LALApps — LSC Algorithm Library Applications

Contact: Duncan Brown duncan@gravity.phys.uwm.edu

March 16, 2008
Authors

Sukanta Bose
Patrick Brady
Duncan Brown
Kipp Cannon
Jolien Creighton
Anand S. Sengupta
Thomas Cokelaer
Rjean Dupuis
Steve Fairhurst
Badri Krishnan
Isabel Leonor
Adam Mercer
Eirini Messaritaki
Jeffrey Noel
Matt Pitkin
Tania Regimbau
Peter Shawhan
Xavier Siemens
Alicia M. Sintes
John T. Whelan
Mark Stephen Williamsen
Rana Adhikari
Ray Majumder Saikat
Reinhard Prix
## Contents

1 LALApps utilities ........................................ 1
   1.1 Header lalapps.h ....................................... 2
       1.1.1 Function set_debug_level ....................... 3
       1.1.2 Function clear_status ......................... 5
       1.1.3 Macro RCSID ..................................... 6
       1.1.4 Macro PRINT_VERSION ........................... 7
       1.1.5 Macro LAL_CALL .................................. 8
   1.2 Source hello.c ........................................ 10
   1.3 Program lalapps_hello ................................ 13
   1.4 Program lalapps_animate ............................. 15

2 Python DAG/Pipeline Modules ............................ 17
   2.1 Python Module pipeline ............................... 18
       2.1.1 Functions ......................................... 18
       2.1.2 Variables ......................................... 18
       2.1.3 Class AnalysisChunk .................. 18
       2.1.4 Class AnalysisJob .................. 18
       2.1.5 Class AnalysisNode .................. 19
       2.1.6 Class CondorDAG .......................... 20
       2.1.7 Class CondorDAGError .................. 20
       2.1.8 Class CondorDAGJob .................. 21
       2.1.9 Class CondorDAGNode .................. 21
       2.1.10 Class CondorDAGNodeError ............. 22
       2.1.11 Class CondorError ...................... 22
       2.1.12 Class CondorJob ...................... 22
       2.1.13 Class CondorJobError ................. 24
       2.1.14 Class CondorSubmitError .......... 24
       2.1.15 Class ScienceData .................. 24
       2.1.16 Class ScienceSegment .......... 25
       2.1.17 Class SegmentError .................. 25
   2.2 Python Module inspiral .............................. 27
       2.2.1 Variables ......................................... 27
       2.2.2 Class DataFindJob .......................... 27
       2.2.3 Class DataFindNode .................. 27
       2.2.4 Class IncaJob .............................. 28
       2.2.5 Class IncaNode ............................. 29
       2.2.6 Class InspiralError .................. 29
       2.2.7 Class InspiralJob ...................... 30
       2.2.8 Class InspiralNode ...................... 30
       2.2.9 Class TmpBankJob ...................... 31
       2.2.10 Class TmpBankNode .................. 31
       2.2.11 Class TmpToTmpJob .................. 32
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.12</td>
<td>Class TrigToTmpltNode</td>
</tr>
<tr>
<td>2.3</td>
<td>Python Module Power</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Variables</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Class BurcaJob</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Class DataFindJob</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Class DataFindNode</td>
</tr>
<tr>
<td>2.3.5</td>
<td>Class PowerError</td>
</tr>
<tr>
<td>2.3.6</td>
<td>Class PowerJob</td>
</tr>
<tr>
<td>2.3.7</td>
<td>Class PowerNode</td>
</tr>
<tr>
<td>2.4</td>
<td>Python Module stochastic</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Functions</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Variables</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Class LSCDataFindJob</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Class LSCDataFindNode</td>
</tr>
<tr>
<td>2.4.5</td>
<td>Class StochasticError</td>
</tr>
<tr>
<td>2.4.6</td>
<td>Class StochasticJob</td>
</tr>
<tr>
<td>2.4.7</td>
<td>Class StochasticNode</td>
</tr>
<tr>
<td>2.4.8</td>
<td>Class StoppJob</td>
</tr>
<tr>
<td>2.4.9</td>
<td>Class StoppNode</td>
</tr>
<tr>
<td>3 Inspiral Search Programs</td>
<td>44</td>
</tr>
<tr>
<td>3.1 Description of the Components of the Pipeline</td>
<td>44</td>
</tr>
<tr>
<td>3.1.1 Template Bank Generation</td>
<td>44</td>
</tr>
<tr>
<td>3.1.2 Matched Filtering and Injections</td>
<td>44</td>
</tr>
<tr>
<td>3.1.3 Triggered Template Bank, Trigger Coincidence and Time Slides</td>
<td>44</td>
</tr>
<tr>
<td>3.1.4 Trigger Summary and Injection Analysis</td>
<td>45</td>
</tr>
<tr>
<td>3.1.5 Injection Parameter Generation</td>
<td>45</td>
</tr>
<tr>
<td>3.1.6 Injection in Frames</td>
<td>45</td>
</tr>
<tr>
<td>3.1.7 Splitting Large Template Banks</td>
<td>45</td>
</tr>
<tr>
<td>3.2 Program lalapps_inspiral_pipe</td>
<td>46</td>
</tr>
<tr>
<td>3.3 Program lalapps_inspiral_hipe</td>
<td>49</td>
</tr>
<tr>
<td>3.4 Program lalappstmplbank</td>
<td>58</td>
</tr>
<tr>
<td>3.5 Program lalapps_inspiral</td>
<td>63</td>
</tr>
<tr>
<td>3.6 Program lalapps_coherent_inspiral</td>
<td>70</td>
</tr>
<tr>
<td>3.7 Program lalapps_inca</td>
<td>73</td>
</tr>
<tr>
<td>3.8 Program lalapps_thinca</td>
<td>79</td>
</tr>
<tr>
<td>3.9 Program lalapps_trigbank</td>
<td>84</td>
</tr>
<tr>
<td>3.10 Program lalapps_sire</td>
<td>86</td>
</tr>
<tr>
<td>3.11 Program lalapps_inspiraj</td>
<td>90</td>
</tr>
<tr>
<td>3.12 Program lalapps_bbbinj</td>
<td>96</td>
</tr>
<tr>
<td>3.13 Program lalapps_inspfrinj</td>
<td>99</td>
</tr>
<tr>
<td>3.14 Program lalapps_inspmultiawg</td>
<td>103</td>
</tr>
<tr>
<td>3.15 Program lalapps_splitbank</td>
<td>106</td>
</tr>
<tr>
<td>3.16 Program lalapps_cointext</td>
<td>108</td>
</tr>
<tr>
<td>4 Ringdown Search Programs</td>
<td>110</td>
</tr>
<tr>
<td>4.1 Program lalapps_ring</td>
<td>110</td>
</tr>
<tr>
<td>5 Stochastic Search Programs</td>
<td>113</td>
</tr>
<tr>
<td>5.1 Program lalapps_olapredfcn</td>
<td>114</td>
</tr>
<tr>
<td>5.2 Running the LALApps Stochastic Pipeline Under Condor</td>
<td>116</td>
</tr>
<tr>
<td>5.2.1 Creating the DAG</td>
<td>116</td>
</tr>
<tr>
<td>5.2.2 Running the DAG</td>
<td>117</td>
</tr>
<tr>
<td>5.3 Program lalapps_stochastic_pipe</td>
<td>118</td>
</tr>
<tr>
<td>5.4 Program lalapps_stochastic</td>
<td>.................................................................</td>
</tr>
<tr>
<td>5.5 Program lalapps_stopp</td>
<td>.................................................................</td>
</tr>
</tbody>
</table>
Chapter 1

LALAppss utilities

Several utilities (macros, global variables, and functions) are provided to assist in writing programs in LALAppss, and for maintaining a standard look-and-feel. This chapter describes these utilities and concludes with the listing of an example program.
1.1 Header `lalapps.h`

Provides utilities for writing programs for LALApps.

Several macros, global variables, and function prototypes are given that will assist in writing LALApps programs, and will aid in maintaining a standard look-and-feel.

To use these utilities, include the header `lalapps.h` and make sure the program links to the object `lalapps.o`. 
1.1. Function **set_debug_level**

**Name**

*set_debug_level* — sets the LAL debug level

**Synopsis**

```c
#include <lalapps.h>
extern int lalDebugLevel;
int set_debug_level( const char *s );
```

**Description**

The function *set_debug_level* sets the LAL debug level to a value determined by the string *s*, which can be an absolute debug level (a string representing an integer) or a string of LAL debug level flags. Allowed flags are:

- **NDEBUG**
  
  No debugging information is printed and memory debugging code is disabled.

- **ERROR**
  
  Error messages are printed.

- **WARNING**
  
  Warning messages are printed.

- **INFO**
  
  Information messages are printed.

- **TRACE**
  
  Function call tracing messages are printed.

- **MEMINFO**
  
  Memory allocation information messages are printed.

- **MEMDBG**
  
  Debugging of memory allocation routines is enabled but no messages are printed.

The following pre-defined composite levels are available:

- **MSGLVL1**
  
  Equivalent to **ERROR**.

- **MSGLVL2**
  
  Equivalent to **ERROR | WARNING**.

- **MSGLVL3**
  
  Equivalent to **ERROR | WARNING | INFO**.

- **ALLDBG**
  
  All debugging messages are printed.

If the argument to *set_debug_level* is **NULL**, then the string stored in the environment variable *LAL_DEBUG_LEVEL* is used. If this environment is not defined, or if no flags or values are specified in the string, the debug level is set to 0, which is equivalent to **NDEBUG**. (This is also the default value for *lalDebugLevel* unless it is set to some other value.)

For example, the statement

```c
set_debug_level( "ERROR | INFO" );
```

will set the debug level so that error and information messages are printed (but not warning messages). Another example is the statement
set_debug_level( "2" );

which would set the debug level to 2 (warning messages are printed).

**Return Value**

The return value is the (integer) debug level that is assigned to `lalDebugLevel`.

**Environment**

`LAL_DEBUG_LEVEL`

Default LAL debug level string to use.
1.1.2 Function **clear_status**

Name

*clear_status* — clears the LAL status structure after a failed LAL function call

Synopsis

```c
#include <lalapps.h>
extern const LALStatus blank_status;
int clear_status( LALStatus *status );
```

Description

Clears the LAL status structure and iteratively frees attached (sic) any linked status structures. This is to be used after a failed LAL function call to restore the status structure to a useable form. The structure `blank_status` contains a blank status structure that can be used to initialize a status structure in the program.

Example

The following program calls a routine `LALFailUnlessNegative` twice, once with a positive argument (which causes the routine to fail) and once with a negative argument (which causes the routine to pass). The function `clear_status` is used to clean up the status structure after the failure and the constant structure `blank_status` is used to initialize the status structure.

```c
#include <lalapps.h>
#include <lal/LALStdlib.h>

extern const LALStatus blank_status;

void LALFailUnlessNegative( LALStatus *status, INT4 n )
{
    INITSTATUS( status, "LALFail", "$Id$" );
    ATTACHSTATUSPTR( status );
    ASSERT( n, status, 1, "Non-negative n" );
    if ( n > 0 )
    {
        TRY( LALFailUnlessNegative( status->statusPtr, n - 1 ), status );
    }
    DETATCHSTATUSPTR( status );
    RETURN( status );
}

int main( void )
{
    LALStatus status = blank_status;
    LALFailUnlessNegative( &status, 5 );
    clear_status( &status );
    LALFailUnlessNegative( &status, -2 );
    return status.statusCode;
}
```
1.1.3 Macro **RCSID**

**Name**

*RCSID* — set the RCS Id variable

**Synopsis**

```c
#include <lalapps.h>
#ifndef RCSID
#define RCSID( id ) static volatile const char *rcsid = (id)
#endif
```

**Description**

*RCSID* sets the static (i.e., internal-linkage) variable *rcsid* to the RCS Id string, `$Id$`, which is given as the argument *id*. The string `$Id$` is expanded by RCS to contain the identification of the source file along with its revision number. For example:

```c
RCSID("$Id$"¦);
```
1.1.4 Macro PRINT_VERSION

Name

PRINT_VERSION — prints the LALApps version of the program

Synopsis

```c
#include <lalapps.h>
static volatile const char *rcsid="$Id$";
#ifndef PRINT_VERSION
#define PRINT_VERSION( program ) \ 
    fprintf( stderr, PACKAGE " %s version " VERSION "\n%s\n", program, rcsid )
#endif
```

Description

PRINT_VERSION prints the version information for program in a standard format, along with the RCS Id information. For example, for the program lalapps_hello, the version information

```
 lalapps hello version 0.1
 $Id$
```

is printed with the command `lalapps_hello -V`. The source code to print this is

```
 PRINT_VERSION( "hello" );
```

Note that PRINT_VERSION requires the string variable rcsid to be set.
1.1.5 Macro LAL_CALL

Name

LAL_CALL — call a LAL routine and handle any errors

Synopsis

```c
#include <lalapps.h>
extern int vrblvl;
extern int (*lal_errhandler)( LALStatus *stat, const char *func, 
    const char *file, const int line, volatile const char *id);
extern lal_errhandler_t lal_errhandler;
static volatile const char *rcsid = "$Id$";

#ifndef LAL_CALL
#define LAL_CALL( function, statusptr ) \
    ((function), lal_errhandler(statusptr, #function, __FILE__, __LINE__,rcsid))
#endif
```

Description

LAL_CALL executes the LAL function `function` and executes the error handler `lal_errhandler`, which examines the status structure `statusptr` to see if an error occurred. Typically the error handler will return with value 0 if there was no error; otherwise it will print a trace of the execution stack and then perform a specific action. The action performed depends on the error handler, which can be set to one of the following:

- **LAL_ERR_DFLT**
  The default error handler (same as LAL_ERR_ABRT).

- **LAL_ERR_ABRT**
  Raises SIGABRT if there is an error.

- **LAL_ERR_EXIT**
  Exits with the returned status code if there is an error.

- **LAL_ERR_RTRN**
  Returns the status code.

Note that LAL_CALL requires the string variable `rcsid` to be set.

Return Value

If LAL_CALL returns (rather than terminating execution), the return value is equal to the status code returned by the LAL function.

Example

The following example program illustrates the use of LAL_CALL. The routine LALInvert is called incorrectly twice. The first time the division by zero error is caught. The second time, the unexpected null pointer error is not caught and the default error handler aborts the program.

```c
#include <stdlib.h>
#include <lalapps.h>
#include <lal/LALStdlib.h>
RCSID( "$Id$" );

extern int vrblvl;
extern const LALStatus blank_status;
```
void LALInvert( LALStatus *status, REAL4 *y, REAL4 x )
{
    INITSTATUS( status, "LALInvert", rcsid );
    ASSERT( y, status, 1, "Null pointer" );
    if ( input == 0 )
    {
        ABORT( status, 1, "Division by zero" );
    }
    *y = 1 / x;
    RETURN( status );
}

int main( void )
{
    LALStatus status = blank_status;
    REAL4 x;
    int code;

    vrblvl = 1;

    lal_errhandler = LAL_ERR_RTRN;
    code = LAL_CALL( LALInvert( &status, &x, 0 ), &status );
    if ( code == 2 )
    {
        puts( "division by zero" );
        clear_status( &status );
    }
    else if ( code )
    {
        exit( code );
    }

    lal_errhandler = LAL_ERR_DFLT;
    LAL_CALL( LALInvert( &status, NULL, 1 ), &status );

    return 0;
}
### 1.2 Source `hello.c`

This is the source code for the program `lalapps_hello`:

```c
/*
 * Copyright (C) 2007 Jolien Creighton
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 */

#include <stdio.h>
#include <unistd.h>
#include <lalapps.h>
#include <lal/LALStdlib.h>
#include <lal/LALHello.h>

#define usgfmt "Usage: %s [options]\n"
#define usage( program ) fprintf( stderr, usgfmt, program )

int main( int argc, char *argv[] )
{
    const char *program = argv[0];
    const char *outfile = NULL;
    const char *dbglvl = NULL;
    lal_errhandler_t default_handler;
    LALStatus status = blank_status;
    int code;
```
int opt;

/* parse options */
while ( 0 < ( opt = getopt( argc, argv, "hVvd:o:" ) ) )
{
    switch ( opt )
    {
    case 'h':
        usage( program );
        return 0;
    case 'V':
        PRINT_VERSION( "hello" );
        return 0;
    case 'v':
        vrbflg = 1;
        break;
    case 'd':
        dbglvl = optarg;
        break;
    case 'o':
        outfile = optarg;
        break;
    default:
        usage( program );
        return 1;
    }
}
if ( optind < argc )
{
    usage( program );
    return 1;
}

/* set debug level */
set_debug_level( dbglvl );

/* try to call LALHello; catch error LALHELLOH_EOPEN */
default_handler = lal_errhandler;
lal_errhandler = LAL_ERR_RTRN;

code = LAL_CALL( LALHello( &status, outfile ), &status );
if ( code == -1 && (status.statusPtr)->statusCode == LALHELLOH_EOPEN )
{
    fprintf( stderr, "warning: couldn't open file %s for output"
            "(using stdout)\n", outfile );
    clear_status( &status );
    lal_errhandler = LAL_ERR_EXIT;
    LAL_CALL( LALHello( &status, NULL ), &status );
}
else if ( code )
{
    exit( code );
}
lal_errhandler = default_handler; /* restore default handler */
LALCheckMemoryLeaks();
return 0;
}
1.3 Program lalapps_hello

Name
lalapps_hello — prints “hello LSC!”

Synopsis
lalapps_hello [-h] [-V] [-v] [-d dbglvl] [-o outfile]

Description
lalapps_hello prints “hello LSC!” to the screen or to an output file.

Options

- **-h**
  Print a help message.

- **-V**
  Print the version information.

- **-v**
  Verbose output.

- **-d dbglvl**
  Set LAL debug level to dbglvl.

- **-o outfile**
  Write the output to file outfile.

Debug levels
The LAL debug level can be specified as an integer or as a string of flags:

- **NDEBUG**
  No debugging information is printed and memory debugging code is disabled.

- **ERROR**
  Error messages are printed.

- **WARNING**
  Warning messages are printed.

- **INFO**
  Information messages are printed.

- **TRACE**
  Function call tracing messages are printed.

- **MEMINFO**
  Memory allocation information messages are printed.

- **MEMDBG**
  Debugging of memory allocation routines is enabled but no messages are printed.

The following composite levels are available:

- **MSGLVL1**
  Equivalent to ERROR

- **MSGLVL2**
  Equivalent to ERROR | WARNING
**MSG_LVL3**
Equivalent to **ERROR | WARNING | INFO**

**ALL_DBG**
All debugging messages are printed.

For example, the command

```
  lalapps_hello -d "ERROR | INFO"
```

will set the debug level so that error and information messages are printed.

**Environment**

**LAL_DEBUG_LEVEL**
Default LAL debug level to use.

**Author**
Jolien Creighton
1.4 Program lalapps_animate

Name

lal_animate — produces an animated display showing the time series output of a selected channel in a lower window, and a simultaneously calculated FFT power spectrum in the upper window.

Synopsis


Description

lal_animate produces an animated display showing the time series output of a selected channel in a lower window, and a simultaneously calculated FFT power spectrum in the upper window. The output from this program must be piped into the graphing program xmgr.

Options

--help
Print a help message.

--channel name
Name of frame channel

--duration secs
How many seconds to look at

--epoch sec nsec
Starting epoch

--framedir dirname
Directory containing frame files

--highpass freq attenuation
High-pass filter parameters

--numpts npoints
Points per graph to display

Example

To run the program, type:

lalapps_animate --channel H2:LSC-AS_Q --framedir ./h1 --numpts 16384 \ 
--epoch 693768272 0 --duration 1 --highpass 300 0.01 | xmgr -pipe

This will search in directory ./h1 for frame files containing the channel H2:LSC-AS_Q and pipe the data starting at 693768272 GPS seconds and 0 GPS nanoseconds to xmgr in segments containing 16384 points until 1 seconds of data has been reviewed. The data is highpass filtered to above 300 Hz with an attenuation of 0.1; the output is shown in Fig. 1.4

Author

Bruce Allen and Patrick Brady
Figure 1.1: Example of output from *lalapps_animate* program
Chapter 2

Python DAG/Pipeline Modules
2.1 Python Module **pipeline**

This module contains objects that make it simple for the user to create python scripts that build Condor DAGs to run code on the LSC Data Grid.

### 2.1.1 Functions

- **s2play**\((t)\)
  - Return 1 if \(t\) is in the S2 playground, 0 otherwise 
  - \(t\) = GPS time to test if playground

### 2.1.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>author</strong></td>
<td>Value: ‘Duncan Brown <a href="mailto:duncan@gravity.phys.uwm.edu">duncan@gravity.phys.uwm.edu</a>’</td>
</tr>
<tr>
<td><strong>date</strong></td>
<td>Value: ‘$Date: 2003/10/01 08:46:06 $’</td>
</tr>
<tr>
<td><strong>version</strong></td>
<td>Value: ‘1.18’</td>
</tr>
</tbody>
</table>

### 2.1.3 Class AnalysisChunk

An AnalysisChunk is the unit of data that a node works with, usually some subset of a ScienceSegment.

Methods

- **init**\((self, start, end)\)
  - start = GPS start time of the chunk. end = GPS end time of the chunk.
- **len**\((self)\)
  - Returns the length of this AnalysisChunk in seconds.
- **repr**\((self)\)
- **dur**\((self)\)
  - Returns the length (duration) of the chunk in seconds.
- **end**\((self)\)
  - Returns the GPS end time of the chunk.
- **start**\((self)\)
  - Returns the GPS start time of the chunk.

### 2.1.4 Class AnalysisJob

Describes a generic analysis job that filters LIGO data as configured by an ini file.

Methods

- **init**\((self, cp)\)
  - \(cp\) = ConfigParser object that contains the configuration for this job.
2.1.5 Class AnalysisNode

Pipeline.CondorDAGNode ─ Analysisinode

Contains the methods that allow an object to be built to analyse LIGO data in a Condor DAG.

Methods

_init_(self)
Overrides: Pipeline.CondorDAGNode._init_

get_end(self)
Get the GPS end time of the node.

get_ifo(self)
Returns the two letter IFO code for this node.

get_input(self)
Get the file that will be passed as input.

get_output(self)
Get the file that will be passed as output.

get_start(self)
Get the GPS start time of the node.

set_cache(self, file)
Set the LAL frame cache to use. The frame cache is passed to the job with the –frame-cache argument. file = calibration file to use.

set_end(self, time)
Set the GPS end time of the analysis node by setting a –gps-end-time option to the node when it is executed. time = GPS end time of job.

set_ifo(self, ifo)
Set the channel name to analyze and add a calibration file for that channel. The name of the ifo is prepended to the channel name obtained from the job configuration file and passed with a –channel-name option. A calibration file is obtained from the ini file and passed with a –calibration-cache option. ifo = two letter ifo code (e.g. L1, H1 or H2).
2.1. **Python Module** `pipeline`  

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set_input(self, file)</code></td>
<td>Add an input to the node by adding a –input option. file = option argument to pass as input.</td>
</tr>
<tr>
<td><code>set_output(self, file)</code></td>
<td>Add an output to the node by adding a –output option. file = option argument to pass as output.</td>
</tr>
<tr>
<td><code>set_start(self, time)</code></td>
<td>Set the GPS start time of the analysis node by setting a –gps-start-time option to the node when it is executed. time = GPS start time of job.</td>
</tr>
</tbody>
</table>

Inherited from `CondorDAGNode`: `repr`, `add_parent`, `add_var_arg`, `add_var_opt`, `job`, `set_log_file`, `set_name`, `set_retry`, `write_job`, `write_parents`, `write_vars`

### 2.1.6 Class CondorDAG

A CondorDAG is a Condor Directed Acyclic Graph that describes a collection of Condor jobs and the order in which to run them. All Condor jobs in the DAG must write their Condor logs to the same file. NOTE: The log file must not be on an NFS mounted system as the Condor jobs must be able to get an exclusive file lock on the log file.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__(self, log)</code></td>
<td>log = path to log file which must not be on an NFS mounted file system.</td>
</tr>
<tr>
<td><code>add_node(self, node)</code></td>
<td>Add a CondorDAGNode to this DAG. The CondorJob that the node uses is also added to the list of Condor jobs in the DAG so that a list of the submit files needed by the DAG can be maintained. Each unique CondorJob will be added once to prevent duplicate submit files being written. node = CondorDAGNode to add to the CondorDAG.</td>
</tr>
<tr>
<td><code>set_dag_file(self, path)</code></td>
<td>Set the name of the file into which the DAG is written. path = path to DAG file.</td>
</tr>
<tr>
<td><code>write_dag(self)</code></td>
<td>Write all the nodes in the DAG to the DAG file.</td>
</tr>
<tr>
<td><code>write_sub_files(self)</code></td>
<td>Write all the submit files used by the dag to disk. Each submit file is written to the file name set in the CondorJob.</td>
</tr>
</tbody>
</table>

### 2.1.7 Class CondorDAGError

```
exceptions.Exception  

<table>
<thead>
<tr>
<th>Base Class</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipeline.CondorError</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong></td>
<td></td>
</tr>
</tbody>
</table>
```

**Methods**

Inherited from `Exception`: `__getitem__`, `__str__`

Inherited from `CondorError`: `__init__`
2.1.8 Class CondorDAGJob

pipeline.CondorJob → CondorDAGJob

A Condor DAG job never notifies the user on completion and can have variable options that are set for a particular node in the DAG. Inherits methods from a CondorJob.

Methods

```python
__init__(self, universe, executable)
```

universe = the condor universe to run the job in. executable = the executable to run in the DAG.

Overrides: pipeline.CondorJob.__init__

```python
add_var_arg(self)
```

Add a command to the submit file to allow variable (macro) arguments to be passed to the executable.

```python
add_var_opt(self, opt)
```

Add a variable (or macro) option to the condor job. The option is added to the submit file and a different argument to the option can be set for each node in the DAG. opt = name of option to add.


2.1.9 Class CondorDAGNode

Known Subclasses: AnalysisNode

A CondorDAGNode represents a node in the DAG. It corresponds to a particular condor job (and so a particular submit file). If the job has variable (macro) options, they can be set here so each nodes executes with the correct options.

Methods

```python
__init__(self, job)
```

job = the CondorJob that this node corresponds to.

```python
__repr__(self)
```

```python
add_parent(self, node)
```

Add a parent to this node. This node will not be executed until the parent node has run sucessfully. node = CondorDAGNode to add as a parent.

```python
add_var_arg(self, arg)
```

Add a variable (or macro) argument to the condor job. The argument is added to the submit file and a different value of the argument can be set for each node in the DAG. arg = name of option to add.

```python
add_var_opt(self, opt, value)
```

Add the a variable (macro) options for this node. If the option specified does not exist in the CondorJob, it is added so the submit file will be correct when written. opt = option name. value = value of the option for this node in the DAG.
2.1. Python Module `pipeline`

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>job(self)</code></td>
<td>Return the CondorJob that this node is associated with.</td>
</tr>
<tr>
<td><code>set_log_file(self, log)</code></td>
<td>Set the Condor log file to be used by this CondorJob. log = path of Condor log file.</td>
</tr>
<tr>
<td><code>set_name(self)</code></td>
<td>Generate a unique name for this node in the DAG.</td>
</tr>
<tr>
<td><code>set_retry(self, retry)</code></td>
<td>Set the number of times that this node in the DAG should retry. retry = number of times to retry node.</td>
</tr>
<tr>
<td><code>write_job(self, fh)</code></td>
<td>Write the DAG entry for this node’s job to the DAG file descriptor. fh = descriptor of open DAG file.</td>
</tr>
<tr>
<td><code>write_parents(self, fh)</code></td>
<td>Write the parent/child relations for this job to the DAG file descriptor. fh = descriptor of open DAG file.</td>
</tr>
<tr>
<td><code>write_vars(self, fh)</code></td>
<td>Write the variable (macro) options and arguments to the DAG file descriptor. fh = descriptor of open DAG file.</td>
</tr>
</tbody>
</table>

2.1.10 Class `CondorDAGNodeError`

```
exceptions.Exception
pipeline.CondorError
CondorDAGNodeError
```

Methods

- Inherited from `Exception`: `__getitem__`, `__str__`
- Inherited from `CondorError`: `__init__`

2.1.11 Class `CondorError`

```
exceptions.Exception
CondorError
```

Known Subclasses: `CondorDAGError`, `CondorDAGNodeError`, `CondorJobError`, `CondorSubmitError`

Error thrown by Condor Jobs

Methods

- `__init__(self, args=None)`
  - Overrides: `exceptions.Exception.__init__`
  - Inherited from `Exception`: `__getitem__`, `__str__`

2.1.12 Class `CondorJob`

Known Subclasses: `CondorDAGJob`

Generic condor job class. Provides methods to set the options in the condor submit file for a particular executable
# Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_init_</code>(self, universe, executable, queue)</td>
<td><code>universe</code> = the condor universe to run the job in. <code>executable</code> = the executable to run. <code>queue</code> = number of jobs to queue.</td>
</tr>
<tr>
<td><code>add_arg</code>(self, arg)</td>
<td>Add an argument to the executable. Arguments are appended after any options and their order is guaranteed. <code>arg</code> = argument to add.</td>
</tr>
<tr>
<td><code>add_condor_cmd</code>(self, cmd, value)</td>
<td>Add a Condor command to the submit file (e.g. a class add or environment). <code>cmd</code> = Condor command directive. <code>value</code> = value for command.</td>
</tr>
<tr>
<td><code>add_ini_opts</code>(self, cp, section)</td>
<td>Parse command line options from a given section in an ini file and pass to the executable. <code>cp</code> = ConfigParser object pointing to the ini file. <code>section</code> = section of the ini file to add to the options.</td>
</tr>
<tr>
<td><code>add_opt</code>(self, opt, value)</td>
<td>Add a command line option to the executable. The order that the arguments will be appended to the command line is not guaranteed, but they will always be added before any command line arguments. The name of the option is prefixed with double hyphen and the program is expected to parse it with getopt.long(). <code>arg</code> = command line option to add. <code>value</code> = value to pass to the option (None for no argument).</td>
</tr>
<tr>
<td><code>get_stderr_file</code>(self)</td>
<td>Get the file to which Condor directs the stderr of the job.</td>
</tr>
<tr>
<td><code>get_stdout_file</code>(self)</td>
<td>Get the file to which Condor directs the stdout of the job.</td>
</tr>
<tr>
<td><code>get_sub_file</code>(self)</td>
<td>Get the name of the file which the Condor submit file will be written to when write_sub_file() is called. <code>path</code> = path to submit file.</td>
</tr>
<tr>
<td><code>set_log_file</code>(self, path)</td>
<td>Set the Condor log file. <code>path</code> = path to log file.</td>
</tr>
<tr>
<td><code>set_notification</code>(self, value)</td>
<td>Set the email address to send notification to. <code>value</code> = email address or never for no notification.</td>
</tr>
<tr>
<td><code>set_stderr_file</code>(self, path)</td>
<td>Set the file to which Condor directs the stderr of the job. <code>path</code> = path to stderr file.</td>
</tr>
<tr>
<td><code>set_stdout_file</code>(self, path)</td>
<td>Set the file to which Condor directs the stdout of the job. <code>path</code> = path to stdout file.</td>
</tr>
<tr>
<td><code>set_sub_file</code>(self, path)</td>
<td>Set the name of the file to write the Condor submit file to when write_sub_file() is called. <code>path</code> = path to submit file.</td>
</tr>
</tbody>
</table>
write_sub_file(self)
Write a submit file for this Condor job.

2.1.13 Class CondorJobError

exceptions.Exception  →
pipeline.CondorError  →  CondorJobError

Methods
Inherited from Exception:  __getitem__,  __str__
Inherited from CondorError:  __init__

2.1.14 Class CondorSubmitError

exceptions.Exception  →
pipeline.CondorError  →  CondorSubmitError

Methods
Inherited from Exception:  __getitem__,  __str__
Inherited from CondorError:  __init__

2.1.15 Class ScienceData

An object that can contain all the science data used in an analysis. Can contain multiple ScienceSegments and has a method to generate these from a text file produces by the LIGOtools segwizard program.

Methods

_init_(self)

_getitem_(self, i)
Allows direct access to or iteration over the ScienceSegments associated with the ScienceData.

_len_(self)
Returns the number of ScienceSegments associated with the ScienceData.

_repr_(self)

make_chunks(self, length, overlap, play)
Divide each ScienceSegment contained in this object into AnalysisChunks. length = length of chunk in seconds. overlap = overlap between segments. play = if true, only generate chunks that overlap with S2 playground data.

read(self, file)
Parse the science segments from the segwizard output contained in file. file = input text file containing a list of science segments generated by segwizard.
2.1.16 Class ScienceSegment

A ScienceSegment is a period of time where the experimenters determine that the interferometer is in a state where the data is suitable for scientific analysis. A science segment can have a list of AnalysisChunks associated with it that break the segment up into (possibly overlapping) smaller time intervals for analysis.

Methods

```python
__init__(self, segment)
segment = a tuple containing the (segment id, gps start time, gps end time, duration) of the segment.
```

```python
__getitem__(self, i)
Allows iteration over and direct access to the AnalysisChunks contained in this ScienceSegment.
```

```python
__len__(self)
Returns the number of AnalysisChunks contained in this ScienceSegment.
```

```python
__repr__(self)
```

```python
add_chunk(self, start, end)
Add an AnalysisChunk to the list associated with this ScienceSegment. start = GPS start time of chunk. end = GPS end time of chunk.
```

```python
dur(self)
Returns the length (duration) in seconds of this ScienceSegment.
```

```python
end(self)
Returns the GPS end time of this ScienceSegment.
```

```python
id(self)
Returns the ID of this ScienceSegment.
```

```python
make_chunks(self, length=0, overlap=0, play=0)
Divides the science segment into chunks of length seconds overlapped by overlap seconds. If the play option is set, only chunks that contain S2 playground data are generated. If the user has a more complicated way of generating chunks, this method should be overridden in a sub-class. Any data at the end of the ScienceSegment that is too short to contain a chunk is ignored. The length of this unused data is stored and can be retrieved with the unused() method. length = length of chunk in seconds. overlap = overlap between chunks in seconds. play = only generate chunks that overlap with S2 playground data.
```

```python
start(self)
Returns the GPS start time of this ScienceSegment.
```

```python
unused(self)
Returns the length of data in the science segment not used to make chunks.
```

2.1.17 Class SegmentError

exceptions.Exception  
| SegmentError |
Methods

```python
_init_(self, args=None)
```
Overrides: exceptions.Exception._init_

Inherited from Exception: __getitem__, __str__
2.2 Python Module *inspiral*

Classes needed for the inspiral analysis pipeline. This script produced the necessary condor submit and dag files to run the standalone inspiral code on LIGO data

### 2.2.1 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>author</em></td>
<td>Value: 'Duncan Brown <a href="mailto:duncan@gravity.phys.uwm.edu">duncan@gravity.phys.uwm.edu</a>'</td>
</tr>
<tr>
<td><em>date</em></td>
<td>Value: '$Date: 2003/10/01 09:05:08 $'</td>
</tr>
<tr>
<td><em>version</em></td>
<td>Value: '1.14'</td>
</tr>
</tbody>
</table>

### 2.2.2 Class DataFindJob

```
pipeline.AnalysisJob
pipeline.CondorJob
pipeline.CondorDAGJob
```

DataFindJob

A LALdataFind job used by the inspiral pipeline. The static options are read from the section [datafind] in the ini file. The stdout from LALdataFind contains the paths to the frame files and is directed to a file in the cache directory named by site and GPS start and end times. The stderr is directed to the logs directory. The job always runs in the scheduler universe. The path to the executable is determined from the ini file.

**Methods**

```
__init__(self, cp)
```

*cp = ConfigParser object from which options are read.*

*Overrides: pipeline.CondorDAGJob.__init__*

**Inherited from AnalysisJob:** calibration, channel, get_config

**Inherited from CondorDAGJob:** add_var_arg, add_var_opt

**Inherited from CondorJob:** add_arg, add_condor_cmd, add_ini_opts, add_opt, get_stderr_file, get_stdout_file, get_sub_file, set_log_file, set_notification, set_stderr_file, set_stdout_file, set_sub_file, write_sub_file

### 2.2.3 Class DataFindNode

```
pipeline.CondorDAGNode
pipeline.AnalysisNode
pipeline.CondorDAGNode
```

DataFindNode

A DataFindNode runs an instance of datafind in a Condor DAG.
Methods

\_\_init\_\_(self, job)

job = A CondorDAGJob that can run an instance of LAL.dataFind.
Overrides: pipeline.AnalysisNode._\_init\_

get_output(self)

Return the output file, i.e. the file containing the frame cache data.
Overrides: pipeline.AnalysisNode.get_output

set_end(self, time)

Set the end time of the datafind query. time = GPS end time of query.
Overrides: pipeline.AnalysisNode.set_end

set_ifo(self, ifo)

Set the IFO to retrieve data for. Since the data from both Hanford interferometers is stored in the same frame file, this takes the first letter of the IFO (e.g. L or H) and passes it to the –instrument option of LAL.dataFind. ifo = IFO to obtain data for.
Overrides: pipeline.AnalysisNode.set_ifo

set_start(self, time)

Set the start time of the datafind query. time = GPS start time of query.
Overrides: pipeline.AnalysisNode.set_start

Inherited from AnalysisNode: get_end, get_ifo, get_input, get_start, set_cache, set_input, set_output
Inherited from CondorDAGNode: __repr__, add_parent, add_var_arg, add_var_opt, job, set_log_file, set_name, set_retry, write_job, write_parents, write_vars

2.2.4 Class IncaJob

pipeline.AnalysisJob

pipeline.CondorJob

pipeline.CondorDAGJob

IncaJob

A lalapps.inca job used by the inspiral pipeline. The static options are read from the section [inca] in the ini file. The stdout and stderr from the job are directed to the logs directory. The job always runs in the scheduler universe. The path to the executable is determined from the ini file.

Methods

\_\_init\_\_(self, cp)

cp = ConfigParser object from which options are read.
Overrides: pipeline.CondorDAGJob._\_init__

Inherited from AnalysisJob: calibration, channel, get_config
Inherited from CondorDAGJob: add_var_arg, add_var_opt
### 2.2.5 Class IncaNode

![Class IncaNode](pipeline.CondorDAGNode) \[
\text{pipeline.AnalysisNode} \]

An IncaNode runs an instance of the inspiral coincidence code in a Condor DAG.

**Methods**

- `_init_(self, job)`
  
  job = A CondorDAGJob that can run an instance of lalapps_inca.

  Overrides: pipeline.AnalysisNode._init_

- `get_ifo_a(self)`
  
  Returns the IFO code of the primary interferometer.

- `get_ifo_b(self)`
  
  Returns the IFO code of the primary interferometer.

- `get_output(self)`
  
  Returns the file name of output from the inca code. This must be kept synchronized with the name of the output file in inca.c.

  Overrides: pipeline.AnalysisNode.get_output

- `set_ifo_a(self, ifo)`
  
  Set the interferometer code to use as IFO A. ifo = IFO code (e.g. L1, H1 or H2).

- `set_ifo_b(self, ifo)`
  
  Set the interferometer code to use as IFO B. ifo = IFO code (e.g. L1, H1 or H2).

**Inherited from** AnalysisNode: `get_end`, `get_ifo`, `get_input`, `get_start`, `set_cache`, `set_end`, `set_ifo`, `set_input`, `get_output`, `set_start`

**Inherited from** CondorDAGNode: `__repr__`, `add_parent`, `add_var_arg`, `add_var_opt`, `job`, `set_log_file`, `set_name`, `set_retry`, `write_job`, `write_parents`, `write_vars`

### 2.2.6 Class InspiralError

![Class InspiralError](exceptions.Exception) \[
\text{InspiralError}
\]

**Methods**

- `_init_(self, args=None)`

  Overrides: exceptions.Exception._init_

**Inherited from** Exception: `_getitem_`, `_str_`
2.2.7 Class InspiralJob

A lalapps_inspiral job used by the inspiral pipeline. The static options are read from the sections [data] and [inspiral] in the ini file. The stdout and stderr from the job are directed to the logs directory. The job runs in the universe specified in the ini file. The path to the executable is determined from the ini file.

Methods

```python
__init__(self, cp)
cp = ConfigParser object from which options are read.
Overrides: pipeline.CondorDAGJob.__init__
Inherited from AnalysisJob: calibration, channel, get_config
Inherited from CondorDAGJob: add_var_arg, add_var_opt
```

2.2.8 Class InspiralNode

An InspiralNode runs an instance of the inspiral code in a Condor DAG.

Methods

```python
__init__(self, job)
job = A CondorDAGJob that can run an instance of lalapps_inspiral.
Overrides: pipeline.AnalysisNode.__init__

get_output(self)
Returns the file name of output from the inspiral code. This must be kept synchronized with the name of the output file in inspiral.c.
Overrides: pipeline.AnalysisNode.get_output

set_bank(self, bank)
Inherited from AnalysisNode: get_end, get_ifo, get_input, get_start, set_cache, set_end, set_ifo, set_input, set_output, set_start
Inherited from CondorDAGNode: __repr__, add_parent, add_var_arg, add_var_opt, job, set_log_file, set_name, set_retry, write_job, write_parents, write_vars
```
2.2.9 Class TmpltBankJob

A lalapps_tmpltbank job used by the inspiral pipeline. The static options are read from the sections [data] and [tmpltbank] in the ini file. The stdout and stderr from the job are directed to the logs directory. The job runs in the universe specified in the ini file. The path to the executable is determined from the ini file.

Methods

```
init(self, cp)
```

* cp = ConfigParser object from which options are read.
  Overrides: pipeline.ConдорDAGJob._init_

Inherited from AnalysisJob: calibration, channel, get_config
Inherited from CondorDAGJob: add_var_arg, add_var_opt

2.2.10 Class TmpltBankNode

A TmpltBankNode runs an instance of the template bank generation job in a Condor DAG.

Methods

```
init(self, job)
```

* job = A CondorDAGJob that can run an instance of lalapps_tmpltbank.
  Overrides: pipeline.AnalysisNode._init_

```
get_output(self)
```

* Returns the file name of output from the template bank code. This must be kept synchronized with the name of the output file in tmpltbank.c.
  Overrides: pipeline.AnalysisNode.get_output

Inherited from AnalysisNode: get_end, get_ifo, get_input, get_start, set_cache, set_end, set_ifo, set_input, set_output, set_start
Inherited from CondorDAGNode: __repr__, add_parent, add_var_arg, add_var_opt, job, set_log_file, set_name, set_retry, write_job, write_parents, write_vars
2.2.11 Class TrigToTmpltJob

pipeline.AnalysisJob
pipeline.CondorJob
pipeline.CondorDAGJob

TrigToTmpltJob

A lalapps_trigtotmplt job used by the inspiral pipeline. The static options are read from the section [trigtotmplt] in the ini file. The stdout and stderr from the job are directed to the logs directory. The job always runs in the scheduler universe. The path to the executable is determined from the ini file.

Methods

```python
__init__(self, cp)
```

`cp` = ConfigParser object from which options are read.
Overrides: pipeline.ConadorDAGJob.__init__

Inherited from AnalysisJob: calibration, channel, get_config
Inherited from CondorDAGJob: add_var_arg, add_var_opt

2.2.12 Class TrigToTmpltNode

pipeline.CondorDAGNode
pipeline.AnalysisNode
pipeline.CondorDAGNode

TrigToTmpltNode

A TrigToTmpltNode runs an instance of the triggered bank generator in a Condor DAG.

Methods

```python
__init__(self, job)
```

`job` = A CondorDAGJob that can run an instance of lalapps_trigtotmplt.
Overrides: pipeline.AnalysisNode.__init__

Inherited from AnalysisNode: get_end, get_ifo, get_input, get_output, get_start, set_cache, set_end, set_ifo, set_input, set_output, set_start
Inherited from CondorDAGNode: __repr__, add_parent, add_var_arg, add_var_opt, job, set_log_file, set_name, set_retry, write_job, write_parents, write_vars
2.3 Python Module Power

Classes needed for the excess power analysis pipeline.

2.3.1 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>author</strong></td>
<td>`Duncan Brown <a href="mailto:duncan@gravity.phys.uwm.edu">duncan@gravity.phys.uwm.edu</a>'</td>
<td>('type=str')</td>
</tr>
<tr>
<td><strong>date</strong></td>
<td><code>$Date: 2004/06/16 04:40:48 $</code></td>
<td>('type=str')</td>
</tr>
<tr>
<td><strong>version</strong></td>
<td><code>1.1</code></td>
<td>('type=str')</td>
</tr>
</tbody>
</table>

2.3.2 Class BurcaJob

A lalapps_burca job used by the power pipeline. The static options are read from the section [burca] in the ini file. The stdout and stderr from the job are directed to the logs directory. The path to the executable is determined from the ini file.

Methods

```python
__init__(self, cp)
```

cp = ConfigParser object from which options are read.

Overrides: pipeline.CondorDAGJob.__init__

Inherited from AnalysisJob: channel, get_config

Inherited from CondorDAGJob: add_arg, add_condor_cmd, add_ini_opts, add_opt, get_stderr_file, get_stdout_file, get_sub_file, set_log_file, set_notification, set_stderr_file, set_stdout_file, set_sub_file, write_sub_file

2.3.3 Class DataFindJob

A LSCdataFind job used by the power pipeline. The static options are read from the section [datafind] in the ini file. The stdout from LSCdataFind contains the paths to the frame files and is directed to a file in the cache directory named by site and GPS start and end times. The stderr is directed to the logs directory. The job always runs in the scheduler universe. The path to the executable is determined from the ini file.
## 2.3. Python Module Power

### Methods

**_init_(self, cp)**

\[ cp = \text{ConfigParser object from which options are read.} \]

Overrides: pipeline._init_

**Inherited from AnalysisJob:** channel, get_config

**Inherited from CondorDAGJob:** add_var_arg, add_var_opt

**Inherited from CondorJob:** add_arg, add_condor_cmd, add_ini_opts, add_opt, get_stderr_file, get_stdout_file, get_sub_file, set_log_file, set_notification, set_stderr_file, set_stdout_file, set_sub_file, write_sub_file

### 2.3.4 Class DataFindNode

pipeline.CondorDAGNode →

pipeline.AnalysisNode →

pipeline.CondorDAGNode

**DataFindNode**

A DataFindNode runs an instance of datafind in a Condor DAG.

### Methods

**_init_(self, job)**

\[ \text{job} = \text{A CondorDAGJob that can run an instance of LSCdataFind.} \]

Overrides: pipeline._init_

**get_output(self)**

Return the output file, i.e. the file containing the frame cache data.

Overrides: pipeline.AnalysisNode.get_output

**set_end(self, time)**

Set the end time of the datafind query. time = GPS end time of query.

Overrides: pipeline.AnalysisNode.set_end

**set_ifo(self, ifo)**

Set the IFO to retrieve data for. Since the data from both Hanford interferometers is stored in the same frame file, this takes the first letter of the IFO (e.g. L or H) and passes it to the –instrument option of LSCdataFind. ifo = IFO to obtain data for.

Overrides: pipeline.AnalysisNode.set_ifo

**set_start(self, time)**

Set the start time of the datafind query. time = GPS start time of query.

Overrides: pipeline.AnalysisNode.set_start

**Inherited from AnalysisNode:** calibration, get_end, get_ifo, get_ifo_tag, get_input, get_start, set_cache, set_ifo_tag, set_input, set_output

**Inherited from CondorDAGNode:** __repr__, add_parent, add_post_script_arg, add_pre_script_arg, add_var_arg, add_var_opt, job, set_log_file, set_name, set_post_script, set_pre_script, set_retry, write_job, write_parents, write_post_script, write_pre_script, write_vars
2.3.5 **Class PowerError**

```python
exceptions.Exception
    └── PowerError
```

Methods

```
__init__ (self, args=None)
 overrides: exceptions.Exception.__init__
```

Inherited from Exception: `__getitem__`, `__str__`

2.3.6 **Class PowerJob**

```python
pipeline.AnalysisJob
    └── pipeline.CondorJob
        └── pipeline.CondorDAGJob
            └── PowerJob
```

A `lalapps.power` job used by the power pipeline. The static options are read from the sections `[data]` and `[power]` in the ini file. The `stdout` and `stderr` from the job are directed to the logs directory. The job runs in the universe specified in the ini file. The path to the executable is determined from the ini file.

Methods

```
__init__ (self, cp)
 overrides: pipeline.CondorDAGJob.__init__
```

Inherited from AnalysisJob: `channel`, `get_config`

Inherited from CondorDAGJob: `add_var_arg`, `add_var_opt`


2.3.7 **Class PowerNode**

```python
pipeline.CondorDAGNode
    └── pipeline.AnalysisNode
        └── pipeline.CondorDAGNode
            └── PowerNode
```

A `PowerNode` runs an instance of the power code in a Condor DAG.

Methods

```
__init__ (self, job)
 overrides: pipeline.AnalysisNode.__init__
```

`job = A CondorDAGJob that can run an instance of lalapps.power.`

Inherited from AnalysisNode: `channel`, `get_config`

Inherited from CondorDAGJob: `add_arg`, `add_condor_cmd`, `add_ini_opts`, `add_opt`, `get_stderr_file`, `get_stdout_file`, `get_sub_file`, `set_log_file`, `set_notification`, `set_stderr_file`, `set_stdout_file`, `set_sub_file`, `write_sub_file`
### get_output(self)

Returns the file name of output from the power code. This must be kept synchronized with the name of the output file in power.c.

Overrides: pipeline.AnalysisNode.get_output

**Inherited from AnalysisNode:** calibration, get_end, get_ifo, get_ifo_tag, get_input, get_start, set_cache, set_end, set_ifo, set_ifo_tag, set_input, set_output, set_start

**Inherited from CondorDAGNode:** __repr__, add_parent, add_post_script_arg, add_pre_script_arg, add_var_arg, add_var_opt, job, set_log_file, set_name, set_post_script, set_pre_script, set_retry, write_job, write_parents, write_post_script, write_pre_script, write_vars
2.4 Python Module stochastic

Classes needed for the stochastic analysis pipeline. This script produces the necessary condor submit and dag files to run the standalone stochastic code on LIGO/GEO data.

2.4.1 Functions

version()

Return version

2.4.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>author</strong></td>
<td>Value: 'Adam Mercer <a href="mailto:ram@star.sr.bham.ac.uk">ram@star.sr.bham.ac.uk</a>' (type=str)</td>
</tr>
<tr>
<td><strong>date</strong></td>
<td>Value: '$Date: 2005/05/06 17:43:19 $' (type=str)</td>
</tr>
<tr>
<td><strong>version</strong></td>
<td>Value: '1.42' (type=str)</td>
</tr>
</tbody>
</table>

2.4.3 Class LSCDataFindJob

An LSCDataFind job used to locate data. The static options are read from the section [datafind] in the ini file. The stdout from LSCDataFind contains the paths to the frame files and is directed to a file in the cache directory named by site and GPS start and end times. The stderr is directed to the logs directory. The job always runs in the scheduler universe. The path to the executable is determined from the ini file.

Note: This class overrides the LSCDataFindJob class within glue.pipeline, it has support for doing runing datafind jobs for multiple frame types within the same DAG. This will be eventually be merged into the main glue.pipeline.

Methods

__init__(self, cache_dir, log_dir, config_file)

Parameters

cache_dir: the directory to write the output lal cache files to.
log_dir: the directory to write the stderr file to.
config_file: ConfigParser object containing the path to the LSCDataFind executable in the [condor] section and a [datafind] section from which the LSCDataFind options are read.

Overrides: glue.pipeline.CondorDAGJob.__init__

get_cache_dir(self)

returns the directory that the cache files are written to.
### 2.4.4 Class LSCDataFindNode

A DataFindNode runs an instance of LSCdataFind in a Condor DAG.

Note: This class overrides the LSCDataFindNode class within glue.pipeline, it has support for doing running datafind jobs for multiple frame types within the same DAG. This will be eventually be merged into the main glue.pipeline.

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__(self, job)</code></td>
<td>A CondorDAGJob that can run an instance of LSCdataFind.</td>
</tr>
<tr>
<td><code>get_output(self)</code></td>
<td>Return the output file, i.e. the file containing the frame cache data.</td>
</tr>
<tr>
<td><code>set_end(self, time)</code></td>
<td>Set the end time of the datafind query.</td>
</tr>
<tr>
<td><code>set_observatory(self, obs)</code></td>
<td>Set the IFO to retrieve data for. Since the data from both Hanford interferometers is stored in the same frame file, this takes the first letter of the IFO (e.g. L or H) and passes it to the --observatory option of LSCdataFind.</td>
</tr>
</tbody>
</table>
2.4. Python Module stochastic

```python
set_start(self, time)
```

Set the start time of the datafind query.

**Parameters**

- `time`: GPS start time of query.

Overrides: glue.pipeline.AnalysisNode.set_start

```python
set_type(self, type)
```

Set the frame type to retrieve data for

**Parameters**

- `type`: Frame type to obtain data for.

Inherited from AnalysisNode: calibration, calibration_cache_path, get_calibration, get_end, get_ifo, get_ifo_tag, get_input, get_start, get_trig_end, get_trig_start, set_cache, set_ifo, set_ifo_tag, set_input, set_output, set_trig_end, set_trig_start

Inherited from CondorDAGNode: __repr__, add_file_arg, add_input_file, add_macro, add_output_file, add_parent, add_post_script_arg, add_pre_script_arg, add_var_arg, add_var_opt, get_args, get_cmd_line, get_input_files, get_name, get_opts, get_output_files, get_vds_group, job, set_log_file, set_name, set_post_script, set_pre_script, set_retry, set_vds_group, write_input_files, write_job, write_output_files, write_parents, write_post_script, write_pre_script, write_vars

### 2.4.5 Class StochasticError

```python
exceptions.Exception
```

**Methods**

- `__init__(self, args=None)`

  Overrides: exceptions.Exception.__init__

  Inherited from Exception: __getitem__, __str__

### 2.4.6 Class StochasticJob

```python
glue.pipeline.AnalysisJob
glue.pipeline.CondorJob
glue.pipeline.CondorDAGJob
```

**StochasticJob**

A lalapps_stochastic job used by the stochastic pipeline. The static options are read from the section [stochastic] in the ini file. The stdout and stderr from the job are directed to the logs directory. The path to the executable and the universe is determined from the ini file.
2.4.7 Class StochasticNode

An StochasticNode runs an instance of the stochastic code in a Condor DAG.

Methods

```python
__init__(self, job)
```

Parameters

- **job**: A CondorDAGJob that can run an instance of lalapps_stochastic.

Overrides: glue.pipeline.AnalysisNode.__init__

```python
get_output(self)
```

Returns the file name of output from the stochastic code. This must be kept synchronized with the name of the output file in stochastic.c.

Overrides: glue.pipeline.AnalysisNode.get_output

```python
set_cache_one(self, file)
```

Set the LAL frame cache to use. The frame cache is passed to the job with the --frame-cache-one argument.

Parameters

- **file**: calibration file to use.
### set_cache_two(self, file)

Set the LAL frame cache to use. The frame cache is passed to the job with the `--frame-cache-two` argument.

**Parameters**

- `file`: calibration file to use.

### set_calibration_one(self, ifo, start)

Set the path to the calibration cache file for the given IFO. During S2, the Hanford 2km IFO had two calibration epochs, so if the start time is during S2, we use the correct cache file.

**Parameters**

- `ifo`: IFO code (e.g. L1, H1, H2 or G1).
- `start`: GPS time for calibration.

### set_calibration_two(self, ifo, start)

Set the path to the calibration cache file for the given IFO. During S2, the Hanford 2km IFO had two calibration epochs, so if the start time is during S2, we use the correct cache file.

**Parameters**

- `ifo`: IFO code (e.g. L1, H1, H2 or G1).
- `start`: GPS time for calibration.

### set_f_max(self, f_max)

Set the maximum frequency.

**Parameters**

- `f_max`: Maximum frequency.

### set_f_min(self, f_min)

Set the minimum frequency.

**Parameters**

- `f_min`: Minimum frequency.

### set_f_ref(self, f_ref)

Set the reference frequency.

**Parameters**

- `f_ref`: Reference frequency.

### set_ifo_one(self, ifo)

Set the interferometer code to use as IFO One.

**Parameters**

- `ifo`: IFO code (e.g. L1, H1, H2 or G1).
2.4. Python Module stochastic

### set_ifo_two(self, ifo)
Set the interferometer code to use as IFO Two.

**Parameters**

ifo: IFO code (e.g. L1, H1, H2 or G1).

### set_output_dir(self, dir)
Set the output directory

**Parameters**

dir: Directory for output files.

### set_user_tag(self, usertag)
Set the user tag

**Parameters**

usertag: User tag to append to job.

Inherited from AnalysisNode: calibration, calibration_cache_path, get_calibration, get_end, get_ifo, get_ifo_tag, get_input, get_start, get_trig_end, get_trig_start, set_cache, set_end, set_ifo, set_ifo_tag, set_input, set_output, set_start, set_trig_end, set_trig_start

Inherited from CondorDAGNode: __repr__, add_file_arg, add_input_file, add_macro, add_output_file, add_parent, add_post_script_arg, add_pre_script_arg, add_var_arg, add_var_opt, get_args, get_cmd_line, get_input_files, get_name, get_opts, get_output_files, get_vds_group, job, set_log_file, set_name, set_post_script, set_pre_script, set_retry, set_vds_group, write_input_files, write_job, write_output_files, write_parents, write_post_script, write_pre_script, write_vars

2.4.8 Class StoppJob

A lalapps_stopp job used by the stochastic pipeline. The static options are read from the section [stopp] in the ini file. The stdout and stderr from the job are directed to the logs directory. The path to the executable and the universe is determined from the ini file.

**Methods**

`__init__(self, cp)`

**Parameters**

cp: ConfigParser object from which options are read.

Overrides: glue.pipeline.CondorDAGJob.__init__

Inherited from AnalysisJob: channel, get_config, is_dax, set_channel

Inherited from CondorDAGJob: add_var_arg, add_var_opt

Inherited from CondorJob: add_arg, add_condor_cmd, add_ini_opts, add_opt, add_short_opt, get_args, get_executable,
2.4.9 **Class StoppNode**

```python
    glue.pipeline.CondorDAGNode ──
                     │
                     │
glue.pipeline.AnalysisNode ──
                               │
                               │
glue.pipeline.CondorDAGNode ── StoppNode
```

An StoppNode runs an instance of the stochastic stopp code in a Condor DAG.

**Methods**

```python
__init__(self, job)
```

**Parameters**

- `job`: A CondorDagNode that can run an instance of lalapps_stopp.

Overrides: glue.pipeline.AnalysisNode.__init__

**Inherited from** **AnalysisNode**: calibration, calibration_cache_path, get_calibration, get_end, get_ifo, get_ifo_tag, get_input, get_output, get_start, get_trig_end, get_trig_start, set_cache, set_end, set_ifo, set_ifo_tag, set_input, set_output, set_start, set_trig_end, set_trig_start

**Inherited from** **CondorDAGNode**: __repr__, add_arg, add_input_file, add_macro, add_output_file, add_parent, add_post_script_arg, add_pre_script_arg, add_var_arg, add_var_opt, get_args, get_cmd_line, get_input_files, get_name, get_opts, get_output_files, get_vds_group, job, set_log_file, set_name, set_post_script, set_pre_script, set_retry, set_vds_group, write_input_files, write_job, write_output_files, write_parents, write_post_script, write_pre_script, write_vars
Chapter 3

Inspiral Search Programs

This section of LALAPPS contains programs that can be used to search interferometer data for inspiral signals using templated matched filtering and associated veto strategies.

3.1 Description of the Components of the Pipeline

A short description of each possible part of the inspiral pipeline follows. The more detailed description of each command-line argument can be found in the specific pages for each part of the code.

3.1.1 Template Bank Generation

`lalapps_tmpltbank`: Calculates the power spectrum and generates a template bank for an analysis chunk of LIGO or GEO data. The bank can be for any of the different kinds of inspiral signals: TaylorF2, BCV etc. The output is written in an aml file.

There is also the capability of outputting the raw data, the response function and the uncalibrated power spectrum in frame files.

3.1.2 Matched Filtering and Injections

`lalapps_inspiral`: Estimates the power spectrum and performs matched filtering for inspiral signals on LIGO or GEO data. The filter can be for either binary neutron star inspirals or for spinning or non-spinning black hole inspirals. It also has the capability of doing software injections of inspiral signals on the data. The resulting triggers are written in an xml file.

There is also the capability of outputting the raw data before and after the injections, the response function, the uncalibrated power spectrum, the snr, the chisq time series, and the complex C time series required by the coherent code, all in frame files.

`lalapps_coherent_inspiral`: Takes C data frame files as outputted by inspiral.c for up to 4 detectors and constructs the optimal coherent SNR time series. Triggers are written to an xml file and the time series is written out as a frame file.

3.1.3 Triggered Template Bank, Trigger Coincidence and Time Slides

`lalapps_inca`: Currently performs three different tasks:

Triggereed template bank generation: Given the triggers from one interferometer in a sngl_inspiral table, it generates a triggered tempalte bank to be used in filtering the data from a second interferometer.

Trigger coincidence: Given the triggers from two interferometers, it checks which of those are time-coincident and coincident in mass (for the BNS search) or in the parameters psi0 and psi3 (for the BCV search).

Time slides: Time slides the data by a given time and performs coincidence, as described above.

All results are written in xml files.
lalapps_thinca: This code is intended as a generalization of inca. It is not fully implemented at the present time. For now, it can take in triggers from any number of instruments and return triggers which are coincident in two of the instruments. The coincidence test is performed over time and mass parameters. Furthermore, the events are labelled with a unique id in order to facilitate post processing.

All results are written in xml files.

lalapps_trigbank: This code generates a triggered template bank. Given an input of a \texttt{sngl_inspiral} table, it generates a triggered template bank to be used in filtering the data from a second interferometer. All results are written in xml files.

3.1.4 Trigger Summary and Injection Analysis

lalapps_sire: Currently performs two different tasks:

Summary and clustering of inspiral triggers: Using the xml files with the inspiral triggers, it summarizes them and time-clusters them, using a specified clustering algorithm.

Injection analysis: Performs time-coincidence between the inspiral triggers and a list of injection parameters. An injection is considered "found" if there is an inspiral trigger that is time-coincident with it, within a specified time interval.

All results are written in xml files.

3.1.5 Injection Parameter Generation

lalapps_inspinj: Given a mass-file, it generates the mass, distance and time parameters of BNS signals to be injected in the data. It currently generates the signals coming from locations within the galaxies of the Local Group, but the ability to do different distributions will be added soon.

The result is an xml file.

lalapps_bbhinj: It generates the mass, distance and time parameters of inspiral signals to be injected in the data. Despite its name, it can be used for BNS and for BBH parameter generation.

The result is an xml file.

3.1.6 Injection in Frames

lalapps_inspfrinj: Given a frame file with LIGO or GEO data, it injects inspiral signals on the data.

3.1.7 Splitting Large Template Banks

lalapps_splitbank: Given a template bank in an xml file, it splits that bank into smaller banks.
3.2 Program `lalapps_inspiral_pipe`

Name

`lalapps_inspiral_pipe` — python script to generate Condor DAGs to run the inspiral pipeline.

Synopsis

```
-h, --help            display this message
-v, --version        print version information and exit
-u, --user-tag TAG   tag the job with TAG (overrides value in ini file)
-d, --datafind       run LALdataFind to create frame cache files
-t, --template-bank  run lalapps_tmpltbank to generate a template bank
-i, --inspiral       run lalapps_inspiral on the first IFO
-T, --triggered-bank run lalapps_trigtotmplt to generate a triggered bank
-I, --triggered-inspiral run lalapps_inspiral on the second IFO
-C, --coincidence    run lalapps_inca on the triggers from both IFOs
-j, --injections     add simulated inspirals from injection file
-p, --playground-only only create chunks that overlap with playground
-F, --priority PRIO  run jobs with condor priority PRIO
-f, --config-file FILE use configuration file FILE
-l, --log-path PATH  directory to write condor log file
```

Description

`lalapps_inspiral_pipe` generates a Condor DAG to run the inspiral analysis pipeline. The configuration file should specify the parameters needed to run the jobs and must be specified with the `--config-file` option. A typical .ini file is the following:

```
; S2 inspiral pipeline configuration script.
;
; $Id: inifile.tex,v 1.2 2004/10/08 22:49:02 emessarit Exp $
;
; this is the configuration file for the inspiral DAG generation program that
; creates a condor DAG to run the inspiral analysis pipeline.

[condor]
universe = standard
datafind = LSCdataFind
tmpltbank = lalapps_tmpltbank
inspiral = lalapps_inspiral
trigtotmplt = lalapps_inca
inca = lalapps_inca

[pipeline]
version = $Id: inifile.tex,v 1.2 2004/10/08 22:49:02 emessarit Exp $
user-tag =
ifo1 = L1
ifo2 = H1
ifo1-snr-threshold = 6.0
ifo2-snr-threshold = 6.0
ifo1-chisq-threshold = 100.0
ifo2-chisq-threshold = 100.0
```
[input]
segments = S2H1L1v04_selectedsegs.txt
channel = LSC-AS_Q

[calibration]
path = /ldas_outgoing/calibration/cache_files
L1 = L1-CAL-V03-729273600-734367600.cache
H1 = H1-CAL-V03-729273600-734367600.cache
H2-1 = H2-CAL-V03-729296220-731849040.cache
H2-2 = H2-CAL-V03-731849076-734367576.cache
H2-cal-epoch-boundary = 731849076

[datafind]
type = RDS_R_L1
lal-cache =

[data]
pad-data = 8
segment-length = 1048576
number-of-segments = 15
sample-rate = 4096
resample-filter = ldas
enable-high-pass = 100.0
spectrum-type = median
low-frequency-cutoff = 100.0
high-pass-order = 8
high-pass-attenuation = 0.1

[tmpltbank]
minimum-mass = 3.0
maximum-mass = 20.0
minimal-match = 0.95
high-frequency-cutoff = 2048.0
order = twoPN
approximant = TaylorF2 (for BNS) BCV (for BCV)
space = Tau0Tau3 (for BNS) Psi0Psi3 (for BCV)
debug-level = 33
; the following are necessary for the BCV search
minimum-psi0 = 10.0
maximum-psi0 = 550000.0
minimum-psi3 = -4000.0
maximum-psi3 = -10.0
alpha = 0.0
maximum-fcut-tmplts = 3
; end of BCV-necessary tmpltbank arguments

[inspiral]
minimal-match = 0.9
segment-overlap = 524288
inverse-spec-length = 16
dynamic-range-exponent = 69.0
enable-output =
enable-event-cluster =
chisq-bins = 0
debug-level = 33
approximant = TaylorF2 {for BNS} BCV {for BCV}

[trigtotmplt]
minimal-match = 0.95
parameter-test = m1_and_m2 {for BNS} psi0_and_psi3 {for BCV}
default-level = 33

[inca]
playground-only =
epsilon = 2.0
kappa = 5000.0
dt = 15.0
dm = 0.03
parameter-test = m1_and_m2 {for BNS} psi0_and_psi3 {for BCV}
default-level = 33
; the following are necessary for the BCV search only
dpsi0 = 0.0
dpsi3 = 0.0
; end of BCV-necessary arguments

A file containing science segments to be analyzed should be specified in the [input] section of the configuration file with a line such as

segments = S2H1L1v03_selectedsegs.txt

This should contain four whitespace separated columns:

    segment_id    gps_start_time    gps_end_time    duration

that define the science segments to be used. Lines starting with an octothorpe are ignored. Segment files can be generated by running segwizard.

The analysis chunk size is determined from the number of data segments and their length and overlap specified in config file. A chunk length is is 2048 seconds for S2. The chunks start and stop times are computed from the science segment times and used to build the DAG.

Once the DAG file has been created it should be submitted to the Condor pool with the condor_submit_dag command.

Options

--help
Display a brief usage summary.

Example

Generate a DAG to run an inspiral search on the first IFO. The generated DAG is then submitted with condor_submit_dag

lalapps_inspiral_pipe --log-path /people/duncan/dag_logs \
--datafind --template-bank --inspiral --playground-only \
--config-file 11_s2.ini

condor_submit_dag 11_s2.dag

Author
Duncan Brown
3.3 Program lalapps_inspiral_hipe

Name
lalapps_inspiral_hipe — python script to generate Condor DAGs to run the inspiral hierarchical pipeline.

Synopsis
```
-h, --help                      show this help message and exit
-v, --version                  print version information and exit
-u USERTAG, --user-tag= USERTAG
                                tag the jobs with USERTAG (overrides value in ini file)
-g, --g1-data                  analyze g1 data
-a, --h1-data                  analyze h1 data
-b, --h2-data                  analyze h2 data
-l, --l1-data                  analyze l1 data
-S, --one-ifo                  analyze single ifo data (not usable for GEO)
-D, --two-ifo                  analyze two interferometer data
-T, --three-ifo                analyze three interferometer data
-Q, --four-ifo                 analyze four interferometer data
-A, --analyze-all              analyze all ifos and all data (over-rides above)
-d, --datafind                 run LSCdataFind to create frame cache files
-t, --template-bank            run lalapps_tmpltbank to generate template banks
-i, --inspiral                 run lalapps_inspiral to generate triggers
-c, --coincidence              run lalapps_thinca to test for coincidence
-B, --trigbank                 run lalapps_trigbank for banks of coinc triggers
-V, --inspiral-veto            run lalapps_inspiral with vetos
-C, --second-coinc             run lalapps_thinca on the inspiral veto triggers
-j, --coherent-bank            run lalapps_coherentbank to make coherent bank
-k, --coherent-inspiral        run lalapps_coherent_inspiral for coherent analysis
-s, --sire                     do sires to sweep up triggers
-R, --read-cache               read cache file from ini-file (if LSCDataFind is broken)
-P PRIO, --priority= PRIO
                                run jobs with condor priority PRIO
-f FILE, --config-file= FILE
                                use configuration file FILE
-p PATH, --log-path= PATH
                                directory to write condor log file
-o, --output-segs              output the segment lists of analyzed data
-x, --dax                      create a dax instead of a dag
```

Description
lalapps_inspiral_hipe generates a Condor DAG to run the hierarchical inspiral analysis pipeline. It currently works for the four LSC interferometers: G1, H1, H2, L1.

The code reads in segment lists for the four instruments. If one of the segment files is not specified or is empty, it is assumed that there is no data from that instrument. From the segment files, the pipeline calculates four lists of single ifo segments, for G1, H1, H2 and L1; six lists of double ifo segments, for G1-H1, G1-H2, G1-L1, H1-H2, H1-L1 and H2-L1; four lists of three ifo data, for G1-H1-H2, G1-H1-L1, G1-H2-L1 and H1-H2-L1, and one list of four ifo segments for G1-H1-H2-L1. The options --g1-data, --h1-data, --h2-data and --l1-data allow you to choose which of the interferometers’ data to analyze. Similarly, the --one-ifo, --two-ifo, --three-ifo and --four-ifo flags determine whether to analyze times during which one, two, three or four instruments respectively were operational. Thus, by specifying --h1-data, --l1-data and --two-ifo, the pipeline will analyze only the H1-L1 double coincident times. If the --analyze-all
flag is set, the pipeline will analyze all data from all instruments. If the `--output-segments` option is chosen, the pipeline will output segment lists for the non-empty data types. The file names are "h1l1_segs_analyzed.txt" etc, or if the analysis is restricted to playground, they are "h1l1_play_segs_analyzed.txt".

The pipeline uses a coincidence stage early on in order to cut down the number of triggers for which we have to perform the computationally costly chi squared and r-squared veto computations. Thus, the pipeline performs a first inspiral stage (without vetoes) which is followed immediately by a coincidence stage. The triggers which survive in coincidence are then passed back to the inspiral code where the chi-squared and r-squared signal based vetoes are computed. The remaining triggers are then tested again for coincidence. Any triggers surviving this second coincidence stage are passed into the coherent analysis. Some of the steps described above require more than one code to be run. For example, before running the inspiral code for the first time, we must first locate the data using datafind, then generate template-banks for the analysis.

At present, the pipeline can perform the following steps of a hierarchical inspiral search: `--datafind`, `--template-bank`, `--inspiral`, `--coincidence`, `--trigbank`, `--inspiral-veto`, `--second-coinc`, `--coherent-bank` and `--coherent-inspiral`. Any or all of these options may be specified. However, each step of the pipeline relies on results files produced by the previous step (and in the case of the `inspiral`, `inspiral-veto` and `coherent-inspiral` steps, the output from `datafind` is also required).

The configuration file specifies the parameters needed to run the analysis jobs contained in the pipeline. It is specified with the `--config-file` option. A typical `.ini` file is the following:

```
; inspiral pipeline configuration script.
;
; $Id: hipeinifile.tex,v 1.4 2006/02/09 02:57:27 sfairhur Exp $
;
; this is the configuration file for the inspiral DAG generation program
; lalapps_inspiral_hipe that creates a condor DAG to run the inspiral
; analysis pipeline. It can be use to perform a simple single interferometer
; or a double coincident analysis.

[condor]
; setup of condor universe and location of executables
universe = standard
datafind = LSCdataFind
tmpltbank = ./lalapps_tmpltbank
inspiral = ./lalapps_inspiral
inca = ./lalapps_inca
thinca = ./lalapps_thinca
trigtotmplt = ./lalapps_trigbank
sire = ./lalapps_sire
cohabank = ./lalapps_coherentbank
chia = ./lalapps_coherent_inspiral

[pipeline]
; tagging information for the configure script
version = $Id: hipeinifile.tex,v 1.4 2006/02/09 02:57:27 sfairhur Exp $
cvs-tag = $Name: HEAD $
; user-tag here can be overridden on the command line of lalapps_inspiral_hipe
user-tag =
; data choice (playground_only|exclude_playground|all_data)
playground-data-mask = playground_only
```

; the segments file should be the output from segwizard with DQ flags applied
; if no segment file if specified, assumed no data from that IFO.
h1-segments = S3H1v05_selectedsegs.txt
h2-segments = S3H2v05_selectedsegs.txt
l1-segments = S3L1v05_selectedsegs.txt
g1-segments =
ligo-channel= LSC-AS_Q
ligo-type = RDS_R_L3
geo-channel = DER_DATA_H
geo-type = G1_RDS_C01_LX
geo-bank = H1_bank_4_G1.xml

; injection file (if blank then no injections)
injection-file =
; num slides (if blank or zero, then no time slides are performed)
num-slides = 20

[calibration]
; location of the calibration cache and the cache files
path = /ldas_outgoing/calibration/cache_files
L1 = L1-CAL-V03-751719553-757687373.cache
H1 = H1-CAL-V03-751651153-757672093.cache
H2 = H2-CAL-V03-751654453-757699693.cache

[datafind]
; type of data to use
match = localhost

[data]
; data conditioning parameters common to tmpltbank and inspiral
pad-data = 8
segment-length = 1048576
number-of-segments = 15
sample-rate = 4096
resample-filter = ldas
spectrum-type = median

[ligo-data]
enable-high-pass = 70.0
high-pass-order = 8
high-pass-attenuation = 0.1
low-frequency-cutoff = 70.0

[geo-data]
; data conditioning specific to GEO detector
calibrated-data = real_8
disable-high-pass =
geo-high-pass-freq = 110.0
geo-high-pass-order = 8
deo-high-pass-atten = 0.1
low-frequency-cutoff = 500.0

[tmpltbank]
; template bank generation parameters -- added to all tmpltbank jobs
minimum-mass = 1.0
maximum-mass = 20.0
minimal-match = 0.97
high-frequency-cutoff = 2048.0
order = twoPN
approximant = TaylorF2
space = Tau0Tau3
debug-level = 33

[inspiral]
; inspiral analysis parameters -- added to all inspiral jobs
approximant = TaylorF2
segment-overlap = 524288
inverse-spec-length = 16
dynamic-range-exponent = 69.0
enable-output =
cluster-method = window
cluster-window = 16
maximization-interval = 10000000
debug-level = 33
minimal-match = 0.55

[no-veto-inspiral]
; inspiral parameters specific to the first set of inspirals (pre coinc)
chisq-bins = 0
disable-rsq-veto =

[veto-inspiral]
; inspiral parameters for the second set of inspirals, after coincidence
chisq-bins = 16
enable-rsq-veto =
rsq-veto-window = 2.0
rsq-veto-threshold = 10.0

[h1-inspiral]
; h1 specific inspiral parameters
snr-threshold = 6.5
chisq-threshold = 5.0

[h2-inspiral]
; h2 specific inspiral parameters
snr-threshold = 6.5
chisq-threshold = 5.0

[l1-inspiral]
; l1 specific inspiral parameters
snr-threshold = 6.5
chisq-threshold = 5.0

[g1-inspiral]
; g1 specific inspiral parameters
snr-threshold = 8.0
chisq-threshold = 10.0
minimal-match = 0.55
inverse-spec-length = 24
dynamic-range-exponent = 65.0

[inca]
; common coincidence parameters -- added to all inca jobs
debug-level = 33

[thinca]
; common coincidence parameters -- added to all thinca jobs
debug-level = 33
parameter-test = mchirp_and_eta
h1-time-accuracy = 1
h2-time-accuracy = 1
l1-time-accuracy = 1
g1-time-accuracy = 2
h1-mchirp-accuracy = 0.01
h2-mchirp-accuracy = 0.01
l1-mchirp-accuracy = 0.01
g1-mchirp-accuracy = 0.05
h1-eta-accuracy = 0.1
h2-eta-accuracy = 0.1
l1-eta-accuracy = 0.1
g1-eta-accuracy = 0.3

[thinca-slide]
; time slide parameters
gl-slide = 15
h1-slide = 0
h2-slide = 10
l1-slide = 5
t1-slide = 20
v1-slide = 25

[trigtotmplt]
parameter-test = m1_and_m2

[sire-cluster]
; clustering parameters for sire
cluster-time = 4000
cluster-algorithm = snr

[sire-inj]
; injection coincidence
injection-coincidence = 3

[cohsbank]
; params for the coherent bank code
debug-level = 33

[coh-trig]
; params for the trigbank code in the coherent stage
debug-level = 33
parameter-test = no_test
[chia]
;params for the coherent code
debg-level = 33
maximize-over-chirp =
cohsnr-threshold = 10.0
low-frequency-cutoff = 70.0
write-cohsnr =
write-events =
sample-rate = 4096
glob-frame-data =
dynamic-range-exponent = 69.0
segment-length = 1048576

The .ini file contains several sections. The [condor] section contains the names of the executables which will run the various stages of the pipeline. The [pipeline] section gives the CVS details of the pipeline, the usertag (which can be overwritten on the command line with the --user-tag option) and the playground-data-mask which must be set to one of playground_only, exclude_playground or all_data. The input section contains the names of the segment files for the four interferometers, the channel names, frame types, injection file name and number of time slides. If any of the segment files are left blank, it is assumed that there are no segments for that instrument. Similarly, a blank injection-file signifies that no injections are to be performed, and a blank num-slides signifies no time slides.

The remaining sections set options for the various jobs to be run in the pipeline. The options in the [datafind], [tmpltbank], [inspiral], [inca], [thinca], [trigtotmplt], [cohbank] and [chia] sections are added to every instance of the relevant executable. Note that these options are set the same for all interferometers. The options in the [data] section are added to all [inspiral] and [tmpltbank] jobs, while the [ligo-data] and [geo-data] commands are added to the LIGO/GEO jobs respectively. The [calibration] information is used to determine the calibration data for these jobs. In the [ifo-thresholds] sections, the ifo specific snr, chi squared and r-squared veto thresholds for the various interferometers are set; while the [no-veto-inspiral] and veto-inspiral sections contain arguments relevant for the first and second inspiral steps respectively.

The science segments are read in from the segment files for each instrument. These science segments are split up into analysis chunks. The analysis chunk size is determined from the number of data segments and their length and overlap specified in config file. Currently, we are using 256 second analysis segments. We use 15 segments in a chunk, and overlap them by 128 seconds to give a chunk length of 2048 seconds. The chunks are constructed for each of the interferometers independently. Any science segment shorter than the length of a chunk is not analyzed. Additionally, we cannot produce triggers for the first and last overlap/2 seconds of a science segment, due to the finite length of the inspiral templates. Using this information, we construct segment lists of analyzable data for each of the fifteen types of data we may have (four single ifo, six two ifo, four three ifo and one four ifo). If the playground only option is specified, the segments are restricted to playground times. We decide which chunks should be analyzed by testing for overlap between the chunk and the data we need to analyze. Note that if the pipeline is restricted to playground data, then only the playground times are analyzed for triggers in the inspiral code. This is done by setting the trig-start-time and trig-end-time for the inspiral jobs appropriately.

Once the DAG file has been created it should be submitted to the Condor pool with the condor_submit_dag command.

Options

--help
Display a brief usage summary.

--version
Display the version information and exit.
3.3. Program **lalapps_inspiral_hipe**

---**user-tag** **USERTAG**
Set the user-tag to **USERTAG**. This overrides the user-tag which may have been set in the ini file. The user-tag will be added to the names of the output files of the pipeline.

---**g1-data**
Analyze the G1 data, the times of which are determined from the **g1-segments** file specified in the ini file. If not set, then no data when G1 was operational will be analyzed.

---**h1-data**
Analyze the H1 data, the times of which are determined from the **h1-segments** file specified in the ini file. If not set, then no data when H1 was operational will be analyzed.

---**h2-data**
Analyze the H2 data, the times of which are determined from the **h2-segments** file specified in the ini file. If not set, then no data when H2 was operational will be analyzed.

---**l1-data**
Analyze the L1 data, the times of which are determined from the **l1-segments** file specified in the ini file. If not set, then no data when H1 was operational will be analyzed.

---**one-ifo**
Analyze any times when one and only one instrument was operational. Note that this option works together with the IFO options given above. For example if **--one-ifo** and **--h2-data** were specified, then only the single IFO H2 times would be analyzed.

---**two-ifo**
Analyze any times when two instruments were operational. Note that this option works together with the IFO options given above. For example if **--two-ifo**, **h1-data** and **--h2-data** were specified, then the times when only H1 and H2 were operational would be analyzed. However, if only **--two-ifo** and **h1-data** were specified, no data would be analyzed.

---**three-ifo**
Analyze any times when three instruments were operational. Note that this option works together with the IFO options given above. For example if **--three-ifo**, **h1-data**, **--h2-data** and **--l1-data** were specified, then the times when H1, H2 and L1 were operational, but G1 was not, would be analyzed.

---**four-ifo**
Analyze any times when all four instruments were operational. Note that this option works together with the IFO options given above. For example if **--four-ifo**, **g1-data**, **h1-data**, **--h2-data** and **--l1-data** were specified, then the times when all of G1, H1, H2 and L1 were operational would be analyzed.

---**analyze-all**
Analyze all ifos and all data. This is equivalent to setting all six of the options above. Then, all the data is analyzed.

---**datafind**
Run the datafind step of the pipeline.

---**template-bank**
Run the template-bank step of the pipeline. Note that the template-bank jobs require the cache files created by datafind, so **--datafind** must either be run in the pipeline or have been run previously.

---**inspiral**
Run the inspiral step of the pipeline. These jobs will take the arguments from the [no-veto-inspiral] section in the ini file. Note that the inspiral jobs require the cache files created by datafind and template banks, so both **--datafind** and **template-bank** must either be run in the pipeline or have been run previously.
--coincidence
Run the coincidence step of the pipeline. Times when two or more detectors were operational are analyzed by the thinca code. This determines all coincidences between two or more operational detectors. If num-slides is specified then time slides are also performed, and output in a separate file named THINCA_SLIDE. Finally, for the times when only one instrument was on, inca is run (in single ifo mode) to sweep up the single ifo triggers. These triggers are not used by later stages of the pipeline. Note that the thinca and inca jobs require the inspiral triggers created by inspiral, so that --inspiral must either be run in the pipeline or have been run previously. For each single IFO segment, the coincidence step simply creates a file containing all the triggers in the time interval. For two/three/four IFO segments, the coincidence step performs coincidence and outputs the double, triple and quadruple coincidence triggers in one file, and time slide coincident triggers in a second file.

--trigbank
Run the triggered bank step of the pipeline. This step takes in the coincident, and thinca slide, triggers. It outputs those triggers which should be filtered by a specific instrument in the follow-up inspiral stage. This is done by keeping those triggers from the relevant ifo, within the times of the chunk to be analyzed by the inspiral code.

--inspiral-veto
Run the second inspiral step of the pipeline. These jobs will take the arguments from the [veto-inspiral] section in the ini file, and are intended to do the computationally costly signal based vetoes of coincident triggers. These jobs are differentiated from the first inspiral jobs by the inclusion of an ifo-tag, so an example job may be named H1-INSPIRAL_H1L1-GPSTIME-DURATION.xml. Note that the inspiral jobs require the cache files created by datafind and trigbanks, so both --datafind and trigbank must either be run in the pipeline or have been run previously.

--second-coinc
Re-run the coincidence step of the pipeline. This runs the thinca code on all times when two or more instruments were operational. As with the first coinc stage, this will perform both the zero lag and time slide analysis if requested. The output from these jobs also have an ifo-tag added, so an example output file might be H1H2L1-THINCA_H1H2L1-GPSTIME-DURATION.xml for the zero lag and H1H2L1-THINCA_SLIDE_H1H2L1-GPSTIME-DURATION.xml for the time slides.

--coherent-bank
Run the coherent bank step of the pipeline. This generates template banks ready for the coherent analysis. These banks are generated from the triggers output in the second coincidence stage of the pipeline.

--coherent-inspiral
Run the coherent inspiral step of the pipeline. This runs the inspiral code, using the coherent-bank. The inspiral code outputs frames containing time series of snr data around the time of a coincident event. These frames are then read in by the coherent inspiral code which calculates the coherent signal to noise ratio of the event.

--sire
Run sire on the various stages of the pipeline. This will collect together the triggers from jobs performed in various stages of the pipeline

--read-cache
Specify a static cache file to use during the analysis. This should only be used if LSCdataFind is not available where the pipeline is being run.

--priority PRIO
Set the condor priority PRIO of the condor jobs to be run in the pipeline.

--config-file config_file
Set the name of the configuration file to be CONFIG_FILE. This is the which is used to determine the parameters for the pipeline. This is a required argument.
--log-path
The directory in which to write the condor log file. This should generally be a local directory of the condor submit machine. This is a required argument.

--output-segs
Output the segment lists of analyzed data. Up to seven files will be output, one for each of the types of interferometer data (H1, H2, L1, H1_H2, H1_L1, H2_L1, H1_H2_L1). Any segment lists which are non-empty will be written.

--dax
Output a dax rather than a dag.

Author
Steve Fairhurst, Darren Woods
3.4 Program *lalapps_tmpltbank*

Name

*lalapps_tmpltbank* — program to generate inspiral template banks.

Synopsis

```bash
lalapps_tmpltbank
[ --help ]
[ --verbose ]
[ --version ]
[ --debug-level <level> ]
[ --user-tag <usertag> ]
[ --comment <comment> ]
--gps-start-time <gps_start>
--gps-end-time <gps_end>
[ --pad-data <time_pad> ]
[ --glob-frame-data ]
[ --frame-type <type> ]
[ --frame-cache <cache_file> ]
--calibration-cache <cal_file>
--glob-calibration-data
--channel-name <channel>
[ --calibrated-data <cal_type> ]
[ --geo-high-pass-freq <geo_freq> ]
[ --geo-high-pass-order <geo_order> ]
[ --geo-high-pass-atten <geo_atten> ]
--sample-rate <sample_freq>
--resample-filter <filter_type>
[ --disable-high-pass ]
[ --enable-high-pass <high_freq> ]
[ --high-pass-order <high_order> ]
[ --high-pass-attenuation <high_atten> ]
--spectrum-type <spectype>
[ --dynamic-range-exponent <exp> ]
--segment-length <seglen>
[ --number-of-segments <segnum> ]
[ --standard-candle ]
[ --candle-snr <candle_snr> ]
[ --candle-mass1 <candle_mass1> ]
[ --candle-mass2 <candle_mass2> ]
--low-frequency-cutoff <cutlow>
--high-frequency-cutoff <cuthigh>
[ --minimum-mass <minmass> ]
[ --maximum-mass <maxmass> ]
[ --minimum-psi0 <psio0min> ]
[ --maximum-psi0 <psio0max> ]
[ --minimum-psi3 <psi3min> ]
[ --maximum-psi3 <psi3max> ]
[ --maximum-fcut-tmplts <maxTemp> ]
[ --alpha <alpha> ]
--minimal-match <match>
--order <order>
--approximant <approx>
--space <space>
[ --write-raw-data ]
```
Options

The following command line arguments are available when running tmpltbank.c

`--alpha <alpha>`
Set BCV amplitude correction to <alpha>.

`--approximant <approx>`
Sets the approximant of the waveform to <approx>. TaylorT2 is the standard stationary phase frequency domain chirp used in the BNS search. Available parameters: TaylorT1, TaylorT2, TaylorT3, TaylorF1, TaylorF2, PadeT1, PadeF1, EOB, BCV, SpinTaylorT3, BCVSpin.

`--calibrated-data <type>`
Calibrated data of <type> real_4 or real_8.

`--calibration-cache <cal_file>`
Obtain calibration from LAL frame cache <cal_file>.

`--candle-mass1 <candle_mass1>`
Mass <candle_mass1> of first component in candle binary. Must be specified is the option `--standard-candle` is set.

`--candle-mass2 <candle_mass2>`
Mass <candle_mass2> of second component in candle binary. Must be specified is the option `--standard-candle` is set.

`--candle-snr <candle_snr>`
Set the signal-to-noise ratio of standard candle to <candle_snr>. Must be specified is the option `--standard-candle` is set.

`--channel-name <channel>`
Read data from interferometer channel <channel>.

`--comment <comment>`
Set the process table comment to <comment>.

`--debug-level <level>`
Sets the LAL debug level to <level>. 1 and 33 are the commonest choices for this code. Debug level 1 is used for developing code because it enables memory and error checking. Since memory checking is slow the correct choice for production code is 33. See the LAL documentation for more information.

`--disable-high-pass`
Turn off the IIR highpass filter. This is an optimistic option. Someday the data will be so good we won’t need high pass filtering!

`--dynamic-range-exponent <exp>`
Set dynamic range scaling to $2^{exp}$.

`--enable-high-pass <high_freq>`
High pass data above <high_freq> Hz using an IIR filter.

`--frame-cache <cache_file>`
This option is used instead of `--glob-frame-data` to read frame data from a frame cache file <cache_file>. 
--frame-type <type>
  This option specified the type of frames containing the input data. This option must be specified with the
  --glob-frame-data option.

--geo-high-pass-atten <geo_atten>
  Set the attenuation of the high pass filter to <geo_atten>. Only if --calibrated-data is set to real_8.

--geo-high-pass-freq <geo_freq>
  This sets the high pass filter frequency for GEO data above <geo_freq> Hz using an IIR filter. Only if --calibrated-data
  is set to real_8.

--geo-high-pass-order <geo_order>
  Set the order of the GEO high pass filter to <geo_order>. Only if --calibrated-data is set to real_8.

--glob-calibration-data
  Is this option is specified, the calibration is obtained by globbing in the working directory.

--glob-frame-data
  This option along with --frame-type can be used instead of --frame-cache to read data stored locally in
  the working directory. It finds files of the specified frame type with a *.gwf extension.

--gps-end-time <gps_end>
  Set the integer part of the GPS time <gps_end> you want to stop reading data.

--gps-start-time <gps_start>
  Set the integer part of the GPS time <gps_start> from which you wish to begin reading data.

--help
  display the help message which gives brief explanations of the command arguments.

--high-frequency-cutoff <cuthigh>
  Do not filter above <cuthigh> Hz.

--high-pass-attenuation <high_atten>
  Set the attenuation of the high pass filter to <high_atten>.

--high-pass-order <high_order>
  Set the order of the high pass filter to <high_order>.

--low-frequency-cutoff <cutlow>
  Do not filter below <cutlow> Hz.

--maximum-fcut-tmplts <maxTemp>
  Set the maximum number of templates in fcut direction to <maxTemp>.

--maximum-mass <maxmass>
  Set maximum component mass of bank to <maxmass>.

--maximum-psi0 <psi0max>
  Set maximum range of BCV parameter psi0 to <psi0max>.

--maximum-psi3 <psi3max>
  Set maximum range of BCV parameter psi3 to <psi3max>.

--minimal-match <match>
  Specifies the minimal match <match> between templates in the bank and all possible signals in the parameter
  space.
Set minimum component mass of bank to `<minmass>`.

Set minimum range of BCV parameter psi0 to `<psi0min>`.

Set minimum range of BCV parameter psi3 to `<psi3min>`.

Set number of data segments to `<segnum>`.

This sets the order of the waveform to `<order>`. Usually it is set to `twoPN` (second order post newtonian). Available parameters: `newtonian`, `oneHalfPN`, `onePN`, `onePointFivePN`, `twoPN`, `twoPointFivePN`, `threePN`, `threePointFivePN`.

This flag specifies an amount of time `<time_pad>` to add to the beginning and end of the input time series data. Padding the data is necessary because resampling and filtering corrupts these portions. 8 seconds is the accepted choice for this parameter. See LAL documentation for a description of resampling and high pass filtering.

Set resample filter `<filter_type>` to `ldas` or `butterworth`. In the normal case the `<ldas>` filter is used.

Specifies the sampling frequency `<sample_freq>` at which you want to filter the data downsampling if necessary.

Set data segment length to `<seglen>` points.

In order to make the template bank coordinates nice and friendly these parameters are used instead of masses. Usually `Tau0Tau3` is used. Available parameters: `Tau0Tau2`, `Tau0Tau3`, `Psi0Psi3`.

Use PSD estimator `<spec_type>` `mean` or `median` to choose how the average is calculated. Since the median average is less affected by a loud glitch `median` is used generally.

Compute a standard candle from the PSD. In that case the arguments `candle-mass1`, `candle-mass2` and `candle-snr` must also be specified.

print progress information as the code executes.

print version information and exit without running the tmpltbank code.

Set the user tag to the string `<usertag>`. This string must not contain spaces or dashes ("-"). This string will appear in the name of the file to which output information is written, and is recorded in the various XML tables within the file.

Write raw data to a frame file.

Write the computed response function to a frame.
--write-spectrum
Write the uncalibrated psd to a frame.

--write-strain-spectrum
Write the calibrated strain psd to a text file.

Description
lalapps_tmpltbank is a stand alone code for generating inspiral template banks for LIGO or GEO data with the LAL bank package. The code generates a calibrated power spectrum at the specified time for the requested channel and uses this to compute the template bank. The number of templates and the values of the bank parameters in the bank also depend on the minimal match, the minimum and maximum values of mass1 and mass2 (for the BNS search) or the minimum and maximum values of psi0, psi3, the bank-alpha and the number of fcut values (for the BCV search), which are all command-line arguments. Other necessary pieces of information are the approximant and its order and the space that the template bank will be laid on. The output of the code is an xml file and the bank is contained in a sngl_inspiral table. The code has also the capability of outputing the raw data, the response function and the calibrated and unclibrated power spectra to frame files. See the LAL bank package documentation for detailed information on the algorithms used to generate the template banks.

Example
lalapps_tmpltbank \
--gps-start-time 734357353 --gps-end-time 734358377 \
--frame-cache_cache/L-734357345-734361107.cache \
--segment-length 1048576 --number-of-segments 7 \
--pad-data 7 --sample-rate 4096 --resample-filter ldas \
--enable-high-pass 5.000000e+01 --spectrum-type median \
--low-frequency-cutoff 7.000000e+01 --high-frequency-cutoff 2.048000e+03 \
--minimum-mass 1.000000e+00 --maximum-mass 3.000000e+00 \
--minimal-match 9.700000e-01 --calibration-cache \
/ldas_outgoing/calibration/cache_files/L1-CAL-V03-729273600-734367600.cache \
--space Tau0Tau3 --approximant TaylorT1 --order twoPN \
--channel-name L1:LSC-AS_Q --debug-level 33

Author
Duncan Brown and Alexander Dietz
3.5 Program lalapps_inspiral

Name
lalapps_inspiral — stand alone inspiral search code

Synopsis
lalapps_inspiral
  [--help]
  [--verbose]
  [--version]
  --debug-level <LEVEL>
  [--user-tag <STRING>]
  [--ifo-tag <STRING>]
  --gps-start-time <SEC>
  --gps-start-time-ns <NS>
  --gps-end-time <SEC>
  --gps-end-time-ns <NS>
  --pad-data <T>
  [--slide-time <T>]
  [--slide-time-ns <T>]
  --glob-frame-data
  --frame-type <TAG>
  --frame-cache <FILE>
  --calibration-cache <FILE>
  --channel-name <CHAN>
  --calibrated-data <TYPE>
  [--geo-high-pass-freq <F>]
  [--geo-high-pass-order <O>]
  [--geo-high-pass-atten <A>]
  --injection-file <FILE>
  --inject-overhead
  --enable-filter-inj-only
  --disable-filter-inj-only
  --bank-file <FILE>
  --minimal-match <M>
  [--start-template <N>]
  [--stop-template <N>]
  --sample-rate <F>
  --resample-filter <TYPE>
  --disable-high-pass
  --enable-high-pass <F>
  --high-pass-order <O>
  --high-pass-attenuation <A>
  --spectrum-type <TYPE>
  --segment-length <N>
  --number-of-segments <N>
  --segment-overlap <N>
  --low-frequency-cutoff <F>
  --inverse-spec-length <T>
  --dynamic-range-exponent <X>
  --approximant <APPROX>
  --chisq-bins <P>
  --snr-threshold <RHO>
  --chisq-threshold <X>
  --cluster-method <MTHD>
Description

lalapps_inspiral is a stand alone code for performing matched filtering for inspiral signals on LIGO or GEO data for gravitational wave signals and Monte Carlo analysis; it also has the capability of doing software signal injections on the data.

Options

--help
Optional. Display a brief usage summary.

--verbose
Optional. Print progress information.

--debug-level <LEVEL>
Set the LAL debug level to <LEVEL>. For example: 1 :developer, 33 :production.

user-tag <STRING>
Optional. Set the process-params usertag to <STRING>.

--ifo-tag <STRING>
Optional. Set the ifotag to <STRING> for file naming.

--gps-start-time <SEC>
GPS seconds <SEC> of data start time.

--gps-start-time-ns <NS>
GPS nanoseconds <NS> of data start time. You have an option of selecting the (start/end time) to be either both GPS seconds or GPS nanoseconds.

--gps-end-time <SEC>
GPS seconds <SEC> of data end time.
--gps-end-time-ns <NS>
  GPS nanoseconds <NS> of data end time.

--pad-data <SEC>
  pad the data start and end time by a number of seconds <SEC>.

--slide-time <SEC>
  Optional. Slide data start epoch by a number of seconds <SEC>.

--slide-time-ns <NS>
  Optional. Slide data start epoch by a number of nanoseconds <NS>.

--glob-frame-data
  Optional. Glob *.gwf files in the pwd to obtain frame data.

--frame-type <TAG>
  Optional. Input data is contained in frames of type <TAG>.

--frame-cache <FILE>
  Obtain frame data from LAL frame cache file <FILE>.

--calibration-cache <FILE>
  Obtain calibration from LAL frame cache file <FILE>.

--channel-name <CHAN>
  Read data from interferometer channel <CHAN> ex: L1-LSC:AS.Q, etc.

--calibrated-data <TYPE>
  Calibrated data of type <TYPE> (real_4|real_8). You have a choice of using the calibration cache or calibrated data.

--geo-high-pass-freq <F>
  Optional. High pass GEO data above a given frequency <F> [Hz] using an IIR filter.

--geo-high-pass-order <O>
  Optional. Set the order of the GEO high pass filter to an order <O>.

--geo-high-pass-atten <A>
  Optional. Set the attenuation of the high pass filter to an attenuation <A>.

--injection-file <FILE>
  Optional. Inject simulated inspiral signals from file <FILE>.

--injection-overhead
  Optional. Inject signals directly overhead detector.

--enable-filter-inj-only
  Enables the mechanism to filter only segments with injections. All other segments are not filtered. Either
  --enable-filter-inj-only or --disable-filter-inj-only must be specified.

--disable-filter-inj-only
  Disables the mechanism to filter only segments with injections. All segments are filtered. Either
  --enable-filter-inj-only or --disable-filter-inj-only must be specified.

--injection-overhead
  Optional. Inject signals directly overhead detector.

--bank-file <FILE>
  Read template bank parameters from file <FILE>.
--minimal-match \textit{<M>}

Override bank minimal match with \textit{<M>} (sets delta). This value is usually set to 0.97 (delta = 0.03). If you plan to do a run with injections (\textit{--injection-file \textit{<FILE>}}, this value should be set to 0.8 (delta = 0.2) to ensure recovering injections.

--start-template \textit{<N>}

Optional. Start filtering at template number \textit{<N>} in bank.

--stop-template \textit{<N>}

Optional. Stop filtering at template number \textit{<N>} in bank.

--sample-rate \textit{<F>}

Filter data at a given frequency \textit{<F>} [Hz], downsampling if necessary.

--resample-filter \textit{<TYPE>}

Set resample filter to a given type \textit{<TYPE>} (\textit{lidas}|\textit{butterworth}).

--disable-high-pass

Turn off the IIR highpass filter.

--enable-high-pass \textit{<F>}

High pass data above \textit{<F>} Hz using an IIR filter.

--high-pass-order \textit{<O>}

Set the order of the high pass filter to a given order \textit{<O>}.

--high-pass-attenuation \textit{<A>}

Set the attenuation of the high pass filter to a given attenuation \textit{<A>}.

--spectrum-type \textit{<TYPE>}

Use PSD estimator type \textit{<TYPE>} (\textit{mean}|\textit{median}).

--segment-length \textit{<N>}

Set data segment length to a number \textit{<N>} of points.

--number-of-segments \textit{<N>}

Set number of data segments to a number \textit{<N>}.  

--segment-overlap \textit{<N>}

Overlap data segments by a number \textit{<N>} of points.

--low-frequency-cutoff \textit{<F>}

Do not filter below a given frequency \textit{<F>} [Hz].

--inverse-spec-length \textit{<T>}

Set length of inverse spectrum to number of a given number of seconds \textit{<T>}.  

--dynamic-range-exponent \textit{<X>}

Set dynamic range scaling to \(2^\textit{X}\).

--approximant \textit{<APPROX>}

Set approximant to be used. (\textit{TaylorF2}|\textit{BCV})

--chisq-bins \textit{<P>}

Set number of chisq veto bins to \textit{<P>}.  

--snr-threshold \textit{<RHO>}

Set signal-to-noise threshold to \textit{<RHO>}.
--chisq-threshold \(<X>\)
Set threshold on \(\chi^2\). Where \(\chi^2 < X \ast (p + \rho \ast \delta^2)\)

--cluster-method \(<MTHD>\)
Set clustering method with which to maximize over a chirp to \(<MTHD>\). \(\text{tmp|window|noClustering}\)

--cluster-window \(<SEC>\)
Optional. If \(<MTHD>\) is set to window, then the window length must be specified in seconds \(<SEC>\).

--maximization-interval \(<MSEC>\)
Optional. The maximization interval is specified in milliseconds \(<MSEC>\). When this option is invoked, the SNR is maximized in fixed intervals of \(<MSEC>\) after all templates have been filtered against all segments. A typical number might be \(<MSEC>=10\).

--ts-cluster \(<MTHD>\)
Optional. When invoked, the raw inspiral triggers over all the templates are clustered over intrinsic parameter space \((\tau_0, \tau_3)\) and end-time \((t_c)\) using the trigScan clustering algorithm. The specified method \(<MTHD>\) should be one of \((T0T3Tc|T0T3TcAS)\). When \(<MTHD>\) is set to \(T0T3Tc\), the stragglers (isolated triggers, or singletons) are not appended to the final list of triggers. For production runs, one should always append the stragglers and \(T0T3TcAS\) should be chosen. Note: the \(--ts-cluster\) option cannot be used in conjunction with the \(--maximization-interval\) option - as they invoke different clustering algorithms. At present trigScan clustering cannot be invoked for BCV searches where the templates are placed on phenomenological parameters \((\Psi_0, \Psi_3)\).

--ts-volume-safety \(<SAFETYFAC>\)
In trigScan clustering algorithm, the raw triggers are modeled as ellipsoids in the space of parameters \(\Lambda \equiv (\tau_0, \tau_3, t_c)\) using the metric over \(\Lambda\). This \(<SAFETYFAC>\) is used to scale-up the volume of the ellipsoids thereby increasing the chances of overlap with neighboring ellipsoids. It is recommended to set \(<SAFETYFAC>\) to about \((1.5)^3 \sim 3.3\).

--ts-endtime-interval \(<MSEC>\)
Optional. In the trigScan algorithm (when \(--ts-cluster\) is set to \(T0T3TcAS\)), one can maximize the stragglers over their SNRs using a \(<MSEC>\) long window slid along \(t_c\) (end time of the triggers) before appending them to the clustered triggers. This option somewhat reduces the total number of triggers at the end of the pipeline. However for safety concerns, it is recommended not to specify this option for production runs. If this option is not specified, all the stragglers are appended to the final list of triggers. Obviously this option is irrelevant when \(--ts-cluster\) is set to \(T0T3Tc\).

--enable-output
Write the results to a LIGO LW XML file.

--disable-output
Do not write LIGO LW XML output file. you have a choice between enabling or disabling the output.

--trig-start-time \(<SEC>\)
Optional. Output only triggers only after a given GPS second \(<SEC>\).

--trig-end-time \(<SEC>\)
Optional. Output only triggers before a given GPS second \(<SEC>\).

--gaussian-noise \(<VAR>\)
Optional. Replace data with gaussian noise of variance \(<VAR>\).

--random-seed \(<SEED>\)
Optional. Set random number seed for injections to \(<SEED>\) \((urandom|integer)\).

\(^1\)Note that, it is a tunable parameter whose optimum value can be determined by trial and error. If the \(<SAFETYFAC>\) is set to too large a value, nearby clusters of triggers can merge into one giving poor results. If set to too small a value (close to 1), clusters are likely to fracture resulting in high trigger rate.
--bank-simulation <N>
  Optional. Perform a number of <N> injections to test the template bank.

--sim-approximant <APX>
  Optional. Set approximant of the injected waveform to <APX>.

--sim-minimum-mass <M>
  Optional. Set minimum mass of bank injected signal to M.

--sim-maximum-mass <M>
  Optional. Set maximum mass of bank injected signal to M.

--data-checkpoint
  Optional. Checkpoint and exit after data is read in.

--checkpoint-path <PATH>
  Optional. Write checkpoint file under a given path <PATH>.

--output-path <PATH>
  Optional. Write output data to a given path <PATH>.

--write-raw-data
  Optional. Write raw data to a frame file.

--write-filter-data
  Optional. Write data that is passed to filter to a frame.

--write-response
  Optional. Write the computed response function to a frame.

--write-spectrum
  Optional. Write the uncalibrated psd to a frame.

--write-snrsq
  Optional. Write the snr time series for each data segment.

--write-chisq
  Optional. Write the r^2 time series for each data segment.

Example
To run the program, type:

lalapps_inspiral
  --verbose
  --debug-level 33
  --gps-start-time 73200096
  --gps-end-time 732902144
  --bank-file L1-TMPLTBANK-732900096-2048.xml
  --calibration-cache L-CAL-729273600-734367600.cache
  --frame-cache L1-732900096-2048.cache
  --channel-name L1:LSC-AS_Q
  --snr-threshold 6.0
  --chisq-threshold 5.0
  --pad-data 8
  --segment-length 1048576
  --number-of-segments 15
  --sample-rate 4096
3.5. Program *lalapps_inspiral*

```
--resample-filter ldas \
--enable-high-pass 100.0 \
--high-pass-order 8 \
--high-pass-attenuation 0.1 \
--spectrum-type median \
--low-frequency-cutoff 100.0 \
--approximant TaylorF2 \
--minimal-match 0.9 \
--segment-overlap 524288 \
--inverse-spec-length 16 \
--cluster-method window \
--cluster-window 0.5 \
--dynamic-range-exponent 69.0 \
--chisq-bins 15 \
--enable-output \\
```

**Author**

Duncan Brown, Andres Rodriguez, Darren Woods, Anand S. Sengupta
3.6 Program **lalapps_coherent_inspiral**

**Name**

lalapps_coherent_inspiral — coherent search code

**Synopsis**

```
lalapps_inspiral
    [--help]
    [--verbose]
    [--version]
    --debug-level <LEVEL>
    --channel-number <CHANNUMBER>
    --bank-file <FILE>
    --sample-rate <F>
    --segment-length <N>
    --low-frequency-cutoff <F>
    --cohsnr-threshold <RHO>
    [--output-path <PATH>]
    [--write-cohsnr]
    [--maximize-over-chirp]
    --H1-framefile <FILE>
    --H2-framefile <FILE>
    --L-framefile <FILE>
    --V-framefile <FILE>
    --G-framefile <FILE>
    --T-framefile <FILE>
```

**Description**

**lalapps_coherent_inspiral** is a code that optimally combines pre-filtered data for up to 4 detectors to produce an event table in the form of an xml file as well as the coherent SNR time series in the form of a frame file if the user so desires. The user must provide 2-4 frame files (1 for each detector in the network) that contain the C data for each detector. To produce these files, run inspiral.c on the raw data for 2-4 detectors (1 at a time of course) with the –write-cdata option specified. Each of these raw data segments should be filtered with the same template for a meaningful coherent SNR. For some networks with 3 or 4 detectors, the coherent code requires that the beam-pattern coefficient files be in the same directory where the code is executed. These files were generated by a mathematica notebook and their filenames are: HBeam.dat, LBeam.dat, VIRGOBeam.dat, TAMABeam.dat, and GEOBeam.dat.

**Options**

`--help`

Optional. Display a brief usage summary.

`--verbose`

Optional. Print progress information.

`--version`

Optional. Print version information and exit.

`--debug-level <LEVEL>`

Set the LAL debug level to <LEVEL>. For example: 1 :developer, 33 :production.

`--channel-number <CHANNUMBER>`

Read data from C data channel number <CHANNUMBER>. After running inspiral.c with the –write-cdata option specified, the output frame file will contain a channel for each segment that the raw data is broken up
into as it is filtered by inspiral.c. Here, the user specifies the number on the end of said channel, or in effect the segment number of the filtered data. (e.g. the 5 in L1:LSC-AS.Q.CData.5)

---bank-file <FILE>
Read template bank parameters from file <FILE>. The raw data for each detector should have been filtered with the same bank for each detector using inspiral.c.

---sample-rate <F>
The sample rate that was used to pre-filter the raw data to obtain the C data frame files using inspiral.c.

---segment-length <N>
The data segment length in terms of <N>, the number of timepoints.

---segment-overlap <N>
Overlap data segments by a number <N> of points.

---low-frequency-cutoff <F>
The low frequency cutoff that was used to pre-filter the data using inspiral.c.

---cohsnr-threshold <RHO>
Set the coherent signal-to-noise threshold to <RHO>.

---output-path <PATH>
Optional. Write output data to a given path <PATH>. The default is to write the output to the same directory in which the executable is run.

---write-cohsnr <FILE>
Optional. Write the coherent snr timeseries to a frame file <FILE> (.gwf).

---maximize-over-chirp
Optional. Do event clustering.

---H1-framefile <FILE>
Specify the frame file containing the C data for H1 if it is to be included in the network whose coherent snr is being computed.

---H2-framefile <FILE>
Specify the frame file containing the C data for H2 if it is to be included in the network whose coherent snr is being computed.

---L-framefile <FILE>
Specify the frame file containing the C data for Livingston if it is to be included in the network whose coherent snr is being computed.

---V-framefile <FILE>
Specify the frame file containing the C data for Virgo if it is to be included in the network whose coherent snr is being computed.

---G-framefile <FILE>
Specify the frame file containing the C data for Geo if it is to be included in the network whose coherent snr is being computed.

---T-framefile <FILE>
Specify the frame file containing the C data for Tama if it is to be included in the network whose coherent snr is being computed.

Example
To run the program, type (for example):
3.6. Program `lalapps_coherent_inspiral`

```bash
lalapps_coherent_inspiral \
  --verbose \ 
  --debug-level 33 \ 
  --low-frequency-cutoff 70.0 \ 
  --bank-file L1-TMPLTBANK-751957200-512.xml \ 
  --sample-rate 4096 \ 
  --segment-length 524288 \ 
  --cohsnr-threshold 10.0 \ 
  --H1-framefile H1-INSPIRAL-751957200-512ZDATA.gwf \ 
  --L-framefile L1-INSPIRAL-751957200-512ZDATA.gwf \ 
  --channel-number 0 \ 
  --write-cohsnr H1-L-COHSNR-751957200-512_0.gwf \ 
  --output-path /home/sseader/LIGO/INSPIRAL/S3/playground \
```

**Author**

Sukanta Bose and Shawn Seader
3.7 Program lalapps_inca

Name
lalapps_inca — program does inspiral coincidence analysis.

Synopsis
[--help ]
[--verbose ]
[--version ]
[--debug-level LEVEL ]
[--user-tag USERTAG ]
[--ifo-tag IFOTAG ]
[--comment STRING ]

--gps-start-time GPSSTARTTIME
--gps-end-time GPSENDTIME

[--slide-time SLIDE_SEC ]
[--slide-time-ns SLIDE_NS ]

--ifo-a IFOA
--ifo-b IFOB

[--single-if0 ]
[--triggered-bank TRIGBANKFILE]
[--minimal-match M ]

[--epsilon EPSILON ]
[--kappa KAPPA ]
[--ifo-b-snr-threshold B_SNR]
[--ifo-b-range-cut ]

--parameter-test TEST
[--dm DM ]
[--dpsi0 DPSI0 ]
[--dpsi3 DPSI3 ]
[--alphaf-cut ALPHAFCUT]
[--dmchirp DM_CHIRP ]
[--deta DETA ]
[--dt DT ]

[--no-playground ]
[--playground-only ]
[--write-uniq-triggers ]

(LIGO LIGHTWEIGHT XML files)

Description — General
lalapps_inca runs in three distinct modes. The first is to perform coincidence between triggers from two distinct interferometers. The second is to create a triggered templatebank from a list of inspiral triggers. The third is to create a list of triggers from a single interferometer in a specified time interval. The way to run the code in these three ways is described below.

Description — Coincidence Testing
We begin with the two interferometer coincidence testing. This is the default behaviour of the code. It takes
in triggers from two interferometers and returns those which pass both time and mass coincidence tests. The two interferometers are specified with the `ifo-a` and `ifo-b` arguments. The triggers are read in from a list of LIGO Lightweight XML files given after the last command line argument. This list must contain at least one from each of the two interferometers. The triggers from these files are read in, and only those triggers which lie in the interval between GPSSTARTTIME and GPSENDTIME are kept. The default behaviour is to keep only playground triggers (this can be explicitly requested with the `playground-only` option). By specifying `no-playground`, only non-playground triggers are kept. The triggers from the two interferometers are then tested for time and mass coincidence. Two triggers are considered time coincident if their end times differ by less than $DT$ milliseconds. If a time slide has been specified, then SLIDE_SEC seconds plus SLIDE_NS nanoseconds is added to the recorded time of each trigger in IFOB before testing for time coincidence. Triggers are then tested for mass coincidence using one of three tests ($m_1$ and $m_2$ | $m_{\text{chirp}}$ and $\eta$ | $\psi_0$ and $\psi_3$). If $m_1$ and $m_2$ is specified then both the mass1 and mass2 fields of the triggers must differ by less than the specified DM. If $m_{\text{chirp}}$ and $\eta$ is specified then the chirp masses must differ by less than $DM_{\text{CHIRP}}$ and the mass ratios $\eta$ must differ by less then $DETA$. Finally, if $\psi_0$ AND $\psi_3$ is specified the $\psi_0$ and $\psi_3$ fields of the trigger must differ by less than $PSI0$ and $PSI3$.

If demanding coincidence over $m_1$ and $m_2$, it then tests that

$$\frac{|D_{\text{IFOA}} - D_{\text{IFOB}}|}{D_{\text{IFOA}}} < \frac{\epsilon}{\rho_{\text{IFOB}}} + \kappa. \quad (3.1)$$

This is equivalent to testing that

$$|\rho_{\text{IFOB}} - \hat{\rho}_{\text{IFOB}}| < \epsilon + \kappa \rho_{\text{IFOB}}, \quad (3.2)$$

where

$$\hat{\rho}_{\text{IFOB}} = \frac{\sigma_{\text{IFOB}}}{\sigma_{\text{IFOA}}} \rho_{\text{IFOA}}. \quad (3.3)$$

If demanding coincidence over $\psi_0$ and $\psi_3$, there is an additional cut applied in the triggers. The single-ifo triggers that have values of $\alpha_F$ greater than ALPHAFCUT (as specified in the command line) are rejected. For that reason, the option `--alphaf-cut` is required, if `--parameter-test` is set to $\psi_0$ and $\psi_3$.

If all the tests are passed, the events are considered to be coincident and written to the output file.

The `--ifo-b-range-cut` option performs a test similar to (3.2) above to see whether we should expect a trigger in IFOB. There are three possibilities, which depend upon the value of the SNRSTAR threshold for IFOB, denoted $\rho^*_{\text{IFOB}}$:

1. In this case, the expected signal to noise ratio in IFOB is above our threshold:

$$\rho^*_{\text{IFOB}} < \frac{(\hat{\rho}_{\text{IFOB}} - \epsilon)}{1 + \kappa}, \quad (3.4)$$

so we look for a coincident trigger. We only keep the IFOA trigger if one is found in coincidence in IFOB.

2. In this case, our the allowed range of signal to noise ratio in IFOB is partly above and partly below our threshold:

$$\frac{(\hat{\rho}_{\text{IFOB}} - \epsilon)}{1 + \kappa} < \rho^*_{\text{IFOB}} < \frac{(\hat{\rho}_{\text{IFOB}} + \epsilon)}{1 - \kappa}. \quad (3.5)$$

We search IFOB for triggers and record a coincident trigger if found. Otherwise, we just record the IFOA trigger.

3. In this case, the trigger is not visible to IFOB:

$$\rho^*_{\text{IFOB}} < \frac{(\hat{\rho}_{\text{IFOB}} + \epsilon)}{1 - \kappa}. \quad (3.6)$$

We do not search IFOB, but do keep the trigger from IFOA.
The triggers which survive coincidence are output to two LIGO lightweight XML files. Two XML output files are written. The output files contain process, process_params and search_summary tables that describe the search. The primary ifo output file contains the triggers from IFOA that are found to be in coincidence with triggers in IFOB. The secondary output file contains the triggers from IFOB that are found to be in coincidence with the triggers from IFOA. Each trigger in the IFOA file corresponds to the coincident trigger in the IFOB file, so there may be duplicate IFOA triggers. To prevent this, specify the write-uniq-triggers option.

The output files are named in the standard way for inspiral pipeline output. The primary triggers are in a file named

    IFOA-INCA_IFOTAG_USERTAG-GPSTARTTIME-DURATION.xml

and the secondary triggers are in a file named

    IFOB-INCA_IFOTAG_USERTAG-GPSTARTTIME-DURATION.xml

If a --user-tag or --ifo-tag is not specified on the command line, the _USERTAG or _IFOTAG part of the filename will be omitted.

Description — Triggered Bank

If the triggered-bank option is specified, then lalapps_inca will produce a triggered template bank from the input xml files. In this case, the code expects triggers from only a single interferometer, IFOA. The triggered bank is formed by first sorting the templates in time, and discarding any which are before the GPSTARTTIME or after the time specified GPSENDTIME. The templates are then sorted according to the given parameter-test, which must be one of m1_and_m2 or psi0_and_psi3. Duplicate templates (those with identical m1 and m2 or psi0 and psi3) are discarded and what remains is output to the TRIGBANKFILE specified by the --triggered-bank argument. The output file contain process, process_params, search_summary and sngl_inspiral tables.

Description — Single IFO mode

If the single-ifo option is specified, then lalapps_inca reads in triggers from a single interferometer and returns those within the specified time window. The time window is specified by GPSTARTTIME and GPSENDTIME. By default, the program returns only playground triggers. This behaviour can be explicitly requested with the playground-only flag. If no-playground is specified then only those triggers outside the playground are written to the output file.

The output file is named in the standard way for inspiral pipeline output. The triggers are in a file named

    IFOA-INCA_IFOTAG_USERTAG-GPSTARTTIME-DURATION.xml

If a --user-tag or --ifo-tag is not specified on the command line, the _USERTAG or _IFOTAG part of the filename will be omitted. The triggers are stored in a sngl_inspiral table. The output file also contains process, process_params and search_summary tables that describe the search.

Options

--triggered-bank TRIGBANKFILE
Optional. Run inca in triggered bank mode. Output the triggered bank to a file named TRIGBANKFILE.

--single-ifo
Optional. Run inca in single ifo mode.

--playground-only
Optional. Record only triggers that occur in the playground times. This is the default behavior.
--no-playground
Optional. Record all triggers that are not in playground data. The default behavior returns only those triggers which lie in the playground data set.

--ifo-a IFOA
Required. This is the name of the interferometer to use as the interferometer A in the coincidence algorithm. It must be a two letter IFO code e.g. L1, H1, etc.

--ifo-b IFOB
Required for coincidence, not for trigbank or single ifo. This is the name of the interferometer to use as the interferometer B in the coincidence algorithm. It must be a two letter IFO code e.g. L1, H1, etc.

--epsilon \( \epsilon \)
Optional. Set the value of \( \epsilon \) in the effective distance test. If not given the default of \( \epsilon = 2 \) will be used.

--kappa \( \kappa \)
Optional. Set the value of \( \kappa \) in the effective distance test. If not given the default of \( \kappa = 0.01 \) will be used.

--ifo-b-range-cut
Optional. Use effective distance test to see whether IFOB has a chance of seeing trigger before performing the search.

--ifo-b-snr-threshold SNRSTAR
Optional. Set the value of the signal to noise threshold for IFOB. This is used in determining which triggers IFOB has a chance to see. If not specified, the default value of 6 is used.

--parameter-test TEST
Required. Choose which parameters to use when testing for coincidence (m1_and_m2—psi0_and_psi3—mchirp_and_eta). Depending on which test is chosen, the allowed windows on the appropriate parameters should be set as described below.

--dm \( \delta m \)
Optional. Accept triggers as coincident if both m1 and m2 agree within \( \delta m \). If not supplied, then \( \delta m = 0 \).

--dpsi0 \( \delta \psi_0 \)
Optional. Accept triggers as coincident if \( \psi_0 \) parameters agree within \( \delta \psi_0 \). If not supplied, then \( \delta \psi_0 = 0 \).

--dpsi3 \( \delta \psi_3 \)
Optional. Accept triggers as coincident if \( \psi_3 \) parameters agree within \( \delta \psi_3 \). If not supplied, then \( \delta \psi_3 = 0 \).

--alphaf-cut ALPHAFCUT
Required only if --parameter-test is set to psi0_and_psi3. Accept only the single-if0 BCV triggers that have \( \alpha_F \) less or equal to ALPHAFCUT. Affects only the coincidence part of the code and not the triggered-bank generation.

--dmchirp \( \delta m_{chirp} \)
Optional. Accept triggers as coincident if mchirp agrees within \( \delta m_{chirp} \). If not supplied, then \( \delta m_{chirp} = 0 \).

--deta \( \delta \eta \)
Optional. Accept triggers as coincident if \( \eta \) agrees within \( \delta \eta \). If not supplied, then \( \delta \eta = 0 \).

--dt \( \delta t \)
Optional. Accept triggers as coincident if their end times agree within \( \delta t \) milliseconds. If not supplied, then \( \delta t = 0 \).

--gps-start-time GPS SECONDS
Required. Look for coincident triggers with end times after GPS SECONDS.
--gps-end-time GPS SECONDS
   Required. Look for coincident triggers with end times before GPS SECONDS.

--slide-time
   SLIDE_SEC Optional. Slide the triggers from IFOB forwards in time by SLIDE_SEC seconds before
testing for coincidence. Only used in the coincidence testing mode of inca.

--slide-time-ns
   SLIDE_NS Optional. Slide the triggers from IFOB forwards in time by SLIDE_NS nano seconds before
testing for coincidence. Only used in the coincidence testing mode of inca.

--write-uniq-triggers
   Optional. The default behavior is to only write all triggers from IFO A. However, a trigger from IFO A may
match two or more triggers from IFO B, so it may be duplicated in the output. Specifying this option causes
only unique IFO A triggers to be written.

--minimal-match M
   Optional. If running in triggered bank mode, set the minimal match in the output file to M.

--comment STRING
   Optional. Add STRING to the comment field in the process table. If not specified, no comment is added.

--user-tag USERTAG
   Optional. Set the user tag for this job to be USERTAG. May also be specified on the command line as
-userTag for LIGO database compatibility. This will affect the naming of the output file.

--ifo-tag IFOTAG
   Optional. Set the user tag for this job to be IFOTAG. This will affect the naming of the output file.

--verbose
   Enable the output of informational messages.

--help
   Optional. Print a help message and exit.

--version
   Optional. Print out the author, CVS version and tag information and exit.

--debug-level LEVEL
   Optional. Set the LAL debug level to LEVEL. If not specified the default is 1.

Arguments

[LIGO Lightweight XML files]
   The arguments to the program should be a list of LIGO Lightweight XML files containing the triggers from
the two interferometers. The input files can be in any order and do not need to be time ordered as inca will
sort all the triggers once they are read in. If the program encounters a LIGO Lightweight XML containing
triggers from an unknown interferometer (i.e. not IFO A or IFO B) it will exit with an error.

Example

lalapps_inca \n   --playground-only --dm 0.03 --kappa 1000.0 --ifo-b H1 --ifo-a L1 \n   --user-tag SNR6_INJ --debug-level 33 --gps-start-time 734323079
   --gps-end-time 734324999 --epsilon 2.0 --dt 11.0 \n   L1-INSPIRAL_INJ-734323015-2048.xml H1-INSPIRAL_INJ-734323015-2048.xml
Algorithm

The code maintains two pointers to triggers from each ifo, `currentTrigger[0]` and `currentTrigger[1]`, corresponding to the current trigger from IFO A and B respectively.

1. An empty linked list of triggers from each interferometer is created. Each input file is read in and the code determines which IFO the triggers in the file correspond to. The triggers are appended to the linked list for the corresponding interferometer.
2. If there are no triggers read in from either of the interferometers, the code exits cleanly.
3. The triggers for each interferometer is sorted by the end time of the trigger.
4. `currentTrigger[0]` is set to point to the first trigger from IFO A that is after the specified GPS start time for coincidence. If no trigger is found after the start time, the code exits cleanly.
5. Loop over each trigger from IFO A that occurs before the specified GPS end time for coincidence:
   (a) `currentTrigger[1]` is set to point to the first trigger from IFO B that is within the time coincidence window, $\delta t$, of `currentTrigger[0]`. If no IFO B trigger exists within this window, `currentTrigger[0]` is incremented to the next trigger from IFO A and the loop over IFO A triggers restarts.
   (b) If the trigger `currentTrigger[0]` is, is not in the playground data, start looping over triggers from IFO B.
      i. For each trigger from IFO B that is within $\delta t$ of `currentTrigger[0]`
      ii. Call `LALCompareSnglInspiral()` to check if the triggers match as determined by the options on the command line. If the trigger match, record them for later output as coincident triggers.
   (c) Increment `currentTrigger[0]` and continue loop over triggers from IFO A.

Author

Patrick Brady, Duncan Brown and Steve Fairhurst
3.8 Program lalapps_thinca

Name
lalapps_thinca — program does the inspiral coincidence analysis.

Synopsis
lalapps_thinca
[--help]
[--verbose]
[--version]
[--debug-level <level>]
[--user-tag <usertag>]
[--ifo-tag <ifotag>]
[--comment <string>]
--gps-start-time <start_time>
--gps-end-time <end_time>
[--check-times]
[--multi-ifo-coinc]
[--maximization-interval <max_dt>]
--parameter-test <(m1_and_m2 | psi0_and_psi3 | mchirp_and_eta)>
--data-type <(playground_only | exclude_play | all_data)>
[--g1-triggers]
[--h1-triggers]
[--h2-triggers]
[--l1-triggers]
[--t1-triggers]
[--v1-triggers]
[--g1-slide <g1_slide>]
[--h1-slide <h1_slide>]
[--h2-slide <h2_slide>]
[--l1-slide <l1_slide>]
[--t1-slide <t1_slide>]
[--v1-slide <v1_slide>]
[--num-slides <num_slides>]
[--g1-time-accuracy <g1_dt>]
[--h1-time-accuracy <h1_dt>]
[--h2-time-accuracy <h2_dt>]
[--l1-time-accuracy <l1_dt>]
[--t1-time-accuracy <t1_dt>]
[--v1-time-accuracy <v1_dt>]
[--g1-mass-accuracy <g1_dm>]
[--h1-mass-accuracy <h1_dm>]
[--h2-mass-accuracy <h2_dm>]
[--l1-mass-accuracy <l1_dm>]
[--t1-mass-accuracy <t1_dm>]
[--v1-mass-accuracy <v1_dm>]
[--g1-mchirp-accuracy <g1_dmchirp>]
[--h1-mchirp-accuracy <h1_dmchirp>]
[--h2-mchirp-accuracy <h2_dmchirp>]
[--l1-mchirp-accuracy <l1_dmchirp>]
[--t1-mchirp-accuracy <t1_dmchirp>]
[--v1-mchirp-accuracy <v1_dmchirp>]
[--g1-eta-accuracy <g1_deta>]
[--h1-eta-accuracy <h1_deta>]
[--h2-eta-accuracy <h2_deta>]

Description — General

*lalapps_thinca* performs a coincidence test between triggers from different interferometers. It reads in triggers from up to four instruments and returns coincident triggers. By default, the code will return all double coincident triggers. If the *multi-ifo-coinc* option is specified then the code will also search for triple and quadruple coincidences.

The user specifies which instruments there will be triggers input from with the *gl-triggers*, *h1-triggers* etc. options (if less than two of these are specified, the program exits as there cannot be coincidence). The triggers are then read in from the list of LIGO Lightweight XML files given after the last command line argument. The code performs some pre-processing of the triggers before searching for coincidences. If the *do-alphaF-cut* option is given, then only triggers with an alphaF value between ALPHA_F_LO and ALPHA_F_HI are kept. If the *maximization-interval* is specified, then only the loudest trigger (per IFO) in each MAX_DT ms is retained. The code only keeps triggers which occur between the START_TIME and the END_TIME.

If the *check-times* option is specified, then the input search summary tables are checked to ensure that we have searched all data between the START_TIME and END_TIME in all relevant ifos. Following this, we discard the non-playground triggers if PLAYGROUND_ONLY was specified and any playground triggers if EXCLUDE_PLAY was specified. At this stage, we check that there are triggers from at least two instruments, if not, the code exits without testing for coincidences.

The code now tests for any pairs of coincident triggers from different instruments. This is done in the function *LALCreateTwoIFOCoincList()* . Triggers are considered coincident if their end times and mass parameters pass coincidence. We test either on the two component masses if *m1_and_m2* is specified, on the chirp mass and mass ratio if *mchirp_and_eta* is specified or on the BCV ψ parameters if *psi0_and_psi3* is specified. To pass time coincidence, the end times must differ by less than (IFOA_DT + IFOB_DT + light travel time). Similarly, we require that the mass parameters agree within (IFOA_DM + IFOB_DM). At the end of the process, we have a list of pairs of triggers from different instruments which pass the time and mass coincidence tests.

For coincident triggers from the two Hanford detectors, it is also possible to perform a distance cut by specifying *h1-h2-distance-cut*. In this case, values of H1_KAPPA, H2_KAPPA, H1_EPSILON and H2_EPSILON
must also be specified. The function XLALInspiralDistanceCut() is used to perform the distance cut. It discards any triggers for which

\[
\frac{|distB - distA|}{distA} > \frac{\epsilon_B}{snr_B} + \kappa_B
\]

where ifo A is the ifo with the greater range at the time (as given by the \(\text{sigmasq}\) values of the triggers).

The list of double coincident triggers is then searched for higher ifo coincidences. This is done in the function LALCreateNIFOCoincList(). If, for example, a double coincidence is a subset of a triple coincidence, then that double is removed. All such repetitions are removed by LALRemoveRepeatedCoincs().

The coincident triggers are written into a single LIGO Lightweight XML file. In order that the coincident triggers can be easily located, the \text{EVENT\_ID} field is populated. This is a \text{UINT8} which is populated with \(10^9 \times \text{START\_TIME} + \text{an\ integer\ identifier}\). The integer identifier is unique within the file, so the overall ID will be unique.

The output file is named

\text{IFOS-THINCA-IFOTAG-USERTAG-GPSSTARTTIME-DURATION.xml}

where IFOS is a list of the active ifos in alphabetical order. The file contains \text{process, process\_params, search\_summvars} and \text{search\_summary} tables that describe the search. Additionally there is a \text{summ\_value} table which contains the summ values which were contained in the input files (in anticipation of performing a distance cut) as well as the \text{snql\_inspiral} table containing the coincident events.

If a non-zero argument is given to \text{num-slides}, then \(2 \times \text{NUM\_SLIDES}\) time slides are performed, half with a positive slide and half with negative. The amount by which the triggers of each instrument are slid is given by the arguments to \text{g1-slide} etc. The slide time specified for each of the instruments must be non-negative and unique. The time slides are performed “on a ring”, so that any trigger which is slid past the \text{END\_TIME} is wrapped to the beginning of the analyzed segment. The triggers are then searched for coincidences in an identical manner to the zero-lag case. Again, a unique \text{EVENT\_ID} is associated with each event. This is a \text{UINT8} which is populated with \(\times 10^9 \times \text{START\_TIME} + 10^5 \times 10^5\) a time slide identifier + an integer identifier. Again, the \text{EVENT\_IDs} serve to uniquely specify the coincidences, and also encode which time slide the coincidence was found in.

When performing time slides, the output file is named

\text{IFOS-THINCA-SLIDE-IFOTAG-USERTAG-GPSSTARTTIME-DURATION.xml}

Options

\text{--data-type (playground\_only | exclude\_play | all\_data)}

\text{Required. Specify whether the code should use only the playground, exclude the playground or use all the data.}

\text{--gps-start-time START\_TIME}

\text{Required. Look for coincident triggers with end times after START\_TIME.}

\text{--gps-end-time END\_TIME}

\text{Required. Look for coincident triggers with end times before END\_TIME.}

\text{--g1-triggers}

\text{Optional. Specify that triggers from G1 will be provided.}

\text{--h1-triggers}

\text{Optional. Specify that triggers from H1 will be provided.}
--h2-triggers
Optional. Specify that triggers from H2 will be provided.

--l1-triggers
Optional. Specify that triggers from L1 will be provided.

--t1-triggers
Optional. Specify that triggers from T1 will be provided.

--v1-triggers
Optional. Specify that triggers from V1 will be provided. Note: while having triggers from each of the
instruments is optional, the code requires triggers from at least two instruments, otherwise it is impossible
to do coincidence.

--check-times
Optional. If this flag is set, the code checks the input search summary tables to verify that the data for each
of the requested interferometers was analyzed once and only once between the START_TIME and END_TIME.
By default, the code will not perform this check.

--maximization interval MAX_DT
Optional. Keep only one trigger per ifo in each MAX_DT ms window. This is the trigger with the highest snr.
If this maximization has not been performed in the inspiral code, it is useful to do so at the start of thinca,
otherwise the code may become swamped by the number of coincidences.

--multi-ifo-coinc
Optional. If this flag is set, the code searches for triple and quadruple coincidences. If a triple coincidence
is found, then the double coincidences which form a subset of this triple are removed from the list.

--parameter-test (m1_and_m2 | psi0_and_psi3 | mchirp_and_eta)
Required. Choose which parameters to use when testing for coincidenc. Depending on which test is chosen,
the allowed windows on the appropriate parameters should be set as described below.

--ifo-time-accuracy IFO_DT
Required for any ifo for which we have triggers. Set the accuracy with which the given ifo can recover the
end time of a signal. The timing accuracy is specified in milliseconds. Here, ifo is one of g1, h1, h2, l1, t1,
v1.

--ifo-mass-accuracy IFO_DM
Optional. Set the accuracy with which the given ifo can recover the component masses of a signal. The
mass accuracy is set in solar masses.

--ifo-mchirp-accuracy IFO_DMCHIRP
Optional. Set the accuracy with which the given ifo can recover the chirp mass of a signal. The chirp mass
accuracy is set in solar masses.

--ifo-eta-accuracy IFO_DETA
Optional. Set the accuracy with which the given ifo can recover the mass ratio η of a signal.

--ifo-psi0-accuracy IFO_DPSI0
Optional. Set the accuracy with which the given ifo can recover the parameter ψ₀ of a signal.

--ifo-psi3-accuracy IFO_DPSI3
Optional. Set the accuracy with which the given ifo can recover the parameter ψ₃ of a signal.

--num-slides NUM_SLIDES
Optional. Specify the number of time slides to perform. Note that if time slides are performed, then the zero
lag coincidences are not found. Also, the output file is named THINCA_SLIDE.
--ifo-slide IFO_SLIDE
Optional. Slide the data from this ifo by multiples of IFO_SLIDE seconds when doing time slides. The values chosen for the time slides must be different for all ifos.

--comment STRING
Optional. Add STRING to the comment field in the process table. If not specified, no comment is added.

--user-tag USERTAG
Optional. Set the user tag for this job to be USERTAG. May also be specified on the command line as -userTag for LIGO database compatibility. This will affect the naming of the output file.

--ifo-tag IFOTAG
Optional. Set the ifo tag for this job to be IFOTAG. This will affect the naming of the output file.

--verbose
Enable the output of informational messages.

--help
Optional. Print a help message and exit.

--version
Optional. Print out the author, CVS version and tag information and exit.

--debug-level LEVEL
Optional. Set the LAL debug level to LEVEL. If not specified the default is 33.

Arguments

[LIGO Lightweight XML files]
The arguments to the program should be a list of LIGO Lightweight XML files containing the triggers from the two interferometers. The input files can be in any order and do not need to be time ordered as thinca will sort all the triggers once they are read in. If the program encounters a LIGO Lightweight XML containing triggers from an unknown interferometer (i.e. not IFO A or IFO B) it will exit with an error.

Example
lalapps_thinca \
--data-type playground_only --h1-triggers --h2-triggers --l1-triggers \
--h1-time-accuracy 1 --h2-time-accuracy 1.5 --l1-time accuracy 1 \
--parameter-test mchirp_and_eta --h1-mchirp-accuracy 0.02 \
--h2-mchirp-accuracy 0.03 --l1-mchirp-accuracy 0.04 \
--h1-eta-accuracy 1 --h2-eta-accuracy 1 --l1-eta-accuracy 1 \
--gps-start-time 777001000 --gps-end-time 777002000 \
H1-INSPIRAL-777000500-2048.xml H2-INSPIRAL-777000700-2048.xml \nL1-INSPIRAL-777000500-2048.xml

Algorithm
Not yet documented.

Author
Steve Fairhurst
3.9 Program lalapps_trigbank

Name
lalapps_trigbank — program to make triggered template banks.

Synopsis
lalapps_trigbank
[--help]
[--verbose]
[--version]
[--debug-level <level>]
[--user-tag <usertag>]
[--ifo-tag <ifotag>]
[--comment]
  --gps-start-time <start_time>
  --gps-end-time <end_time>
  [--check-times]
  --input-ifo <inputifo>
  --output-ifo <outputifo>
  --parameter-test <(m1_and_m2—psi0_and_psi3—mchirp_and_eta)>
  --data-type <(playground_only—exclude_play—all_data)>
LIGOLW XML input files

(LIGO LIGHTWEIGHT XML files)

Description — General
lalapps_trigbank produces a triggered template bank from the input xml files. The code takes in a list of triggers from the specified input files. First, it discards any triggers which do not come from the specified INPUTIFO. Then, it keeps only those triggers which occur between the START_TIME and the END_TIME. If the check-times option is specified, then the input search summary tables are checked to ensure that we have searched all data between the START_TIME and END_TIME in all relevant ifos. Following this, we discard the non-playground triggers if PLAYGROUND_ONLY was specified and any playground triggers if EXCLUDE_PLAY was specified.

The remaining triggers are sorted according to the given parameter-test, which must be one of m1_and_m2 or psi0_and_psi3. Duplicate templates (those with identical m1 and m2 or psi0 and psi3) are discarded and what remains is output as the triggered template bank.

The output file contains process, process_params, search_summary and sngl_inspiral tables. The output file name is

    OUTPUTIFO-TRIGBANK_IFOTAG_USERTAG-GPSSTARTTIME-DURATION.xml

where the input and output instruments are specified on the command line.

Options

  --data-type (playground_only—exclude_play—all_data)
    Required. Specify whether the code should use only the playground, exclude the playground or use all the data.

  --gps-start-time START_TIME
    Required. Look for coincident triggers with end times after START_TIME.

  --gps-end-time END_TIME
    Required. Look for coincident triggers with end times before END_TIME.
3.9. Program `lalapps_trigbank`

```
--check-times
Optional. If this flag is set, the code checks the input search summary tables to verify that the data for
the input interferometers was analyzed once and only once between the START_TIME and END_TIME. By
default, the code will not perform this check.

--parameter-test (m1_and_m2—psi0_and_psi3)
Required. Choose which parameters to use when testing for repeated triggers. If two triggers have the same
value of mass1 and mass2 or psi0 and psi3 then only one copy is put into the triggered bank.

--input-ifo INPUTIFO
Required. The triggers from INPUTIFO are used to create the triggered bank.

--output-ifo OUTPUTIFO
Required. This gives the instrument for which we are creating the triggered bank. It is only used in naming
the output file.

--ifo-tag IFOTAG
Optional. Used in naming the output file.

--comment STRING
Optional. Add STRING to the comment field in the process table. If not specified, no comment is added.

--user-tag USERTAG
Optional. Set the user tag for this job to be USERTAG. May also be specified on the command line as
-userTag for LIGO database compatibility. This will affect the naming of the output file.

--verbose
Enable the output of informational messages.

--help
Optional. Print a help message and exit.

--version
Optional. Print out the author, CVS version and tag information and exit.

--debug-level LEVEL
Optional. Set the LAL debug level to LEVEL. If not specified the default is 1.
```

Arguments

```
[LIGO Lightweight XML files]
The arguments to the program should be a list of LIGO Lightweight XML files containing the triggers from
the two interferometers. The input files can be in any order and do not need to be time ordered as trigbank
will sort all the triggers once they are read in.

Example
```
lalapps_trigbank \
--data-type playground_only --input-ifo H1 --output-ifo H1 --ifo-tag H1H2 \
--debug-level 33 --gps-start-time 734323079 --gps-end-time 734324999 \
H1-INSPIRAL-734323015-2048.xml
```

Algorithm

None.

Author

Patrick Brady, Duncan Brown and Steve Fairhurst
3.10 Program **lalapps_sire**

**Name**

*lalapps_sire* — single inspiral trigger reader and inspiral injections analysis

**Synopsis**

```
lalapps_sire
  --all-data
  [--cluster-algorithm <choice>]
  [--cluster-time <t>]
  [--comment <string>]
  [--debug-level <level>]
  [--disable-trig-start-time]
  --exclude-playground
  --glob <globfiles>
  [--hardware-injections <thardware>]
  [--help]
  [--injection-file <injfile>]
  [--injection-coincidence <t_inj>]
  --input <inputfiles>
  [--missed-injections <missedfile>]
  --output <outfile>
  --playground-only
  [--snr-treshold <rho>]
  [--summary-file <file>]
  [--tama-output <file>]
  [--user-tag <string>]
  [--verbose]
  --version
```

**Description**

*lalapps_sire* processes the LIGO lightweight XML files produced by the standalone inspiral analysis code *lalapps_inspiral* or the inspiral coincidence analysis code *lalapps_inca*. It can be used to concatenate individual *sngl_inspiral* tables from multiple XML files which contain a *search_summary* table into a single XML file. This may be performed with or without clustering and signal-to-noise ratio cuts. It can also write a summary file containing the number of triggers and the total time analyzed, computed from the *search_summary* table.

The list of input files may be specified by either of POSIX system glob, `<globfiles>`, or by giving the path to a text file, `<inputfile>`, that contains relative or absolute paths to the required input files.

If the `--injection-file` option is specified, *lalapps_sire* also reads in a list of *sim_inspiral* rows from the file `<injfile>`. It determines how many of the injections have a trigger coincident to within `<t_inj>` milliseconds. The output file, `<outfile>`, will contain *sim_inspiral* rows for all coincident (found) events. The summary file will contain numbers of missed and found events and the efficiency of detection of the injections. Only injections that are within the input data times are processed, so the injection file can span a time larger than the input data and efficiencies will be correct. Missed injections can be written to the file `<missedfile>`, if desired.

Note that the default LAL debug level for *lalapps_sire* is 33, rather than the usual 1. This turns of memory allocation and leak checking to increase the speed of run time. The `--debug-level` option can be used to increase the level of debug checking used by the LAL functions.

**Options**
--all-data
Either this option or one of the options --exclude-playground or --playground-only must be specified. Using this option all triggers (and injections) from the input files are analyzed.

--cluster-algorithm <choice>
Use the clustering algorithm CHOICE to cluster the sngl_inspiral rows in the output file before writing them to disk. The options for CHOICE are snr_and_chisq, snrsq_over_chisq or snr. The clustering is performed by the LAL function LALClusterSnglInspiralTable() and documentation for the clustering can be found in the tools package of the LAL Software Documentation.

--cluster-time <t>
Required if the --cluster-algorithm option is specified. Use the time window <t> for the clustering algorithm.

--comment <string>
Add the string <comment> to the process table in the output XML file.

--debug-level <level>
Sets the LAL debug level to <level>. The default value is LALMSG_LVL2. A useful setting is 65 which turns off memory padding, but keeps memory tracking and error messages. If you want to turn off memory tracking completely, then use 33.

--disable-trig-start-time
This option should only be used by maintainers. Disable checking of the --trig-start-time option in the input files. Using this option may caused total analyzed times to be reported incorrectly. See note in algorithm section below.

--exclude-playground
Either this option or one of the option --all-data or --playground-only must be specified. Using this option only triggers (and injections) that are not in playground times specified by the post-S1 playground algorithm are analyzed.

--glob <globfiles>
Must be given if the --input option is not used. Read the input triggers from the LIGO lightweight XML files that match the regular expression <globfiles>. The POSIX system call glob() is used to determine which files are read in. Mutually exclusive with the --input option.

--help
Display a usage message and exit.

--injection-file <infilename>
Use <file name> as a LIGO lightweight XML file containing a list of injections to be made. The file should contain a sim_burst table which is used to set information about the types of injections to be made. This file may be constructed by hand, or one can use the lalapps_bijn program described in Section ??.

--input <inputfile>
Must be given if the --glob option is not used. Read the input triggers from the list of LIGO lightweight XML files in <inputfile> which must be a plain text file containing relative or absolute paths to the files. Mutually exclusive with the --glob option.

--hardware-injections <t_hardware>
This option can only be specified if --injection-file has been specified. Increment the end times of the injections read from <infilename> by t_hardware. Seconds. Used for injection analysis of hardware injections where the input sim_inspiral rows contain the time offset of the injection from t_hardware.

--injection-file <infilename>
If this option is given, lalapps_sire reads in sim_inspiral rows from the file <infilename> and performs an injection analysis of the triggers.
3.10. Program `lalapps_sire`

--injection-coincidence \(<t_{\text{inj}}>\)
This option is required if the `--injection-file` option is specified. Set the injection coincidence window to \(\pm t_{\text{inj}}\) milliseconds.

--missed-injections <file>
This option can only be specified if `--injection-file` has been specified. If any injections are not found, write the `sim_inspiral` rows for these missed injections to the LIGO lightweight file <file>.

--output <outfile>
Write the concatenated `sngl_inspiral` tables to the LIGO lightweight XML file <outfile>. If injection analysis is performed the `sim_inspiral` rows from the input injection file that are coincident with a trigger are also written to this file (i.e. the found injections).

--playground-only
Either this option or one of the option `--exclude-playground` or `--all-data` must be specified. Using this option only triggers (and injections) that are in playground times specified by the post-S1 playground algorithm are analyzed.

--snr-threshold <\(\rho_\ast\)>
Discard all input triggers that have a signal-to-noise ratio \(\rho < \rho_\ast\).

--summary-file <file>
With this option a summary file <file> is created containing the number of triggers and the total time analyzed, computed from the `search_summary` table.

--tama-output <file>
If specified produces an output text file <file> for use in collaboration with TAMA, in addition to the usual LIGO lightweight XML file. The following quantities are recorded for each trigger in the text file:

- trigger time (as a double precision real)
- total mass, \(M_{\text{TOT}}\)
- the mass ratio, \(\eta\)
- the signal to noise ratio, \(\rho\)
- the value of \(\chi^2\)
- the effective distance to the trigger, \(d_{\text{eff}}\).

--user-tag <comment>
Set the user tag to the string <comment>. This string must not contain spaces or dashes ("-"). This string will appear in the name of the file to which output information is written, and is recorded in the various XML tables within the file.

--verbose
Enable the output of informational messages.

--version
Print the CVS id and exit.

Example 1
Read in all playground triggers files from the current directory that match the expression

`L1-INSPIRAL_INJ-7*`

Discard all triggers below signal-to-noise ratio 10 and report the number of injections from file

`HL-INJECTIONS_45-729273613-5094000.xml`
that are coincident to within 20 milliseconds with the remaining triggers. Write an XML file containing the coincident triggers and injections, an XML file containing the injections not coincident with a trigger and a text summary file of the analysis, which will contain the total time analyzed and the efficiency. Report the progress to the standard output and perform LAL memory checking:

```bash
lalapps_sire \
--glob "L1-INSPIRAL_INJ-7*"\n--output out_10.xml\n--summary-file summ_10.txt \n--playground-only\n--verbose\n--debug-level 1\n--snr-threshold 10.0 \n--injection-file HL-INJECTIONS_45-729273613-5094000.xml \n--injection-coincidence 20\n--missed-injections missed_10.xml
```

Example 2

Read in all the XML files from the list in the plain text file `H1-INSPIRAL.txt` and discard all the triggers that are in the playground. Write the remaining triggers to the XML file `H1-INSPIRAL.xml` and write a text summary file containing the time analyzed to `H1-INSPIRAL_summary.txt`:

```bash
lalapps_sire \
--input H1-INSPIRAL.txt\n--exclude-playground \n--output H1-INSPIRAL.xml\n--summary-file H1-INSPIRAL_summary.txt
```

Notes

1. The post-S1 playground algorithm is defined to be

\[ t \text{ is playground } \Leftrightarrow t - 729273613 < 600(\text{mod } 6370). \]  

(3.8)

2. If a given trigger `end_time,end_time_ns`, `t_{trig}` is coincident to within `\pm t_{inj}` seconds of an injection site end time, given by `h_{end_time},h_{end_time_ns}` or `l_{end_time},l_{end_time_ns}` then the injection is considered to be found and the trigger coincident with an injection.

3. Early versions of the inspiral code contained a bug that causes the `out_start_time` column of the `search_summary` table to be set incorrectly if a non-zero `--trig-start-time` option is specified. `lalapps_sire` corrects for this by checking for the value of `--trig-start-time` in the `process_params` table and using it to override the value of `out_start_time` in the `search_summary` table. To disable this behaviour, use the `--disable-trig-start-time` option. Note that specifying this option may cause some analyzed data times to be double counted and so the amount of analyzed data will be incorrectly reported.

Author

Patrick Brady, Duncan Brown, Alexander Dietz and Steve Fairhurst
3.11 Program lalapps_inspinj

Name
lalapps_inspinj — produces inspiral injection data files.

Synopsis
lalapps_inspinj
[--help]
--source-file SFILE
--mass-file MFILE
[--gps-start-time TSTART]
[--gps-end-time TEND]
[--time-step TSTEP]
[--time-interval TINTERVAL]
[--seed SEED]
[--waveform WAVE]
[--lal-eff-dist]
[--usertag TAG]
[--tama-output]
[--write-eff-dist]
[--ilwd]

Description
lalapps_inspinj generates a number of inspiral parameters suitable for using in a Monte Carlo injection to test the efficiency of a inspiral search. The various parameters (detailed below) are randomly chosen and are appropriate for a particular population of binary neutron stars whose spatial distribution includes the Milky Way and a number of extragalactic objects that are input in a datafile. The possible mass pairs for the binary neutron star companions are also specified in a (different) datafile.

The output of this program is a list of the injected events, starting at the specified start time and ending at the specified end time. One injection with random inspiral parameters will be made every specified time step, and will be randomly placed within the specified time interval. The output is written to a file name in the standard inspiral pipeline format:

HL-INJECTIONS_USERTAG_SEED-GPSSTART-DURATION.xml

where USERTAG is TAG as specified on the command line, SEED is the value of the random number seed chosen and GPSSTART and DURATION describes the GPS time interval that the file covers. The file is in the standard LIGO lightweight XML format containing a sim_inspiral table that describes the injections. In addition, an ASCII log file called injlog.txt is also written. If a --user-tag is not specified on the command line, the USERTAG part of the filename will be omitted.

Options

--help
Print a help message.

--source-file SFILE
Optional. Data file containing spatial distribution of extragalactic objects. Default is the file inspsrcs.dat provided by LALApps. If that file is empty, all signals are in the Milky Way.

--mass-file MFILE
Optional. Data file containing mass pairs for the binary neutron star companions. Default is the file BNSMasses.dat provided by LALApps.
3.11. Program `lalapps_inspinj`

**--gps-start-time** `TSTART`
Optional. Start time of the injection data to be created. Defaults to the start of S2, Feb 14 2003 16:00:00 UTC (GPS time 729273613).

**--gps-end-time** `TEND`
Optional. End time of the injection data to be created. Defaults to the end of S2, Apr 14 2003 15:00:00 UTC (GPS time 734367613).

**--time-step** `TSTEP`
Optional. Sets the time step interval between injections. The injections will occur with an average spacing of `TSTEP` seconds. Defaults to $\frac{2630}{\pi}$.

**--time-interval** `TINTERVAL`
Optional. Sets the time interval during which an injection can occur. Injections are uniformly distributed over the interval. Setting `TSTEP` to 6370 and `TINTERVAL` to 600 guarantees there will be one injection into each playground segment and they will be randomly distributed within the playground times - taken the fact that your gps start time coincides with start of a playground segment.

**--seed** `SEED`
Optional. Seed the random number generator with the integer `SEED`. Defaults to 1.

**--waveform** `WAVE`
Optional. The string `WAVE` will be written into the `waveform` column of the `sim_inspiral` table output. This is used by the inspiral code to determine which type of waveforms it should inject into the data. Defaults is `GeneratePPNtwoPN`.

**--lal-eff-dist**
Optional. If this option is specified, the effective distance will be calculated using routines from LAL. Otherwise, the default behaviour is to use an independent method contained in `inspinj.c`. There is good agreement between these two methods, see below for more details.

**--user-tag** `STRING`
Optional. Set the user tag for this job to be `STRING`. May also be specified on the command line as `--userTag` for LIGO database compatibility.

**--tama-output**
Optional. If this option is given, `lalapps_inspinj` also produces a text output file:

```
HLT-INJECTIONS_USERTAG_SEED-GPSSTART-DURATION.txt
```

which contains the following fields:

- geocentric end time
- Hanford end time
- Livingston end time
- TAMA end time
- total mass, $M_{TOT}$
- mass ratio, $\eta$
- distance to source (in kpc)
- longitude
- latitude
- inclination
- coalescence phase
- polarization
- TAMA polarization
• end time GMST

In the above, all times are recorded as double precision real numbers and all angles are in radians. The TAMA polarization is calculated using

\[ \tan(\psi_T) = \frac{x \cdot T_z}{y \cdot T_z} \]  

(3.9)

Here \( x \) and \( y \) are the x,y axes of the radiation frame expressed in earth fixed coordinates (3.13, 3.14). \( T_z \) is a unit vector in earth fixed coordinates which is orthogonal to the two arms of the TAMA detector (3.23). It is given by

\[ T_z = (-0.6180, +0.5272, +0.5832) \]  

(3.10)

**--write-eff-dist**

Optional. If this option is given, three extra columns are added to the TAMA output file described above. They are

- Hanford effective distance (kpc)
- Livingston effective distance (kpc)
- TAMA effective distance (kpc)

These entries are added to the list immediately after TAMA end time and before total mass.

**--ilwd**

Optional. If this option is given, `lalapps_inspinj` also produces two ILWD-format files, injepochs.ilwd and injparams.ilwd, that contain, respectively, the GPS times suitable for inspiral injections, and the intrinsic inspiral signal parameters to be used for those injections.

The file injepochs.ilwd contains a sequence of integer pairs representing the injection GPS time in seconds and residual nano-seconds. The file injparams.ilwd contains the intrinsic binary parameters for each injection, which is a sequence of eight real numbers representing (in order) (1) the total mass of the binary system (in solar masses), (2) the dimensionless reduced mass — reduced mass per unit total mass — in the range from 0 (extreme mass ratio) to 0.25 (equal masses), (3) the distance to the system in meters, (4) the inclination of the binary system orbit to the plane of the sky in radians, (5) the coalescence phase in radians, (6) the longitude to the direction of the source in radians, (7) the latitude to the direction of the source in radians, (8) and the polarization angle of the source in radians.

**Example**

```
lalapps_inspinj --seed 45\n--source-file inspsrcs.dat --mass-file BNSMasses.dat
```

**Algorithm**

The algorithm for computing the effective distance will be described in some detail below. The method is to compute both the strain due to the inspiral and the detector response in the earth fixed frame. This frame is such that the z-axis points from the earth’s centre to the North Pole, the x-axis points from the centre to the intersection of the equator and the prime meridian and the y-axis is chosen to complete the orthonormal basis. The coordinates of the injection are specified by longitude (or right ascension) \( \alpha \) and latitude (or declination) \( \delta \). The polarization is appropriate for transferring from the radiation to earth fixed frame. These are then converted to the earth fixed frame by

\[ \theta = \frac{\pi}{2} - \delta \]  

(3.11)

\[ \phi = \alpha - \text{gmst} \]  

(3.12)
Here, gmst is the Greenwich Mean sidereal time of the injection. The axes of the radiation frame (x,y,z) can be expressed in terms of the earth fixed coordinates as:

\[
\begin{align*}
    x(1) &= +\left(\sin(\phi) \cos(\psi) - \sin(\psi) \cos(\phi) \cos(\theta)\right) \\
    x(2) &= -\left(\cos(\phi) \cos(\psi) + \sin(\psi) \sin(\phi) \cos(\theta)\right) \\
    x(3) &= \sin(\psi) \sin(\theta) \tag{3.13}
\end{align*}
\]

\[
\begin{align*}
    y(1) &= -\left(\sin(\phi) \sin(\psi) + \cos(\psi) \cos(\phi) \cos(\theta)\right) \\
    y(2) &= +\left(\cos(\phi) \sin(\psi) - \cos(\psi) \sin(\phi) \cos(\theta)\right) \\
    y(3) &= \cos(\psi) \sin(\theta) \tag{3.14}
\end{align*}
\]

Making use of these expressions, we can express the gravitational wave strain in earth fixed coordinates as

\[
h_{ij} = \left( h^+ (t) e^+_{ij} + h^\times (t) e^\times_{ij} \right) \tag{3.15}
\]

where

\[
e^+_{ij} = x_i \ast x_j - y_i \ast y_j \quad \text{and} \quad e^\times_{ij} = x_i \ast y_j + y_i \ast x_j. \tag{3.16}
\]

For the case of a binary inspiral signal, the two polarizations \( h^+ \) and \( h^\times \) of the gravitational wave are given by

\[
\begin{align*}
    h^+ (t) &= \frac{A}{r} \left( 1 + \cos^2(\iota) \right) \ast \cos(\Phi(t)) \tag{3.17} \\
    h^\times (t) &= \frac{A}{r} \ast \left( 2 \cos(\iota) \right) \ast \sin(\Phi(t)) \tag{3.18}
\end{align*}
\]

where \( A \) is a mass and frequency dependent amplitude factor, \( r \) is the physical distance at which the injection is located and \( \iota \) is the inclination angle.

Next, we can write the detector response function as

\[
d^{ij} = \left( \frac{1}{2} \right) \left( n^i_x n^j_x - n^i_y n^j_y \right). \tag{3.19}
\]

Here, \( n_x \) and \( n_y \) are unit vectors directed along the arms of the detector. Specifically, for the Hanford, Livingston, GEO, TAMA and Virgo detectors we use:

\[
\begin{align*}
    H_x &= (-0.2239, +0.7998, +0.5569) \tag{3.20} \\
    H_y &= (-0.9140, +0.0261, -0.4049) \\
    L_x &= (-0.9546, -0.1416, -0.2622) \\
    L_y &= (+0.2977, -0.4879, -0.8205) \tag{3.21} \\
    G_x &= (-0.6261, -0.5522, +0.5506) \\
    G_y &= (-0.4453, +0.8665, +0.2255) \tag{3.22} \\
    T_x &= (+0.6490, +0.7608, +0.0000) \\
    T_y &= (-0.4437, +0.3785, -0.8123) \tag{3.23} \\
    V_x &= (-0.7005, +0.2085, +0.6826) \\
    V_y &= (-0.0538, -0.9691, +0.2408) \tag{3.24}
\end{align*}
\]

The response of an interferometric detector with arm locations given by \( n_x \) and \( n_y \) to an inspiralling binary system described by (3.15) is

\[
h(t) = h^+ (t) (d^{ij} e^+_{ij}) + h^\times (t) (d^{ij} e^\times_{ij}) \]

\[
= \left( \frac{A}{r} \right) \left[ \left( 1 + \cos^2(\iota) \right) F_+ \cos(\Phi(t)) + 2 \cos(\iota) F_\times \sin(\Phi(t)) \right], \tag{3.25}
\]

where we have introduced

\[ F_+ = d^i_j e^+_{ij} \quad \text{and} \quad F_\times = d^i_j e^\times_{ij} \]  

(3.26)

Finally, to calculate the effective distance, we note that the two contributions to \( h(t) \) are \( \pi/2 \) radians out of phase, and hence orthogonal. Thus, we can compute the effective distance to be:

\[ D_{\text{eff}} = r \left( \frac{(1 + \cos^2(\iota))^2}{4} F_+^2 + \cos^2(\iota) F_\times^2 \right) \]  

(3.27)

The algorithm to calculate effective distances described above is completely contained within inspinj.c. There is an independent method of computing effective distances can also be called by inspinj. It is contained in the LAL function `LALPopulateSimInspiralSiteInfo()`. This function populates the site end time and effective distance for all the interferometer sites. It makes use of LAL functionality in the tools and date packages. These same functions are used when generating the injection waveform which is added to the data stream (in `lalapps_inspiral`). As a check that these two calculations produce the same effective distance, `lalapps_inspinj` was run twice, once with the `--lal-eff-dist` option and once without. Figure ?? shows the fractional difference in effective distance between the two methods for a set of injections. We see that the distances agree within 1% for all injections, with the largest differences occurring for the largest effective distances, i.e. close to the dead spot of the instrument. For injections which initial LIGO is sensitive to, the accuracy is few \( \times 10^{-4} \).

Environment

**LALAPPS_DATA_PATH**

Directory to look for the default mass file `BNSMasses.dat` and the default source file `inspsrcs.dat`.

Author

Jolien Creighton, Patrick Brady, Duncan Brown
Figure 3.1: Comparison of effective distance computed by inspinj.c and LAL routines
3.12 Program lalapps_bbhinj

Name
lalapps_bbhinj — produces inspiral injection data files.

Synopsis
lalapps_bbhinj
[--help]
[--verbose]
[--gps-start-time TSTART]
[--gps-end-time TEND]
[--time-step TSTEP]
[--time-interval TINTERVAL]
[--seed SEED]
[--usertag TAG]
[--min-mass MMIN]
[--max-mass MMAX]
[--max-total-mass TOTALMASS]
[--min-distance DMIN]
[--max-distance DMAX]
[--m-distr MDISTR]
[--d-distr DDISTR]
[--waveform WVF]

Description
lalapps_bbhinj generates a number of inspiral parameters suitable for using in a Monte Carlo injection to
test the efficiency of an inspiral search. The various parameters (detailed below) are randomly chosen and are
appropriate for a population of binaries that extends over all space between the minimum and maximum distances
specified. Despite its name, it can be used for BNS and for BBH parameter generation.

The output of this program is a list of the injected events, starting at the specified start time and ending at the
specified end time. One injection with random inspiral parameters will be made every specified time step, and
will be randomly placed within the specified time interval. The output is written to a file name in the standard
inspiral pipeline format:

HL-INJECTIONS_USERTAG_SEED-GPSSTART-DURATION.xml

where USERTAG is TAG as specified on the command line, SEED is the value of the random number seed chosen
and GPSSTART and DURATION describes the GPS time interval that the file covers. The file is in the standard
LIGO lightweight XML format containing a sim_inspiral table that describes the injections. In addition, an
ASCII log file called injlog.txt is also written. If a --user-tag is not specified on the command line, the
USERTAG part of the filename will be omitted.

The GEO, TAMA and VIRGO end times and effective distances are currently not being populated.

Options

--help
Print a help message.

--gps-start-time TSTART
Optional. Start time of the injection data to be created. Defaults to the start of S2, Feb 14 2003 16:00:00
UTC (GPS time 729273613)

--gps-end-time TEND
Optional. End time of the injection data to be created. Defaults to the end of S2, Apr 14 2003 15:00:00 UTC
(GPS time 734367613).
--\textbf{time-step} TSTEP  
Optional. Sets the time step interval between injections. The injections will occur with an average spacing of TSTEP seconds. Defaults to $2630/\pi$.

--\textbf{time-interval} TINTERVAL  
Optional. Sets the time interval during which an injection can occur. Injections are uniformly distributed over the interval. Setting TSTEP to 6370 and TINTERVAL to 600 guarantees there will be one injection into each playground segment and they will be randomly distributed within the playground times.

--\textbf{seed} SEED  
Optional. Seed the random number generator with the integer SEED. Defaults to 1.

--\textbf{user-tag} STRING  
Optional. Set the user tag for this job to be STRING. May also be specified on the command line as -userTag for LIGO database compatibility.

--\textbf{min-mass} MMIN  
Optional. The minimum component mass of the binary. If not specified, it is automatically set to 3 $M_\odot$.

--\textbf{max-mass} MMAX  
Optional. The maximum component mass of the binary. If not specified, it is automatically set to 20 $M_\odot$.

--\textbf{max-total-mass} TOTALMASS  
Optional. The maximum of the total masses of the two components. If not specified, no restriction is set to the total mass of the two components.

--\textbf{min-distance} DMIN  
Optional. The minimum distance (in kpc) from the earth of the signals. If not specified, it is automatically set to 1 kpc.

--\textbf{max-distance} DMAX  
Optional. The maximum distance (in kpc) from the earth of the signals. If not specified, it is automatically set to 20 Mpc (20000 kpc).

--\textbf{m-distr} MDISTR  
Optional. It lets the user specify the mass distribution of the population. The choices are:

- $MDISTR = 0$ the distribution is uniform over total mass (DEFAULT VALUE)
- $MDISTR = 1$ the distribution is uniform over mass1 and over mass2

--\textbf{d-distr} DDISTR  
Optional. It lets the used specify the distance distribution of the population. The choices are:

- $DDISTR = 0$ uniform-distance distribution (DEFAULT VALUE)
- $DDISTR = 1$ uniform log(distance) distribution
- $DDISTR = 2$ uniform volume distribution

--\textbf{waveform} WVF  
Optional. Sets the waveforms to be injected to WVF. The WVF consists of two parts, which are specified as one word. The first part is one of:

- EOB
- PadeT1
- TaylorT1
- TaylorT3
- GeneratePPN

The second part is one of:
3.12. Program \texttt{lalapps\_bbhinj}

- newtonian
- onePN
- onePointFivePN
- twoPN (ONLY CHOICE if GeneratePPN is used before!)
- twoPointFivePN
- threePN

If nothing is specified, the default value is EOBtwoPN. It should be noted that if GeneratePPNtwoPN is used as the waveform, the code used to perform the injections is different than otherwise. For GeneratePPNtwoPN, the older injection code (that does only standard post-Newtonian injections) is used. That is the recommended approximant for the case of binary neutron star injections.

Example:
The following command will produce an injection file for a population of binary black holes of total mass between 6 and 20 $M_\odot$ (component mass between 3 and 10 $M_\odot$), with uniform-distance distribution between 100 kpc and 3 Mpc. Since no mass-distribution is specified, the mass-distribution will be uniform over total mass (default value).

\begin{verbatim}
lalapps_bbhinj --min-mass 3.0 --max-mass 10.0 --min-distance 100
--max-distance 3000 --d-distr 0 --waveform PadeT1twoPN
\end{verbatim}

Author
Jolien Creighton, Patrick Brady, Duncan Brown and Eirini Messaritaki
3.13 Program lalapps_inspfrinj

Name
lalapps_inspfrinj — performs inspiral injections into frame data.

Synopsis
lalapps_inspfrinj [options]

[--help ]
[--verbose ]
[--version ]
[--debug-level LEVEL ]
[--user-tag STRING ]
[--comment STRING ]

--gps-start-time SEC
[--gps-start-time-ns NS ]
--gps-end-time SEC
[--gps-end-time-ns NS ]

[--frame-cache FRAME_CACHE
--channel-name CHAN ]

[--ifo IFO
--sample-rate SAMPLE_RATE ]

--calibration-cache CAL_CACHE
--calibrated-data TYPE

[--num-resp-points N ]

--injection-file FILE
--injection-channel INJ
[--inject-overhead ]
[--inject-safety SEC ]

[--write-raw-data ]
[--write-inj-only ]
[--write-raw-plus-inj ]

--output-frame-length OUTPUT_LENGTH
[--output-file-name OUTPUT_NAME ]

Description
lalapps_frinspinj generates injections and stores them as frame files. This code is essentially a copy of those parts of lalapps_inspiral responsible for reading in the data and performing the injections. The injection details are either read in from frame files containin an injection channel or from the sim_inspiral table of a LIGO lightweight xml file or Is a sim_inspiral table is given, the injections are generated using LALFindChirpInjectSignals. The injection data is output to frame files. The length of the output frames is specified by output-frame-length.

The code is capable of generating both calibrated and uncalibrated injections. To obtain calibrated h(t) injections the option calibrated-data must be specified (at present the code only works for real4 data). Alternatively, if a CAL_CACHE is given, then uncalibrated injections will be produced using the a response function generated from the calibration data in the cache. The calibration coefficients will by default be averaged over the duration...
of the data. When working with uncalibrated data, the number of points used to determine the response function will have an effect, albeit minor, on the injection signals. By default, the value `num-resp-points` is set to 4194304. This matches the value used by `lalapps_inspiral` when working with 256 second segments and a channel sampled at 16384 Hz.

If input data is given by a FRAME_CACHE, the code will read in this data. In this case, the sample rate and ifo will be determined from the data and channel name respectively. The output frames can contain any of the following: raw data, injection only and raw plus injection.

If no input data is provided, then the IFO and SAMPLE_RATE must be given on the command line. Since no input data is read in, the output data can only contain the injection only channel.

The output of `lalapps_frinspinj` is an xml file and one or several frame files. The xml file contains `process`, `process_params`, `search_summary` and `sim_inspiral` tables and is named (unless set on the command line):

```
IFO-INSPPFRINJ-USERTAG-GPSSTARTTIME-DURATION.xml
```

where `GPSSTARTTIME` and `DURATION` are the start time and duration passed to the code. If `--output-file-name` is set to `OUTPUT` on the command line, the xml will be named

```
OUTPUT-USERTAG-GPSSTARTTIME-DURATION.xml
```

The `sim_inspiral` table contains a list of all injections which were performed into the specified data. The length of output frame files is specified by `output-frame-length`. Unless otherwise set on the command line, output frames are named:

```
IFO-INSPPFRINJ-USERTAG-FRAMESTARTTIME-FRAMELENGTH.gwf
```

The channels stored in the output frame are determined by which of `--write-raw-data`, `--write-inj-only` and `--write-raw-plus-inj` are selected. The channel names are `CHAN`, `CHAN_INSP_INJ_ONLY` and `CHAN_RAW_PLUS_INSP_INJ`. In the case where no input data is given, the single output channel name is `IFO:STRAIN_INSP_INJ_ONLY`.

Options

```
--help
Optional. Print a help message and exit.

--verbose
Optional. Print out the author, CVS version and tag information and exit.

--version
Optional. Print out the author, CVS version and tag information and exit.

--debug-level LEVEL
Optional. Set the LAL debug level to LEVEL. If not specified the default is 1.

--user-tag USERTAG
Optional. Set the user tag for this job to be USERTAG. May also be specified on the command line as `-userTag` for LIGO database compatibility. This will affect the naming of the output file.

--comment STRING
Optional. Add STRING to the comment field in the process table. If not specified, no comment is added.

--gps-start-time GPS SECONDS
Required. Start time, GPS seconds.
```
Program lalapps_inspfrinj

--gps-start-time-ns GPS SECONDS
Optional. Start time, GPS nanoseconds. If not specified, then set to zero.

--gps-end-time GPS SECONDS
Required. End time, GPS seconds.

--gps-end-time-ns GPS SECONDS
Optional. Start time, GPS nanoseconds. If not specified, then set to zero.

--frame-cache FRAME_CACHE
Optional. Name of the frame cache file. If this is not specified then the code will not read in any input data. Hence it will be unable to produce the raw or raw_plus_inj frames.

--channel-name CHAN
Optional. Input data channel name. This must be specified if the frame-cache file is specified.

--ifo IFO
Optional. Set input interferometer name to IFO. This information is required unless a channel-name is specified (in this case the IFO is obtained from the channel name).

--sample-rate SAMPLE_RATE
Optional. Set the sample rate (in Hz) at which the injections should be produced. This is required unless a frame cache is given (in this case, the sample rate of the injections matches the input data).

--calibration-cache CAL_CACHE
Optional. Specify the CAL_CACHE to obtain the calibration information.

--calibrated-data TYPE
Optional. Use calibrated data as input/output. This will produce strain data injections. This option must be specified if no calibration cache is given.

--num-resp-points N
Optional. The number of points used to generate the response function. If generating uncalibrated injections, this can actually change slightly the injection data. If not specified, then set to default value of 4194304.

--injection-channel INJ
Optional. If this option is specified, the code will read in injection data from the INJ channel and add it to the raw data. Either this option or --injection-file must be specified.

--injection-file FILE
Optional. Read in the injection data from FILE. This should contain a sim_inspiral table with a list of injections. Either this option or --injection-channel must be specified.

--inject-overhead
Optional. Perform the injections directly overhead the interferometer and optimally oriented.

--inject-safety SEC
Optional. Inject signals whose end time is up to SEC after the gps end time.

--write-raw-data
Optional. Write out a frame containing the raw data. This can only be done if the raw data has been read in from frames.

--write-inj-only
Optional. Write out a frame containing only the injection data.

--write-raw-plus-inj
Optional. Write out a frame containing the raw data with the injections added.
--output-frame-length OUTPUT_LENGTH
   Required. Specify the length of the output frames to be OUTPUT_LENGTH seconds.

--output-file-name OUTPUT_NAME
   Optional. Set the output file name. This overrides the default file naming convention given above.

Example
lalapps_frinspinj --gps-start-time 732758030 --gps-end-time 732760078 \
    --frame-cache cache/L-732758022-732763688.cache --channel-name L1:LSC-AS_Q\ 
    --calibration-cache cache_files/L1-CAL-V03-729273600-734367600.cache \
    --injection-file HL-INJECTIONS_4096-729273613-5094000.xml \
    --inject-safety 50 --output-frame-length 16 \
    --write-raw-data --write-inj-only --write-raw-plus-inj

Author
   Steve Fairhurst
3.14 Program **lalapps_inspmultiawg**

**Name**

lalapps_inspmultiawg — injects specified inspiral chirps into zero data. Intended for producing the hardware injection data.

**Synopsis**

```
--help     display this message
--source   source file containing details of injection
--actuation ACTFILE file containing the actuation function
--darm2inj DCFACTOR calibration between darm_ctrl and injection point
--summary  SUMFILE write details of injections to file
--ifo      IFO name of interferometer (optional)
--flow     FSTART start frequency of injection (default 40 Hz)
--fhigh    FSTOP end frequency of injection (default: end at ISCO)
--smooth   QFAC ringdown the end of the injection with Q factor Qfac
--length   LENGTH length of the data (default 64 seconds)
--samplerate FREQ rate at which data is sampled (default 16384Hz)
--debug-level DEBUG the lal debug level
--user-tag TAG user-tag added to output file names
```

**Description**

lalapps_inspmultiawg injects inspiral chirps into zero data. The details of several chirps can be specified using the command **--source**, otherwise, a single inspiral of two 1.4 solar mass neutron stars will be injected. Each chirp is injected into a new file containing zero data of LENGTH seconds, sampled at FREQ Hz, and each injection begins at the beginning of the data. The actuation function can be provided in ACTFILE. Additionally, there is a parameter DCFACTOR which is used to convert between the actuation function and the “response function” between ETM_EXC and strain. The chirp is output to a file named

```
TAG_inspiral_NUMBER_IFO.out
```

where TAG is the user-tag, IFO is the name of the interferometer, and NUMBER is the injection number. A summary of the injections performed can be saved in SUMFILE.

**Options**

```
--sourcefile SFILE
  Optional. Reads source information from the file SFILE. If absent, it injects a single 1.4M⊙–1.4M⊙ inspiral, optimally oriented, at a distance of 1.0Mpc.

--actuation ACTFILE
  Optional. Reads a detector actuation function from the file ACTFILE. If absent, it generates raw dimensionless strain. The actuation function should contain 3 columns; frequency, followed by amplitude and phase.

--darm2inj DCFACTOR
  Optional. Multiplicative factor used in calculating the response function from the actuation function. More specifically, the actuation function is given by
  \[ x_C = A(f) \times \text{DARM}_\text{CTRL} \]  \hspace{1cm} (3.28)
  while we are taking the response function to be:
  \[ h(f) = R(f) \times \text{ETM}_\text{EXC} \]  \hspace{1cm} (3.29)
  Then, DCFACTOR is given by
  \[ A(f) = \text{DCFACTOR} \times R(f) \]  \hspace{1cm} (3.30)
```
--smooth QFAC
Optional. This will smooth out the end of the injected inspiral. After the inspiral, the frequency of the waveform will be held constant while the amplitude is exponentially decreased as \( a = a_{\text{final}} \exp(-\pi f_{\text{final}} t / QFAC) \). Here, \( t \) is the amount of time after the end of the inspiral.

--summary SUMFILE
Optional. The SUMFILE format is LIGO lightweight XML with process, process params and sim_inspiral tables. The sim_inspiral table contains details of all the injections performed. The details of the injection are obtained from the source file.

--length SEC
Optional. Specify the length of data into which the signal will be injected. The default is 64 seconds.

--samplerate FREQ
Optional. Specify the rate at which the data is sampled. The default is 16384 Hz.

--ifo IFO
Optional. Give the name of the interferometer for which the injections are intended. This is only used in naming the output files.

--user-tag TAG
Optional. A user-tag which is used in naming the output files.

--flow FSTART
Optional. Give the start frequency FSTART for the inspiral. The default is 40 Hz

--fhigh FSTOP
Optional. Give the end frequency FSTOP for the inspiral. The default behaviour is that the inspiral will continue to ISCO. If set to a negative number, the generator will use its absolute value as the terminating frequency, but will ignore post-Newtonian breakdown.

--debug-level DEBUG
Optional. Set the LAL debug level. The default is 33.

--help
Optional. Print a help message.

Format for sourcefile: The source file consists of any number of lines of data, each specifying a chirp waveform. Each line must begin with a character code (CHAR equal to one of ‘i’, ‘f’, or ‘c’), followed by 6 whitespace-delimited numerical fields: the epoch of the chirp (INT8 seconds), the two binary masses (REAL4 \( M_\odot \)), the distance to the source (REAL4 Mpc), and the source’s inclination and phase at coalescence (REAL4 degrees). The character codes have the following meanings:

‘i’ The epoch represents the GPS time of the start of the chirp waveform.
‘f’ The epoch represents the GPS time of the end of the chirp waveform.
‘c’ The epoch represents the GPS time when the binaries would coalesce in the point-mass approximation.

Since the injection is started at time \( t = 0 \), it is recommended that the ‘i’ option is used.

Thus a typical input line for two \( 1.4M_\odot \) objects at 1.1 Mpc inclined 30° with an initial phase of 45°, beginning at 70 seconds (after the start of the injections), will have the following line in the input file:

\[ i \quad 70 \quad 1.4 \quad 1.4 \quad 1.1 \quad 30.0 \quad 45.0 \]

The time parameter (in this case 70 sec) does not affect the output data in any way. It is simply stored in the sim_inspiral table of the SUMFILE, in order to make analysis of the injections easier.
Format for actfile:  The actuation function \( A(f) \) gives the transformation from DARM_CRTL to \( X_C \) via \( X_C = A(f) \times \text{DARM}_\text{CTRL} \). It is combined with \( \text{DCFAC} \text{TOR} \) as described above to give the response function from the exitation channel to strain as \( \tilde{h}(f) = R(f) \times \text{ETM}_\text{EXC}(f) \). This is inverted internally to give the transfer function \( T(f) = 1/R(f) \).

The format for actfile is three columns of \texttt{REAL4} data giving the frequency followed by the amplitude and phase of the actuation function at that frequency. This is the format which is produced by the calibration team.

Format for the data output:  The data output in the files \texttt{TAG_inspiral\_NUMBER\_IFO.txt} is a single column of \texttt{REAL4} ADC data.

Example

```
lalapps_inspmultiawg --source s3.sources \
  --actuation E10-H1-ACTUATION.txt --darm2inj 8000\n  --summary summ.xml --ifo H1 --smooth 5
```

Author

Steve Fairhurst
3.15 Program `lalapps_splitbank`

Name

`lalapps_splitbank` — splits a template bank file into several smaller files

Synopsis

```
lalapps_splitbank
   --bank-file <file>
   --comment <comment>
   --debug-level <level>
   --help
   --minimal-match <m>
   --number-of-banks <n>
   --user-tag <comment>
   --verbose
   --version
```

Description

`lalapps_splitbank` splits a LIGO_LW XML file containing inspiral templates in a `sngl.inspiral` table into several smaller bank files. This allows a template bank to be split across several inspiral jobs and then recombined with `lalapps_inca` or `lalapps_sire`.

The name of the output template bank files is derived from the name of the input bank file and the number of files that the bank should be split into. For example, if the input bank file:

```
H1-TRIGBANK_L1-729330491-2048.xml
```

is split into 3 output files, then these will be named:

```
H1-TRIGBANK_L1-00-729330491-2048.xml
H1-TRIGBANK_L1-01-729330491-2048.xml
H1-TRIGBANK_L1-02-729330491-2048.xml
```

The naming convention is to insert the bank file number after the usertag part of the filename and before the GPS start time part of the file name.

In the case that the input file contains no templates, empty output bank files are generated. This is done since DAGman does not implement decision rules yet, so the nodes in the DAG must be identical regardless of the data flowing through them.

Options

```
--bank-file <file>
   Read the templates from the `sngl.inspiral` table in the file `<file>`.

--comment <comment>
   Add the string `<comment>` to the `process` table in the output XML file.

--debug-level <level>
   Sets the LAL debug level to `<level>`. The default value is `LALMSGLVL2`. A useful setting is 65 which turns off memory padding, but keeps memory tracking and error messages. If you want to turn off memory tracking completely, then use 33.
```
--help
Display a usage message and exit.

--minimal-match <m>
Set the minimal match of the output template bank file to <m>. This option is not really needed for running lalapps_splitbank, it just put that value of <m> for the minimal match in all splited template banks.

--number-of-banks <n>
Split the input template banks into <n> separate output bank files.

--user-tag <comment>
Set the user tag to the string <comment>. This string must not contain spaces or dashes ("-"). This string will appear in the name of the file to which output information is written, and is recorded in the various XML tables within the file.

--verbose
Print debugging information to the standard output while executing.

--version
Print the CVS id and exit.

Example
lalapps_splitbank --bank-file L1-TMPLTBANK-732488741-2048.xml \
--number-of-banks 3 --minimal-match 0.97

Algorithm
lalapps_splitbank counts the number of templates in the input file. It increments this by one and divides by the number of template banks to generate using standard integer division. This gives the upper limit on the number of templates in a single output file.

Author
Duncan Brown and Alexander Dietz
3.16 Program **lalapps_coinext**

**Name**

lalapps_coinext — analyzing coincident external triggers online

**Synopsis**

```
coinExt
[--help]
[--test]
[--dirInspiral <dir_insp>]
[--dirData <dir_data>]
[--refresh <t_refresh>]
[--timeWindow <t_window>]
[--snrCut <snr>]
[--ifo <name_ifo>]
[--trigger <trigger_file>]
[--restart]
[--recalc]
```

**Description**

This online looks for coincidences on external and inspiral triggers, creating and transferring a XML file containing both data.

After a time $<t_{\text{refresh}}>$ has elapsed the actual list of external triggers, placed on http://www.uoregon.edu/ileonor/lib is downloaded. This file is updated whenever an external trigger from the GCN (The GRB Coordinates Network, see http://gcn.gsfc.nasa.gov/) has been received. Since it takes some time from the observation of a Gamma-Ray Burst to the notification, about 10-20 minutes elapses until the trigger is placed into the downloaded file.

**coinext** looks either for inspiral triggers specified in the directory $<\text{dir}_{\text{insp}}>$ or it creates an own DAG for creating the inspiral triggers by itself (when setting the option --recalc). The DAG is run in the directory OnlineAnalysis/Jobs and the final inspiral triggers are put into Trigger.

All inspiral triggers within a time window of $<t_{\text{window}}>$ and the information of the external trigger are stored into a XML file in the directory Data or $<\text{dir}_{\text{data}}>$. The XML file also is transferred to the machine adietz.phys.lsu.edu for further analysis.

If the option --restart is specified, all intermediate data files will be deleted, such as the local list of external triggers with the state of each trigger (later more).

**Options**

```
--dirData <dir_data>
  Optional argument, specifying the directory where the resulting XML files are written to. The default directory is CoinExt/Data.

--dirInspiral <dir_insp>
  Optional argument, specifying the directory searched for inspiral trigger XML-files. The default directory is Triggers/Inspiral/???

--help
  Prints a help message and exits.
```
--refresh $t_{refresh}$
  Setting the refresh rate to $t_{refresh}$ minutes. This is the time the program waits until it checks for new external triggers. The default value is 30 minutes.

--test
  With this option set the test mode is activated. The external triggers are read from a file CoinExt/ProgCoin/Test/triggers.xml? and the refresh rate is set to 60 seconds.

--timeWindow $t_{window}$
  Sets the coincident time window that is searched for coincidences.

--snrCut $cut$
  Sets a threshold for the selection of inspiral triggers. Only triggers with a SNR value greater than $cut$ are considered.

--ifo $ifo_{name}$
  Specifies at which site the code is running. Possible parameters are LLO and LHO.

--trigger $trigger_{file}$
  When specifying this option coinExt used always the trigger file $trigger_{file}$ instead downloading a new one.

--restart
  With this option the whole analysis on the triggers is restarted.

--recalc
  With this option set for each GRb trigger an inspiral DAG will be run on the cluster to calculate the inspiral triggers.

Example 1
  will follow soon

Notes
  This program requires tconvert to be installed.

Author
  Alexander Dietz
Chapter 4

Ringdown Search Programs

4.1 Program lalapps_ring

Name
lalapps_ring — filters data through a bank of ringdown filters.

Synopsis

Description
lalapps_ring uses matched filtering to search for ringdown waveforms in gravitational wave data.

Options

- **-h**
  Print a help message.

- **-v**
  Print the version information.

- **-v**
  Verbose output.

- **-d dbglvl**
  Set LAL debug level to dbglvl.

- **-i infile**
  Read filter parameters from input file in file [stdin].

- **-o outfile**
  Write the output to file outfile [stdout].

- **-k**
  Keep filtering results (for use with option -s).

- **-s**
  Save intermediate filtering results as .dat and snr- files.

- **-f framefile**
  Read channel data from framefile [*.gwf].
-r resfile
   Read response function data from resfile [response.asc].

-c channel
   Use channel channel data [H1:LSC-AS_Q].

-n numpoints
   Use numpoints points of data [65536].

-t starttime
   Use data starting at GPS time starttime [start of frame data].

-b b0,b1
   Filter only template numbers b0 to b1 in bank [0,end of bank].

--filterparams
   Specify filter parameters as command line arguments filterparams (see below for filter parameters).

Filter parameters
The filter parameters can be specified either on the command line as arguments following the -- option or in a resource file that is input using the -i option (or from stdin). As a resource file, each option-value pair should have their own line.

-segsz npts
   Set the size of segments analyzed to npts points.

-speclen len
   Set the size of inverse spectrum truncation to len points [0].

-flow flow
   Set the low frequency cutoff to flow Hz.

-fmin fmin
   Set the minimum frequency for the bank to fmin Hz.

-fmax fmax
   Set the maximum frequency for the bank to fmax Hz.

-qmin qmin
   Set the minimum quality for the bank to qmin.

-qmax qmax
   Set the maximum quality for the bank to qmax.

-maxmm maxmm
   Set the maximum allowed mismatch for the bank to maxmm.

-thresh thresh
   Set the ringdown event signal-to-noise ratio threshold to thresh.

-scale scale
   Scale the response function by a dynamic range factor of scale [1].

Debug levels
The LAL debug level can be specified as an integer or as a string of flags:

NDEBUG
   No debugging information is printed and memory debugging code is disabled.
4.1. Program \texttt{lalapps\_ring}

\begin{verbatim}
ERROR
Error messages are printed.

WARNING
Warning messages are printed.

INFO
Information messages are printed.

TRACE
Function call tracing messages are printed.

MEMINFO
Memory allocation information messages are printed.

MEMDBG
Debugging of memory allocation routines is enabled but no messages are printed.
\end{verbatim}

The following composite levels are available:

\begin{verbatim}
MSGLVL1
Equivalent to \texttt{ERROR}

MSGLVL2
Equivalent to \texttt{ERROR | WARNING}

MSGLVL3
Equivalent to \texttt{ERROR | WARNING | INFO}

ALLDBG
All debugging messages are printed.
\end{verbatim}

For example, the command

\texttt{lalapps\_ring -d "ERROR | INFO" ...}

will set the debug level so that error and information messages are printed.

\textbf{Environment}

\begin{verbatim}
LAL\_DEBUG\_LEVEL
Default LAL debug level to use.
\end{verbatim}

\textbf{Author}

Jolien Creighton
Chapter 5

Stochastic Search Programs
5.1 Program \texttt{lalapps\_olapredfcn}

Name
\texttt{lalapps\_olapredfcn} — computes overlap reduction function given a pair of known detectors.

Synopsis
\texttt{lalapps\_olapredfcn}
\begin{itemize}
  \item \texttt{-h}
  \item \texttt{-q}
  \item \texttt{-v}
  \item \texttt{-d <debugLevel>}
  \item \texttt{-s <siteID1>}
  \item \texttt{-a <azimuth1>}
  \item \texttt{-t <siteID2>}
  \item \texttt{-b <azimuth2>}
  \item \texttt{-f <fLow>}
  \item \texttt{-e <deltaF>}
  \item \texttt{-n <numPoints>}
  \item \texttt{-o <outfile>}
\end{itemize}

Description
\texttt{lalapps\_olapredfcn} computes the overlap reduction function $\gamma(f)$ for a pair of known gravitational wave detectors. It uses the LAL function \texttt{LALOverlapReductionFunction()}, which is documented in the LAL Software Documentation under the \texttt{stochastic} package.

Options
\begin{itemize}
  \item \texttt{-h}
    Print a help message.
  \item \texttt{-q}
    Run silently (redirect standard input and error to /dev/null).
  \item \texttt{-v}
    Run in verbose mode.
  \item \texttt{-d <debugLevel>}
    Set the LAL debug level to \texttt{<debugLevel>}.\end{itemize}

\begin{itemize}
  \item \texttt{-s <siteID1>, -t <siteID2>}
    Use detector sites identified by \texttt{<siteID1>} and \texttt{<siteID2>}; ID numbers between \texttt{LALNumCachedDetectors} (defined in the \texttt{tools} package of LAL) refer to detectors cached in the constant array \texttt{lalCachedDetectors[]}. (At this point, these are all interferometers.) Additionally, the five resonant bar detectors of the IGEC collaboration can be specified. The bar geometry data (summarized in table 5.1) is used by the function \texttt{LALCreateDetector()} from the \texttt{tools} package of LAL to generate the Cartesian position vector and response tensor which are used to calculate the overlap reduction function. The ID numbers for the bars depend on the value of \texttt{LALNumCachedDetectors}; the correct ID numbers can be obtained by with the command

\begin{verbatim}
> lalapps_olapredfcn -h
\end{verbatim}

\item \texttt{-a <azimuth1>, -b <azimuth2>}
    If \texttt{<siteID1>} (\texttt{<siteID2>}) is a bar detector, assume it has an azimuth of \texttt{<azimuth1>} (\texttt{<azimuth2>}) degrees East of North rather than the default IGEC orientation given in table 5.1. Note that this convention, measuring azimuth in degrees clockwise from North is not the same as that used in LAL (which comes from the frame spec). Note also that any specified azimuth angle is ignored if the corresponding detector is an interferometer.
5.1. Program \texttt{lalapps\_olapredfcn}  \hfill Page 115

<table>
<thead>
<tr>
<th>Name</th>
<th>Longitude</th>
<th>Latitude</th>
<th>Azimuth</th>
</tr>
</thead>
<tbody>
<tr>
<td>AURIGA</td>
<td>11°56'54&quot;E</td>
<td>45°21'12&quot;N</td>
<td>N44°E</td>
</tr>
<tr>
<td>NAUTILUS</td>
<td>12°40'21&quot;E</td>
<td>41°49'26&quot;N</td>
<td>N44°E</td>
</tr>
<tr>
<td>EXPLORER</td>
<td>6°12'E</td>
<td>46°27'N</td>
<td>N39°E</td>
</tr>
<tr>
<td>ALLEGRO</td>
<td>91°10'43&quot;766W</td>
<td>30°24'45&quot;110N</td>
<td>N40°W</td>
</tr>
<tr>
<td>NIOBE</td>
<td>115°49'E</td>
<td>31°56'S</td>
<td>N0°E</td>
</tr>
</tbody>
</table>

Table 5.1: Location and orientation data for the five IGEC resonant bar detectors, stored in the \texttt{lalCachedBars[]} array. The data are taken from \url{http://igec.lng.infn.it/cgi-bin/browser.pl?Level=0,3,1} except for the latitude and longitude of ALLEGRO, which were taken from Finn \& Lazzarini, gr-qc/0104040. Note that the elevation above the WGS-84 reference ellipsoid and altitude angle for each bar is not given, and therefore set to zero.

\begin{itemize}
\item \texttt{-f <fLow>}
  \begin{itemize}
  \item Begin the frequency series at a frequency of \texttt{<fLow>} Hz; if this is omitted, the default value of 0 Hz is used.
  \end{itemize}
\item \texttt{-e <deltaF>}
  \begin{itemize}
  \item Construct the frequency series with a frequency spacing of \texttt{<deltaF>} Hz
  \end{itemize}
\item \texttt{-n <numPoints>}
  \begin{itemize}
  \item Construct a frequency series with \texttt{<numPoints>} points.
  \end{itemize}
\item \texttt{-o <outfile>}
  \begin{itemize}
  \item Write the output to file \texttt{outfile}. The format of this file is that output by the routine \texttt{LALPrintFrequencySeries()} in the \texttt{support} package of LAL, which consists of a header describing metadata followed by two-column rows, each containing the doublet \{\texttt{f}, \gamma(\texttt{f})\}.
  \end{itemize}
\end{itemize}

\textbf{Example}

To compute the overlap reduction function for LIGO Hanford and LIGO Livingston, with a resolution of 1 Hz from 0 Hz to 1024 Hz:

\begin{verbatim}
> lalapps_olapredfcn -s 0 -t 1 -e 1 -n 1025 -o LHOLLO.dat
\end{verbatim}

To compute the overlap reduction function for ALLEGRO in its optimal orientation of 72°08 West of South (see Finn \& Lazzarini, gr-qc/0104040) and LIGO Livingston, with a resolution of 0.5 Hz from 782.5 Hz to 1032 Hz (assuming \texttt{lalNumCachedBars} is 6):

\begin{verbatim}
> lalapps_olapredfcn -s 9 -a 252.08 -t 1 -f 782.5 -e 0.5 -n 500 -o ALLEGROLHO.dat
\end{verbatim}

\textbf{Author}

John T. Whelan
5.2 Running the LALApps Stochastic Pipeline Under Condor

This section describes how to run the LALApps stochastic analysis pipeline under Condor at the various LSC Data Grid Sites, such as CIT, UWM, and the observatory clusters at LHO and LL0.

The pipeline is constructed as a directed acyclic graph (DAG) run under Condor. A DAG represents a set of programs where the input, output, or execution of one or more programs is dependent upon one or more other programs. The programs are represented by nodes (vertices) in the graph, and the edges (arcs) identifies the dependencies. There are three different nodes that make up the Stochastic analysis DAG - datafind, stochastic and stopp. A short description of the nodes that make up the stochastic DAG follows:

Datafind Node

The datafind node, as its name suggests, is for determining the location of the data that is to be analysed. It uses LSCdataFind to query the LDR database for the location of the data on the cluster. This step in the pipeline does not need to be run for every analysis, only when the location of the data changes.

Stochastic Node

The stochastic node runs an instance of lalapps_stochastic, the main stochastic analysis code. See Section 5.4 for more information on lalapps_stochastic.

Stopp Node

The stopp node runs an instance of lalapps_stopp, the STOchastic Post Processing code. See Section 5.5 for more information on lalapps_stopp.

5.2.1 Creating the DAG

In order to create the DAG, a configuration file is required, examples of which can be found in the lalapps/src/stochastic/example directory. This configuration specifies the parameters to use for each step of the pipeline. See Section 5.3 for details regarding the layout of the configuration file.

Any combination of nodes, see above, can be present in the DAG, assuming that the dependencies for each node are satisfied. The datafind node has no dependencies, the stochastic node requires cache files to be available, so if cache have already been generated and the location of the data on the cluster has not changed then there is no read for the datafind node to be run, and finally the stopp node requires that the xml output file from lalapps_stochastic are available.

The following example will create a DAG containing all nodes, datafind, stochastic and stopp.

```
> lalapps_stochastic_pipe --datafind --stochastic --stopp \
   --config-file example_S3_H1L1.ini --log-path logs
```

In addition to creating the DAG it will also create several other file required for running the search under condor, descriptions of these files can be found below:

- **example_S3_H1L1.dag** This is main dag file that includes all the required information for running the pipeline. This includes the dependency information for each job within the DAG along with all the dynamic command line arguments for each job.

- **example_S3_H1L1.datafind.sub** This is the submit file used to submit datafind jobs to condor. It includes all the static command line options along with variable command line options, for each job, that are defined within the main .dag file.

- **example_S3_H1L1.stochastic.sub** This is the submit file used to submit stochastic jobs to condor. It includes all the static command line options along with variable command line options, for each job, that are defined within the main .dag file.
5.2. Running the LALApps Stochastic Pipeline Under Condor

example_S3_H1L1.stopp.sub This is the submit file used to submit stopp jobs to condor. It includes all the static command line options along with variable command line options, for each job, that are defined within the main .dag file.

example_S3_H1L1.pipeline.log This is the log file that details how the data, from the input segment list, has been split up into different jobs.

5.2.2 Running the DAG

When the DAG has been created successfully, a message will be displayed describing what needs to be done inorder to run the analysis pipeline at a LSC DataGrid Site.

LSCdataFind works by querying the datafind server for the location of the requested data, this requires the user running LSCdataFind to have a valid grid proxy. Therefore if the DAG includes any datafind nodes, a valid grid proxy must be available prior to the job being submitted to the cluster. This is achieved by running the following commands:

> unset X509_USER_PROXY
> grid-proxy-init

This will remove any existing proxy and recreate another, requesting the grid certificates pass phase. The default duration for the proxy to be valid is 12 hours, if a longer duration is required it can be increased by appending “-hours <HOURS>” to the call to grid-proxy-init.

Now that everything is setup, the DAG is ready to be submitted to condor for execution. This is achieved with the following command.

> condor_submit_dag example_S3_H1L1.dag

The progress of the DAG can be monitored by studying the DAGman output file, example_S3_H1L1.dag.dagman.out. This contains information regarding the current state of the search, what jobs have been completed successfully, those that have failed, and those that fail to run due to unsatisfied dependencies. Once the DAG has completed, if and jobs have failed, a rescue DAG will be written out. This can be then submitted to the cluster in the same way as before, the difference is that this will only attempt to run jobs that previously failed.

NOTE: currently there is a problem with running a large number of stochastic jobs at any one time. As condor is designed to run jobs that take over an hour to complete and the average runtime for a stochastic job is a matter a minutes, running a large number of stochastic jobs at the same time will greatly reduce the computational efficiency of a cluster. Whilst this problem it resolved it is recommended to submit the DAG with the following command, limiting the total number of jobs to run to a maximum of 30:

> condor_submit_dag -maxjobs 30 example_S3_H1L1.dag
5.3 Program `lalapps_stochastic.pipe`

**Name**

`lalapps_stochastic.pipe` — python script to generate Condor DAGs to run the stochastic pipeline.

**Synopsis**

```
lalapps_stochastic.pipe
   [--help]
   [--version]
   [--user-tag <TAG>]
   [--datafind]
   [--stochastic]
   [--stopp]
   [--priority]
   --config-file <FILE>
   --log-path <PATH>
```

**Description**

`lalapps_stochastic.pipe` builds a stochastic search DAG suitable for running at the various LSC Data Grid sites. The configuration file specifies the parameters needed to run the analysis jobs contained within the pipeline. It is specified with the `--config-file` option. Examples of typical .ini files can be found in the directory `lalapps/src/stochastic/example`.

The .ini file contains several sections. The `[condor]` section contains the names of executables which will run the various stages of the pipeline. The `[pipeline]` section gives the CVS details of the pipeline. The `[input]` section specifies the segment list and the minimum and maximum segment duration for the jobs. The `[datafind]` section specifies the frame types for the two input streams for passing onto LSCdataFind. The `[detectors]` section specifies the detector pair to cross correlate. The `[calibration]` section specifies the location and name of the various calibration cache files. The `[stochastic]` section specifies static options to pass to `lalapps_stochastic`, i.e. options that are not automatically generated. Finally the `[stopp]` section specifies the static options to pass onto `lalapps_stopp`.

**Options**

- **--help**
  Display usage information

- **--version**
  Display version information

- **--user-tag <TAG>**
  The tag for the job

- **--datafind**
  Run LSCdataFind as part of the DAG to create the cache files for each science segment

- **--stochastic**
  Run `lalapps_stochastic` on the data

- **--stopp**
  Run `lalapps_stopp` on the data

- **--priority <PRIO>**
  Run jobs with condor priority `<PRIO>`

- **--config-file <FILE>**
  Use configuration file `<FILE>`
5.3. Program `lalapps_stochastic.pipe`

```
--log-path <PATH>
Directory to write condor log file
```

**Configuration File Options**

The configuration file is a standard python configuration that can be easily parsed using the standard ConfigParser module from python. The configuration file consists of sections, led by a “[section]” header and followed by “name: value” entries. The optional values can contain format strings which refer to other values in the same section. Lines beginning with “#” or “;” are ignored and may be used to provide comments.

The first section required is “[condor]”, this specifies all the parameters associated with condor.

**universe**

Specifies the condor universe under which to run, this should be set to ”standard” if LALApps has been compiled with the `--enable-condor`, otherwise it should be set to “vanilla”.

**datafind**

Specifies the location of the `LSCdataFind` script.

**stochastic**

Specifies the location of the `lalapps_stochastic` binary.

**stopp**

Specifies the location of the `lalapps_stopp` binary.

The next section is “[pipeline]”, this specifies version information of the pipeline, currently only the version of configuration file is specified.

**version**

A CVS `$Id$` keyword specifying the version of the configuration file.

The next section is ”[input]” which specifies parameters regarding the input data.

**segments**

Specifies the location of the segment list, outputed from `SegWizard` listing the Science Segments to analyse.

**min_length**

Specifies the minimum length of Science Segment to analyse

**max_length**

Specifies the maximum length at which the split the Science Segments into.

**channel**

Currently the pipeline infrastructure requires that the ”[input]” section contains the option ”channel”, this can be set to anything as it not used within the `LSCdataFind` class.

The next section is “[datafind]” which specifies parameters for `LSCdataFind`.

**type-one**

Specifies the frame type of the first data stream for `LSCdataFind` to find.

**type-two**

Specifies the frame type of the second data stream for `LSCdataFind` to find.

**server**

Specifies which LDRdataFindServer to use.

The next section is ”[detectors]” which specifies the detectors to use for the search.
5.3. Program `lalapps_stochastic.pipe`

**detector-one**
- Specifies the detector for the first data stream.

**detector-two**
- Specifies the detector for the second data stream.

The next section is “[calibration]” which specifies the location of the calibration files.

**path**
- Specifies the path to the calibration cache files.

**L1**
- Specifies the cache file for the Livingston 4 km detector.

**H1**
- Specifies the cache file for the Hanford 4 km detector.

**H2**
- Specifies the cache file for the Hanford 2 km detector.

The next section is “[stochastic]”, this is used to specify any static options to pass onto `lalapps_stochastic`, i.e. options that are not automatically generated. See Section 5.4 for the accepted options for `lalapps_stochastic`. The options that are automatically generated are, `--gps-start-time`, `--gps-end-time`, `--ifo-one`, `--ifo-two`, `--frame-cache-one`, `--frame-cache-two`, `--calibration-cache-one`, and `--calibration-cache-two`.

The next section is “[stopp]”, this is used to specify any static options to pass onto `lalapps_stopp`, i.e. options that are not automatically generated. See Section 5.5 for the accepted options for `lalapps_stopp`. The XML output files from `lalapps_stochastic` are automatically added.

**Example**
- Generate a DAG to run a stochastic search on a pair of interferometers specified in the configuration file. The generated DAG is then submitted with `condor_submit_dag`

  ```bash
  > lalapps_stochastic_pipe --log-path /home/ram/dag_logs \
  >   --datafind --stochastic --stopp --config-file stochastic_H1L1.ini
  > condor_submit_dag stochastic_H1L1.dag
  ```

**Author**
- Adam Mercer
5.4  Program **lalapps_stochastic**

**Name**

*lalapps_stochastic* — standalone stochastic analysis code.

**Synopsis**

```
lalapps_stochastic
  [--help]
  [--version]
  [--verbose]
  [--debug]
  [--debug-level <N>]
  [--user-tag <STRING>]
  [--comment <STRING>]
  [--output-dir <DIR>]
  [--cc-spectra]
  --gps-start-time <N>
  --gps-end-time <N>
  --interval-duration <N>
  --segment-duration <N>
  --resample-rate <N>
  --f-min <N>
  --f-max <N>
  --ifo-one <IFO>
  --ifo-two <IFO>
  --channel-one <CHANNEL>
  --channel-two <CHANNEL>
  --frame-cache-one <FILE>
  --frame-cache-two <FILE>
  --calibration-cache-one <FILE>
  --calibration-cache-two <FILE>
  --calibration-offset <N>
  [--apply-mask <N>]
  --mask-bin <N>]
  [--overlap-hann
  --hann-duration <N>]
  [--high-pass-filter
   --hpf-frequency <N>
   --hpf-attenuation <N>
   --hpf-order <N>]
  --recentre
  --middle-segment
  [--geo-hpf-frequency <N>
   --geo-hpf-attenuation <N>
   --geo-hpf-order <N>]
  [--alpha <N>]
  [--f-ref <N>]
  [--omega0 <N>]
```

**Description**

*lalapps_stochastic* runs the standalone stochastic analysis code.

**Options**
--help
Display usage information and exit.

--version
Display version information and exit.

--verbose
Enable the output of informational messages.

--debug
Run in debug mode, saves out all intermediate products as ASCII files.

--debug-level <N>
Sets the LAL debug level to <N>. The default value is LALMSGLVL2, displaying error and warning messages. A useful setting is 65 which turns off memory padding, but keeps memory tracking and error messages. If you want to turn off memory tracking completely, then use 33.

--user-tag <STRING>
Set the user tag to the string <STRING>. This string must not contain spaces or dashes (“-”). This string will appear in the name of the file to which output information is written, and is recorded in the various XML tables within the file.

--comment <STRING>
Set the process table comment to <STRING>

--output-dir <DIR>
Set the output directory for search results to <DIR>

--cc-spectra
Save out cross correlation spectra as frame files.

--gps-start-time <N>
Sets the GPS time from which data should be read to <N>

--gps-end-time <N>
Sets the GPS time to which data should be read to <N>

--interval-duration <N>
Sets the interval duration to <N>

--segment-duration <N>
Sets the segment duration to <N>

--resample-rate <N>
Down-convert the input data stream to a sample rate of <N> samples per second prior to analysis. This can be used to reduce the number of CPU cycles required to analyze a given quantity of input data.

--f-min <N>
Sets the minimum frequency of the search band to <N>

--f-max <N>
Sets the maximum frequency of the search band to <N>

--ifo-one <IFO>
Sets the IFO for the first stream to be <IFO>, currently supported IFO’s are H1, H2, L1 and G1

--ifo-two <IFO>
Sets the IFO for the second stream to be <IFO>, currently supported IFO’s are H1, H2, L1 and G1
--channel-one <CHANNEL>
    Sets the channel for the first stream to be <CHANNEL>

--channel-two <CHANNEL>
    Sets the channel for the second stream to be <CHANNEL>

--frame-cache-one <FILE>
    Obtain the locations of input .gwf frame files from the LAL frame cache file <FILE> for the first detector. LAL frame cache files are explained in the “framedata” package in LAL and can be constructed by using LSCdataFind on supported systems.

--frame-cache-two <FILE>
    Obtain the locations of input .gwf frame files from the LAL frame cache file <FILE> for the second detector. LAL frame cache files are explained in the “framedata” package in LAL and can be constructed by using LSCdataFind on supported systems.

--calibration-cache-one <FILE>
    Specify the location of calibration information for the first detector. <FILE> gives the path to a LAL-format frame cache file describing locations of .gwf frame files that provide the calibration data (\(\alpha\) and \(\beta\) coefficients) for the analysis. Frame cache files are explained in the “framedata” package in LAL.

--calibration-cache-two <FILE>
    Specify the location of calibration information for the second detector. <FILE> gives the path to a LAL-format frame cache file describing locations of .gwf frame files that provide the calibration data (\(\alpha\) and \(\beta\) coefficients) for the analysis. Frame cache files are explained in the “framedata” package in LAL.

--calibration-offset <N>
    Sets the calibration offset to <N>

--apply-mask
    Apply frequency masking

--mask-bin <N>
    Set the number of bins to mask per frequency to <N>

--overlap-hann
    Use overlapping Hann windows for data segments

--hann-duration <N>
    Set the Hann duration of the data segment window to <N>, 0 for Rectangular windowing, 1 for Tukey windowing and 60 for Hann windowing

--high-pass-filter
    Apply a high pass filter to the input data

--hpf-frequency <N>
    Set the knee frequency of the high pass filter to <N>

--hpf-attenuation <N>
    Set the attenuation coefficient for the high pass filter to <N>

--hpf-order <N>
    Sets the high pass filter order to <N>

--recentre
    Centre the data

--middle-segment
    Include the middle segment in the power spectra estimation
--geo-hpf-frequency <N>
Set the knee frequency for the GEO high pass filter to <N>

--geo-hpf-attenuation <N>
Set the attenuation coefficient for the GEO high pass filter to <N>

--geo-hpf-order <N>
Set the GEO high pass filter order to <N>

--alpha <N>
Exponent for $\Omega_{GW}$ for construction of the optimal filter.

--f-ref <N>
Reference frequency for $\Omega_{GW}$ for the construction of the optimal filter.

--omega0 <N>
Reference $\Omega_0$ for $\Omega_{GW}$ for the construction of the optimal filter.

Example

lalapps_stochastic is generally run as part of a DAG, as created by the pipeline generation scripts, lalapps_stochastic_pipe or lalapps_stochastic_bayes, however an example usage can be seen below.

```bash
> lalapps_stochastic --debug-level 33 --verbose \
> --gps-start-time 752242398 --gps-end-time 752242758 \
> --interval-duration 180 --segment-duration 60 \
> --resample-rate 1024 --f-min 50 --f-max 250 --ifo-one H1 \
> --ifo-two H2 --channel-one LSC-AS_Q --channel-two LSC-AS_Q \
> --frame-cache-one H1.cache --frame-cache-two H2.cache \
> --calibration-cache-one H1-CAL-V02-751651244-757699245.cache \
> --calibration-cache-two H2-CAL-V02-751651244-757699245.cache \
> --calibration-offset 0 --hann-duration 1 --cc-spectra
```

Author

Adam Mercer, Tania Regimbau
5.5 Program `lalapps_stopp`

Name

`lalapps_stopp` — Stochastic Post Processing.

Synopsis

```
  lalapps_stopp <options> <xml files>
  --help
  --version
  --verbose
  --cat-only
  --analyse-only
  --text
  --output <FILE>
```

Description

`lalapps_stopp` performs post processing upon output from `lalapps_stochastic`.

Options

```
--help
  Display usage information

--version
  Display version information

--verbose
  Verbose mode

--cat-only
  Only cat XML files together

--analyse-only
  Only combine statistics

--text
  Output file as text

--output <FILE>
  write output data to <FILE>
```

Example

`lalapps_stopp` is generally run as part of a DAG, as created by the pipeline generation script `lalapps_stochastic.pipe`, however an example usage can be seen below.

```
> lalapps_stopp --output S3-H1L1-STOCHASTIC.xml \ 
>    H1L1-STOCHASTIC-753601044-753601242.xml \ 
>    H1L1-STOCHASTIC-753620042-753620352.xml \ 
>    H1L1-STOCHASTIC-753638864-753639462.xml \ 
>    H1L1-STOCHASTIC-753785374-753785707.xml \ 
>    H1L1-STOCHASTIC-753791744-753792342.xml
```

Author

Adam Mercer
Bibliography


KEY: Anderson:2000yy


KEY: Einstein:1905


KEY: LAL
ANNOTATION: Manual for the LSC Algorithm Library (LAL)
URL [http://www.lsc-group.phys.uwm.edu/lal](http://www.lsc-group.phys.uwm.edu/lal)


KEY: Press:1992
# Index

<table>
<thead>
<tr>
<th>ALLDBG</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank_status</td>
<td>5</td>
</tr>
<tr>
<td>clear_status</td>
<td>5</td>
</tr>
</tbody>
</table>

## Debug Level

- ALLDBG | 3 |
- ERROR | 3 |
- INFO | 3 |
- lalDebugLevel | 3 |
- MEMDBG | 3 |
- MEMINFO | 3 |
- MSGVL1 | 3 |
- MSGVL2 | 3 |
- MSGLVL3 | 3 |
- NDEBUG | 3 |
- TRACE | 3 |
- WARNING | 3 |

## Environment

- LAL_DEBUG_LEVEL | 3 |
- ERROR | 8 |
- LAL_ERR_ABORT | 8 |
- LAL_ERR_DEFAULT | 8 |
- LAL_ERR_EXIT | 8 |
- LAL_ERR_RTRN | 8 |

## Function

- clear_status | 5 |
- lal_errhandler | 8 |
- set_debug_level | 3 |

## hello.c

- 10 |

## INFO

- 5 |

## inspiral (module)

- 27 | 32 |

## DataFindJob (class)

- 27 |
- _init_ (method) | 27 |

## DataFindNode (class)

- 27 | 28 |
- _init_ (method) | 28 |
- get_output (method) | 28 |
- set_end (method) | 28 |
- set_ifo (method) | 28 |
- set_start (method) | 28 |

## IncaJob (class)

- 28 |
- _init_ (method) | 28 |

## IncaNode (class)

- 28 | 29 |
- _init_ (method) | 29 |
- get_ifo_a (method) | 29 |
- get_ifo_b (method) | 29 |
- get_output (method) | 29 |
- set_ifo_a (method) | 29 |
- set_ifo_b (method) | 29 |

## InspiralError (class)

- 29 |
- _init_ (method) | 29 |

## InspiralJob (class)

- 29 | 30 |
- _init_ (method) | 30 |

## InspiralNode (class)

- 30 |
- _init_ (method) | 30 |
- get_output (method) | 30 |

## TrigToTmpltJob (class)

- 31 | 32 |
- _init_ (method) | 32 |

## TrigToTmpltNode (class)

- 32 |
- _init_ (method) | 32 |

## inspiral_hipe.in

- 49 |

## inspiral_pipeline.py

- 46 |

## LAL_CALL

- 8 |

## LAL_DEBUG_LEVEL

- 3 |

## LAL_ERR_ABORT

- 8 |

## LAL_ERR_DEFAULT

- 8 |

## LAL_ERR_EXIT

- 8 |

## LAL_ERR_RTRN

- 8 |

## lal_errhandler

- 8 |

## lal_errhandler_t

- 8 |

## lalapps_animate

- 15 |

## lalapps_bbhjun

- 96 |

## lalapps_coherent_inspiral

- 70 |

## lalapps_coinext

- 108 |

## lalapps_hello

- 13 |

## lalapps_inca

- 73 |

## lalapps_inspfrunj

- 99 |

## lalapps_inspinj

- 90 |
Index

lalapps_inspiral, 63
lalapps_inspmultiawg, 103
lalapps_cola_predfcn, 114
lalapps_ring, 110
lalapps_sire, 86
lalapps_splitbank, 106
lalapps_stochastic, 121
lalapps_stochastic_pipe, 118
lalapps_stopp, 125
lalapps_thinca, 79
lalapps_tmpltbank, 58
lalapps_trigbank, 84
lalDebugLevel, 3

Macro

LAL_CALL, 8
PRINT_VERSION, 7
RCSID, 6
MEMDBG, 3
MEMINFO, 3
MSGLVL1, 3
MSGLVL2, 3
MSGLVL3, 3
NDEBUG, 3

pipeline (module), 18–26
AnalysisChunk (class), 18
__init__ (method), 18
__len__ (method), 18
__repr__ (method), 18
dur (method), 18
end (method), 18
start (method), 18
AnalysisJob (class), 18–19
__init__ (method), 18
calibration (method), 18
channel (method), 19
get_config (method), 19
AnalysisNode (class), 19–20
__init__ (method), 19
gt_end (method), 19
gt_ifo (method), 19
gt_input (method), 19
gt_output (method), 19
gt_start (method), 19
set_cache (method), 19
set_end (method), 19
set_ifo (method), 19
set_input (method), 19
set_output (method), 20
set_start (method), 20
CondorDAG (class), 20
__init__ (method), 20
add_node (method), 20
set_dag_file (method), 20
write_dag (method), 20
write_sub_files (method), 20
CondorDAGError (class), 20
CondorDAGJob (class), 20–21
__init__ (method), 21
add_var_arg (method), 21
add_var_opt (method), 21
CondorDAGNode (class), 21–22
__init__ (method), 21
__repr__ (method), 21
add_parent (method), 21
add_var_arg (method), 21
add_var_opt (method), 21
job (method), 21
set_log_file (method), 22
set_name (method), 22
set_retry (method), 22
write_job (method), 22
write_parents (method), 22
write_vars (method), 22
CondorDAGNodeError (class), 22
CondorError (class), 22
__init__ (method), 23
CondorJob (class), 22–24
__init__ (method), 23
add_arg (method), 23
add_condor_cmd (method), 23
add_ini_opts (method), 23
add_opt (method), 23
gt_stderr_file (method), 23
gt_stdout_file (method), 23
gt_sub_file (method), 23
set_log_file (method), 23
set_notification (method), 23
set_stderr_file (method), 23
set_stdout_file (method), 23
set_sub_file (method), 23
write_sub_file (method), 23
CondorJobError (class), 24
CondorSubmitError (class), 24
s2play (function), 18
ScienceData (class), 24
__getitem__ (method), 24
__init__ (method), 24
__len__ (method), 24
__repr__ (method), 24
make_chunks (method), 24
read (method), 24
ScienceSegment (class), 24–25
__getitem__ (method), 25
__init__ (method), 25
__len__ (method), 25
__repr__ (method), 25
add_chunk (method), 25
dur (method), 25
end (method), 25
id (method), 25
make_chunks (method), 25
start (method), 25
unused (method), 25
SegmentError (class), 25
__init__ (method), 26
power (module), 33
BurcaJob (class), 33
__init__ (method), 33
DataFindJob (class), 33
__init__ (method), 34
DataFindNode (class), 34
__init__ (method), 34
get_output (method), 34
set_end (method), 34
set_ifo (method), 34
set_start (method), 34
PowerError (class), 34
__init__ (method), 35
PowerJob (class), 35
__init__ (method), 35
PowerNode (class), 35
__init__ (method), 35
get_output (method), 35
PRINT_VERSION, 7
Program
inspiral_hipe.in, 49
inspiral_pipeline.py, 46
lalapps_animate, 15
lalapps_bbhinj, 96
lalapps_coherent_inspiral, 70
lalapps_coinext, 108
lalapps_hello, 13
lalapps_inca, 73
lalapps_inspfrinj, 99
lalapps_inspinj, 90
lalapps_inspiral, 63
lalapps_inspmulti, 103
lalapps_opalredfcn, 114
lalapps_ring, 110
lalapps_sire, 86
lalapps_splitbank, 106
lalapps_stochastic, 121
lalapps_stochastic_pipe, 118
lalapps_stopp, 125
lalapps_thinca, 79
lalapps_tmpltbank, 58
lalapps_trigbank, 84
set_debug_level, 3
Source
hello.c, 10
stochastic (module), 37
LSCDataFindJob (class), 37
__init__ (method), 37
get_cache_dir (method), 37
LSCDataFindNode (class), 38
__init__ (method), 38
get_output (method), 38
set_end (method), 38
set_observatory (method), 38
set_start (method), 38
set_type (method), 39
StochasticError (class), 39
__init__ (method), 39
StochasticJob (class), 40
__init__ (method), 40
StochasticNode (class), 40
__init__ (method), 40
get_output (method), 40
set_cache_one (method), 40
set_cache_two (method), 40
set_calibration_one (method), 41
set_calibration_two (method), 41
set_f_max (method), 41
set_f_min (method), 41
set_f_ref (method), 41
set_ifo_one (method), 41
set_ifo_two (method), 41
set_output_dir (method), 42
set_user_tag (method), 42
StoppJob (class), 42
__init__ (method), 42
StoppNode (class), 43
__init__ (method), 43
version (function), 37
TRACE, 3
Type
lal_errhandler_t, 8
Variable
blank_status, 5
lalDebugLevel, 5
rcsid, 6
vrblvl, 8
vrblvl, 8
WARNING, 5