diff);`

      v. Now calculate the sum of the real and imaginary components expected output
      `real_tot_expected=sum(real(CCspec(:,1)));`
      `imag_tot_expected=sum(imag(CCspec(:,1)));`
      and likewise for the output files
      `real_tot_database=sum(real(output_spectra));`
      `imag_tot_database=sum(imag(output_spectra));`

      vi. Finally, calculate the fractional errors
      `magn_tot_expected = sqrt (real_tot_expectedˆ2 + imag_tot_expectedˆ2);`
      `max_real_diff/magn_tot_expected`
      `max_imag_diff/magn_tot_expected`

      vii. The fractional error in the real and imaginary parts should be less than 10^{-7}.

      viii. Repeat this for the other 9 output spectra by reading in the appropriate output file
      `p = ilwdread('../output/stochasticbar00001_output_2.ilwd');`
      and incrementing the second index in `CCspec` to the appropriate spectra.
      `real_diff=abs(real(CCspec(:,2))-real(output_spectra));`

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

_____

New faults submitted – list bug IDs: _____

_____

Notes: _____

_____

TEST RESULT **Pass / Fail**

A.10.3 `stochasticbar00020`

Test Case: `stochasticbar00020`

Purpose: Test `libldasstochasticbar.so` behavior for $\sim$ 15 minutes of 920 Hz (13 Hz heterodyned) sine-wave data, filtered assuming flat PSD, flat response and coincident & coaligned detectors

Tester:

Test machine:

Date (mm/dd/yy): **Time:**

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed. It will also require Matlab and the `"ilwdread"` matlab function distributed as part of the `ligotools` package for the data comparison tests.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/stochasticbar00020.ilwd`,it can be found in the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/command/`
   and execute the script
   `./runtest.tclsh stochasticbar00020`
   It will run an MPI job with the following schema, from the file `stochasticbar00020.schema` in the same directory:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: stochasticbar00020.schema,v 1.1 2001/09/06 22:23:20 heng Exp $
#
# libstochasticbar.so should be invoked with the following arguments
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
# argv[18] = allegroLatitude   Latitude of ALLEGRO
# argv[19] = allegroLongitude  Longitude of ALLEGRO
# argv[20] = allegroAzimuth    Azimuth of ALLEGRO
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochasticbar.so" -dataAPI="(datahost,5678)"
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/stochasticba
filterparams="(10,22500,24900,782.5,0.01,120,784,2,L1\:LSC-AS_Q,AL\:GW_CHAN,response1,response2,1,0,0,spectrum1,spectrum2,0,0,0)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/output/`, prepending to each filename the prefix `stochasticbar00020_`.

2. Examine the output files as follows:

   (a) Go to the ouput data directory at `../output/stochasticbar00020_output_1.ilwd`

   i. `start_frequency:summ_spectrum:dataBase` and `start_freq` are 782.5=7.825e+2 Hz
   ii. `stop_frequency:summ_spectrum:dataBase` and `stop_freq` are 1031.49=1.03149e+3 Hz
   iii. `delta_frequency:summ_spectrum:dataBase` and `freq:step_size` are 0.01 Hz
   iv. The type of the output spectrum is `complex_8`.
   v. There are 24900 points in the output frequency series.
   vi. The units of the container labelled `data` in the `filter_output::sequence` are $\mathrm{s}/\mathrm{Hz} = \mathrm{s}^2$

   Pass / Fail

   (b) i. To compare the output data with the expected output, go to `/.../ldasmdc/burst-stochastic/test/expected/` You should see a series of Matlab files. If these files are gzipped, type `make` and all the Matlab files will be unzipped.

   ii. Start up matlab by typing `matlab` and load the file `stochasticbar00020Expected.mat`.

   iii. Load the first output file by typing from within Matlab
   `p = ilwdread('../output/stochasticbar00020_output_1.ilwd');` and extract the spectra by typing
   `output_spectra=p.m1.m4.m9.complex_8;`
   Also, create a corresponding frequency vector by doing
   `f1=[782.5:0.01:1031.49];`

   iv. Calculate the absolute differences in the real and imaginary components like so
   ```
   real_diff=abs(real(CCspec(:,1))-real(output_spectra));
   imag_diff=abs(imag(CCspec(:,1))-imag(output_spectra));
   ```
   and find the maximum
   ```
   max_real_diff=max(real_diff);
   max_imag_diff=max(imag_diff);
   ```

   v. Now calculate the sum of the real and imaginary components expected output
   ```
   real_tot_expected=sum(real(CCspec(:,1)));
   imag_tot_expected=sum(imag(CCspec(:,1)));
   ```
   and likewise for the output files
   ```
   real_tot_database=sum(real(output_spectra));
   imag_tot_database=sum(imag(output_spectra));
   ```

   vi. Finally, calculate the fractional errors
   ```
   magn_tot_expected = sqrt (real_tot_expectedˆ2 + imag_tot_expectedˆ2);
   max_real_diff/magn_tot_expected
   max_imag_diff/magn_tot_expected
   ```

   vii. The fractional error in the real and imaginary parts should be less than 10^{-7}.

viii. Repeat this for the other 9 output spectra by reading in the appropriate output file
`p = ilwdread('../output/stochasticbar00001_output_2.ilwd');`
and incrementing the second index in `CCspec` to the appropriate spectra.
`real_diff=abs(real(CCspec(:,2))-real(output_spectra));`

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

_____

New faults submitted – list bug IDs: _____

_____

Notes: _____

_____

TEST RESULT **Pass / Fail**

A.10.4 `stochasticbar00011`

Test Case: `stochasticbar00011`

Purpose: Test `libldasstochasticbar.so` behavior for ∼ 900 sec of white noise with a non-zero stochastic signal, filtered assuming a flat PSD and response function. The detectors are coincident & coaligned detectors

Tester:

Test machine:

Date (mm/dd/yy): **Time:**

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on `ldas-pcdev1.mit.edu` as user `ldas_mdc` with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed. It will also require Matlab and the `"ilwdread"` matlab function distributed as part of the `ligotools` package for the data comparison tests.

LAM must be running on the machine; if it's not, issue the command
`lamboot -v`.

This test uses the file `input/stochasticbar00011.ilwd`. To make sure all the files needed for the standalone IFO-bar tests are ready, issue the command
`make data`
from the directory
`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/command/`
   and execute the script
   `./runtest.tclsh stochasticbar00011`
   It will run an MPI job with the following schema, from the file `stochasticbar00011.schema` in the same directory:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: stochasticbar00011.schema,v 1.1 2001/09/06 22:23:20 heng Exp $
#
# libstochasticbar.so should be invoked with the following arguments
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
# argv[18] = allegroLatitude   Latitude of ALLEGRO
# argv[19] = allegroLongitude  Longitude of ALLEGRO
# argv[20] = allegroAzimuth    Azimuth of ALLEGRO
#
```

```
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochasticbar.so" -dataAPI="(datahost,5678)"
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/stochasticba
filterparams="(10,22500,24900,782.5,0.01,120,784,2,L1\:LSC-AS_Q,AL\:GW_CHAN,response1,response2,1,0,0,spectrum1,spectrum2,0,0,0)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/output/`, prepending to each filename the prefix `stochasticbar00011_`.

2. Examine the output files as follows:

   (a) Go to the ouput data directory at `../output/stochasticbar00011_output_1.ilwd`

   i. `start_frequency:summ_spectrum:dataBase` and `start_freq` are 782.5=7.825e+2 Hz
   ii. `stop_frequency:summ_spectrum:dataBase` and `stop_freq` are 1031.49=1.03149e+3 Hz
   iii. `delta_frequency:summ_spectrum:dataBase` and `freq:step_size` are 0.01 Hz
   iv. The type of the output spectrum is `complex_8`.
   v. There are 24900 points in the output frequency series.
   vi. The units of the container labelled `data` in the `filter_output::sequence` are $\mathrm{s}/\mathrm{Hz} = \mathrm{s}^2$

   Pass / Fail

   (b) i. To compare the output data with the expected output, go to `/.../ldasmdc/burst-stochastic/test/expected/` You should see a series of Matlab files. If these files are gzipped, type `make` and all the Matlab files will be unzipped.

   ii. Start up matlab by typing `matlab` and load the file `stochasticbar00011Expected.mat`.

   iii. Load the first output file by typing from within Matlab
   `p = ilwdread('../output/stochasticbar00011_output_1.ilwd');` and extract the spectra by typing
   `output_spectra=p.m1.m4.m9.complex_8;`
   Also, create a corresponding frequency vector by doing
   `f1=[782.5:0.01:1031.49];`

   iv. Calculate the absolute differences in the real and imaginary components like so
   `real_diff=abs(real(CCspec(:,1))-real(output_spectra));`
   `imag_diff=abs(imag(CCspec(:,1))-imag(output_spectra));`
   and find the maximum
   `max_real_diff=max(real_diff);`
   `max_imag_diff=max(imag_diff);`

   v. Now calculate the sum of the real and imaginary components expected output
   `real_tot_expected=sum(real(CCspec(:,1)));`
   `imag_tot_expected=sum(imag(CCspec(:,1)));`
   and likewise for the output files
   `real_tot_database=sum(real(output_spectra));`
   `imag_tot_database=sum(imag(output_spectra));`

   vi. Finally, calculate the fractional errors
   `magn_tot_expected = sqrt (real_tot_expectedˆ2 + imag_tot_expectedˆ2);`
   `max_real_diff/magn_tot_expected`
   `max_imag_diff/magn_tot_expected`

vii. The fractional error in the real and imaginary parts should be less than 10^{-7}.

viii. Repeat this for the other 9 output spectra by reading in the appropriate output file
`p = ilwdread('../output/stochasticbar00001_output_2.ilwd');`
and incrementing the second index in `CCspec` to the appropriate spectra.
`real_diff=abs(real(CCspec(:,2))-real(output_spectra));`

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

_____

New faults submitted – list bug IDs: _____

_____

Notes: _____

_____

TEST RESULT **Pass / Fail**

A.10.5 stochasticbar00101

Test Case: stochasticbar00101

Purpose: Test libldasstochasticbar.so behavior for ∼ 15 minutes of white noise, filtered assuming flat PSD, flat response and LLO and ALLEGRO (1996-2000) detector orientations

Tester:

Test machine:

Date (mm/dd/yy): **Time:**

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on ldas-pcdev1.mit.edu as user ldas_mdc with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed. It will also require Matlab and the "ilwdread" matlab function distributed as part of the ligotools package for the data comparison tests.

LAM must be running on the machine; if it's not, issue the command
lamboot -v.

This test uses the file input/stochasticbar00101.ilwd. It is gzip-compressed in the repository and must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-bar tests are ready, issue the command
make data
from the directory
/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/.

PROCEDURE

1. Set the current directory to
   /.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/command/
   and execute the script
   ./runtest.tclsh stochasticbar00101
   It will run an MPI job with the following schema, from the file stochasticbar00101.schema in the same directory:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: stochasticbar00101.schema,v 1.1 2001/09/06 22:23:20 heng Exp $
#
# libstochasticbar.so should be invoked with the following arguments
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
# argv[18] = allegroLatitude   Latitude of ALLEGRO
# argv[19] = allegroLongitude  Longitude of ALLEGRO
```

```
# argv[20] = allegroAzimuth    Azimuth of ALLEGRO
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochasticbar.so" -dataAPI="(datahost,5678)"
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/stochasticba
filterparams="(10,22500,24900,782.5,0.01,120,784,2,L1\:LSC-AS_Q,AL\:GW_CHAN,response1,response2,1,0,0,spectrum1,spectrum2,-
91.2,30.4,-40)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/output/`, prepending to each filename the prefix `stochasticbar00101_`.

2. Examine the output files as follows:

   (a) Go to the ouput data directory at `../output/stochasticbar00101_output_1.ilwd`

   i. `start_frequency:summ_spectrum:dataBase` and `start_freq` are 782.5=7.825e+2 Hz
   ii. `stop_frequency:summ_spectrum:dataBase` and `stop_freq` are 1031.49=1.03149e+3 Hz
   iii. `delta_frequency:summ_spectrum:dataBase` and `freq:step_size` are 0.01 Hz
   iv. The type of the output spectrum is `complex_8`.
   v. There are 24900 points in the output frequency series.
   vi. The units of the container labelled `data` in the `filter_output::sequence` are $\mathrm{s}/\mathrm{Hz} = \mathrm{s}^2$

   Pass / Fail

   (b) i. To compare the output data with the expected output, go to `/.../ldasmdc/burst-stochastic/test/expected/` You should see a series of Matlab files. If these files are gzipped, type `make` and all the Matlab files will be unzipped.

   ii. Start up matlab by typing `matlab` and load the file `stochasticbar00101Expected.mat`.

   iii. Load the first output file by typing from within Matlab
   `p = ilwdread('../output/stochasticbar00101_output_1.ilwd');` and extract the spectra by typing
   `output_spectra=p.m1.m4.m9.complex_8;`
   Also, create a corresponding frequency vector by doing
   `f1=[782.5:0.01:1031.49];`

   iv. Calculate the absolute differences in the real and imaginary components like so
   `real_diff=abs(real(CCspec(:,1))-real(output_spectra));`
   `imag_diff=abs(imag(CCspec(:,1))-imag(output_spectra));`
   and find the maximum
   `max_real_diff=max(real_diff);`
   `max_imag_diff=max(imag_diff);`

   v. Now calculate the sum of the real and imaginary components expected output
   `real_tot_expected=sum(real(CCspec(:,1)));`
   `imag_tot_expected=sum(imag(CCspec(:,1)));`
   and likewise for the output files
   `real_tot_database=sum(real(output_spectra));`
   `imag_tot_database=sum(imag(output_spectra));`

   vi. Finally, calculate the fractional errors
   `magn_tot_expected = sqrt (real_tot_expected^2 + imag_tot_expected^2);`
   `max_real_diff/magn_tot_expected`

```
max_imag_diff/magn_tot_expected
```

vii. The fractional error in the real and imaginary parts should be less than 10^{-7}.

viii. Repeat this for the other 9 output spectra by reading in the appropriate output file
```
p = ilwdread('../output/stochasticbar00001_output_2.ilwd');
```
and incrementing the second index in `CCspec` to the appropriate spectra.
```
real_diff=abs(real(CCspec(:,2))-real(output_spectra));
```

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

_____

New faults submitted – list bug IDs: _____

_____

Notes: _____

_____

TEST RESULT **Pass / Fail**

A.10.6 stochasticbar00111

Test Case: stochasticbar00111

Purpose: Test libldasstochasticbar.so behavior for ∼ 900 sec of white noise, filtered assuming flat PSD and flat response, with LLO and ALLEGRO (1996-2000) orientation.

Tester:

Test machine:

Date (mm/dd/yy): **Time:**

ENVIRONMENT AND PREREQUISITES

During the MDC, this test should be run on ldas-pcdev1.mit.edu as user ldas_mdc with the most up-to-date versions of LAL, LALWrapper, and the PRE05MDC010904 version of wrapperAPI installed. It will also require Matlab and the "ilwdread" matlab function distributed as part of the ligotools package for the data comparison tests.

LAM must be running on the machine; if it's not, issue the command
lamboot -v.

This test uses the file input/stochasticbar00111.ilwd. This files are gzip-compressed in the repository and must be uncompressed in order to run the test. To make sure all the files needed for the standalone IFO-bar tests are ready, issue the command
make data
from the directory
/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/.

PROCEDURE

1. Set the current directory to
   /.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/command/
   and execute the script
   ./runtest.tclsh stochasticbar00111
   It will run an MPI job with the following schema, from the file stochasticbar00111.schema in the same directory:

```
#
# lam boot schema to run the stochastic shared object
#
# $Id: stochasticbar00111.schema,v 1.1 2001/09/06 22:23:21 heng Exp $
#
# libstochasticbar.so should be invoked with the following arguments
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies in optimal filter
# argv[4]  = f0                Start frequency for optimal filter
# argv[5]  = deltaF            Frequency spacing for optimal filter
# argv[6]  = outputNumFreq     Number of frequencies in CC spectrum
# argv[7]  = outputF0          Start frequency for CC spectrum
# argv[8]  = outputDeltaF      Frequency spacing for CC spectrum
# argv[9]  = dmro1name         Name of first data channel
# argv[10] = dmro2name         Name of second data channel
# argv[11] = response1name     Name of first response fcn
# argv[12] = response2name     Name of second response fcn
# argv[13] = epochsMatch       Check that time streams are simultaneous
# argv[14] = siteID1           Index of first detector in lalCachedDetectors
# argv[15] = siteID2           Index of second detector in lalCachedDetectors
# argv[16] = spectrum1name     Name of first PSD
# argv[17] = spectrum2name     Name of second PSD
# argv[18] = allegroLatitude   Latitude of ALLEGRO
# argv[19] = allegroLongitude  Longitude of ALLEGRO
```

```
# argv[20] = allegroAzimuth    Azimuth of ALLEGRO
#
h -np 2 wrapperAPI -mpiAPI="(localhost,1000)" -nodelist="(1)" -dynlib="/ldcg/lib/lalwrapper/libldasstochasticbar.so" -dataAPI="(datahost,5678)"
resultAPI"=(eventmon,1002)" -realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8 -uniqueID=9.0 -inputFile="../input/stochasticba
filterparams="(10,22500,24900,782.5,0.01,120,784,2,L1\:LSC-AS_Q,AL\:GW_CHAN,response1,response2,1,0,0,spectrum1,spectrum2,-
91.2,30.4,-40)"
```

and then move the output and process ILWDs into the directory `/.../ldasmdc/burst-stochastic/test/stochastic_ifo-bar/output/`, prepending to each filename the prefix `stochasticbar00111_`.

2. Examine the output files as follows:

   (a) Go to the ouput data directory at `../output/stochasticbar00111_output_1.ilwd`

   i. `start_frequency:summ_spectrum:dataBase` and `start_freq` are 782.5=7.825e+2 Hz
   ii. `stop_frequency:summ_spectrum:dataBase` and `stop_freq` are 1031.49=1.03149e+3 Hz
   iii. `delta_frequency:summ_spectrum:dataBase` and `freq:step_size` are 0.01 Hz
   iv. The type of the output spectrum is `complex_8`.
   v. There are 24900 points in the output frequency series.
   vi. The units of the container labelled `data` in the `filter_output::sequence` are $\mathrm{s}/\mathrm{Hz} = \mathrm{s}^2$

   Pass / Fail

   (b) i. To compare the output data with the expected output, go to `/.../ldasmdc/burst-stochastic/test/expected/` You should see a series of Matlab files. If these files are gzipped, type `make` and all the Matlab files will be unzipped.

   ii. Start up matlab by typing `matlab` and load the file `stochasticbar00111Expected.mat`.

   iii. Load the first output file by typing from within Matlab
   `p = ilwdread('../output/stochasticbar00111_output_1.ilwd');` and extract the spectra by typing
   `output_spectra=p.m1.m4.m9.complex_8;`
   Also, create a corresponding frequency vector by doing
   `f1=[782.5:0.01:1031.49];`

   iv. Calculate the absolute differences in the real and imaginary components like so
   `real_diff=abs(real(CCspec(:,1))-real(output_spectra));`
   `imag_diff=abs(imag(CCspec(:,1))-imag(output_spectra));`
   and find the maximum
   `max_real_diff=max(real_diff);`
   `max_imag_diff=max(imag_diff);`

   v. Now calculate the sum of the real and imaginary components expected output
   `real_tot_expected=sum(real(CCspec(:,1)));`
   `imag_tot_expected=sum(imag(CCspec(:,1)));`
   and likewise for the output files
   `real_tot_database=sum(real(output_spectra));`
   `imag_tot_database=sum(imag(output_spectra));`

   vi. Finally, calculate the fractional errors
   `magn_tot_expected = sqrt (real_tot_expected^2 + imag_tot_expected^2);`
   `max_real_diff/magn_tot_expected`

```
max_imag_diff/magn_tot_expected
```

vii. The fractional error in the real and imaginary parts should be less than 10^{-7}.

viii. Repeat this for the other 9 output spectra by reading in the appropriate output file
```
p = ilwdread('../output/stochasticbar00001_output_2.ilwd');
```
and incrementing the second index in `CCspec` to the appropriate spectra.
```
real_diff=abs(real(CCspec(:,2))-real(output_spectra));
```

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

_____

New faults submitted – list bug IDs: _____

_____

Notes: _____

_____

TEST RESULT **Pass / Fail**

A.11 Stochastic pipeline tests

The tests in this section verify the behavior of the shared object `libldasstochastic.so` within the full LDAS pipeline.

The numbering system for the tests `PIPESTOCHIFOIFO####` is as follows:

- The first digit (0/1) indicates the length of the time series as well as the frequency series sampling parameters.
- The second digit (0/9) indicates whether the data are trivial (a 1 at the start of each segment followed by a string of 0s) or actual engineering run data.
- The third digit (0/1) indicates whether the PSDs used in constructing the filter are supplied as auxiliary data or calculated from the time series data.
- The fourth digit (0/1) indicates whether the simulation and analysis are performed for coïncident and coäligned detectors, or for correlations between LIGO Hanford and LIGO Livingston

The input data should reside on the LDAS system on which the pipeline jobs are run, in the form of frames and ILWDs; they were created indirectly from the following datasets (which are created by the matlab code in the directory `../stochastic_ifo-ifo/src`)

- `src/dmro_trivial.488815210.10x92138.1024.dat` Trivial time series data consisting of 10 identical segments of $92138 = (3^4 * 5^2 * 7^1 * 13 + 1)/2$ points each; each segment has a 1 in the first element, with 0s in the rest; the dataset contains 102380 points in all, representing 99 seconds, 980468750 nanoseconds of data sampled at 1024 Hz. The GPS start time for this data is 488815210 seconds, 0 nanoseconds (1995 July 3 07:00 PDT).
- `src/resp_ident.40-500.1_4.dat` A complex frequency series containing a trivial response function, with every element having a real part of 1 and an imaginary part of 0, containing 1840 points with a start frequency of 40 Hz and a frequency spacing of 0.25 Hz.
- `src/spec_white.-512-512.1024_92138.dat` A trivial power spectral density consisting of all 1s, containing 4096 points with a start frequency of -512 Hz and a frequency spacing of 0.25 Hz.

Additionally, the file `../stochastic_ifo-ifo/input/STOCHASTICIFOIFOSO00000.ilwd`, described in section A.9.2, is used to generate data. All of the PIPESTOCHIFOIFO#### tests retrieve their data products with the script
`command/fetchproducts.tclsh`, which has the following source:

```
#!/ldcg/bin/tclsh
##
## $Id: fetchproducts.tclsh,v 1.4 2001/09/08 22:11:07 whelan Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  set output [ read $sid ]
  close $sid
  return $output
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
```

```
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set job     [ lindex $argv 0 ]
set test     [ lindex $argv 1 ]

if { $argc <= 2 || [ lindex $argv 2 ] != "blob" } {
    set file    jobs/$job/results.F
    set subject "proc frame data from job $job now ready to be retrieved as ASCII ILWD"

    set cmd "ldasJob {-name $user -password $pwrd -email $email } {getFrameData
    -subject \{$subject\} -returnprotocol http:/results -outputformat
    \{ilwd ascii\} -frames $file -framequery full(0) }"

    puts $cmd

    set msg [ sendCmd $cmd $host $port ]

    regexp {NORMAL\d+} $msg framefid

    puts "Conversion of proc frame data to ASCII ILWD initiated by command"
    puts ""
    puts $cmd
    puts ""
    puts "proceeding with ack msg"
    puts ""
    puts $msg
    puts ""
}
if { $argc <= 2 || [ lindex $argv 2 ] != "frame" } {
    regexp {\d+} $job jobid
    set subject "database summ_spectrum data from job $job now ready to be retrieved as ASCII ILWD"

    set cmd "ldasJob {-name $user -password $pwrd -email $email } {getMetaData
    -subject \{$subject\} -returnprotocol http:/results -returnformat \{ilwd
    ascii\} -sqlquery \{select * from summ_spectrum where process_id in (select
    process_id from process where jobid=$jobid) order by start_time,start_time_ns
    for read only\} }"

    set msg [ sendCmd $cmd $host $port ]

    regexp {NORMAL\d+} $msg blobfid

    puts ""
    puts "Conversion of database summ_spectrum data to ASCII ILWD initiated by command"
    puts ""
    puts $cmd
    puts ""
    puts "proceeding with ack msg"
    puts ""
    puts $msg
    puts ""
}
if { $argc <= 2 || [ lindex $argv 2 ] != "blob" } {
    set file $test
    append file "_results.F"
    set cmd "lynx --source http://$host/ldas_outgoing/jobs/$job/results.F > ../output/$file"
    puts "Note that you can retrieve the original output frame file"
    puts "(at any time) with the command"
    puts ""
    puts $cmd
    puts ""

    puts "Once you receive an email at $email telling you that job $framefid has"
    puts "completed successfully, you can retrieve the data with the command"
    puts ""

    set file $test
    append file "_frame.ilwd"
    puts "lynx --source http://$host/ldas_outgoing/jobs/$framefid/results.ilwd > ../output/$file"
    puts ""
}
if { $argc <= 2 || [ lindex $argv 2 ] != "frame" } {
    puts "Once you receive an email at $email telling you that job $blobfid has"
    puts "completed successfully, you can retrieve the data with the command"
    puts ""

    set file $test
    append file "_database.ilwd"
    puts "lynx --source http://$host/ldas_outgoing/jobs/$blobfid/results.ilwd > ../output/$file"
    puts ""
}
puts "(Be sure you do not have 'noclobber' set.)"
```

The tests which analyze trivial data, namely PIPESTOCHIFOIFO0000, PIPESTOCHIFOIFO1000, and

PIPESTOCHIFOIFO1010, compare the data in their output frames to the expected outputs with the script `command/trivdatadiff.tclsh`, which has the following source:

```
#!/ldcg/bin/tclsh
# $Id: trivdatadiff.tclsh,v 1.2 2001/09/08 14:41:29 whelan Exp $
# Script to compare ILWD outputs to expected output

set test [lindex $argv 0]

set verbose [lindex $argv 1]

set filename ../output/$test
append filename _frame.ilwd

set file [open $filename]

set framedata ""
set i 0

while { [gets $file line] != -1 } {
    if { [regexp {\<complex_8[^\>]+\>([e\d\+\-. ]+)\<} $line -> data] } {
incr i
puts "reading spectrum #$i from frame..."
if { $framedata != "" } {
    if { $data != $framedata } {
puts "$i-th frame entry "
puts $data
puts "differs from first entry"
puts $framedata
    }
} else {
    set framedata $data
}
    }
}

close $file

set framearray [split $framedata]

# Now read in BLObs from database ILWD

set filename ../output/$test
append filename _database.ilwd

set file [open $filename]

puts "extracting BLObs from $filename ..."

set blobdata ""
set i 1

while { [gets $file line] != -1 } {
    if { [regexp {\<char_u[^\>]+\>((\\\d{3})+)?\</char_u\>} $line -> binarydata] } {
if { $blobdata != "" } {
    if { $data != $blobdata } {
puts "$i-th database entry "
puts $data
puts "differs from first entry"
puts $blobdata
    }
    incr i
} else {
    set blobdata $data
}
    }
}

binary scan [subst $binarydata] f* blobdata

set blobarray [split $blobdata]

set filename ../expected/$test
append filename _results.dat
set expfile [open $filename]

set exparrayreal [list]
set exparrayimag [list]

while {[gets $expfile line] >= 0} {
    if {[scan $line "%f %f" real imag] != 2} {
  puts "error reading"
return 1
    }
    lappend exparrayreal $real
    lappend exparrayimag $imag
}

close $expfile

if {[llength $framearray] != [llength $blobarray]} {
  puts "error reading"
```

```
return 1
}

set num [ expr [llength $framearray] / 2 ]

if {$num != [llength $exparrayreal]} {
  puts "error reading"
return 1
}

if {$num != [llength $exparrayimag]} {
  puts "error reading"
return 1
}

set realframediff -1
set imagframediff -1
set realblobdiff -1
set imagblobdiff -1

set realframetot 0
set imagframetot 0
set realblobtot 0
set imagblobtot 0
set realexptot 0
set imagexptot 0

for {set i 0} {$i < $num} {incr i} {
    set realframe [lindex $framearray [expr 2 * $i] ]
    set imagframe [lindex $framearray [expr 2 * $i + 1] ]
    set realblob [lindex $blobarray [expr 2 * $i] ]
    set imagblob [lindex $blobarray [expr 2 * $i + 1] ]
    set realexp [lindex $exparrayreal $i]
    set imagexp [lindex $exparrayimag $i]

    set realexptot [expr $realexptot + $realexp]
    set imagexptot [expr $imagexptot + $imagexp]

    set realblobtot [expr $realblobtot + $realblob]
    set imagblobtot [expr $imagblobtot + $imagblob]

    set realframetot [expr $realframetot + $realframe]
    set imagframetot [expr $imagframetot + $imagframe]

# Compare data in frame output to data in expected output
    set diff [expr abs($realframe - $realexp)]
    if {$diff > $realframediff} {
set realframediff $diff
set realframeind $i
    }
    if {$verbose != ""} {
puts "$i exp $realexp frame $realframe diff [expr $realframe - $realexp]"
    }
    set diff [expr abs($imagframe - $imagexp)]
    if {$diff > $imagframediff} {
set imagframediff $diff
set imagframeind $i
    }
# Compare data in blob output to data in expected output
    set diff [expr abs($realblob - $realexp)]
    if {$diff > $realblobdiff} {
set realblobdiff $diff
set realblobind $i
    }
    if {$verbose != ""} {
puts "$i exp $realexp blob $realblob diff [expr $realblob - $realexp]"
    }
    set diff [expr abs($imagblob - $imagexp)]
    if {$diff > $imagblobdiff} {
set imagblobdiff $diff
set imagblobind $i
    }

}
puts "Compared $num entries"
puts "Expected total $realexptot + i $imagexptot"
puts "Actual total in frame output is $realframetot + i $imagframetot"
puts "Maximum error in real frame output is $realframediff at $realframeind"
puts "Maximum error in imag frame output is $imagframediff at $imagframeind"
puts "Actual total in database output is $realblobtot + i $imagblobtot"
puts "Maximum error in real database output is $realblobdiff at $realblobind"
puts "Maximum error in imag database output is $imagblobdiff at $imagblobind"
puts "====================================="
puts "Fractional error in database total is [expr ($realblobtot - $realexptot)/abs($realexptot)]\
+ i [expr ($imagblobtot - $imagexptot)/abs($realexptot)]"
puts "(max real database err) / (real total) = [expr $realblobdiff / abs($realexptot)]"
puts "(max imag database err) / (real total) = [expr $imagblobdiff / abs($realexptot)]"
puts "================================="
puts "Fractional error in frame total is [expr ($realframetot - $realexptot)/abs($realexptot)]\
+ i [expr ($imagframetot - $imagexptot)/abs($realexptot)]"
puts "(max real frame err) / (real total) = [expr $realframediff / abs($realexptot)]"
puts "(max imag frame err) / (real total) = [expr $imagframediff / abs($realexptot)]"
```

these same tests compare the data in their output frames to the data they write to the metadata database with the script `command/trivdatacomp.tclsh`, which has the following source:

```
#!/ldcg/bin/tclsh
# $Id: trivdatacomp.tclsh,v 1.2 2001/09/08 14:41:29 whelan Exp $
# Script to compare ILWD outputs to each other

set test [lindex $argv 0]

set verbose [lindex $argv 1]

set filename ../output/$test
append filename _frame.ilwd

set file [open $filename]

set framedata ""
set i 0

while { [gets $file line] != -1 } {
    if { [regexp {\<complex_8[^\>]+\>([e\d\+\-. ]+)\<} $line -> data] } {
incr i
puts "reading spectrum #$i from frame..."
if { $framedata != "" } {
    if { $data != $framedata } {
puts "$i-th frame entry "
puts $data
puts "differs from first entry"
puts $framedata
    }
} else {
    set framedata $data
}
    }
}

close $file

set framearray [split $framedata]

# Now read in BLObs from database ILWD

set filename ../output/$test
append filename _database.ilwd

set file [open $filename]

puts "extracting BLObs from $filename ..."

set blobdata ""
set i 1

gets $file line

while { ![regexp {SPECTRUM} $line ] } {
    gets $file line
}

while { [gets $file line] != -1 } {
    if { [regexp {\<char_u[^\>]+\>((\\\d{3})+)?\</char_u\>} $line -> binarydata] } {
#   puts $i
#   puts $binarydata
if { $blobdata != "" } {
    if { $binarydata != $blobdata } {
puts "$i-th database entry "
puts $binarydata
puts "differs from first entry"
puts $blobdata
       } else {
   puts "$i-th database entry same as first entry"
#   puts $blobdata
    }
} else {
    set blobdata $binarydata
}
incr i
    }
}

binary scan [subst $binarydata] f* blobdata

set blobarray [split $blobdata]

set filename ../expected/$test
append filename _results.dat

if {[llength $framearray] != [llength $blobarray]} {
  puts "error reading"
return 1
}

set num [ expr [llength $framearray] / 2 ]
```

```
set realdiff -1
set imagdiff -1

set realframetot 0
set imagframetot 0
set realblobtot 0
set imagblobtot 0

for {set i 0} {$i < $num} {incr i} {
    set realframe [lindex $framearray [expr 2 * $i] ]
    set imagframe [lindex $framearray [expr 2 * $i + 1] ]
    set realblob [lindex $blobarray [expr 2 * $i] ]
    set imagblob [lindex $blobarray [expr 2 * $i + 1] ]

    set realblobtot [expr $realblobtot + $realblob]
    set imagblobtot [expr $imagblobtot + $imagblob]

    set realframetot [expr $realframetot + $realframe]
    set imagframetot [expr $imagframetot + $imagframe]

# Compare data in frame output to data in blob output
    set diff [expr abs($realframe - $realblob)]
    if {$diff > $realdiff} {
set realdiff $diff
set realind $i
    }
    if {$verbose != ""} {
puts "$i blob $realblob frame $realframe diff [expr $realframe - $realblob]"
    }
    set diff [expr abs($imagframe - $imagblob)]
    if {$diff > $imagdiff} {
set imagdiff $diff
set imagind $i
    }
}
puts "Compared $num entries"
puts "Actual total in frame output is $realframetot + i $imagframetot"
puts "Actual total in database output is $realblobtot + i $imagblobtot"
puts "Maximum discrepancy in real output is $realdiff at $realind"
puts "Maximum discrepancy in imag output is $imagdiff at $imagind"
puts "================================="
puts "Fractional discrepancy in total is [expr ($realframetot - $realblobtot)/abs($realframetot)]\
+ i [expr ($imagblobtot - $imagframetot)/abs($realframetot)]"
puts "(max real disc) / (real total) = [expr $realdiff / abs($realframetot)]"
puts "(max imag disc) / (real total) = [expr $imagdiff / abs($realframetot)]"
```

Test PIPESTOCHIFOIFO1911, which operates on engineering run frame data, compares the data in its output frames to the data it writes to the metadata database with the script `command/multidatacomp.tclsh`, which has the following source:

```
#!/ldcg/bin/tclsh
# $Id: multidatacomp.tclsh,v 1.2 2001/09/08 14:41:29 whelan Exp $
# Script to compare ILWD outputs to each other

set test [lindex $argv 0]

set verbose [lindex $argv 1]

set filename ../output/$test
append filename _frame.ilwd

set file [open $filename]

set framearrays [list]
set i 0

while { [gets $file line] != -1 } {
    if { [regexp {\<complex_8[^\>]+\>([e\d\+\-. ]+)\<} $line -> data] } {
incr i
puts "reading spectrum #$i from frame..."
lappend framearrays [split $data]
    }
}

close $file

# Now read in BLObs from database ILWD

set filename ../output/$test
append filename _database.ilwd

set file [open $filename]

puts "extracting BLObs from $filename ..."

set blobarrays [list]
set i 0

gets $file line
```

```
while { ![regexp {SPECTRUM} $line ] } {
    gets $file line
}

while { [gets $file line] != -1 } {
    if { [regexp {\<char_u[^\>]+\>((\\\d{3})+)?\</char_u\>} $line -> binarydata] } {
# puts $i
# puts $binarydata
incr i
puts "reading spectrum #$i from database..."
binary scan [subst $binarydata] f* data
lappend blobarrays [split $data]
    }
}

set filename ../expected/$test
append filename _results.dat

set numsegs [llength $framearrays]

if {$numsegs != [llength $blobarrays]} {
  puts "error reading"
return 1
}

for {set n 0} {$n < $numsegs} {incr n} {
    set framearray [ lindex $framearrays $n ]
    set blobarray [ lindex $blobarrays $n ]

    if {[llength $framearray] != [llength $blobarray]} {
  puts "error reading"
return 1
    }

    set num [ expr [llength $framearray] / 2 ]

    set realdiff -1
    set imagdiff -1

    set realframetot 0
    set imagframetot 0
    set realblobtot 0
    set imagblobtot 0

    for {set i 0} {$i < $num} {incr i} {
set realframe [lindex $framearray [expr 2 * $i] ]
set imagframe [lindex $framearray [expr 2 * $i + 1] ]
set realblob [lindex $blobarray [expr 2 * $i] ]
set imagblob [lindex $blobarray [expr 2 * $i + 1] ]

set realblobtot [expr $realblobtot + $realblob]
set imagblobtot [expr $imagblobtot + $imagblob]

set realframetot [expr $realframetot + $realframe]
set imagframetot [expr $imagframetot + $imagframe]

# Compare data in frame output to data in blob output
set diff [expr abs($realframe - $realblob)]
if {$diff > $realdiff} {
    set realdiff $diff
    set realind $i
}
if {$verbose != ""} {
    puts "$i blob $realblob frame $realframe diff [expr $realframe - $realblob]"
}
set diff [expr abs($imagframe - $imagblob)]
if {$diff > $imagdiff} {
    set imagdiff $diff
    set imagind $i
}
    }
    puts ""
    puts "Compared $num entries in [expr $n + 1]-th entry"
    puts "Actual total in frame output is $realframetot + i $imagframetot"
    puts "Actual total in database output is $realblobtot + i $imagblobtot"
    puts "Maximum discrepancy in real output is $realdiff at $realind"
    puts "Maximum discrepancy in imag output is $imagdiff at $imagind"
    puts "================================="
    puts "Fractional discrepancy in total is [expr ($realframetot - $realblobtot)/abs($realframetot)]\
+ i [expr ($imagblobtot - $imagframetot)/abs($realframetot)]"
    puts "(max real disc) / (real total) = [expr $realdiff / abs($realframetot)]"
    puts "(max imag disc) / (real total) = [expr $imagdiff / abs($realframetot)]"
}
```

A.11.1 PIPELINE00

Test Case: PIPELINE00

Purpose: Verify that trivial data can pass unmolested through LDAS

Tester: Warren G. Anderson

Test machine: My laptop

Date (mm/dd/yy): 09/10/01 **Time:** 8:35 AM - 8:45 AM

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS. It depends on the successful of completion of all test checklists in section A.1.

PROCEDURE

Run the test DATAPIPELINE01 from section 02ldaspipe of the inspiral MDC. Ensure that all pass criteria are met.

1. DATAPIPELINE01

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.11.2 PIPESTOCHIFOIFOERR

Test Case: PIPESTOCHIFOIFOERR

Purpose: Tests that `libldasstochastic.so` correctly rejects improper search parameters in the LDAS environment.

Tester: Joseph D. Romano

Test machine: My laptop

Date (mm/dd/yy): 09/09/01 **Time:** 14:46 EDT

This test checks that the error conditions from test STOCHASTICIFOIFOSOERR in section A.9.1 also generate the correct error messages within the LDAS pipeline, the job is shut down cleanly, and the errors are logged.

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS using the tcl scripts:

```
PIPESTOCHIFOIFOERR00.tclsh
PIPESTOCHIFOIFOERR01.tclsh
PIPESTOCHIFOIFOERR02.tclsh
PIPESTOCHIFOIFOERR03.tclsh
PIPESTOCHIFOIFOERR04.tclsh
PIPESTOCHIFOIFOERR05.tclsh
PIPESTOCHIFOIFOERR06.tclsh
PIPESTOCHIFOIFOERR07.tclsh
PIPESTOCHIFOIFOERR08.tclsh
PIPESTOCHIFOIFOERR09.tclsh
PIPESTOCHIFOIFOERR10.tclsh
PIPESTOCHIFOIFOERR11.tclsh
```

At least two frames and four input ilwd files are required, although the contents should be irrelevant since the shared object is not expected to carry out any analysis. For this purpose, all tests will use the input files used in the pipeline test `PIPESTOCHIFOIFO0000`.

This test depends on the successful of completion of all test checklists in section A.1 and in section A.9.1.

PROCEDURE

Set the current directory to

```
/.../ldasmdc/burst-stochastic/test/pipeline/command/
```

and perform the following tests:

1. Test that the shared object correctly rejects input with the wrong number of arguments by executing the command:
   `./PIPESTOCHIFOIFOERR00.tclsh`
   This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR00
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR00.tclsh,v 1.1 2001/09/06 18:10:43 warren Exp $
##
```

```
## ****************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##########################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR00

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR00"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
```

```
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1

# other inputs used by the data-conditioning api
set responsefiles "
                  file:$spec1file,pass
                  file:$spec2file,pass
                  file:$resp1file,pass
                  file:$resp2file,pass
     "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                  gwH1 = $ch1alias;
                  gwH2 = $ch2alias;
     "

set algorithms "
                  slice(gwH1, 0, $totalpoints, 1);
                  intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                  slice(gwH2, 0, $totalpoints, 1);
     "

set framequery "
                  { P {} $times Adc($channel1) }
                  { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Wrong number of arguments`.

Pass

2. Test that the shared object correctly rejects input that does not specify a positive number of segments to process by executing the command:
`./PIPESTOCHIFOIFOERR01.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR01
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR01.tclsh,v 1.1 2001/09/06 18:10:43 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR01

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR01"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
```

```
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams -32,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                  file:$spec1file,pass
                  file:$spec2file,pass
                  file:$resp1file,pass
                  file:$resp2file,pass
     "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                  gwH1 = $ch1alias;
                  gwH2 = $ch2alias;
     "

set algorithms "
                  slice(gwH1, 0, $totalpoints, 1);
                  intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                  slice(gwH2, 0, $totalpoints, 1);
     "

set framequery "
                  { P {} $times Adc($channel1) }
                  { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
```

```
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Number of data segments is zero or negative`.

Pass

3. Test that the shared object correctly rejects input files that do not specify a positive number of data points to analyze by executing the command:
`./PIPESTOCHIFOIFOERR02.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR02
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR02.tclsh,v 1.1 2001/09/06 18:10:43 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##########################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR02

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR02"
```

```
# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,-32,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                file:$spec1file,pass
                file:$spec2file,pass
                file:$resp1file,pass
                file:$resp2file,pass
    "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                gwH1 = $ch1alias;
                gwH2 = $ch2alias;
    "

set algorithms "
                slice(gwH1, 0, $totalpoints, 1);
                intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                slice(gwH2, 0, $totalpoints, 1);
    "

set framequery "
```

```
                { P {} $times Adc($channel1) }
                { Q {} $times Adc($channel2) }
    "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Number of points in a data segment is zero or negative`.

Pass

4. Test that the shared object correctly rejects input not requesting a positive number of frequencies for the optimal filter by executing the command:
`./PIPESTOCHIFOIFOERR03.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR03
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR03.tclsh,v 1.1 2001/09/06 18:10:43 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
```

```
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR03

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR03"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,0,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
```

```
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                  file:$spec1file,pass
                  file:$spec2file,pass
                  file:$resp1file,pass
                  file:$resp2file,pass
     "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                  gwH1 = $ch1alias;
                  gwH2 = $ch2alias;
     "

set algorithms "
                  slice(gwH1, 0, $totalpoints, 1);
                  intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                  slice(gwH2, 0, $totalpoints, 1);
     "

set framequery "
                  { P {} $times Adc($channel1) }
                  { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Number of frequencies is zero or negative`.

Pass

5. Test that the shared object correctly rejects input requesting a negative starting frequency for the optimal filter by executing the command:
`./PIPESTOCHIFOIFOERR04.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ****************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR04
##
## The dataPipeline user command documentation is found at:
```

```
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR04.tclsh,v 1.1 2001/09/06 18:10:43 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR04

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR04"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2
```

```
# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,-32,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                  file:$spec1file,pass
                  file:$spec2file,pass
                  file:$resp1file,pass
                  file:$resp2file,pass
     "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                  gwH1 = $ch1alias;
                  gwH2 = $ch2alias;
     "

set algorithms "
                  slice(gwH1, 0, $totalpoints, 1);
                  intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                  slice(gwH2, 0, $totalpoints, 1);
     "

set framequery "
                  { P {} $times Adc($channel1) }
                  { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job.

The test will have passed if this these messages contain the phrase:
`ABORT: Start frequency is negative`.

Pass

6. Test that the shared object correctly rejects input that does not specify a positive frequency bin size for the optimal filter by executing the command:
`./PIPESTOCHIFOIFOERR05.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR05
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR05.tclsh,v 1.1 2001/09/06 18:10:43 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR05

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR05"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
```

```
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,$flow,-32,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                file:$spec1file,pass
                file:$spec2file,pass
                file:$resp1file,pass
                file:$resp2file,pass
    "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                gwH1 = $ch1alias;
                gwH2 = $ch2alias;
    "

set algorithms "
                slice(gwH1, 0, $totalpoints, 1);
                intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                slice(gwH2, 0, $totalpoints, 1);
    "

set framequery "
                { P {} $times Adc($channel1) }
                { Q {} $times Adc($channel2) }
    "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
```

```
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Frequency spacing is zero or negative`.

Pass

7. Test that the shared object correctly rejects input that does not specify a positive number of frequencies for the cross-correlation spectral densities by executing the command:
`./PIPESTOCHIFOIFOERR06.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR06
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR06.tclsh,v 1.1 2001/09/06 18:10:43 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR06

# this will be used as the subject for the returned e-mail,
```

```
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR06"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,-3,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                file:$spec1file,pass
                file:$spec2file,pass
                file:$resp1file,pass
                file:$resp2file,pass
    "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                gwH1 = $ch1alias;
                gwH2 = $ch2alias;
    "

set algorithms "
                slice(gwH1, 0, $totalpoints, 1);
                intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                slice(gwH2, 0, $totalpoints, 1);
    "
```

```
set framequery "
                  { P {} $times Adc($channel1) }
                  { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Number of frequencies is zero or negative`.

Pass

8. Test that the shared object correctly rejects input that specifies a negative starting frequency for the cross-correlation spectral densities by executing the command:
`./PIPESTOCHIFOIFOERR07.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR07
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR07.tclsh,v 1.1 2001/09/06 18:10:43 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
```

```
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR07

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR07"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name
```

```
set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams -80,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                  file:$spec1file,pass
                  file:$spec2file,pass
                  file:$resp1file,pass
                  file:$resp2file,pass
     "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                  gwH1 = $ch1alias;
                  gwH2 = $ch2alias;
     "

set algorithms "
                  slice(gwH1, 0, $totalpoints, 1);
                  intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                  slice(gwH2, 0, $totalpoints, 1);
     "

set framequery "
                  { P {} $times Adc($channel1) }
                  { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Start frequency is negative`.

Pass

9. Test that the shared object correctly rejects input that specifies a negative starting frequency for the cross-correlation spectral densities by executing the command:
`./PIPESTOCHIFOIFOERR08.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR08
```

```
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR08.tclsh,v 1.1 2001/09/06 18:10:43 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

###########################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR08

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR08"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
```

```
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,-64,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                  file:$spec1file,pass
                  file:$spec2file,pass
                  file:$resp1file,pass
                  file:$resp2file,pass
     "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                  gwH1 = $ch1alias;
                  gwH2 = $ch2alias;
     "

set algorithms "
                  slice(gwH1, 0, $totalpoints, 1);
                  intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                  slice(gwH2, 0, $totalpoints, 1);
     "

set framequery "
                  { P {} $times Adc($channel1) }
                  { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Frequency spacing is zero or negative`.

Pass

10. Test that the shared object correctly rejects input that specifies a detector site number out of range for detector one by executing the command:
`./PIPESTOCHIFOIFOERR09.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR09
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR09.tclsh,v 1.1 2001/09/06 18:10:44 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##########################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR09

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR09"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
```

```
# argv[5]  = deltaF           Frequency spacing
# argv[6]  = dmro1name        Name of first data channel
# argv[7]  = dmro2name        Name of second data channel
# argv[8]  = response1name    Name of first response fcn
# argv[9]  = response2name    Name of second response fcn
# argv[10]  = siteID1         Index of first detector in lalCachedDetectors
# argv[11]  = siteID2         Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name   Name of first PSD
# argv[13]  = spectrum2name   Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,250,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                file:$spec1file,pass
                file:$spec2file,pass
                file:$resp1file,pass
                file:$resp2file,pass
    "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                gwH1 = $ch1alias;
                gwH2 = $ch2alias;
    "

set algorithms "
                slice(gwH1, 0, $totalpoints, 1);
                intermediate(,mpi,gwH1,H1 gravitational wave channel!);
                slice(gwH2, 0, $totalpoints, 1);
    "

set framequery "
                { P {} $times Adc($channel1) }
                { Q {} $times Adc($channel2) }
    "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
```

```
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Detector Site ID out of range`.

Pass

11. Test that the shared object correctly rejects input that specifies a detector site number out of range for detector two by executing the command:
`./PIPESTOCHIFOIFOERR10.tclsh`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFOERR10
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFOERR10.tclsh,v 1.1 2001/09/06 18:10:44 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  puts [ read $sid ]
  close $sid
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFOERR10
```

```
# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test PIPESTOCHIFOIFOERR10"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 spectrum1
set spectrum2 spectrum2

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,-1,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                file:$spec1file,pass
                file:$spec2file,pass
                file:$resp1file,pass
                file:$resp2file,pass
    "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                gwH1 = $ch1alias;
                gwH2 = $ch2alias;
    "

set algorithms "
                slice(gwH1, 0, $totalpoints, 1);
                intermediate(,mpi,gwH1,H1 gravitational wave channel!);
```

```
                slice(gwH2, 0, $totalpoints, 1);
     "

set framequery "
                { P {} $times Adc($channel1) }
                { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget mpi
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

puts $cmd
#exit

## BOILERPLATE NOT MODIFIED PER TEST

sendCmd $cmd $host $port

## END OF BOILERPLATE
```

Error messages should be generated in the log files and also mailed in the e-mail report for the job. The test will have passed if this these messages contain the phrase:
`ABORT: Detector Site ID out of range`.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes: I had to rerun `./PIPESTOCHASTIC09.tclsh`, since it initially generated a bad_list element error. This was probably due to the truncation of the command by the managerAPI.

TEST RESULT **Pass**

A.11.3 PIPESTOCHIFOIFO0000

Test Case: PIPESTOCHIFOIFO0000

Purpose: Verify that the data from test STOCHASTICIFOIFOSO00000 produces the correct output when run with synthetically generated frames and auxiliary ILWD files

Tester: Joseph D. Romano

Test machine: My laptop

Date (mm/dd/yy): 09/09/01 **Time:** 16:32 EDT

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS. It depends on the successful completion of all test checklists in section A.1. The `datadiff.tclsh` and `datacomp.tclsh` need to be run on a machine with the same architecture (little-endian or big-endian) as the nodes in the LDAS beowulf. Since those are all linux boxes, this means the test should be run from a little-endian machine like a linux box, rather than a big-endian system like Solaris.

The input data and filter parameters are identical to those used in test STOCHASTICIFOIFOSO00000 described in section A.9.2. The power spectral densities and response functions were extracted by hand from the file
`../stochastic_ifo-ifo/input/STOCHASTICIFOIFOSO00000.ilwd` into the files
`input/resp1_ident.40-500.2.ilwd`
`input/resp2_ident.40-500.2.ilwd`
`input/spec1_white.-512-512.2.ilwd`
`input/spec2_white.-512-512.2.ilwd`
which are included for reference in the repository in gzip-compressed form, in the directory
`/.../ldasmdc/burst-stochastic/test/pipeline/src/`
These files need to reside on the LDAS machine where the job is run, in the directory
`/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/`
During the MDC, their existence can be confirmed by pointing a web browser at
`ftp://ldas.mit.edu/ldasmdc_data/burst-stochastic/`

The input data themselves were converted to frames using the script
`src/PIPESTOCHIFOIFO0000_frames.tclsh`
There are 200 frames in total, named `P-488811610.F` through `P-488811709.F` and `Q-488811610.F` through `Q-488811709.F`. For reference, they are contained in the repository as the contents of
`input/PIPESTOCHIFOIFO0000_frames.tar.gz`
The frames need to reside on the LDAS machine where the job is run, somewhere below the mount point. During the MDC, their existence can be confirmed by pointing a web browser at
`ftp://ldas.mit.edu/ldasmdc_data/burst-stochastic/FULL_frames/PIPESTOCHIFOIFO0000/`

The test compares its output to `expected/PIPESTOCHIFOIFO0000_results.dat`; this is just a copy of the expected output for test STOCHASTICIFOIFOSO00000, which can be confirmed by changing to the directory `/.../ldasmdc/burst-stochastic/test/pipeline/expected/` and issuing the command

```
diff PIPESTOCHIFOIFO0000_results.dat \
  ../../stochastic_ifo-ifo/expected/STOCHASTICIFOIFOSO00000_output_1.dat
```

The file `expected/PIPESTOCHIFOIFO0000_results.dat` is gzip-compressed in the repository, but must be uncompressed in order to run the test. To make sure all the files needed for the pipeline tests are ready, issue the command

`make data`

from the directory

`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to

   `/.../ldasmdc/burst-stochastic/test/pipeline/command/`

   and execute the script

   `./PIPESTOCHIFOIFO0000.tclsh`

   which has the following source:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFO0000
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFO0000.tclsh,v 1.6 2001/09/07 15:02:45 whelan Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  set output [ read $sid ]
  close $sid
  return $output
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFO0000

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test $testname complete.  Data products ready to be retrieved with fetchproducts.tclsh"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488811610
set srate 1024
```

```
set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 10238
set numfreq 230
set flow 40.0
set freqres 2
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 0
set spectrum1 spectrum1
set spectrum2 spectrum2
set target mpi

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.2.ilwd
set spec2name spec2_white.-512-512.2.ilwd
set resp1name resp1_ident.40-500.2.ilwd
set resp2name resp2_ident.40-500.2.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                file:$spec1file,pass
                file:$spec2file,pass
                file:$resp1file,pass
                file:$resp2file,pass
    "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                gwH1 = $ch1alias;
                gwH2 = $ch2alias;
    "

set algorithms "
                slice(gwH1, 0, $totalpoints, 1);
                intermediate(,$target,gwH1,H1 gravitational wave channel!);
                slice(gwH2, 0, $totalpoints, 1);
    "

set framequery "
                { P {} $times Adc($channel1) }
                { Q {} $times Adc($channel2) }
    "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
```

```
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget $target
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

## BOILERPLATE NOT MODIFIED PER TEST

set msg [ sendCmd $cmd $host $port ]

regexp {NORMAL\d+} $msg jobid

puts "Execution of test $testname initiated by command"
puts ""
puts $cmd

puts ""
puts "proceeding with ack msg"
puts ""
puts $msg
puts ""
puts "Once you receive an email at $email telling you that job $jobid has"
puts "completed successfully, you can fetch the data with the command"
puts ""
puts "./fetchproducts.tclsh $jobid $testname"
puts ""

## END OF BOILERPLATE

exit
```

This will initiate an `ldasJob` command, the contents of which are dumped to the screen, followed by an acknowledgement message and instructions to wait for an email that the job has completed and then run the `fetchproducts.tcl` script. Note that the last line of the output contains the command you will need to issue for the next part of this test. This part of the test passes if and when you receive via the MDC mailing list an email from the LDAS General Account with a subject line containing "NORMAL#### MDC test PIPESTOCHIFOIFO0000 complete", where "NORMAL####" is the job ID. **Note the job ID here:** 5951

Pass

2. Once you get the email that your job has completed, run the script

   `./fetchproducts.tclsh NORMAL#### PIPESTOCHIFOIFO0000`

   where `NORMAL####` is the job ID you entered above. (The exact command, including job ID, is included at the end of the output of the `PIPESTOCHIFOIFO0000.tclsh`.) This script will launch two LDAS jobs, one of which converts the output frame to an ASCII ILWD, and the other of which extracts the summary spectrum from the database, and print both LDAS job commands followed by instructions on how to retrieve the original frame as well as both ILWDs over the web using lynx. Issue the first lynx command (to retrieve the frame), then wait until you receive the appropriate emails from LDAS, and issue the other two lynx commands (to retrieve the ILWDs). This part of the test passes if both LDAS jobs are successful and the you successfully retrieve the frame and ILWDs (which will automatically end up in the directory `../output`) using the specified lynx commands.

   Pass

3. Verify that the correct metadata are contained in the file `../output/PIPESTOCHIFOIFO0000_database.ilwd` describing the contents of the database table entry:

   (a) The `START_FREQUENCY` array has dims=10, and all ten entries are 40=4e+1 Hz.

   (b) The `DELTA_FREQUENCY` array has dims=10, and all ten entries are 2 Hz.

   (c) The `SPECTRUM` container has size=10, and all ten arrays have dims=1840, corresponding to $230 * 8 = 1840$ bytes (230 8-byte complex numbers).

   (d) The `START_TIME`, `START_TIME_NS`, `END_TIME`, `END_TIME_NS`, and `FRAMES_USED` arrays each have dims=10 and the values agree with those listed in table 26.

Segment	START_TIME	START_TIME_NS	END_TIME	END_TIME_NS	FRAMES_USED
1	488811610	0	488811619	998046875	10
2	488811619	998046875	488811629	996093750	11
3	488811629	996093750	488811639	994140625	11
4	488811639	994140625	488811649	992187500	11
5	488811649	992187500	488811659	990234375	11
6	488811659	990234375	488811669	988281250	11
7	488811669	988281250	488811679	986328125	11
8	488811679	986328125	488811689	984375000	11
9	488811689	984375000	488811699	982421875	11
10	488811699	982421875	488811709	980468750	11

Table 26: Start and stop times for `../output/PIPESTOCHIFOIFO0000_database.ilwd`.

   (e) The `SPECTRUM_LENGTH` array has dims=10, and all ten entries are 230

Pass

4. Next, verify that the values in the ILWDs corresponding to the output frame and to the database entry agree with one another by running the script
   `./trivdatacomp.tclsh PIPESTOCHIFOIFO0000`
   and filling out table 27; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) must be less than 10^{-6}.

Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
4.4e-1 + i 0.0	1.1e-9 + i 0.0	1.1e-9	0.0

Table 27: Discrepancies between `PIPESTOCHIFOIFO0000_database.ilwd` and `PIPESTOCHIFOIFO0000_frame.ilwd`; fill in the columns to two significant figures.

Pass

5. Finally, verify that the values in the frame output agree with those in `expected/PIPESTOCHIFOIFO0000_results.dat` by running the script
   `./trivdatadiff.tclsh PIPESTOCHIFOIFO0000`
   and filling out table 28; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) must be less than 10^{-4}.

Pass

Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
4.4e-1 + i 0.0	1.8e-7 + i 0.0	1.8e-8	0.0

Table 28: Errors in `PIPESTOCHIFOIFO0000_frame.ilwd`; fill in the columns to two significant figures.

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.11.4 PIPESTOCHIFOIFO1000

Test Case: PIPESTOCHIFOIFO1000

Purpose: Verify that ~ 15 min of trivial data produces the correct output when run with synthetically generated frames and auxiliary ILWD files

Tester: Joseph D. Romano

Test machine: My laptop

Date (mm/dd/yy): 09/09/01 **Time:** 17:50 EDT

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS. It depends on the successful completion of all test checklists in section A.1. The `datadiff.tclsh` and `datacomp.tclsh` need to be run on a machine with the same architecture (little-endian or big-endian) as the nodes in the LDAS beowulf. Since those are all linux boxes, this means the test should be run from a little-endian machine like a linux box, rather than a big-endian system like Solaris.

The input data, response function, and spectra, are those produced by the script `src/PIPESTOCHIFOIFO1000.ilwd.sh`, but with a GPS start time of 488815210 rather than 488811610. The PSDs and response functions were extracted by hand from the resulting file `src/PIPESTOCHIFOIFO1000.ilwd`, which is included (gzip-compressed) for reference in the MDC repository. They were extracted into the files

`resp1_ident.40-500.1_4.ilwd`
`resp2_ident.40-500.1_4.ilwd`
`spec1_white.-512-512.1024_92138.ilwd`
`spec2_white.-512-512.1024_92138.ilwd`

which are included for reference in the repository in gzip-compressed form, in the directory

`/.../ldasmdc/burst-stochastic/test/pipeline/src/`

These files need to reside on the LDAS machine where the job is run, in the directory

`/ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/`

During the MDC, their existence can be confirmed by pointing a web browser at

`ftp://ldas.mit.edu/ldasmdc_data/burst-stochastic/`

The input data themselves were converted to frames using the script

`src/PIPESTOCHIFOIFO1000_frames.tclsh`

There are 1800 frames in total, named `P-488815210.F` through `P-488816109.F` and `Q-488815210.F` through `Q-488816109.F`. For reference, they are contained in the repository as the contents of

`input/PIPESTOCHIFOIFO1000_frames.tar.gz`

The frames need to reside on the LDAS machine where the job is run, somewhere below the mount point. During the MDC, their existence can be confirmed by pointing a web browser at

`ftp://ldas.mit.edu/ldasmdc_data/burst-stochastic/FULL_frames/PIPESTOCHIFOIFO1000/`

The test compares its output to `expected/PIPESTOCHIFOIFO1000_results.dat`, which is gzip-compressed in the repository, but must be uncompressed in order to run the test. To make sure all the files needed for the pipeline tests are ready, issue the command

`make data`

from the directory

`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/pipeline/command/`
   and execute the script
   `./PIPESTOCHIFOIFO1000.tclsh`
   which has the following source:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFO1000
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFO1000.tclsh,v 1.8 2001/09/07 15:02:45 whelan Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  set output [ read $sid ]
  close $sid
  return $output
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##############################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFO1000

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test $testname complete.  Data products ready to be retrieved with fetchproducts.tclsh"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488815210
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
```

```
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 92138
set numfreq 1840
set flow 40.0
set freqres 0.25
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 0
set spectrum1 spectrum1
set spectrum2 spectrum2
set target mpi

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set spec1name spec1_white.-512-512.1024_92138.ilwd
set spec2name spec2_white.-512-512.1024_92138.ilwd
set resp1name resp1_ident.40-500.1_4.ilwd
set resp2name resp2_ident.40-500.1_4.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set spec1file $suppdir/$spec1name
set spec2file $suppdir/$spec2name
set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                  file:$spec1file,pass
                  file:$spec2file,pass
                  file:$resp1file,pass
                  file:$resp2file,pass
     "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                  gwH1 = $ch1alias;
                  gwH2 = $ch2alias;
     "

set algorithms "
                  slice(gwH1, 0, $totalpoints, 1);
                  intermediate(,$target,gwH1,H1 gravitational wave channel!);
                  slice(gwH2, 0, $totalpoints, 1);
     "

set framequery "
                  { P {} $times Adc($channel1) }
                  { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
```

```
      -datacondtarget $target
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

## BOILERPLATE NOT MODIFIED PER TEST

set msg [ sendCmd $cmd $host $port ]

regexp {NORMAL\d+} $msg jobid

puts "Execution of test $testname initiated by command"
puts ""
puts $cmd

puts ""
puts "proceeding with ack msg"
puts ""
puts $msg
puts ""
puts "Once you receive an email at $email telling you that job $jobid has"
puts "completed successfully, you can fetch the data with the command"
puts ""
puts "./fetchproducts.tclsh $jobid $testname"
puts ""

## END OF BOILERPLATE

exit
```

This will initiate an `ldasJob` command, the contents of which are dumped to the screen, followed by an acknowledgement message and instructions to wait for an email that the job has completed and then run the `fetchproducts.tcl` script. Note that the last line of the output contains the command you will need to issue for the next part of this test. This part of the test passes if and when you receive via the MDC mailing list an email from the LDAS General Account with a subject line containing "NORMAL#### MDC test PIPESTOCHIFOIFO1000 complete", where "NORMAL####" is the job ID. **Note the job ID here:** 5970

Pass

2. Once you get the email that your job has completed, run the script

   ```
   ./fetchproducts.tclsh NORMAL#### PIPESTOCHIFOIFO1000
   ```

   where `NORMAL####` is the job ID you entered above. (The exact command, including job ID, is included at the end of the output of the `PIPESTOCHIFOIFO1000.tclsh`.) This script will launch two LDAS jobs, one of which converts the output frame to an ASCII ILWD, and the other of which extracts the summary spectrum from the database, and print both LDAS job commands followed by instructions on how to retrieve the original frame as well as both ILWDs over the web using lynx. Issue the first lynx command (to retrieve the frame), then wait until you receive the appropriate emails from LDAS, and issue the other two lynx commands (to retrieve the ILWDs). This part of the test passes if both LDAS jobs are successful and the you successfully retrieve the frame and ILWDs (which will automatically end up in the directory `../output`) using the specified lynx commands.

   Pass

3. Verify that the correct metadata are contained in the file `../output/PIPESTOCHIFOIFO1000_database.ilwd` describing the contents of the database table entry:

   (a) The `START_FREQUENCY` array has dims=10, and all ten entries are 40=4e+1 Hz.

   (b) The `DELTA_FREQUENCY` array has dims=10, and all ten entries are 0.25=2.5e-1 Hz.

   (c) The `SPECTRUM` container has size=10, and all ten arrays have dims=14720, corresponding to $1840 * 8 = 14720$ bytes (1840 8-byte complex numbers).

(d) The START_TIME, START_TIME_NS, END_TIME, END_TIME_NS, and FRAMES_USED arrays each have dims=10 and the values agree with those listed in table 29.

Segment	START_TIME	START_TIME_NS	END_TIME	END_TIME_NS	FRAMES_USED
1	488815210	0	488815299	978515625	90
2	488815299	978515625	488815389	957031250	91
3	488815389	957031250	488815479	935546875	91
4	488815479	935546875	488815569	914062500	91
5	488815569	914062500	488815659	892578125	91
6	488815659	892578125	488815749	871093750	91
7	488815749	871093750	488815839	849609375	91
8	488815839	849609375	488815929	828125000	91
9	488815929	828125000	488816019	806640625	91
10	488816019	806640625	488816109	785156250	91

Table 29: Start and stop times for ../output/PIPESTOCHIFOIFO1000_database.ilwd.

(e) The SPECTRUM_LENGTH array has dims=10, and all ten entries are 1840.

Pass

4. Next, verify that the values in the ILWDs corresponding to the output frame and to the database entry agree with one another by running the script
./trivdatacomp.tclsh PIPESTOCHIFOIFO1000
and filling out table 30; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) must be less than 10^{-6}.

Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
4.7e-1 + i 0.0	5.1e-10 + i 0.0	1.1e-10	0.0

Table 30: Discrepancies between PIPESTOCHIFOIFO1000_database.ilwd and PIPESTOCHIFOIFO1000_frame.ilwd; fill in the columns to two significant figures.

Pass

5. Finally, verify that the values in the frame output agree with those in expected/PIPESTOCHIFOIFO1000_results.dat by running the script
./trivdatadiff.tclsh PIPESTOCHIFOIFO1000
and filling out table 31; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) must be less than 10^{-4}.

Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
4.7e-1 + i 0.0	2.5e-7 + i 0.0	3.9e-7	0.0

Table 31: Errors in PIPESTOCHIFOIFO1000_frame.ilwd; fill in the columns to two significant figures.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.11.5 PIPESTOCHIFOIFO1010

Test Case: PIPESTOCHIFOIFO1010

Purpose: Verify that ~ 15 min of trivial data produces the correct output when run with synthetically generated frames, response function from an auxiliary ILWD file, and PSD calculated by the data conditioning API

Tester: Joseph D. Romano

Test machine: My laptop

Date (mm/dd/yy): 09/09/01 **Time:** 18:18 EDT

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS. It depends on the successful completion of all test checklists in section A.1. The `datadiff.tclsh` and `datacomp.tclsh` need to be run on a machine with the same architecture (little-endian or big-endian) as the nodes in the LDAS beowulf. Since those are all linux boxes, this means the test should be run from a little-endian machine like a linux box, rather than a big-endian system like Solaris.

The input data, response functions, and filter parameters are identical to those used in test PIPESTOCHIFOIFO1000 described in section A.11.4. The only difference between that test and this one is that that in this test, the power spectral densities are calculated by the data conditioning API, and the parameters used to construct the PSD are chosen so that it should agree, up to a constant (to which the LAL stochastic code is insensitive), with the ILWDs used in test PIPESTOCHIFOIFO1000.

Therefore, it needs the same frames and ILWDs (except for the the ones containing power spectra) to reside on the LDAS machine where the job is run.

The test compares its output to `expected/PIPESTOCHIFOIFO1010_results.dat`; this is just a copy of the expected output for test PIPESTOCHIFOIFO1000, which can be confirmed by changing to the directory `/.../ldasmdc/burst-stochastic/test/pipeline/expected/` and issuing the command

```
diff PIPESTOCHIFOIFO1010_results.dat.gz PIPESTOCHIFOIFO1000_results.dat.gz
```

The file `expected/PIPESTOCHIFOIFO1010_results.dat` is gzip-compressed in the repository, but must be uncompressed in order to run the test. To make sure all the files needed for the pipeline tests are ready, issue the command

```
make data
```

from the directory

`/.../ldasmdc/burst-stochastic/test/stochastic_ifo-ifo/`.

PROCEDURE

1. Set the current directory to

   `/.../ldasmdc/burst-stochastic/test/pipeline/command/`

   and execute the script

   `./PIPESTOCHIFOIFO1010.tclsh`

   which has the following source:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which tests PIPESTOCHIFOIFO1010
```

```
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: PIPESTOCHIFOIFO1010.tclsh,v 1.4 2001/09/07 15:02:45 whelan Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  set output [ read $sid ]
  close $sid
  return $output
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

###########################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFO1010

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test $testname complete.  Data products ready to be retrieved with fetchproducts.tclsh"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 488815210
set srate 1024

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 92138
set numfreq 1840
set flow 40.0
set freqres 0.25
set channel1 H1:LSC-AS_Q
set channel2 H2:LSC-AS_Q
set response1 response1
set response2 response2
set site1 0
set site2 0
```

```
set spectrum1 $channel1
set spectrum2 $channel2
set target mpi

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set resp1name resp1_ident.40-500.1_4.ilwd
set resp2name resp2_ident.40-500.1_4.ilwd

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ]
set dt [ expr $totalpoints/double($srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel1,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set responsefiles "
                file:$resp1file,pass
                file:$resp2file,pass
     "

set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set aliases "
                gwH1 = $ch1alias;
                gwH2 = $ch2alias;
     "

# PSD options
set fftlen $numpoints
set overlap 0

set algorithms "
                gwH1s = slice(gwH1, 0, $totalpoints, 1);
                intermediate(,$target,gwH1s,H1 gravitational wave channel!);

                gwH2s = slice(gwH2, 0, $totalpoints, 1);
                intermediate(,$target,gwH2s,H2 gravitational wave channel!);

                w = RectangularWindow();
                psd1 = psd(gwH1s, $fftlen, w, $overlap);
                intermediate(,$target, psd1, PSD1);

                psd2 = psd(gwH2s, $fftlen, w, $overlap);
     "

set framequery "
                { P {} $times Adc($channel1) }
                { Q {} $times Adc($channel2) }
     "

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget $target
      -responsefiles { $responsefiles }
      -aliases { $aliases }
      -outputformat { ilwd ascii }
      -algorithms { $algorithms }
      # Frame API options
      -framequery { $framequery }
}"

## BOILERPLATE NOT MODIFIED PER TEST
```

```
set msg [ sendCmd $cmd $host $port ]

regexp {NORMAL\d+} $msg jobid

puts "Execution of test $testname initiated by command"
puts ""
puts $cmd

puts ""
puts "proceeding with ack msg"
puts ""
puts $msg
puts ""
puts "Once you receive an email at $email telling you that job $jobid has"
puts "completed successfully, you can fetch the data with the command"
puts ""
puts "./fetchproducts.tclsh $jobid $testname"
puts ""

## END OF BOILERPLATE

exit
```

This will initiate an `ldasJob` command, the contents of which are dumped to the screen, followed by an acknowledgement message and instructions to wait for an email that the job has completed and then run the `fetchproducts.tcl` script. Note that the last line of the output contains the command you will need to issue for the next part of this test. This part of the test passes if and when you receive via the MDC mailing list an email from the LDAS General Account with a subject line containing "NORMAL#### MDC test PIPESTOCHIFOIFO1010 complete", where "NORMAL####" is the job ID. **Note the job ID here:** 5979

Pass

2. Once you get the email that your job has completed, run the script

   `./fetchproducts.tclsh NORMAL#### PIPESTOCHIFOIFO1010`

   where `NORMAL####` is the job ID you entered above. (The exact command, including job ID, is included at the end of the output of the `PIPESTOCHIFOIFO1010.tclsh`.) This script will launch two LDAS jobs, one of which converts the output frame to an ASCII ILWD, and the other of which extracts the summary spectrum from the database, and print both LDAS job commands followed by instructions on how to retrieve the original frame as well as both ILWDs over the web using lynx. Issue the first lynx command (to retrieve the frame), then wait until you receive the appropriate emails from LDAS, and issue the other two lynx commands (to retrieve the ILWDs). This part of the test passes if both LDAS jobs are successful and the you successfully retrieve the frame and ILWDs (which will automatically end up in the directory `../output`) using the specified lynx commands.

   Pass

3. Next, verify that the values in the ILWDs corresponding to the output frame and to the database entry agree with one another by running the script
   `./trivdatacomp.tclsh PIPESTOCHIFOIFO1010`
   and filling out table 32; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) must be less than 10^{-6}.

Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
4.7e-1 + i 0.0	8.1e-10 + i 0.0	1.1e-10	0.0

Table 32: Discrepancies between `PIPESTOCHIFOIFO1010_database.ilwd` and `PIPESTOCHIFOIFO1010_frame.ilwd`; fill in the columns to two significant figures.

Pass

4. Finally, verify that the values in the frame output agree with those in `expected/PIPESTOCHIFOIFO1010_results.dat` by running the script `./trivdatadiff.tclsh PIPESTOCHIFOIFO1010` and filling out table 33; to pass, the imaginary errors must be zero and the fractional error in the real parts (total and maximum) must be less than 10^{-4}.

Expected total	Fractional error in frame total	(max real frame err) / (real total)	(max imag frame err) / (real total)
4.7e-1 + i 0.0	2.6e-7 + i 0.0	3.9e-7	0.0

Table 33: Errors in `PIPESTOCHIFOIFO1010_frame.ilwd`; fill in the columns to two significant figures.

Pass

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

A.11.6 PIPESTOCHIFOIFO1911

Test Case: PIPESTOCHIFOIFO1911

Purpose: Verify that real frame data can be run through the LDAS pipeline for the shared object `libldasstochastic.so`, with appropriate data conditioning, and write results to proc frames and the metadata database without error.

Tester: Joseph D. Romano

Test machine: My laptop

Date (mm/dd/yy): 09/09/01 **Time:** 18:40 EDT

ENVIRONMENT AND PREREQUISITES

This test is executed through the user `ldas_mdc` on LDAS. It depends on the successful completion of all test checklists in section A.1. The `datadiff.tclsh` and `datacomp.tclsh` need to be run on a machine with the same architecture (little-endian or big-endian) as the nodes in the LDAS beowulf. Since those are all linux boxes, this means the test should be run from a little-endian machine like a linux box, rather than a big-endian system like Solaris.

The response functions used for this test are identical to those used in tests PIPESTOCHIFOIFO1000 and PIPESTOCHIFOIFO1010 described in sections A.11.4 and A.11.5, respectively, which means that the files

`input/resp1_ident.40-500.2.ilwd`
`input/resp2_ident.40-500.2.ilwd`

must reside on the LDAS machine where the job is run, as described in the ENVIRONMENT section of A.11.4.

The input data for this test are contained in actual Livingston and Hanford frames from the E5 engineering run. There need to be 1800 frames in total, named `H-680938500.F` through `H-680939399.F` and `L-680938500.F` through `L-680939399.F`, and they must reside on the LDAS machine where the job is run, somewhere below the mount point. During the MDC, their existence can be confirmed by pointing a web browser at

`ftp://ldas.mit.edu/ldasmdc_data/burst-stochastic/FULL_frames/LHOE5seg/`

and

`ftp://ldas.mit.edu/ldasmdc_data/burst-stochastic/FULL_frames/LLOE5seg/`

There is no expected output to which the results of this test are to be compared, and the input files need to reside on the remote LDAS system rather than the local machine, so no local data needs to be uncompressed.

PROCEDURE

1. Set the current directory to
   `/.../ldasmdc/burst-stochastic/test/pipeline/command/`
   and execute the script
   `./PIPESTOCHIFOIFO1911.tclsh`
   which has the following source:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which test PIPESTOCHIFOIFO01
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
```

```
##
## $Id: PIPESTOCHIFOIFO1911.tclsh,v 1.9 2001/09/07 20:45:27 charlton Exp $
##
## ***************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  set output [ read $sid ]
  close $sid
  return $output
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}

##########################################################################

# Test-specific variables

set testname PIPESTOCHIFOIFO1911

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test $testname complete.  Data products ready to be retrieved with fetchproducts.tclsh"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 680938500

#  Sampling rate of GW channel(s)
set srate 16384

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 92138
set numfreq 1840
set flow 40.0
set freqres 0.25
set channel0 "L1:LSC-AS_Q"
set channel1 "L0:PEM-EX_V1"
set channel2 "H1:LSC-AS_Q"
set channel3 "H0:PEM-EX_V1"
set response1 response1
set response2 response2
set site1 0
```

```
set site2 1
set spectrum1 $channel0
set spectrum2 $channel2
set target mpi

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set resp1name resp1_cmplx.40-500.1_4.ilwd
set resp2name resp2_cmplx.40-500.1_4.ilwd

# Resampling options for GW channel
set gw_p 1
set gw_q 16

# Resampling options for voltage channel
set v_p 1
set v_q 2

# PSD options
set fftlen 16384
set overlap [ expr $fftlen/2 ]

# oelslr options - remove lines at frequencies (k*basefreq) +/- freqbound,
# where k = 1 to numharm
set basefreq 60.0
set freqbound 1.0
set numharm 3

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ] ;## Points sent to wrapper after
                                             ;## resampling

# Work out how many frames we need to read to encompass the data
set dt [ expr $totalpoints*$gw_q/double($gw_p*$srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
set times ${startsec}-${endsec}

set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name
set linfilt_a_file $suppdir/highpass-50-a.ilwd
set linfilt_b_file $suppdir/highpass-50-b.ilwd

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel0,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set ch0alias [ mkchannel $channel0 $startsec ]
set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set ch3alias [ mkchannel $channel3 $startsec ]

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget $target
      -responsefiles {
          file:$linfilt_a_file,push,a
          file:$linfilt_b_file,push,b
          file:$resp1file,pass
          file:$resp2file,pass
      }
      -aliases {
          lgw = $ch0alias;
          lpl = $ch1alias;
          hgw = $ch2alias;
          hpl = $ch3alias;
      }
      -algorithms {
          # Downsample the gw and voltage channels, clearing the input
          # data after use to reduce memory overhead
```

```
          ly = resample(lgw, $gw_p, $gw_q);
          clear(lgw);
          lu = resample(lpl, $v_p, $v_q);
          clear(lpl);

          hy = resample(hgw, $gw_p, $gw_q);
          clear(hgw);
          hu = resample(hpl, $v_p, $v_q);
          clear(hpl);

          # Do line removal
          lw = oelslr(ly, lu, $basefreq, $freqbound, $numharm);
          lz = sub(ly, lw);
          lz = linfilt(b, a, lz);
  lz = slice(lz, 0, $totalpoints, 1);
          intermediate(,$target, gwL1, L1 gravitational wave channel);

          hw = oelslr(hy, hu, $basefreq, $freqbound, $numharm);
          hz = sub(hy, hw);
          hz = linfilt(b, a, hz);
  hz = slice(hz, 0, $totalpoints, 1);
          intermediate(,$target, gwH1, H1 gravitational wave channel);

          # Do spectra
          lpsd = psd(lz, $fftlen, _, $overlap);
          intermediate(,$target, lpsd, LLO PSD);

          hpsd = psd(hz, $fftlen, _, $overlap);
      }
      # Frame API options
      -framequery {
          { L {} $times Adc($channel0,$channel1) }
          { H {} $times Adc($channel2,$channel3) }
      }
}"

## BOILERPLATE NOT MODIFIED PER TEST

set msg [ sendCmd $cmd $host $port ]

regexp {NORMAL\d+} $msg jobid

puts "Execution of test $testname initiated by command"
puts ""
puts $cmd

puts ""
puts "proceeding with ack msg"
puts ""
puts $msg
puts ""
puts "Once you receive an email at $email telling you that job $jobid has"
puts "completed successfully, you can fetch the data with the command"
puts ""
puts "./fetchproducts.tclsh $jobid $testname"
puts ""

## END OF BOILERPLATE

exit
```

This will initiate an `ldasJob` command, the contents of which are dumped to the screen, followed by an acknowledgement message and instructions to wait for an email that the job has completed and then run the `fetchproducts.tcl` script. Note that the last line of the output contains the command you will need to issue for the next part of this test. This part of the test passes if and when you receive via the MDC mailing list an email from the LDAS General Account with a subject line containing "NORMAL#### MDC test PIPESTOCHIFOIFO1911 complete", where "NORMAL####" is the job ID. **Note the job ID here:** 5999

Pass

2. Once you get the email that your job has completed, run the script

   `./fetchproducts.tclsh NORMAL#### PIPESTOCHIFOIFO1911`

   where `NORMAL####` is the job ID you entered above. (The exact command, including job ID, is included at the end of the output of the `PIPESTOCHIFOIFO1911.tclsh`.) This script will launch two LDAS jobs, one of which converts the output frame to an ASCII ILWD, and the other of which extracts the summary spectrum from the database, and print both LDAS job commands followed by

instructions on how to retrieve the original frame as well as both ILWDs over the web using lynx. Issue the first lynx command (to retrieve the frame), then wait until you receive the appropriate emails from LDAS, and issue the other two lynx commands (to retrieve the ILWDs). This part of the test passes if both LDAS jobs are successful and the you successfully retrieve the frame and ILWDs (which will automatically end up in the directory `../output`) using the specified lynx commands.

Pass

3. Verify that the correct metadata are contained in the file `../output/PIPESTOCHIFOIFO1911_database.ilwd` describing the contents of the database table entry:

   (a) The `START_FREQUENCY` array has dims=10, and all ten entries are 40=4e+1 Hz.

   (b) The `DELTA_FREQUENCY` array has dims=10, and all ten entries are 0.25=2.5e-1 Hz.

   (c) The `SPECTRUM` container has size=10, and all ten arrays have dims=14720, corresponding to $1840 * 8 = 14720$ bytes (1840 8-byte complex numbers).

   (d) The `START_TIME`, `START_TIME_NS`, `END_TIME`, `END_TIME_NS`, and `FRAMES_USED` arrays each have dims=10 and the values of the start and end times agree with those listed in table 34 shifted by a fixed negative time offset. **Record the time offset here:** 009765625 ns

Segment	START_TIME	START_TIME_NS	STOP_TIME	STOP_TIME_NS	FRAMES_USED
1	680938500	0	680938589	978515625	91
2	680938589	978515625	680938679	957031250	91
3	680938679	957031250	680938769	935546875	91
4	680938769	935546875	680938859	914062500	91
5	680938859	914062500	680938949	892578125	91
6	680938949	892578125	680939039	871093750	91
7	680939039	871093750	680939129	849609375	91
8	680939129	849609375	680939219	828125000	91
9	680939219	828125000	680939309	806640625	91
10	680939309	806640625	680939399	785156250	91

Table 34: Start and stop times for `../output/PIPESTOCHIFOIFO1911_database.ilwd`.

   (e) The `SPECTRUM_LENGTH` array has dims=10, and all ten entries are 1840.

Pass

4. Finally, verify that the values in the ILWDs corresponding to the output frame and to the database entry agree with one another by running the script
   `./multidatacomp.tclsh PIPESTOCHIFOIFO1911`
   and filling out table 35; to pass, all fractional errors (total and maximum, real and imaginary) must be less than 10^{-6}.

Pass

SUMMARY

Known faults encountered – list bug IDs:

Segment	Total in frame output	Fractional discrepancy in total	(max real disc) / (real total)	(max imag disc) / (real total)
1	-1.0e+1 -i 1.1e+1	3.6e-8 - i 1.8e-8	4.7e-9	4.8e-9
2	4.1e0 + i 5.5e0	-7.2e-9 - i 6.2e-9	1.1e-8	7.4e-9
3	6.4e0 + i 3.4e0	1.0e-8 - i 1.0e-8	7.4e-9	7.8e-10
4	-1.9e-2 + i 3.8e-1	9.6e-7 + i 4.0e-7	8.3e-7	1.9e-6
5	-1.0e0 + i 5.3e-1	3.5e-8 + i 2.0e-8	4.1e-8	3.1e-8
6	-1.4e0 + i 3.8e-1	5.2e-9 + i 4.4e-8	3.6e-9	1.7e-8
7	-3.3e-1 + i 3.6e0	6.6e-8 - i 6.2e-8	3.9e-8	1.5e-8
8	2.0e0 + i 2.2e0	-8.0e-10 + i 4.2e-9	2.3e-9	2.4e-9
9	1.6e0 - i 3.3e0	1.1e-8 - i 4.7e-9	2.0e-8	3.1e-9
10	-1.3e0 - i 1.3e0	3.9e-9 + i 7.7e-9	3.6e-9	3.7e-9

Table 35: Discrepancies between `PIPESTOCHIFOIFO1911_database.ilwd` and `PIPESTOCHIFOIFO1911_frame.ilwd`; fill in the columns to two significant figures.

New faults submitted – list bug IDs:

Notes: We should have a datacondAPI time-shift action for time-series, to compensate for the time delay introduced by resample.

TEST RESULT **Pass**

A.11.7 PIPESTOCHIFOIFOFRAMETA

Test Case: PIPESTOCHIFOIFOFRAMETA

Purpose: Verify that the spectra written into frames by the `libldasstochastic.so` in LDAS pipeline mode contain the correct metadata.

Tester: Joseph D. Romano

Test machine: My laptop

Date (mm/dd/yy): 09/10/01 **Time:** 08:26 EDT

ENVIRONMENT AND PREREQUISITES

This test involves analysis of the outputs of tests PIPESTOCHIFOIFO1000, PIPESTOCHIFOIFO1010, PIPESTOCHIFOIFO1911 described in sections A.11.4, A.11.5, and A.11.6, respectively, so the following output files need to be present:

```
output/PIPESTOCHASTICIFOIFO0000_frames.ilwd
output/PIPESTOCHASTICIFOIFO1000_frames.ilwd
output/PIPESTOCHASTICIFOIFO1911_frames.ilwd
```

PROCEDURE

1. Verify that the correct metadata are contained in the file `../output/PIPESTOCHIFOIFO0000_frame.ilwd` describing the contents of the output frame:
   (a) The `GTime` entry is 488811610 seconds, 0 nanoseconds.
   (b) The `dt` entry is 99 seconds, 980468750 nanoseconds = 9.998046875e+1 seconds
   (c) Each of the ten `sampleRate` entries is $230 * 2 = 460$=4.6e+2 Hz. (If the frequency series were the Fourier transform of a heterodyned time series, the sampling rate of the time series would correspond to the bandwidth of the frequency series.)
   (d) Each of the ten `fShift` entries is 40=4e+1 Hz.
   (e) The `timeOffset` entries argee with those listed in table 36.

Segment	offset seconds	offset nanoseconds
1	0	0
2	9	998046875
3	19	996093750
4	29	994140625
5	39	992187500
6	49	990234375
7	59	988281250
8	69	986328125
9	79	984375000
10	89	982421875

Table 36: Start and stop times for `../output/PIPESTOCHASTICIFOIFO0000_frames.ilwd`.

Fail

2. Verify that the correct metadata are contained in the file `../output/PIPESTOCHIFOIFO1000_frame.ilwd` describing the contents of the output frame:

   (a) The `GTime` entry is 488815210 seconds, 0 nanoseconds.

   (b) The `dt` entry is 899 seconds, 785156250 nanoseconds = 8.9978515625e+2 seconds

   (c) Each of the ten `sampleRate` entries is $1840 * 1/4 = 460$=4.6e+2 Hz. (If the frequency series were the Fourier transform of a heterodyned time series, the sampling rate of the time series would correspond to the bandwidth of the frequency series.)

   (d) Each of the ten `fShift` entries is 40=4e+1 Hz.

   (e) The `timeOffset` entries argee with those listed in table 37.

Segment	offset seconds	offset nanoseconds
1	0	0
2	89	978515625
3	179	957031250
4	269	935546875
5	359	914062500
6	449	892578125
7	539	871093750
8	629	849609375
9	719	828125000
10	809	806640625

Table 37: Start and stop times for `../output/PIPESTOCHASTICIFOIFO1000_frames.ilwd`.

Fail

3. Verify that the correct metadata are contained in the file `../output/PIPESTOCHIFOIFO1911_frame.ilwd` describing the contents of the output frame:

   (a) The `GTime` entry is 680938500 seconds, 0 nanoseconds minus some time offset.

   (b) The `dt` entry is 899 seconds, 785156250 nanoseconds = 8.9978515625e+2 seconds

   (c) Each of the ten `sampleRate` entries is $1840 * 1/4 = 460$=4.6e+2 Hz. (If the frequency series were the Fourier transform of a heterodyned time series, the sampling rate of the time series would correspond to the bandwidth of the frequency series.)

   (d) Each of the ten `fShift` entries is 40=4e+1 Hz.

   (e) The `timeOffset` entries argee with those listed in table 37.

Fail

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes: The above failures were due to incorrect values for `sampleRate` and `fShift`. An e-mail was sent to Albert Lazzarini and Benoit Mours asking how best to store frequency series metadata in proc frames.

TEST RESULT **Fail**

A.12 Stochastic longterm test

Run the stochastic IFO-IFO shared object repeatedly in a loop to ensure that there are no long term operation problems, such as memory leaks.

A.12.1 LONGTERMSTOCHIFOIFO

Test Case: LONGTERMSTOCHIFOIFO

Purpose: Run stochastic IFO-IFO shared object as continually as possible for 2 hours to test for longterm operation problems.

Tester: Warren G. Anderson

Test machine: My laptop

Date (mm/dd/yy): 09/08/01 **Time:** 8:37 AM - 10:45 AM

This test runs the stochastic IFO-IFO shared object in a loop for 2 hours, analyzing approximately 6 hours of data from the engineering run E5. The *modus operandi* of this test is to run the pipeline test found in A.11.6 in a loop, launching each new `sendCmd` statement after five minutes (execution time on ldas-mit was approximately two minutes when the script was written). The script exercises all of the ldas components that are expected to be used in an actual stochastic IFO-IFO cross-correlation gravitational wave search.
ENVIRONMENT AND PREREQUISITES This test requires the tcl script:
`LONGTERMSTOCHIFOIFO.tclsh`
to be run on an LDAS system. It depends on successful completion of all tests in A.1 and A.12.1.

PROCEDURE Set the current directory to:
`/.../ldasmdc/burst-stochastic/test/longterm/command/`
and execute the script:
`LONGTERMSTOCHIFOIFO.tclsh >& .../output/LONGTERMSTOCHIFOIFO.txt`
This file contains:

```
#!/ldcg/bin/tclsh
## ******************************************************
##
## Description:
## Tcl script which test LONGTERMSTOCHIFOIFO01
##
## The dataPipeline user command documentation is found at:
## http://www.ldas-dev.ligo.caltech.edu/doc/userAPI/html/dataPipeline.html
##
## $Id: LONGTERMSTOCHIFOIFO.tclsh,v 1.4 2001/09/08 21:31:48 warren Exp $
##
## ******************************************************

proc sendCmd {cmd host port} {
  regsub -all -- {#[^\n]*} $cmd {} cmd
  regsub -all -- {[\n\s\t]+} $cmd { } cmd
  set sid [ socket $host $port ]
  puts $sid $cmd
  flush $sid
  set output [ read $sid ]
  close $sid
  return $output
}

proc mkchannel { chan stime } {
    regsub -all -- {:} $chan {\:} chalias
    set chalias ${chalias}::AdcData:$stime:0:Frame
}

;## Default settings
set user ""
set pwrd ""
set email ""
set host ldas-dev.ligo.caltech.edu
set port 10001

;## Source resource file
set rcfile "../../../misc/.datacondAPI.rc"
catch {source $rcfile} err

;## Process command-line arguments
set opt [list user pwrd email]
for {set idx 0} {$idx < $::argc} {incr idx} {
    if {$idx >= [llength $opt]} {break}
    set [lindex $opt $idx] [lindex $::argv $idx]
}
```

```
##############################################################################

# Test-specific variables

set testname LONGTERMSTOCHIFOIFO

# this will be used as the subject for the returned e-mail,
# so it should be descriptive of the test.
set subject "MDC test $testname complete.  Data products ready to be retrieved with fetchproducts.tclsh"

# actual name attached to the result object
set result_name "stochastic dso"

# comment attached to the result object
set result_comment "test $testname"

set startsec 680938500

#  Sampling rate of GW channel(s)
set srate 16384

set shared_object /ldcg/lib/lalwrapper/libldasstochastic.so

# params for your shared object
# The parameters for the stochastic dso are:
#
# argv[0]  = "-filterparams"
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = numFrequencies    Number of frequencies
# argv[4]  = f0                Start frequency
# argv[5]  = deltaF            Frequency spacing
# argv[6]  = dmro1name         Name of first data channel
# argv[7]  = dmro2name         Name of second data channel
# argv[8]  = response1name     Name of first response fcn
# argv[9]  = response2name     Name of second response fcn
# argv[10]  = siteID1          Index of first detector in lalCachedDetectors
# argv[11]  = siteID2          Index of second detector in lalCachedDetectors
# argv[12]  = spectrum1name    Name of first PSD
# argv[13]  = spectrum2name    Name of second PSD

set numsegs 10
set numpoints 92138
set numfreq 1840
set flow 40.0
set freqres 0.25
set channel0 "L1:LSC-AS_Q"
set channel1 "L0:PEM-EX_V1"
set channel2 "H1:LSC-AS_Q"
set channel3 "H0:PEM-EX_V1"
set response1 response1
set response2 response2
set site1 0
set site2 1
set spectrum1 $channel0
set spectrum2 $channel2
set target mpi

# Variables for reading using -responsefiles
set suppdir jobs/ldasmdc_data/burst-stochastic

set resp1name resp1_cmplx.40-500.1_4.ilwd
set resp2name resp2_cmplx.40-500.1_4.ilwd

# Resampling options for GW channel
set gw_p 1
set gw_q 16

# Resampling options for voltage channel
set v_p 1
set v_q 2

# PSD options
set fftlen 16384
set overlap [ expr $fftlen/2 ]

# oelslr options - remove lines at frequencies (k*basefreq) +/- freqbound,
# where k = 1 to numharm
set basefreq 60.0
set freqbound 1.0
set numharm 3

##############################################################################

# Derived variables
set totalpoints [ expr $numsegs*$numpoints ] ;## Points sent to wrapper after
                                             ;## resampling

# Work out how many frames we need to read to encompass the data
set dt [ expr $totalpoints*$gw_q/double($gw_p*$srate) ]
set numframes [ expr int(ceil($dt)) ]
set endsec [ expr $startsec + $numframes - 1 ]
```

```
set times ${startsec}-${endsec}

set resp1file $suppdir/$resp1name
set resp2file $suppdir/$resp2name
set linfilt_a_file $suppdir/highpass-50-a.ilwd
set linfilt_b_file $suppdir/highpass-50-b.ilwd

set filterparams $numsegs,$numpoints,$numfreq,$flow,$freqres,$numfreq,
append filterparams $flow,$freqres,$channel0,$channel2,$response1,$response2,
append filterparams 1,$site1,$site2,$spectrum1,$spectrum2

# other inputs used by the data-conditioning api
set ch0alias [ mkchannel $channel0 $startsec ]
set ch1alias [ mkchannel $channel1 $startsec ]
set ch2alias [ mkchannel $channel2 $startsec ]
set ch3alias [ mkchannel $channel3 $startsec ]

## values hard coded in command are defaults not normally
## modified by users.

set cmd "
ldasJob { -name $user -password $pwrd -email $email }
  {
   dataPipeline
      # Final results
      -subject { $subject }
      -resultcomment { $result_comment }
      -resultname { $result_name }
      # Eventmon API options
      -mddapi frame
      # mpi/wrapper API DSO options
      -dynlib $shared_object
      -filterparams ($filterparams)
      -np 2
      -realtimeratio 0.9
      -datadistributor WRAPPER
      # Data-conditioning API options
      -datacondtarget $target
      -responsefiles {
          file:$linfilt_a_file,push,a
          file:$linfilt_b_file,push,b
          file:$resp1file,pass
          file:$resp2file,pass
      }
      -aliases {
          lgw = $ch0alias;
          lpl = $ch1alias;
          hgw = $ch2alias;
          hpl = $ch3alias;
      }
      -algorithms {
          # Downsample the gw and voltage channels, clearing the input
          # data after use to reduce memory overhead
          ly = resample(lgw, $gw_p, $gw_q);
          clear(lgw);
          lu = resample(lpl, $v_p, $v_q);
          clear(lpl);

          hy = resample(hgw, $gw_p, $gw_q);
          clear(hgw);
          hu = resample(hpl, $v_p, $v_q);
          clear(hpl);

          # Do line removal
          lw = oelslr(ly, lu, $basefreq, $freqbound, $numharm);
          lz = sub(ly, lw);
          lz = linfilt(b, a, lz);
  lz = slice(lz, 0, $totalpoints, 1);
          intermediate(,$target, gwL1, L1 gravitational wave channel);

          hw = oelslr(hy, hu, $basefreq, $freqbound, $numharm);
          hz = sub(hy, hw);
          hz = linfilt(b, a, hz);
  hz = slice(hz, 0, $totalpoints, 1);
          intermediate(,$target, gwH1, H1 gravitational wave channel);

          # Do spectra
          lpsd = psd(lz, $fftlen, _, $overlap);
          intermediate(,$target, lpsd, LLO PSD);

          hpsd = psd(hz, $fftlen, _, $overlap);
      }
      # Frame API options
      -framequery {
          { L {} $times Adc($channel0,$channel1) }
          { H {} $times Adc($channel2,$channel3) }
      }
}"

puts "$testname initiated"
puts ""
set N 25
```

```
for {set i 1} {$i < $N} {incr i} {
   set msg [ sendCmd $cmd $host $port ]
   if [regexp {NORMAL\d+} $msg jobid] {
     puts "Analyzing segment $i of [expr $N-1] with job $jobid"
     after 1000
  } else {
     puts "ERROR: $msg"
  }
}
puts ""
puts "Test $testname completed. Please check logs to make sure all jobs"
puts "completed without error."

## END OF BOILERPLATE

exit
```

Job id's will be printed for each segment analyzed. When you see that the phrase:
`Test LONGTERMSTOCHIFOIFO completed.`
check the job logs and determine whether all jobs completed without error. If no errors appear in the job logs, the test passes. **Pass**

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

Notes:

TEST RESULT **Pass**

B Burst Filter packages used in this MDC

Three distinct burst filter packages were tested during this MDC. They were all under intense development during and after this exercise. In the following pages we provide brief documentation on these packages as they were realized during this MDC.

B.1 The Slope Filter

B.1.1 The MDC Test Slope Filter

The Slope Filter package consists of a collection of linear and nonlinear filters applied to LIGO data. As implied by the name of the package, many of the filters work by measuring the slope of time–domain data, using the fact that on the leading edge of a burst or pulse the data will have a positive gradient for some time interval.

Emphasis during the MDC was placed on testing the infrastructure (the wrapper) that sends data to the algorithm library of filters, searches the filtered data and sends triggers to a database when the filtered data exceeds some threshold. No attempt was made to optimize the slope detector filter for efficiency or to compare the relative sensitivities of different filtering schemes. Indeed, throughout the MDC, a single filter in the slope detector library was used. Let x_i denote data samples taken at a regularly spaced set of times $\{t_i\}$, and y_i denotes the filtered data. The test filter implements the algorithm

$$y_i = \frac{\frac{1}{N}\sum_{i=1}^{N} x_i t_i - \left(\frac{1}{N}\sum_{i=1}^{N} x_i\right)\left(\frac{1}{N}\sum_{i=1}^{N} t_i\right)}{\frac{1}{N}\sum_{i=1}^{N} t_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} t_i\right)^2}, \tag{2}$$

where N is the number of successive time samples over which the averages are measured. N is a free parameter of this slope filter, determined by the anticipated time duration of the burst.

B.1.2 Command Line Arguments

Command line arguments appear in pairs, a flag with a dash, then a space, then the parameter value. The pairs may appear in any order.

`-c <channel name>`

`-smw <number of bins` N `over which to measure slope>`

`-pw <rejection region around a single event passing threshold>`

`-sr <channel sampling rate>`

`-t <threshold in filtered data>`

Some further explanation of the `-smw` and `-pw` command line arguments may be helpful. When a sample of filtered data passes the threshold, it is likely that some neighboring bins of data may pass the threshold too. To avoid double counting peaks, it is necessary to select a single bin to generate one trigger corresponding to a peak that may cover many neighboring bins. The number of bins N used in generating the filtered data should be matched to the anticipated duration of the leading edge of the pulse. This number of bins is set using the `-smw` argument. The peak search algorithm examines the filtered data samples in time order. When it finds one that passes the threshold, it searches for the maximum bin in some number of bins after the bin that initially passes threshold. This second number of bins is set using the `-pw` argument.

B.1.3 Continuing Development of the Slope Detector Library

The filter used in the MDC was a placeholder to test infrastructure and interfacing to the rest of the LDAS / LAL software and hardware. Future work on the slope detector library will concentrate on implementing more time domain filter algorithms.

B.2 LALWrapper DSO: `power`

The excess power method distinguishes itself for the detection of signals of known duration and frequency band by a single compelling feature: in the absence of any other knowledge about the signal, *the method can be shown to be optimal*. Furthermore, it can be shown that for binary black hole (BBH) mergers of a sufficiently short duration and narrow frequency band, it performs nearly as well as matched filtering.

The essence of the power filter is that one compares the power of the data in the estimated frequency band and for the estimated duration to the known statistical distribution of noise power. It is straightforward to show that if the detector output consists solely of stationary Gaussian noise, the power in the band will follow a χ^2 distribution with the number of degrees of freedom being twice the estimated time-frequency volume (i.e., the product of the time duration and the frequency band of the signal). If a gravitational wave of sufficiently large amplitude is also present in the detector output, an excess of power will be observed; in this case, the power is distributed as a non-central χ^2 distribution with non-centrality parameter given by the signal power. The signal is detectable if the excess power is much greater than the fluctuations in the noise power which scales as the square-root of the time-frequency volume. Thus, the viability of the excess power method depends on the expected duration and bandwidth of the gravitational wave as well as on its intrinsic strength. For instance, the method is not competitive with matched filtering in detecting binary neutron star inspirals, since the time-frequency volume for such signals is very large, $> 10^4$.

To implement this method, one needs to decide the range of frequency bands and durations to search over. For initial LIGO, the most sensitive frequency band is $\sim$ 100–300 Hz, and it makes sense to search just in this band. For binary black hole mergers, signal durations might be of order tens or hundreds of milliseconds, depending on the black hole masses and spins [6]. Thus, the time-frequency volume of a merger signal can be as large as ~ 100, and its power would need to be more than one tenth as large as the noise power for detectability with the excess power method.

One might establish operational lower bounds on the time durations and frequency bands of interest. Because the largest operational frequency bandwidth is 200 Hz for the initial LIGO interferometers, the shortest duration of signal that need be considered is 5 ms (for a minimum time-frequency volume of unity). Similarly, for a maximum duration of 0.5 s, the smallest bandwidth that needs to be considered is 2 Hz. The excess power in any of the allowed bandwidths and durations can thus be obtained by judiciously summing up power that is output from a bank of one hundred 2 Hz band-pass filters (spanning the 200 Hz of peak interferometer sensitivity) for the required duration.

Having established the statistic and its operational range of parameters, the following simple algorithm for implementing the excess power method emerges naturally:

1. Pick a start time t_s, a time duration δt (containing N data samples), and a frequency band $[f_s, f_s+\delta f]$.
2. Fast Fourier transform (FFT) the block of (time domain) detector data for the chosen duration and start time.
3. Sum the power in each of the $\sim$ one hundred 2 Hz bands spanning the peak sensitivity region of the detector.
4. Further sum the power in the 2 Hz bands which correspond to the chosen frequency band.
5. Calculate the probability of having obtained the summed power from Gaussian noise alone using a χ^2 distribution with $2 \times \delta t \times \delta f$ degrees of freedom.
6. If the probability is significant, record a detection.
7. Repeat the process for all allowable choices of start times t_s, durations δt, starting frequencies f_s and bandwidths δf.

B.2.1 Command Line Arguments

The command line arguments for the power search code follow. **Note: this code is under active development. You should always refer to the LALWrapper documentation for the most up to date documentation. This material was correct during the MDC**. The `libldaspower.so` should be used with the following arguments to the wrapperAPI:

`argv[0] = "-filterparams"`

`argv[1] = numPoints` Number of data points in a segment. Note that the current implementation of the code must have this number of the form $2^{p+1} + 2$. The maximum expected time duration of the signal should be `numPoints`$\times$ sampling rate of data passed to the code.

`argv[2] = numSegments` Number of overlapping segments into which data should be divided for filtering.

`argv[3] = ovrlap` Number of points overlap between segments. This is an argument for completeness, but in general it should be `numPoints/2`

`argv[4] = overlapFactor` Amount of overlap between neighboring TF tiles. Currently see LAL `burstsearch` package for details.

`argv[5] = minFreqBins` Smallest extent in freq of TF tiles to search. Currently see LAL `burstsearch` package for details.

`argv[6] = minTimeBins` Smallest extent in time of TF tiles to search Currently see LAL `burstsearch` package for details.

`argv[7] = flow` Lowest frequency in Hz to be searched Currently see LAL `burstsearch` package for details.

`argv[8] = deltaF` Frequency resolution of first TF plane Currently see LAL `burstsearch` package for details.

`argv[9] = length` Length (N_F) of first TF plane (with $N_T = 1$) Currently see LAL `burstsearch` package for details.

`argv[10] = numSigmaMin` threshold number of sigma Currently see LAL `burstsearch` package for details.

`argv[11] = alphaDefault` default alpha value for tiles with sigma ¡ numSigmaMin Currently see LAL `burstsearch` package for details.

`argv[12] = segDutyCycle` Number of segments sent to slave for analysis Currently see LAL `burstsearch` package for details.

`argv[13] = alphaThreshold` Identify events with alpha less than this Currently see LAL `burstsearch` package for details.

`argv[14] = events2Master` Number of events to be accepted from a search over `numPoints` of data.

`argv[15] = channelName` The name used to identify the data to be analyzed in the multiDimData.

Note the constraints mentioned here and in the LAL documentation. The code can run several times real time on a single slave node. Optimal efficiency is achieved by insuring that each node spends more time computing the results than communicating them. For the parameters below, this needs `segDutyCylcle` to be greater than 200.

B.2.2 A typical LDAS user command

The following is a typical job running on 512s of data at 16384Hz. The data is resampled to 1024 Hz, divided into 1024 segments each of 1 second duration. The overlap between neighboring time-frequency tiles is 3 frequency bins. Tiles down to one frequency bin and one time bin are searched. Frequencies from 100Hz to 356Hz are searched.

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name patrick -password ****** -email patrick@gravity.phys.uwm.edu }
  {
   dataPipeline
      -framequery {H {} 600000000-600000513 Adc(H2:LSC-AS_Q)}
      -np 4
      -dynlib /home/patrick/src/lalwrapper/contrib/power/src/.libs/libldaspower.so
      -filterparams (1026,1024,513,3,1,1,100.0,1.0,256,3.0,0.5,205,0.99,1,channel)
      -subject newpower
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { newpower }
      -resultname { newpower }
      -responsefiles {
          file:/home/patrick/iulmdc/ldasmdc/inspiral/test/13challenge/input/response.ilwd,pass
       }
      -aliases gw=_ch0
      -algorithms {
         rgw = resample(gw, 1, 16);
         srgw = slice(rgw,0,524288,1);
         intermediate(,mpi,srgw,resampled gw timeseries);
         p = psd( rgw, 1026 );
      }
  }"

set sid [ socket ldas.phys.uwm.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

B.3 The tfclusters Filter

B.3.1 LAL Package: `tfclusters`

This package implements a nonlinear search algorithm for arbitrary transients, by (i) applying a threshold on a spectrogram, (ii) applying a set of clustering analysis algorithms, and (iii) applying a threshold on the total power in the cluster.

- (i) Thresholded Spectrogram: The spectrogram is computed from stacked periodograms, i.e., from the norm square of normalized Fourier transforms of adjacent, non-overlaping segments of data. No window is used to multiply the data before the transform. A threshold (which could be frequency dependent) is applied on the power in the spectrogram, to give a binary map representation of the time-frequency plane. By convention, pixels with power above the threshold will be called *black* pixels, and those below will be *white* pixels. In general, the threshold on the power should be chosen so that every pixel in the binary map has an equal probability of being black. The black pixel probability is labelled p.

- (ii) Clustering Analysis: The pixels in the time-frequency plane are labelled by two positive integers (i, j) refering respectively to there time and frequency indices. The distance between two pixels is defined by $d_{12} = |i_1 - i_2| + |j_1 - j_2|$. Pixels are said to be *nearest neighbours* if $d_{12} = 1$. A *cluster* is a set of one or more black pixels such that each black pixel has at least one black neighbour at a distance of 1, and such that no black pixel outside the set is at a distance of 1 of any pixel in the set. The size of a cluster is the cardinality of the set. The first threshold to be applied on clusters is that the size $s \geq \sigma$, with $\sigma \geq 1$. For $\sigma > 1$, a second set of threshold is applied on clusters with $s < \sigma$: All pairs of clusters are considered. For a pair with sizes s_1 and s_2, if $d_{12} \leq \delta_{s_1,s_2}$, the two clusters are merged and are called a *generalized cluster*. δ_{s_1,s_2} is an integer; there are $\sigma(\sigma - 1)/2$ such numbers. When a given cluster can pair with more than one other cluster and satisfy the distance thresholds, all these clusters are merged together into a single generalized cluster. By an abuse of language, 'generalized clusters' will often be called simply 'clusters' below.

- (iii) Power Threshold: For a given (generalized) cluster of size s, its *total power* P is defined as the sum of the power over all its pixels. If the data are Gaussian noise, and if the threshold on the spectrogram power in step (i) above was T, then $P - sT \sim \chi^2_{2s}$, i.e., is a chi-square variable with $2s$ degrees of freedom. The third threshold is therefore $Q(P - sT, 2s) < \alpha$, with $0 < \alpha < 1$ and $Q(x, a) = \int_x^\infty e^{-t} t^{a-1} dt / \Gamma(a)$. Any cluster that has survived cuts (i) and (ii) and for which this inequality is true will be called a *significant event*.

B.3.2 LALWrapper Package: `tfclusters`

The time series of $N_{\rm total}$ data points (sampled at frequency f_s) is first splitted into $n_{\rm nodes}$ segments of $N_{\rm seg}$ data points, where $n_{\rm nodes}$ is usually taken to be the number of nodes available for computation. The segments should overlap by $N_{\rm overlap}$ samples, where $N_{\rm overlap}/f_s$ is at least as large as the longest expected burst in the data. For every segment, the analysis is performed with the arguments set as explained in the LAL library documentation. The output of every one of these analyses is a list of detected bursts. The last step is the merging of all these lists (which is done on the serach master node), in order to remove duplicated bursts. Note that there are two different ways to specify the frequency dependent power thresholds. They can be set to $-\ln p$ by setting the absolute error goal on power threshld to a negative value (below). This is suitable for a white gaussian noise of unit variance. When the absolute error goal on power threshld is positive, the code will estimate the power thresholds so that the pixels have a uniform balck pixel probability in time and frequency. This is done by fitting a Rice distribution to the power as a function of time at every

frequency. The power threshold is obtained from the integral of the fitted distribution, at the precision level specified by the absolute error goal.

B.3.3 Command Line Arguments

The `libldastfclusters.so` requires the following arguments to the wrapperAPI:

argv[0] = "-filterparams"

argv[1] = channel name; actually use only 1st two letters to get IFO name

argv[2] = metadata database table (GDS_TRIGGER or SNGL_BURST)

argv[3] = total number of data (N_{total})

argv[4] = number of samples in one segment (N_{seg})

argv[5] = time resolution in seconds (T)

argv[6] = sampling frequency (f_s)

argv[7] = overlap between segments (number of samples) (N_{overlap})

argv[8] = black pixel probability (p)

argv[9] = absolute error goal on power threshold (Set to <0 for white noise of unit variance at input)

argv[10] = α

argv[11] = minimum frequency to process (Hz)

argv[12] = maximum frequency to process (Hz)

argv[13] = σ

argv[14] = distance for s1 = 1, s2 = 1, i.e. δ_{s_1,s_2}

argv[15] = distance for s1 = 1, s2 = 2

…

argv[$13+\sigma(\sigma-1)$**] = distance for s1 =** σ**-1, s2 =** σ**-1**

B.3.4 Special Notes:

- `argv[1]` and `argv[2]` were not present during September's MDC.
- $\frac{N_{\text{total}}-N_{\text{overlap}}}{n_{\text{nodes}}} + N_{\text{overlap}} = N_{\text{seg}}$ must always be respected.
- N_{seg}/f_s must be a positive integer.

The following is a typical job running on 512s of data at 16384Hz. The noise is white and unit variance, so no threshold estimation occurs. The black pixel probability and clustering parameters are set so that the probability to have a false alarm in 512s of data at full bandwidth is 0.0718.

```
#!/ldcg/bin/tclsh

set cmd "ldasJob
{ -name ldas_mdc -password beowulf -email julien@ligo.mit.edu }
  {
   dataPipeline
      -np 10
      -dynlib /ldcg/lib/lalwrapper/libldastfclusters.so
      -filterparams (8388608,1277952,0.03125,16384,262144,0.0213,-1,1,0,8192,6,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
      -subject TFCLUSTERS1301
      -datacondtarget mpi
      -mddapi ligolw
      -state {}
      -resultcomment { TFCLUSTERS1301 }
      -resultname { TFCLUSTERS1301 }
      -aliases x=_ch0
      -algorithms {m=mean(x);
                   y=sub(x,m);
                   }
      -frames /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_512.F
      -framequery Adc(0)
  }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```