GW Analysis Tools

Generated by Doxygen 1.8.16

Chapter 1

Gravitational Waves Analysis Tools

A suite of analysis tools useful for gravitational wave science. All code is written in C++, with some of the interface classes wrapped in Cython to allow for python-access.

1.1 Compatibility

Known to work with gcc/g++-7

Known to work with gcc/g++-9

Need nvcc – known to work with v9.1 of CUDA

1.2 Required Software

Required non-standard C libraries: FFTW3 ADOL-C – (must be compiled with OpenMP option) GSL CUDA

Required non-standard Python packages: Cython

Required non-standard packages for documentation: Doxygen

1.3 Current Development

NOTE: currently using static parameters to share data between threads for mcmc_gw.cpp. This could cause issues when running multiple samplers at the same time. Investigating further.

To do:

Change MCMC_MH to use the more general threadPool class instead of a custom threadpool, incorporate job class and comparator

1.4 Installation

For proper compilation, update or create the enviornment variables CPATH, LIBRARY_PATH,and LD_LIBRARY↩ _PATH, which should point to header files and lib files, respectively. Specifically, these variables should point to the above libraries.

Also, the PYTHONPATH environment variables must point to /gw_analysis_tools_py/src because I can't figure how to get this shit to work.

In the root directory of the project, run 'make' to compile source files, create the library file and create the cython modules, and create the documentation.

To just create C++/C files, run 'make c'.

Run 'make test' to build a test program that will create an executable.

1.5 Supported Functionality

1.5.1 Waveform Generation

IMRPhenomD, IMRPhenomPv2

1.5.2 Modified Gravity

ppE_IMRPhenomD_Inspiral ppE_IMRPhenomD_IMR ppE_IMRPhenomPv2_Inspiral ppE_IMRPhenomPv2_IMR

1.5.3 Fisher Analysis

utilizes the above waveform templates

1.5.4 MCMC Routines

Has a generic MCMC sampler, MCMC_MH, that utilizes gaussian steps, differential evolution steps, and Fisher informed steps. Includes wrapping MCMC_MH_GW for GW specific sampling, currently only for one detector.

Includes log likelihood caclulation for implementation in other samplers.

1.6 Usage

1.6.1 Environment variables

The environment variable PYTHONPATH should include the directory $(PROJECT_DIR)

1.6.2 Include

To include header files, use -I$(PROJECT_DIRECTORY)/include

1.6.3 Link

To link object files, use -L$(PROJECT_DIRECTORY)/lib -lgwat (the -L command is un-needed if you add /lib to the environment variable CPATH)

For dynamic linking, the following environment variables for Linux (MacOs) should be updated to include /lib – LD_LIBRARY_PATH (DYLD_LIBRARY_PATH)

For Cuda code: use -lcuda -lcudart

For Cuda, may need to link to /usr/local/cuda/lib64/ (or wherever this library is on your machine)

1.6.4 Python Importable Code

Two modules currently available:

1.6.4.1 gw_analysis_tools_py.mcmc_routines_ext

mcmc_routines_ext.pyx wraps the log_likelihood functions in mcmc_routines.cpp

1.6.4.2 gw_analysis_tools_py.waveform_generator_ext

waveform_generator_ext.pyx wraps the fourier_waveform function in waveform_generator.cpp

Also contains the SNR calculation function

1.6.4.3 Custom Waveforms

If adding waveforms and to have full accesibility:

Create class, using other waveforms as template – need interface to create full waveform (plus,cross polarization), and amplitude/phase

Add the option as a waveform to waveform_generation.cpp, including the header file at the top of the waveform_↩ generation.cpp file

Add option to check_mod in util.h if applicable.

For Fishers and MCMC – write the class as a template with double and adouble types for all variables. Add the option to unpack_parameters, repack_parameters, and repack_non_parameters. Add 'MCMC_" if using MCMC-specific version separate from the fisher version.

Author

Scott Perkins

Contact: scottep3@illinois.edu

Chapter 2

gw_analysis_tools

A suite of tools useful for doing statistical studies on gravitational wave science, including routines useful in MC↩ MC studies, wave template generation, Fisher analysis, etc. Written in C++ and wrapped in Cython for access in Python.

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

alpha_coeffs< T > . . . ??
Comparator . . . ??
comparator_ac_fft . . . ??
comparator_ac_serial . . . ??
Comparatorswap . . . ??
default_comp< jobtype > . . . ??
epsilon_coeffs< T > . . . ??
fftw_outline . . . ??
gen_params_base< T > . . . ??
gen_params_base< double > . . . ??
    gen_params . . . ??
GPUplan . . . ??
gsl_subroutine . . . ??
IMRPhenomD< T > . . . ??
    gIMRPhenomD< T > . . . ??
    IMRPhenomPv2< T > . . . ??
        ppE_IMRPhenomPv2_Inspiral< T > . . . ??
            ppE_IMRPhenomPv2_IMR< T > . . . ??
    ppE_IMRPhenomD_Inspiral< T > . . . ??
        dCS_IMRPhenomD< T > . . . ??
        dCS_IMRPhenomD_log< T > . . . ??
        EdGB_IMRPhenomD< T > . . . ??
        EdGB_IMRPhenomD_log< T > . . . ??
        ppE_IMRPhenomD_IMR< T > . . . ??
lambda_parameters< T > . . . ??
mcr_job . . . ??
mcr_sampler . . . ??
sampler . . . ??
source_parameters< T > . . . ??
sph_harm< T > . . . ??
threaded_ac_jobs_fft . . . ??
threaded_ac_jobs_serial . . . ??
threadPool< jobtype, comparator > . . . ??
ThreadPool . . . ??
useful_powers< T > . . . ??

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

alpha_coeffs< T > . . . ??
Comparator
Class to facilitate the comparing of chains for priority . . . ??
comparator_ac_fft
Comparator to sort ac-jobs . . . ??
comparator_ac_serial
Comparator to sort ac-jobs . . . ??
Comparatorswap . . . ??
dCS_IMRPhenomD< T > . . . ??
dCS_IMRPhenomD_log< T > . . . ??
default_comp< jobtype >
Default comparator for priority_queue in threadPool – no comparison . . . ??
EdGB_IMRPhenomD< T > . . . ??
EdGB_IMRPhenomD_log< T > . . . ??
epsilon_coeffs< T > . . . ??
fftw_outline . . . ??
gen_params
Convience wrapper for the gen_params_base class . . . ??
gen_params_base< T > . . . ??
gIMRPhenomD< T > . . . ??
GPUplan . . . ??
gsl_subroutine . . . ??
IMRPhenomD< T > . . . ??
IMRPhenomPv2< T > . . . ??
lambda_parameters< T > . . . ??
mcr_job . . . ??
mcr_sampler . . . ??
ppE_IMRPhenomD_IMR< T > . . . ??
ppE_IMRPhenomD_Inspiral< T > . . . ??
ppE_IMRPhenomPv2_IMR< T > . . . ??
ppE_IMRPhenomPv2_Inspiral< T > . . . ??
sampler . . . ??
source_parameters< T > . . . ??
sph_harm< T > . . . ??
threaded_ac_jobs_fft
Class to contain spectral method jobs . . . ??

threaded_ac_jobs_serial
Class to contain serial method jobs . . . . . ??
threadPool< jobtype, comparator >
Class for creating a pool of threads to asynchronously distribute work . . . . . ??
ThreadPool . . . . . ??
useful_powers< T >
To speed up calculations within the for loops, we pre-calculate reoccuring powers of M∗F and Pi, since the pow() function is prohibatively slow . . . . . ??

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

include/gwat/autocorrelation.h . . . ??
include/gwat/autocorrelation_cuda.h . . . ??
include/gwat/autocorrelation_cuda.hu . . . ??
include/gwat/**D_Z_Config.h** . . . ??
include/gwat/detector_util.h . . . ??
include/gwat/fisher.h . . . ??
include/gwat/gIMRPhenomD.h . . . ??
include/gwat/**GWATConfig.h** . . . ??
include/gwat/IMRPhenomD.h . . . ??
include/gwat/IMRPhenomP.h . . . ??
include/gwat/mc_reject.h . . . ??
include/gwat/mcmc_gw.h . . . ??
include/gwat/mcmc_sampler.h . . . ??
include/gwat/mcmc_sampler_internals.h . . . ??
include/gwat/pn_waveform_util.h . . . ??
include/gwat/ppE_IMRPhenomD.h . . . ??
include/gwat/ppE_IMRPhenomP.h . . . ??
include/gwat/QNM_data.h . . . ??
include/gwat/threadPool.h . . . ??
include/gwat/util.h . . . ??
include/gwat/waveform_generator.h . . . ??
include/gwat/waveform_generator_C.h . . . ??
include/gwat/waveform_util.h . . . ??
src/autocorrelation.cpp . . . ??
src/autocorrelation_cuda.cu . . . ??
src/detector_util.cpp . . . ??
src/gIMRPhenomD.cpp . . . ??
src/IMRPhenomD.cpp . . . ??
src/IMRPhenomP.cpp . . . ??
src/mc_reject.cpp . . . ??
src/mcmc_gw.cpp . . . ??
src/mcmc_sampler.cpp . . . ??
src/mcmc_sampler_internals.cpp . . . ??
src/pn_waveform_util.cpp . . . ??
src/ppE_IMRPhenomD.cpp . . . ??

src/ppE_IMRPhenomP.cpp . . . . . ??
src/util.cpp . . . . . ??
src/waveform_generator.cpp . . . . . ??
src/waveform_util.cpp . . . . . ??

Chapter 6

Class Documentation

6.1 alpha_coeffs< T > Struct Template Reference

Public Attributes

- T **coeff1**
- T **coeff2**
- T **coeff3**
- T **coeff4**
- T **coeff5**

The documentation for this struct was generated from the following file:

- include/gwat/IMRPhenomP.h

6.2 Comparator Class Reference

Class to facilitate the comparing of chains for priority.

Public Member Functions

- bool **operator()** (int i, int j)

6.2.1 Detailed Description

Class to facilitate the comparing of chains for priority.

3 levels of priority: 0 (high) 1 (default) 2 (low)

The documentation for this class was generated from the following file:

- src/mcmc_sampler.cpp

6.3 comparator_ac_fft Class Reference

comparator to sort ac-jobs

`#include <autocorrelation.h>`

Public Member Functions

- bool **operator()** (threaded_ac_jobs_fft t, threaded_ac_jobs_fft k)

6.3.1 Detailed Description

comparator to sort ac-jobs

Starts with the longest jobs, then works down the list

The documentation for this class was generated from the following file:

- include/gwat/autocorrelation.h

6.4 comparator_ac_serial Class Reference

comparator to sort ac-jobs

`#include <autocorrelation.h>`

Public Member Functions

- bool **operator()** (threaded_ac_jobs_serial t, threaded_ac_jobs_serial k)

6.4.1 Detailed Description

comparator to sort ac-jobs

Starts with the longest jobs, then works down the list

The documentation for this class was generated from the following file:

- include/gwat/autocorrelation.h

6.5 Comparatorswap Class Reference

Public Member Functions

- bool **operator()** (int i, int j)

The documentation for this class was generated from the following file:

- src/mcmc_sampler.cpp

6.6 dCS_IMRPhenomD< T > Class Template Reference

Inheritance diagram for dCS_IMRPhenomD< T >:

Collaboration diagram for dCS_IMRPhenomD< T >:

Public Member Functions

- virtual int construct_waveform (T *frequencies, int length, std::complex< T > *waveform, source_parameters< T > *params)

  Constructs the waveform as outlined by.
- virtual T **dCS_phase_mod** (source_parameters< T > *param)
- virtual T **dCS_phase_factor** (source_parameters< T > *param)
- virtual int construct_amplitude (T *frequencies, int length, T *amplitude, source_parameters< T > *params)

  Constructs the Amplitude as outlined by IMRPhenomD.
- virtual int construct_phase (T *frequencies, int length, T *phase, source_parameters< T > *params)

  Constructs the Phase as outlined by IMRPhenomD.

6.6.1 Member Function Documentation

6.6.1.1 construct_amplitude()

```
template<class T >
int dCS_IMRPhenomD< T >::construct_amplitude (
            T * frequencies,
            int length,
            T * amplitude,
            source_parameters< T > * params )  [virtual]
```

Constructs the Amplitude as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output amplitude, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaulated at
length	integer length of the input array of frequencies and the output array
amplitude	output T array for the amplitude
params	Structure of source parameters to be initilized before computation

Reimplemented from IMRPhenomD< T >.

6.6.1.2 construct_phase()

```
template<class T >
int dCS_IMRPhenomD< T >::construct_phase (
            T * frequencies,
            int length,
            T * phase,
            source_parameters< T > * params )  [virtual]
```

Constructs the Phase as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output phase, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the input and output arrays
phase	output T array for the phasee
params	structure of source parameters to be calculated before computation

Reimplemented from IMRPhenomD< T >.

6.6.1.3 construct_waveform()

```
template<class T >
int dCS_IMRPhenomD< T >::construct_waveform (
            T * frequencies,
            int length,
            std::complex< T > * waveform,
            source_parameters< T > * params )  [virtual]
```

Constructs the waveform as outlined by.

arguments: array of frequencies, length of that array, a complex array for the output waveform, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the array of frequencies and the waveform
waveform	complex T array for the waveform to be output

Reimplemented from IMRPhenomD< T >.

The documentation for this class was generated from the following files:

- include/gwat/ppE_IMRPhenomD.h
- src/ppE_IMRPhenomD.cpp

6.7 dCS_IMRPhenomD_log< T > Class Template Reference

Inheritance diagram for dCS_IMRPhenomD_log< T >:

Collaboration diagram for dCS_IMRPhenomD_log< T >:

Public Member Functions

- virtual int construct_waveform (T *frequencies, int length, std::complex< T > *waveform, source_parameters< T > *params)

  Constructs the waveform as outlined by.
- virtual T **dCS_phase_mod** (source_parameters< T > *param)
- virtual T **dCS_phase_factor** (source_parameters< T > *param)
- virtual int construct_amplitude (T *frequencies, int length, T *amplitude, source_parameters< T > *params)

  Constructs the Amplitude as outlined by IMRPhenomD.
- virtual int construct_phase (T *frequencies, int length, T *phase, source_parameters< T > *params)

  Constructs the Phase as outlined by IMRPhenomD.

6.7.1 Member Function Documentation

6.7.1.1 construct_amplitude()

```
template<class T >
int dCS_IMRPhenomD_log< T >::construct_amplitude (
            T * frequencies,
            int length,
            T * amplitude,
            source_parameters< T > * params )  [virtual]
```

Constructs the Amplitude as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output amplitude, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaulated at
length	integer length of the input array of frequencies and the output array
amplitude	output T array for the amplitude
params	Structure of source parameters to be initilized before computation

Reimplemented from IMRPhenomD< T >.

6.7.1.2 construct_phase()

```
template<class T >
int dCS_IMRPhenomD_log< T >::construct_phase (
            T * frequencies,
            int length,
            T * phase,
            source_parameters< T > * params )  [virtual]
```

Constructs the Phase as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output phase, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the input and output arrays
phase	output T array for the phasee
params	structure of source parameters to be calculated before computation

Reimplemented from IMRPhenomD< T >.

6.7.1.3 construct_waveform()

```
template<class T >
int dCS_IMRPhenomD_log< T >::construct_waveform (
            T * frequencies,
            int length,
            std::complex< T > * waveform,
            source_parameters< T > * params )  [virtual]
```

Constructs the waveform as outlined by.

arguments: array of frequencies, length of that array, a complex array for the output waveform, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the array of frequencies and the waveform
waveform	complex T array for the waveform to be output

Reimplemented from IMRPhenomD< T >.

The documentation for this class was generated from the following files:

- include/gwat/ppE_IMRPhenomD.h
- src/ppE_IMRPhenomD.cpp

6.8 default_comp< jobtype > Class Template Reference

Default comparator for priority_queue in threadPool – no comparison.

`#include <threadPool.h>`

Public Member Functions

- bool **operator()** (jobtype j, jobtype k)

6.8.1 Detailed Description

template<class jobtype>
class default_comp< jobtype >

Default comparator for priority_queue in threadPool – no comparison.

First in first out, not sorting

The documentation for this class was generated from the following file:

- include/gwat/threadPool.h

6.9 EdGB_IMRPhenomD< T > Class Template Reference

Inheritance diagram for EdGB_IMRPhenomD< T >:

Collaboration diagram for EdGB_IMRPhenomD< T >:

Public Member Functions

- virtual int construct_waveform (T *frequencies, int length, std::complex< T > *waveform, source_parameters< T > *params)

  Constructs the waveform as outlined by.
- virtual T **EdGB_phase_mod** (source_parameters< T > *param)
- virtual T **EdGB_phase_factor** (source_parameters< T > *param)
- virtual int construct_amplitude (T *frequencies, int length, T *amplitude, source_parameters< T > *params)

  Constructs the Amplitude as outlined by IMRPhenomD.
- virtual int construct_phase (T *frequencies, int length, T *phase, source_parameters< T > *params)

  Constructs the Phase as outlined by IMRPhenomD.

6.9.1 Member Function Documentation

6.9.1.1 construct_amplitude()

```
template<class T >
int EdGB_IMRPhenomD< T >::construct_amplitude (
            T * frequencies,
            int length,
            T * amplitude,
            source_parameters< T > * params )  [virtual]
```

Constructs the Amplitude as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output amplitude, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaulated at
length	integer length of the input array of frequencies and the output array
amplitude	output T array for the amplitude
params	Structure of source parameters to be initilized before computation

Reimplemented from IMRPhenomD< T >.

6.9.1.2 construct_phase()

```
template<class T >
int EdGB_IMRPhenomD< T >::construct_phase (
            T * frequencies,
            int length,
            T * phase,
            source_parameters< T > * params )  [virtual]
```

Constructs the Phase as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output phase, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the input and output arrays
phase	output T array for the phasee
params	structure of source parameters to be calculated before computation

Reimplemented from IMRPhenomD< T >.

6.9.1.3 construct_waveform()

```
template<class T >
int EdGB_IMRPhenomD< T >::construct_waveform (
            T * frequencies,
            int length,
            std::complex< T > * waveform,
            source_parameters< T > * params )  [virtual]
```

Constructs the waveform as outlined by.

arguments: array of frequencies, length of that array, a complex array for the output waveform, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the array of frequencies and the waveform
waveform	complex T array for the waveform to be output

Reimplemented from IMRPhenomD< T >.

The documentation for this class was generated from the following files:

- include/gwat/ppE_IMRPhenomD.h
- src/ppE_IMRPhenomD.cpp

6.10 EdGB_IMRPhenomD_log< T > Class Template Reference

Inheritance diagram for EdGB_IMRPhenomD_log< T >:

Collaboration diagram for EdGB_IMRPhenomD_log< T >:

Public Member Functions

- virtual int construct_waveform (T *frequencies, int length, std::complex< T > *waveform, source_parameters< T > *params)

  Constructs the waveform as outlined by.
- virtual T **EdGB_phase_mod** (source_parameters< T > *param)
- virtual T **EdGB_phase_factor** (source_parameters< T > *param)
- virtual int construct_amplitude (T *frequencies, int length, T *amplitude, source_parameters< T > *params)

  Constructs the Amplitude as outlined by IMRPhenomD.
- virtual int construct_phase (T *frequencies, int length, T *phase, source_parameters< T > *params)

  Constructs the Phase as outlined by IMRPhenomD.

6.10.1 Member Function Documentation

6.10.1.1 construct_amplitude()

```
template<class T >
int EdGB_IMRPhenomD_log< T >::construct_amplitude (
            T * frequencies,
            int length,
            T * amplitude,
            source_parameters< T > * params )  [virtual]
```

Constructs the Amplitude as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output amplitude, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaulated at
length	integer length of the input array of frequencies and the output array
amplitude	output T array for the amplitude
params	Structure of source parameters to be initilized before computation

Reimplemented from IMRPhenomD< T >.

6.10.1.2 construct_phase()

```
template<class T >
int EdGB_IMRPhenomD_log< T >::construct_phase (
            T * frequencies,
            int length,
            T * phase,
            source_parameters< T > * params )  [virtual]
```

Constructs the Phase as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output phase, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the input and output arrays
phase	output T array for the phasee
params	structure of source parameters to be calculated before computation

Reimplemented from IMRPhenomD< T >.

6.10.1.3 construct_waveform()

```
template<class T >
int EdGB_IMRPhenomD_log< T >::construct_waveform (
            T * frequencies,
            int length,
            std::complex< T > * waveform,
            source_parameters< T > * params )  [virtual]
```

Constructs the waveform as outlined by.

arguments: array of frequencies, length of that array, a complex array for the output waveform, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the array of frequencies and the waveform
waveform	complex T array for the waveform to be output

Reimplemented from IMRPhenomD< T >.

The documentation for this class was generated from the following files:

- include/gwat/ppE_IMRPhenomD.h
- src/ppE_IMRPhenomD.cpp

6.11 epsilon_coeffs< T > Struct Template Reference

Public Attributes

- T **coeff1**
- T **coeff2**

- T **coeff3**
- T **coeff4**
- T **coeff5**

The documentation for this struct was generated from the following file:

- include/gwat/IMRPhenomP.h

6.12 fftw_outline Struct Reference

Public Attributes

- fftw_complex * **in**
- fftw_complex * **out**
- fftw_plan **p**

The documentation for this struct was generated from the following file:

- include/gwat/util.h

6.13 gen_params Class Reference

convience wrapper for the gen_params_base class

`#include <util.h>`

Inheritance diagram for gen_params:

Collaboration diagram for gen_params:

Additional Inherited Members

6.13.1 Detailed Description

convience wrapper for the gen_params_base class

If using the code in the intended way, this is all the user should ever have to use. Just allows the user to drop the template parameter

Also implemented for backwards compatibility with previous versions of the code

The documentation for this class was generated from the following file:

- include/gwat/util.h

6.14 gen_params_base< T > Class Template Reference

Collaboration diagram for gen_params_base< T >:

Public Attributes

- std::string **cosmology** ="PLANCK15"
- T mass1
- T mass2
- T Luminosity_Distance
- T spin1 [3]
- T spin2 [3]
- T tc =0
- T **psi** =0
- T incl_angle
- bool equatorial_orientation =false
- T theta_l
- T phi_l
- bool horizon_coord =false
- T theta
- T phi
- T RA
- T DEC
- double gmst
- bool NSflag1 =false
- bool **NSflag2** =false
- T f_ref =0
- bool shift_time = true
- bool shift_phase = true
- T **phiRef** =0
- T phic =0
- bool **sky_average** =false
- T thetaJN = -10
- T **alpha0** = 0
- T **chip** = 0
- T **phip** = -1
- bool **precess_reduced_flag** =false
- T **LISA_alpha0** =0
- T **LISA_phi0** =0
- T LISA_thetal
- T **theta_j_ecl**
- T LISA_phil
- T **phi_j_ecl**
- int **Nmod_beta** =0
- int **Nmod_alpha** =0
- int **Nmod_sigma** =0
- int **Nmod_phi** =0
- int * **betai**
- int * **alphai**
- int * **sigmai**
- int * **phii**
- T * **delta_beta**
- T * **delta_alpha**

- T * **delta_sigma**
- T * **delta_phi**
- int * bppe
- T * betappe
- int Nmod =0
- T **chi1_l** = 0
- T **chi2_l** = 0
- T **phiJL** = 0
- T **thetaJL** = 0
- T **zeta_polariz** =0
- T **phi_aligned** = 0
- T **chil** = 0
- gsl_spline * **Z_DL_spline_ptr** =NULL
- gsl_interp_accel * **Z_DL_accel_ptr** =NULL

6.14.1 Member Data Documentation

6.14.1.1 betappe

```
template<class T>
T* gen_params_base< T >::betappe
```

ppE coefficient for the phase modification - vector for multiple modifications

6.14.1.2 bppe

```
template<class T>
int* gen_params_base< T >::bppe
```

ppE b parameter (power of the frequency) - vector for multiple modifications

6.14.1.3 DEC

```
template<class T>
T gen_params_base< T >::DEC
```

Equatorial coordinates of source DEC

6.14.1.4 equatorial_orientation

```
template<class T>
bool gen_params_base< T >::equatorial_orientation =false
```

boolean flag indicating equatorial orientation coordinates should be used

6.14.1.5 f_ref

```
template<class T>
T gen_params_base< T >::f_ref =0
```

Reference frequency for PhenomPv2

6.14.1.6 gmst

```
template<class T>
double gen_params_base< T >::gmst
```

Greenwich Mean Sidereal time (for detector orientation - start of data

6.14.1.7 horizon_coord

```
template<class T>
bool gen_params_base< T >::horizon_coord =false
```

Boolean flag indicating local, horizon coordinates should be used

6.14.1.8 incl_angle

```
template<class T>
T gen_params_base< T >::incl_angle
```

∗angle between angular momentum and the total momentum

6.14.1.9 LISA_phil

```
template<class T>
T gen_params_base< T >::LISA_phil
```

Azimuthal angle in ecliptic coordinates

6.14.1.10 LISA_thetal

```
template<class T>
T gen_params_base< T >::LISA_thetal
```

Polar angle in ecliptic coordinates

6.14.1.11 Luminosity_Distance

```
template<class T>
T gen_params_base< T >::Luminosity_Distance
```

Luminosity distance to the source

6.14.1.12 mass1

```
template<class T>
T gen_params_base< T >::mass1
```

mass of the larger body in Solar Masses

6.14.1.13 mass2

```
template<class T>
T gen_params_base< T >::mass2
```

mass of the smaller body in Solar Masses

6.14.1.14 Nmod

```
template<class T>
int gen_params_base< T >::Nmod =0
```

Number of phase modificatinos

6.14.1.15 NSflag1

```
template<class T>
bool gen_params_base< T >::NSflag1 =false
```

BOOL flag for early termination of NS binaries

6.14.1.16 phi

```
template<class T>
T gen_params_base< T >::phi
```

azimuthal angle in detector-centered coordinates

6.14.1.17 phi_l

```
template<class T>
T gen_params_base< T >::phi_l
```

Equatorial Spherical angles for the orbital angular momentum

6.14.1.18 phic

```
template<class T>
T gen_params_base< T >::phic =0
```

coalescence phase of the binary

6.14.1.19 RA

```
template<class T>
T gen_params_base< T >::RA
```

Equatorial coordinates of source RA

6.14.1.20 shift_phase

```
template<class T>
bool gen_params_base< T >::shift_phase = true
```

Shift time detemines if phic or phiRef is used

6.14.1.21 shift_time

```
template<class T>
bool gen_params_base< T >::shift_time = true
```

Shift time detemines if times are shifted so coalescence is more accurately

6.14.1.22 spin1

```
template<class T>
T gen_params_base< T >::spin1[3]
```

Spin vector of the larger mass [Sx,Sy,Sz]

6.14.1.23 spin2

```
template<class T>
T gen_params_base< T >::spin2[3]
```

Spin vector of the smaller mass [Sx,Sy,Sz]

6.14.1.24 tc

```
template<class T>
T gen_params_base< T >::tc =0
```

coalescence time of the binary

6.14.1.25 theta

```
template<class T>
T gen_params_base< T >::theta
```

Polar angle in detector-centered coordinates

6.14.1.26 theta_l

```
template<class T>
T gen_params_base< T >::theta_l
```

Equatorial Spherical angles for the orbital angular momentum

6.14.1.27 thetaJN

```
template<class T>
T gen_params_base< T >::thetaJN = -10
```

thetaJ – optional domain is [0,M_PI]

The documentation for this class was generated from the following file:

- include/gwat/util.h

6.15 gIMRPhenomD< T > Class Template Reference

Inheritance diagram for gIMRPhenomD< T >:

Collaboration diagram for gIMRPhenomD< T >:

Public Member Functions

- virtual T phase_ins (T f, source_parameters< T > *param, T *pn_coeff, lambda_parameters< T > *lambda, useful_powers< T > *pow)

  Calculates the inspiral phase for frequency f with precomputed powers of MF and PI for speed.
- virtual T Dphase_ins (T f, source_parameters< T > *params, T *pn_coeff, lambda_parameters< T > *lambda)

  Calculates the derivative of the inspiral phase for frequency f.
- virtual void assign_lambda_param (source_parameters< T > *source_param, lambda_parameters< T > *lambda)

  Wrapper for the Lambda parameter assignment that handles the looping.
- virtual void assign_static_pn_phase_coeff (source_parameters< T > *source_param, T *coeff)

  Calculates the static PN coeffecients for the phase - coeffecients 0,1,2,3,4,7.

6.15.1 Member Function Documentation

6.15.1.1 Dphase_ins()

```
template<class T >
T gIMRPhenomD< T >::Dphase_ins (
            T f,
            source_parameters< T > * param,
            T * pn_coeff,
            lambda_parameters< T > * lambda )  [virtual]
```

Calculates the derivative of the inspiral phase for frequency f.

For phase continuity and smoothness return a T

Reimplemented from IMRPhenomD< T >.

6.15.1.2 phase_ins()

```
template<class T >
T gIMRPhenomD< T >::phase_ins (
            T f,
            source_parameters< T > * param,
            T * pn_coeff,
            lambda_parameters< T > * lambda,
            useful_powers< T > * pow )  [virtual]
```

Calculates the inspiral phase for frequency f with precomputed powers of MF and PI for speed.

return a T

extra argument of precomputed powers of MF and pi, contained in the structure useful_powers<T>

Reimplemented from IMRPhenomD< T >.

The documentation for this class was generated from the following files:

- include/gwat/gIMRPhenomD.h
- src/gIMRPhenomD.cpp

6.16 GPUplan Struct Reference

Public Attributes

- int **device_id**
- double * **device_data**
- double * **host_data**
- int * **host_lag**
- int * **device_lag**
- int * **device_lags**
- int * **initial_lag**
- cudaStream_t **stream**

The documentation for this struct was generated from the following file:

- include/gwat/autocorrelation_cuda.hu

6.17 gsl_subroutine Struct Reference

Collaboration diagram for gsl_subroutine:

Public Attributes

- string **detector**
- string **generation_method**
- string **sensitivity_curve**
- gen_params * **gen_params_in**
- int **dim**
- int **id1**
- int **id2**
- int * **waveform_tapes**
- int * **time_tapes**
- int * **phase_tapes**
- double * **freq_boundaries**
- double * **grad_freqs**
- int **boundary_num**
- bool * **log_factors**

The documentation for this struct was generated from the following file:

- include/gwat/fisher.h

6.18 IMRPhenomD< T > Class Template Reference

Inheritance diagram for IMRPhenomD< T >:

Public Member Functions

- virtual void **fisher_calculation_sky_averaged** (double *frequency, int length, gen_params *parameters, double **amplitude_deriv, double **phase_deriv, double *amplitude, int *amp_tapes, int *phase_tapes)
- virtual void change_parameter_basis (T *old_param, T *new_param, bool sky_average)

  Convience method to change parameter basis between common Fisher parameters and the intrinsic parameters of IMRPhenomD.
- virtual void construct_amplitude_derivative (double *frequencies, int length, int dimension, double **amplitude_derivative, source_parameters< double > *input_params, int *tapes=NULL)

  Construct the derivative of the amplitude for a given source evaluated by the given frequency.
- virtual void construct_phase_derivative (double *frequencies, int length, int dimension, double **phase_derivative, source_parameters< double > *input_params, int *tapes=NULL)

  Construct the derivative of the phase for a given source evaluated by the given frequency.
- virtual void amplitude_tape (source_parameters< double > *input_params, int *tape)

  Creates the tapes for derivatives of the amplitude.
- virtual void phase_tape (source_parameters< double > *input_params, int *tape)

  Creates the tapes for derivatives of phase.
- virtual int construct_waveform (T *frequencies, int length, std::complex< T > *waveform, source_parameters< T > *params)

Constructs the waveform as outlined by.

- virtual std::complex< T > construct_waveform (T frequency, source_parameters< T > *params)

  overloaded method to evaluate the waveform for one frequency instead of an array
- virtual int construct_amplitude (T *frequencies, int length, T *amplitude, source_parameters< T > *params)

  Constructs the Amplitude as outlined by IMRPhenomD.
- virtual int construct_phase (T *frequencies, int length, T *phase, source_parameters< T > *params)

  Constructs the Phase as outlined by IMRPhenomD.
- virtual T build_amp (T f, lambda_parameters< T > *lambda, source_parameters< T > *params, useful_powers< T > *pows, T *amp_coeff, T *deltas)

  constructs the IMRPhenomD amplitude for frequency f
- virtual T build_phase (T f, lambda_parameters< T > *lambda, source_parameters< T > *params, useful_powers< T > *pows, T *phase_coeff)

  constructs the IMRPhenomD phase for frequency f
- virtual T assign_lambda_param_element (source_parameters< T > *source_param, int i)

  Calculate the lambda parameters from Khan et al for element i.
- virtual void assign_lambda_param (source_parameters< T > *source_param, lambda_parameters< T > *lambda)

  Wrapper for the Lambda parameter assignment that handles the looping.
- virtual void precalc_powers_ins (T f, T M, useful_powers< T > *Mf_pows)

  Pre-calculate powers of Mf, to speed up calculations for the inspiral waveform (both amplitude and phase.
- virtual void precalc_powers_PI (useful_powers< T > *PI_pows)

  Pre-calculate powers of pi, to speed up calculations for the inspiral phase.
- virtual void precalc_powers_ins_phase (T f, T M, useful_powers< T > *Mf_pows)

  Pre-calculate powers of Mf, to speed up calculations for the inspiral phase.
- virtual void precalc_powers_ins_amp (T f, T M, useful_powers< T > *Mf_pows)

  Pre-calculate powers of Mf, to speed up calculations for the inspiral amplitude.
- virtual void assign_pn_amplitude_coeff (source_parameters< T > *source_param, T *coeff)

  Calculates the static PN coeffecients for the amplitude.
- virtual void assign_static_pn_phase_coeff (source_parameters< T > *source_param, T *coeff)

  Calculates the static PN coeffecients for the phase - coeffecients 0,1,2,3,4,7.
- virtual void assign_nonstatic_pn_phase_coeff (source_parameters< T > *source_param, T *coeff, T f)

  Calculates the dynamic PN phase coefficients 5,6.
- virtual void assign_nonstatic_pn_phase_coeff_deriv (source_parameters< T > *source_param, T *Dcoeff, T f)

  Calculates the derivative of the dynamic PN phase coefficients 5,6.
- virtual void post_merger_variables (source_parameters< T > *source_param)

  Calculates the post-merger ringdown frequency and dampening frequency.
- virtual void **calc_fring** (source_parameters< T > *source_params)
- virtual void **calc_fdamp** (source_parameters< T > *source_params)
- virtual T **final_spin** (source_parameters< T > *params)
- virtual T FinalSpin0815_s (T eta, T s)
- virtual T FinalSpin0815 (T eta, T chi1, T chi2)
- virtual T EradRational0815_s (T eta, T s)
- virtual T EradRational0815 (T eta, T chi1, T chi2)
- virtual T fpeak (source_parameters< T > *params, lambda_parameters< T > *lambda)

  Solves for the peak frequency, where the waveform transitions from intermediate to merger-ringdown.
- virtual T amp_ins (T f, source_parameters< T > *param, T *pn_coeff, lambda_parameters< T > *lambda, useful_powers< T > *pow)

  Calculates the scaled inspiral amplitude A/A0 for frequency f with precomputed powers of MF and PI.
- virtual T Damp_ins (T f, source_parameters< T > *param, T *pn_coeff, lambda_parameters< T > *lambda)

  Calculates the derivative wrt frequency for the scaled inspiral amplitude A/A0 for frequency f.

- virtual T phase_ins (T f, source_parameters< T > *param, T *pn_coeff, lambda_parameters< T > *lambda, useful_powers< T > *pow)

  Calculates the inspiral phase for frequency f with precomputed powers of MF and PI for speed.
- virtual T Dphase_ins (T f, source_parameters< T > *param, T *pn_coeff, lambda_parameters< T > *lambda)

  Calculates the derivative of the inspiral phase for frequency f.
- virtual T amp_mr (T f, source_parameters< T > *param, lambda_parameters< T > *lambda)

  Calculates the scaled merger-ringdown amplitude A/A0 for frequency f.
- virtual T phase_mr (T f, source_parameters< T > *param, lambda_parameters< T > *lambda)

  Calculates the merger-ringdown phase for frequency f.
- virtual T Damp_mr (T f, source_parameters< T > *param, lambda_parameters< T > *lambda)

  Calculates the derivative wrt frequency for the scaled merger-ringdown amplitude A/A0 for frequency f.
- virtual T Dphase_mr (T f, source_parameters< T > *param, lambda_parameters< T > *lambda)

  Calculates the derivative of the merger-ringdown phase for frequency f.
- virtual T amp_int (T f, source_parameters< T > *param, lambda_parameters< T > *lambda, T *deltas)

  Calculates the scaled intermediate range amplitude A/A0 for frequency f.
- virtual T phase_int (T f, source_parameters< T > *param, lambda_parameters< T > *lambda)

  Calculates the intermediate phase for frequency f.
- virtual T Dphase_int (T f, source_parameters< T > *param, lambda_parameters< T > *lambda)

  Calculates the derivative of the intermediate phase for frequency f.
- virtual void phase_connection_coefficients (source_parameters< T > *param, lambda_parameters< T > *lambda, T *pn_coeffs)

  Calculates the phase connection coefficients alpha{0,1} and beta{0,1}.
- virtual T **calculate_beta1** (source_parameters< T > *param, lambda_parameters< T > *lambda, T *pn↩_coeffs)
- virtual T **calculate_beta0** (source_parameters< T > *param, lambda_parameters< T > *lambda, T *pn↩_coeffs)
- virtual T **calculate_alpha1** (source_parameters< T > *param, lambda_parameters< T > *lambda)
- virtual T **calculate_alpha0** (source_parameters< T > *param, lambda_parameters< T > *lambda)
- virtual void amp_connection_coeffs (source_parameters< T > *param, lambda_parameters< T > *lambda, T *pn_coeffs, T *coeffs)

  Solves for the connection coefficients to ensure the transition from inspiral to merger ringdown is continuous and smooth.
- virtual T calculate_delta_parameter_0 (T f1, T f2, T f3, T v1, T v2, T v3, T dd1, T dd3, T M)

  Calculates the delta_0 component.
- virtual T calculate_delta_parameter_1 (T f1, T f2, T f3, T v1, T v2, T v3, T dd1, T dd3, T M)

  Calculates the delta_1 component.
- virtual T calculate_delta_parameter_2 (T f1, T f2, T f3, T v1, T v2, T v3, T dd1, T dd3, T M)

  Calculates the delta_2 component.
- virtual T calculate_delta_parameter_3 (T f1, T f2, T f3, T v1, T v2, T v3, T dd1, T dd3, T M)

  Calculates the delta_3 component.
- virtual T calculate_delta_parameter_4 (T f1, T f2, T f3, T v1, T v2, T v3, T dd1, T dd3, T M)

  Calculates the delta_4 component.
- template<>
  void **calc_fring** (source_parameters< double > *source_param)
- template<>
  void **calc_fdamp** (source_parameters< double > *source_param)
- template<>
  void **calc_fring** (source_parameters< adouble > *source_param)
- template<>
  void **calc_fdamp** (source_parameters< adouble > *source_param)

6.18.1 Member Function Documentation

6.18.1.1 amp_ins()

```
template<class T >
T IMRPhenomD< T >::amp_ins (
            T f,
            source_parameters< T > * param,
            T * pn_coeff,
            lambda_parameters< T > * lambda,
            useful_powers< T > * pow )  [virtual]
```

Calculates the scaled inspiral amplitude A/A0 for frequency f with precomputed powers of MF and PI.

return a T

additional argument contains useful powers of MF and PI in structure userful_powers

6.18.1.2 amp_int()

```
template<class T >
T IMRPhenomD< T >::amp_int (
            T f,
            source_parameters< T > * param,
            lambda_parameters< T > * lambda,
            T * deltas )  [virtual]
```

Calculates the scaled intermediate range amplitude A/A0 for frequency f.

return a T

6.18.1.3 amp_mr()

```
template<class T >
T IMRPhenomD< T >::amp_mr (
            T f,
            source_parameters< T > * param,
            lambda_parameters< T > * lambda )  [virtual]
```

Calculates the scaled merger-ringdown amplitude A/A0 for frequency f.

return a T

6.18.1.4 amplitude_tape()

```
template<class T >
void IMRPhenomD< T >::amplitude_tape (
            source_parameters< double > * input_params,
            int * tape )  [virtual]
```

Creates the tapes for derivatives of the amplitude.

For efficiency in long runs of large sets of fishers, the tapes can be precomputed and reused

Parameters

input_params	source parameters structure of the desired source
tape	tape ids

Reimplemented in ppE_IMRPhenomD_IMR< T >, and ppE_IMRPhenomD_Inspiral< T >.

6.18.1.5 assign_nonstatic_pn_phase_coeff()

```
template<class T >
void IMRPhenomD< T >::assign_nonstatic_pn_phase_coeff (
            source_parameters< T > * source_param,
            T * coeff,
            T f )  [virtual]
```

Calculates the dynamic PN phase coefficients 5,6.

f is in Hz

6.18.1.6 assign_nonstatic_pn_phase_coeff_deriv()

```
template<class T >
void IMRPhenomD< T >::assign_nonstatic_pn_phase_coeff_deriv (
            source_parameters< T > * source_param,
            T * Dcoeff,
            T f )  [virtual]
```

Calculates the derivative of the dynamic PN phase coefficients 5,6.

f is in Hz

6.18.1.7 build_amp()

```
template<class T >
T IMRPhenomD< T >::build_amp (
            T f,
            lambda_parameters< T > * lambda,
            source_parameters< T > * params,
            useful_powers< T > * pows,
            T * amp_coeff,
            T * deltas )  [virtual]
```

constructs the IMRPhenomD amplitude for frequency f

arguments: numerical parameters from Khan et al lambda_parameters structure, source_parameters structure, useful_powers<T> structure, PN parameters for the inspiral portions of the waveform, and the delta parameters for the intermediate region, numerically solved for using the amp_connection_coeffs function

6.18.1.8 build_phase()

```
template<class T >
T IMRPhenomD< T >::build_phase (
            T f,
            lambda_parameters< T > * lambda,
            source_parameters< T > * params,
            useful_powers< T > * pows,
            T * phase_coeff )  [virtual]
```

constructs the IMRPhenomD phase for frequency f

arguments: numerical parameters from Khan et al lambda_parameters structure, source_parameters structure, useful_powers structure, PN parameters for the inspiral portions of the waveform

6.18.1.9 calculate_delta_parameter_0()

```
template<class T >
T IMRPhenomD< T >::calculate_delta_parameter_0 (
            T f1,
            T f2,
            T f3,
            T v1,
            T v2,
            T v3,
            T dd1,
            T dd3,
            T M )  [virtual]
```

Calculates the delta_0 component.

Solved in Mathematica and imported to C

6.18.1.10 calculate_delta_parameter_1()

```
template<class T >
T IMRPhenomD< T >::calculate_delta_parameter_1 (
            T f1,
            T f2,
            T f3,
            T v1,
            T v2,
            T v3,
            T dd1,
            T dd3,
            T M )  [virtual]
```

Calculates the delta_1 component.

Solved in Mathematica and imported to C

6.18.1.11 calculate_delta_parameter_2()

```
template<class T >
T IMRPhenomD< T >::calculate_delta_parameter_2 (
            T f1,
            T f2,
            T f3,
            T v1,
            T v2,
            T v3,
            T dd1,
            T dd3,
            T M )  [virtual]
```

Calculates the delta_2 component.

Solved in Mathematica and imported to C

6.18.1.12 calculate_delta_parameter_3()

```
template<class T >
T IMRPhenomD< T >::calculate_delta_parameter_3 (
            T f1,
            T f2,
            T f3,
            T v1,
            T v2,
            T v3,
            T dd1,
            T dd3,
            T M )  [virtual]
```

Calculates the delta_3 component.

Solved in Mathematica and imported to C

6.18.1.13 calculate_delta_parameter_4()

```
template<class T >
T IMRPhenomD< T >::calculate_delta_parameter_4 (
            T f1,
            T f2,
            T f3,
            T v1,
            T v2,
            T v3,
            T dd1,
            T dd3,
            T M )  [virtual]
```

Calculates the delta_4 component.

Solved in Mathematica and imported to C

6.18.1.14 change_parameter_basis()

```
template<class T >
void IMRPhenomD< T >::change_parameter_basis (
            T * old_param,
            T * new_param,
            bool sky_average )  [virtual]
```

Convience method to change parameter basis between common Fisher parameters and the intrinsic parameters of IMRPhenomD.

Takes input array of old parameters and ouputs array of transformed parameters

Parameters

old_param	array of old params, order {A0, tc, phic, chirpmass, eta, spin1, spin2}
new_param	output new array: order {m1,m2,DL, spin1,spin2,phic,tc}

6.18.1.15 construct_amplitude()

```
template<class T >
int IMRPhenomD< T >::construct_amplitude (
            T * frequencies,
            int length,
            T * amplitude,
            source_parameters< T > * params )  [virtual]
```

Constructs the Amplitude as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output amplitude, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaulated at
length	integer length of the input array of frequencies and the output array
amplitude	output T array for the amplitude
params	Structure of source parameters to be initilized before computation

Reimplemented in EdGB_IMRPhenomD< T >, EdGB_IMRPhenomD_log< T >, dCS_IMRPhenomD< T >, and dCS_IMRPhenomD_log< T >.

6.18.1.16 construct_amplitude_derivative()

```
template<class T >
void IMRPhenomD< T >::construct_amplitude_derivative (
```

```
            double * frequencies,
            int length,
            int dimension,
            double ** amplitude_derivative,
            source_parameters< double > * input_params,
            int * tapes = NULL )  [virtual]
```

Construct the derivative of the amplitude for a given source evaluated by the given frequency.

Order of output: dh/d \theta : \theta \el {A0,tc, phic, chirp mass, eta, symmetric spin, antisymmetric spin}

Parameters

frequencies	input array of frequency
length	length of the frequency array
amplitude_derivative	< dimension of the fisher output array for all the derivatives double[dimension][length]
input_params	Source parameters structure for the source
tapes	int array of tape ids, if NULL, these will be calculated

Reimplemented in ppE_IMRPhenomD_IMR< T >, and ppE_IMRPhenomD_Inspiral< T >.

6.18.1.17 construct_phase()

```
template<class T >
int IMRPhenomD< T >::construct_phase (
            T * frequencies,
            int length,
            T * phase,
            source_parameters< T > * params )  [virtual]
```

Constructs the Phase as outlined by IMRPhenomD.

arguments: array of frequencies, length of that array, T array for the output phase, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the input and output arrays
phase	output T array for the phasee
params	structure of source parameters to be calculated before computation

Reimplemented in EdGB_IMRPhenomD< T >, EdGB_IMRPhenomD_log< T >, dCS_IMRPhenomD< T >, and dCS_IMRPhenomD_log< T >.

6.18.1.18 construct_phase_derivative()

```
template<class T >
void IMRPhenomD< T >::construct_phase_derivative (
```

```
            double * frequencies,
            int length,
            int dimension,
            double ** phase_derivative,
            source_parameters< double > * input_params,
            int * tapes = NULL )  [virtual]
```

Construct the derivative of the phase for a given source evaluated by the given frequency.

Order of output: dh/d \theta : \theta \el {A0,tc, phic, chirp mass, eta, symmetric spin, antisymmetric spin}

Parameters

frequencies	input array of frequency
length	length of the frequency array
phase_derivative	< dimension of the fisher output array for all the derivatives double[dimension][length]
input_params	Source parameters structure for the source
tapes	int array of tape ids, if NULL, these will be calculated

Reimplemented in ppE_IMRPhenomD_IMR< T >, and ppE_IMRPhenomD_Inspiral< T >.

6.18.1.19 construct_waveform() [1/2]

```
template<class T >
int IMRPhenomD< T >::construct_waveform (
            T * frequencies,
            int length,
            std::complex< T > * waveform,
            source_parameters< T > * params )  [virtual]
```

Constructs the waveform as outlined by.

arguments: array of frequencies, length of that array, a complex array for the output waveform, and a source_parameters structure

Parameters

frequencies	T array of frequencies the waveform is to be evaluated at
length	integer length of the array of frequencies and the waveform
waveform	complex T array for the waveform to be output

Reimplemented in EdGB_IMRPhenomD< T >, EdGB_IMRPhenomD_log< T >, dCS_IMRPhenomD< T >, and dCS_IMRPhenomD_log< T >.

6.18.1.20 construct_waveform() [2/2]

```
template<class T >
std::complex< T > IMRPhenomD< T >::construct_waveform (
```

```
            T frequency,
            source_parameters< T > * params )  [virtual]
```

overloaded method to evaluate the waveform for one frequency instead of an array

Parameters

frequency	T array of frequencies the waveform is to be evaluated at

6.18.1.21 Damp_ins()

```
template<class T >
T IMRPhenomD< T >::Damp_ins (
            T f,
            source_parameters< T > * param,
            T * pn_coeff,
            lambda_parameters< T > * lambda )  [virtual]
```

Calculates the derivative wrt frequency for the scaled inspiral amplitude A/A0 for frequency f.

This is an analytic derivative for the smoothness condition on the amplitude connection

return a T

6.18.1.22 Damp_mr()

```
template<class T >
T IMRPhenomD< T >::Damp_mr (
            T f,
            source_parameters< T > * param,
            lambda_parameters< T > * lambda )  [virtual]
```

Calculates the derivative wrt frequency for the scaled merger-ringdown amplitude A/A0 for frequency f.

This is an analytic derivative for the smoothness condition on the amplitude connection

The analytic expression was obtained from Mathematica - See the mathematica folder for code

return a T

6.18.1.23 Dphase_ins()

```
template<class T >
T IMRPhenomD< T >::Dphase_ins (
            T f,
            source_parameters< T > * param,
            T * pn_coeff,
            lambda_parameters< T > * lambda )  [virtual]
```

Calculates the derivative of the inspiral phase for frequency f.

For phase continuity and smoothness return a T

Reimplemented in gIMRPhenomD< T >, ppE_IMRPhenomD_Inspiral< T >, and ppE_IMRPhenomPv2_Inspiral< T >.

6.18.1.24 Dphase_int()

```
template<class T >
T IMRPhenomD< T >::Dphase_int (
            T f,
            source_parameters< T > * param,
            lambda_parameters< T > * lambda )  [virtual]
```

Calculates the derivative of the intermediate phase for frequency f.

For phase continuity and smoothness return a T

Reimplemented in ppE_IMRPhenomD_IMR< T >, and ppE_IMRPhenomPv2_IMR< T >.

6.18.1.25 Dphase_mr()

```
template<class T >
T IMRPhenomD< T >::Dphase_mr (
            T f,
            source_parameters< T > * param,
            lambda_parameters< T > * lambda )  [virtual]
```

Calculates the derivative of the merger-ringdown phase for frequency f.

For phase continuity and smoothness return a T

Reimplemented in ppE_IMRPhenomD_IMR< T >, and ppE_IMRPhenomPv2_IMR< T >.

6.18.1.26 EradRational0815()

```
template<class T >
T IMRPhenomD< T >::EradRational0815 (
            T eta,
            T chi1,
            T chi2 )  [virtual]
```

Wrapper function for EradRational0815_s.

6.18.1.27 EradRational0815_s()

```
template<class T >
T IMRPhenomD< T >::EradRational0815_s (
            T eta,
            T s )  [virtual]
```

Formula to predict the total radiated energy. Equation 3.7 and 3.8 arXiv:1508.07250 Input parameter s defined around Equation 3.7 and 3.8.

6.18.1.28 FinalSpin0815()

```
template<class T >
T IMRPhenomD< T >::FinalSpin0815 (
            T eta,
            T chi1,
            T chi2 )  [virtual]
```

Wrapper function for FinalSpin0815_s.

6.18.1.29 FinalSpin0815_s()

```
template<class T >
T IMRPhenomD< T >::FinalSpin0815_s (
            T eta,
            T s )  [virtual]
```

Formula to predict the final spin. Equation 3.6 arXiv:1508.07250 s defined around Equation 3.6.

6.18.1.30 fpeak()

```
template<class T >
T IMRPhenomD< T >::fpeak (
            source_parameters< T > * params,
            lambda_parameters< T > * lambda )  [virtual]
```

Solves for the peak frequency, where the waveform transitions from intermediate to merger-ringdown.

returns Hz

6.18.1.31 phase_connection_coefficients()

```
template<class T >
void IMRPhenomD< T >::phase_connection_coefficients (
            source_parameters< T > * param,
            lambda_parameters< T > * lambda,
            T * pn_coeffs )  [virtual]
```

Calculates the phase connection coefficients alpha{0,1} and beta{0,1}.

Note: these coefficients are stored in the lambda parameter structure, not a separate array

6.18.1.32 phase_ins()

```
template<class T >
T IMRPhenomD< T >::phase_ins (
            T f,
            source_parameters< T > * param,
            T * pn_coeff,
            lambda_parameters< T > * lambda,
            useful_powers< T > * pow )  [virtual]
```

Calculates the inspiral phase for frequency f with precomputed powers of MF and PI for speed.

return a T

extra argument of precomputed powers of MF and pi, contained in the structure useful_powers<T>

Reimplemented in ppE_IMRPhenomD_Inspiral< T >, gIMRPhenomD< T >, and ppE_IMRPhenomPv2_Inspiral< T >.

6.18.1.33 phase_int()

```
template<class T >
T IMRPhenomD< T >::phase_int (
            T f,
            source_parameters< T > * param,
            lambda_parameters< T > * lambda )  [virtual]
```

Calculates the intermediate phase for frequency f.

return a T

Reimplemented in ppE_IMRPhenomD_IMR< T >, and ppE_IMRPhenomPv2_IMR< T >.

6.18.1.34 phase_mr()

```
template<class T >
T IMRPhenomD< T >::phase_mr (
            T f,
            source_parameters< T > * param,
            lambda_parameters< T > * lambda )  [virtual]
```

Calculates the merger-ringdown phase for frequency f.

return a T

Reimplemented in ppE_IMRPhenomD_IMR< T >, and ppE_IMRPhenomPv2_IMR< T >.

6.18.1.35 phase_tape()

```
template<class T >
void IMRPhenomD< T >::phase_tape (
            source_parameters< double > * input_params,
            int * tape )  [virtual]
```

Creates the tapes for derivatives of phase.

For efficiency in long runs of large sets of fishers, the tapes can be precomputed and reused

Parameters

input_params	source parameters structure of the desired source
tape	tape ids

Reimplemented in ppE_IMRPhenomD_IMR< T >, and ppE_IMRPhenomD_Inspiral< T >.

6.18.1.36 post_merger_variables()

```
template<class T >
void IMRPhenomD< T >::post_merger_variables (
            source_parameters< T > * source_param )  [virtual]
```

Calculates the post-merger ringdown frequency and dampening frequency.

Returns in Hz - assigns fRD to var[0] and fdamp to var[1]

6.18.1.37 precalc_powers_ins()

```
template<class T >
void IMRPhenomD< T >::precalc_powers_ins (
            T f,
            T M,
            useful_powers< T > * Mf_pows )  [virtual]
```

Pre-calculate powers of Mf, to speed up calculations for the inspiral waveform (both amplitude and phase.

It seems the pow() function is very slow, so to speed things up, powers of Mf will be precomputed and passed to the functions within the frequency loops

6.18.1.38 precalc_powers_ins_amp()

```
template<class T >
void IMRPhenomD< T >::precalc_powers_ins_amp (
            T f,
            T M,
            useful_powers< T > * Mf_pows )  [virtual]
```

Pre-calculate powers of Mf, to speed up calculations for the inspiral amplitude.

It seems the pow() function is very slow, so to speed things up, powers of Mf will be precomputed and passed to the functions within the frequency loops

6.18.1.39 precalc_powers_ins_phase()

```
template<class T >
void IMRPhenomD< T >::precalc_powers_ins_phase (
            T f,
            T M,
            useful_powers< T > * Mf_pows )  [virtual]
```

Pre-calculate powers of Mf, to speed up calculations for the inspiral phase.

It seems the pow() function is very slow, so to speed things up, powers of Mf will be precomputed and passed to the functions within the frequency loops

6.18.1.40 precalc_powers_PI()

```
template<class T >
void IMRPhenomD< T >::precalc_powers_PI (
            useful_powers< T > * PI_pows )  [virtual]
```

Pre-calculate powers of pi, to speed up calculations for the inspiral phase.

It seems the pow() function is very slow, so to speed things up, powers of PI will be precomputed and passed to the functions within the frequency loops

The documentation for this class was generated from the following files:

- include/gwat/IMRPhenomD.h
- src/IMRPhenomD.cpp

6.19 IMRPhenomPv2< T > Class Template Reference

Inheritance diagram for IMRPhenomPv2< T >:

Collaboration diagram for IMRPhenomPv2< T >:

Public Member Functions

- virtual T **alpha** (T omega, T q, T chi2l, T chi2)
- virtual T **epsilon** (T omega, T q, T chi2l, T chi2)
- virtual void calculate_euler_coeffs (alpha_coeffs< T > *acoeffs, epsilon_coeffs< T > *ecoeffs, source_parameters< T > *params)

  Pre calculate euler angle coefficients.
- virtual T **d** (int l, int mp, int m, T s)
- virtual void PhenomPv2_JSF_from_params (gen_params_base< T > *params, T *JSF)

  Calculate the unit vector in the direction of the total angular momentum.
- virtual int construct_waveform (T *frequencies, int length, std::complex< T > *waveform_plus, std::complex< T > *waveform_cross, source_parameters< T > *params)

  Constructs the waveform for IMRPhenomPv2 - uses IMRPhenomD, then twists up.
- virtual int construct_phase (T *frequencies, int length, T *phase_plus, T *phase_cross, source_parameters< T > *params)

  Constructs the phase for IMRPhenomPv2 - uses IMRPhenomD, then twists up.
- virtual T **calculate_time_shift** (source_parameters< T > *params, useful_powers< T > *pows, T *pn_phase_coeffs, lambda_parameters< T > *lambda)
- virtual void **WignerD** (T d2[5], T dm2[5], useful_powers< T > *pows, source_parameters< T > *params)
- virtual void **calculate_twistup** (T alpha, std::complex< T > *hp_factor, std::complex< T > *hc_factor, T d2[5], T dm2[5], sph_harm< T > *sph_harm)
- virtual void **calculate_euler_angles** (T *alpha, T *epsilon, useful_powers< T > *pows, alpha_coeffs< T > *acoeffs, epsilon_coeffs< T > *ecoeffs)
- virtual void PhenomPv2_Param_Transform (source_parameters< T > *params)
- virtual void PhenomPv2_Param_Transform_J (source_parameters< T > *params)
- virtual void PhenomPv2_Param_Transform_reduced (source_parameters< T > *params)
- virtual T **L2PN** (T eta, useful_powers< T > *pow)
- virtual T FinalSpinIMRPhenomD_all_in_plane_spin_on_larger_BH (T m1, T m2, T chi1_l, T chi2_l, T chip)
- virtual T **final_spin** (source_parameters< T > *params)
- template<>
  double calculate_time_shift (source_parameters< double > *params, useful_powers< double > *pows, double *pn_phase_coeffs, lambda_parameters< double > *lambda)

  Shifts the time of coalescence to the desired value.
- template<>
  adouble calculate_time_shift (source_parameters< adouble > *params, useful_powers< adouble > *pows, adouble *pn_phase_coeffs, lambda_parameters< adouble > *lambda)

  Shifts the time of coalescence to the desired value.

6.19.1 Member Function Documentation

6.19.1.1 calculate_euler_coeffs()

```
template<class T >
void IMRPhenomPv2< T >::calculate_euler_coeffs (
            alpha_coeffs< T > * acoeffs,
            epsilon_coeffs< T > * ecoeffs,
            source_parameters< T > * params )  [virtual]
```

Pre calculate euler angle coefficients.

Straight up stolen from LALsuite

6.19.1.2 calculate_time_shift() [1/2]

```
template<>
adouble IMRPhenomPv2< adouble >::calculate_time_shift (
            source_parameters< adouble > * params,
            useful_powers< adouble > * pows,
            adouble * pn_phase_coeffs,
            lambda_parameters< adouble > * lambda )
```

Shifts the time of coalescence to the desired value.

Because GSL interpolation must have double (not adouble), the two cases must behandled separately, explicitly.

6.19.1.3 calculate_time_shift() [2/2]

```
template<>
double IMRPhenomPv2< double >::calculate_time_shift (
            source_parameters< double > * params,
            useful_powers< double > * pows,
            double * pn_phase_coeffs,
            lambda_parameters< double > * lambda )
```

Shifts the time of coalescence to the desired value.

Because GSL interpolation must have double (not adouble), the two cases must behandled separately, explicitly.

6.19.1.4 construct_phase()

```
template<class T >
int IMRPhenomPv2< T >::construct_phase (
            T * frequencies,
            int length,
            T * phase_plus,
            T * phase_cross,
            source_parameters< T > * params )  [virtual]
```

Constructs the phase for IMRPhenomPv2 - uses IMRPhenomD, then twists up.

arguments: array of frequencies, length of that array, a complex array for the output waveform, and a source_parameters structure