GW Analysis Tools

Generated by Doxygen 1.8.13

Contents

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 Software

Required non-standard C libraries: FFTW3 ADOL-C GSL

Required non-standard Python packages: Cython

Required non-standard packages for documentation: Doxygen

1.2 Installation

For proper compilation, update or create the enviornment variables CDIR and LD_LIBRARY_PATH, which should point to header files and lib files, respectively. Specifically, these variables should point to the above libraries.

1.3 Functionality

1.3.1 Generation

IMRPhenomD, IMRPhenomPv2

1.3.2 Gravity

ppE_IMRPhenomD_Inspiral ppE_IMRPhenomD_IMR ppE_IMRPhenomP_Inspiral ppE_IMRPhenomP_IMR

1.3.3 Analysis

utilizes the above waveform templates

1.3.4 Routines

Includes log likelihood caclulation

Author

Scott Perkins

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:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Class Documentation

6.1 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/mcmc_routines.h

6.2 gen_params Struct Reference

Public Attributes

- double mass1
- double mass2
- double Luminosity_Distance
- double spin1 [3]
- double spin2 [3]
- double phic
- double tc
- int bppe
- double betappe
- double incl_angle
- double theta
- double **phi**
- bool NSflag

6.2.1 Member Data Documentation

6.2.1.1 betappe

```
double gen_params::betappe
```

ppE coefficient for the phase modification

6.2.1.2 bppe

```
int gen_params::bppe
```

ppE b parameter (power of the frequency)

6.2.1.3 incl_angle

```
double gen_params::incl_angle
```

∗angle between angular momentum and the total momentum

6.2.1.4 Luminosity_Distance

```
double gen_params::Luminosity_Distance
```

Luminosity distance to the source

6.2.1.5 mass1

```
double gen_params::mass1
```

mass of the larger body in Solar Masses

6.2.1.6 mass2

```
double gen_params::mass2
```

mass of the smaller body in Solar Masses

6.2.1.7 NSflag

```
bool gen_params::NSflag
```

BOOL flag for early termination of NS binaries

6.2.1.8 phic

```
double gen_params::phic
```

coalescence phase of the binary

6.2.1.9 spin1

```
double gen_params::spin1[3]
```

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

6.2.1.10 spin2

```
double gen_params::spin2[3]
```

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

6.2.1.11 tc

```
double gen_params::tc
```

coalescence time of the binary

6.2.1.12 theta

```
double gen_params::theta
```

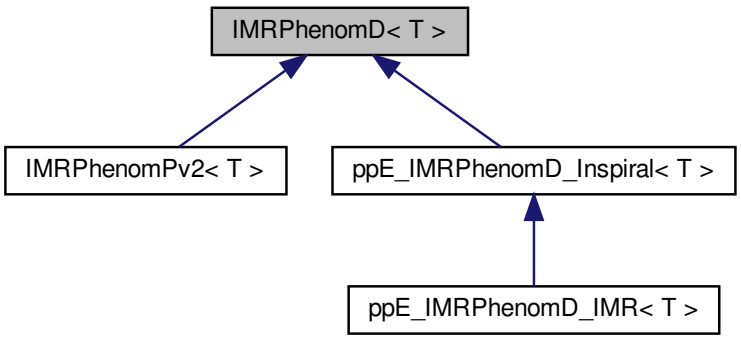
spherical angles for the source location relative to the detector

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

- include/util.h

6.3 IMRPhenomD< T > Class Template Reference

Inheritance diagram for IMRPhenomD< T >:



Public Member Functions

- virtual void **fisher_calculation** (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)

  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)

  Overloaded version for a single frequency instead of a whole array.
- 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 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.

6.3.1 Member Function Documentation

6.3.1.1 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

6.3.1.2 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

6.3.1.3 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

6.3.1.4 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 >.

6.3.1.5 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

6.3.1.6 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

6.3.1.7 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

6.3.1.8 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

6.3.1.9 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

6.3.1.10 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

6.3.1.11 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

6.3.1.12 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

6.3.1.13 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

6.3.1.14 change_parameter_basis()

```
template<class T >
void IMRPhenomD< T >::change_parameter_basis (
            T * old_param,
            T * new_param )  [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}

6.3.1.15 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

6.3.1.16 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 : {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 >.

6.3.1.17 construct_phase()

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

Overloaded version for a single frequency instead of a whole array.

This will be a SLOWER evaluation, only being defined for internal evaluations of derivatives

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

6.3.1.18 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 : {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 >.

6.3.1.19 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

6.3.1.20 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

6.3.1.21 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

6.3.1.22 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

6.3.1.23 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 ppE_IMRPhenomD_Inspiral< T >.

6.3.1.24 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 >.

6.3.1.25 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 >.

6.3.1.26 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

6.3.1.27 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

6.3.1.28 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 >.

6.3.1.29 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 >.

6.3.1.30 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 >.

6.3.1.31 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 >.

6.3.1.32 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]

6.3.1.33 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

6.3.1.34 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

6.3.1.35 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

6.3.1.36 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.

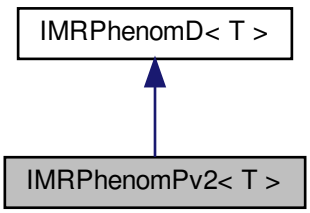
It seems the pow() function is very slow, so to speed things up, powers of PI will be precomputed and passed to the functions within the frequency loops

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

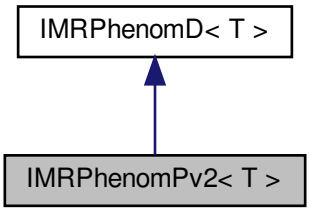
- include/IMRPhenomD.h
- src/IMRPhenomD.cpp

6.4 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 T **d** (int l, int mp, int m, T s)

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

- include/IMRPhenomP.h
- src/IMRPhenomP.cpp

6.5 lambda_parameters< T > Struct Template Reference

Public Attributes

- T **rho** [4]
- T **v2**
- T **gamma** [4]
- T **sigma** [5]
- T **beta** [5]
- T **alpha** [7]

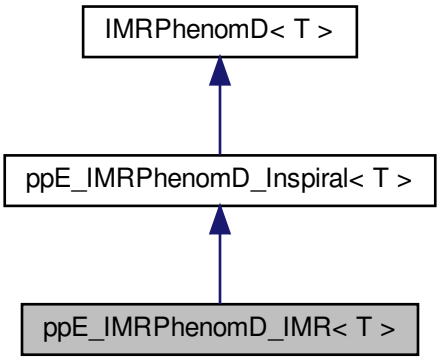
The documentation for this struct was generated from the following file:

- include/IMRPhenomD.h

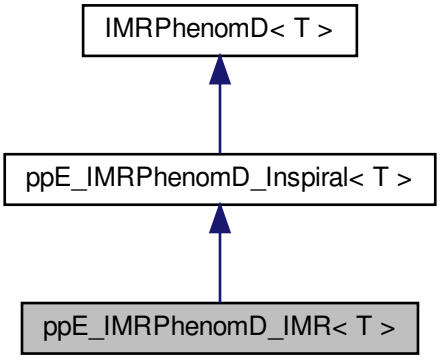
6.6 ppE_IMRPhenomD_IMR< T > Class Template Reference

`#include <ppE_IMRPhenomD.h>`

Inheritance diagram for ppE_IMRPhenomD_IMR< T >:



Collaboration diagram for ppE_IMRPhenomD_IMR< T >:



Public Member Functions

- 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 phase_mr (T f, source_parameters< T > *param, lambda_parameters< T > *lambda)
  Calculates the merger-ringdown phase 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 **fisher_calculation** (double *frequency, int length, gen_params *parameters, double **amplitude_deriv, double **phase_deriv, double *amplitude, int *amp_tapes, int *phase_tapes)
- 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 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.

6.6.1 Detailed Description

template<class T>
class ppE_IMRPhenomD_IMR< T >

Class that extends the IMRPhenomD waveform to include non-GR terms in the full phase. This is an appropriate waveform choice for propagation effects

6.6.2 Member Function Documentation

6.6.2.1 amplitude_tape()

```
template<class T >
void ppE_IMRPhenomD_IMR< 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 from ppE_IMRPhenomD_Inspiral< T >.

6.6.2.2 construct_amplitude_derivative()

```
template<class T >
void ppE_IMRPhenomD_IMR< 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 : {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 from ppE_IMRPhenomD_Inspiral< T >.

6.6.2.3 construct_phase_derivative()

```
template<class T >
void ppE_IMRPhenomD_IMR< 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 : {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 from ppE_IMRPhenomD_Inspiral< T >.

6.6.2.4 Dphase_int()

```
template<class T >
T ppE_IMRPhenomD_IMR< 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 from IMRPhenomD< T >.

6.6.2.5 Dphase_mr()

```
template<class T >
T ppE_IMRPhenomD_IMR< 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 from IMRPhenomD< T >.

6.6.2.6 phase_int()

```
template<class T >
T ppE_IMRPhenomD_IMR< 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 from IMRPhenomD< T >.

6.6.2.7 phase_mr()

```
template<class T >
T ppE_IMRPhenomD_IMR< 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 from IMRPhenomD< T >.

6.6.2.8 phase_tape()

```
template<class T >
void ppE_IMRPhenomD_IMR< 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 from ppE_IMRPhenomD_Inspiral< T >.

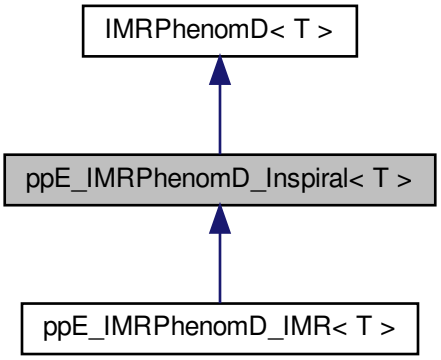
The documentation for this class was generated from the following files:

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

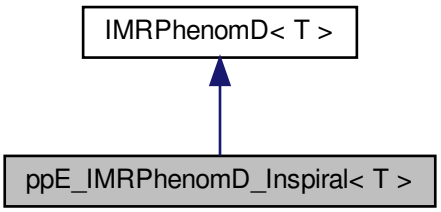
6.7 ppE_IMRPhenomD_Inspiral< T > Class Template Reference

```
#include <ppE_IMRPhenomD.h>
```

Inheritance diagram for ppE_IMRPhenomD_Inspiral< T >:



Collaboration diagram for ppE_IMRPhenomD_Inspiral< 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)

  Overloaded method for the inspiral portion of the phase.
- 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 void **fisher_calculation** (double *frequency, int length, gen_params *parameters, double **amplitude_deriv, double **phase_deriv, double *amplitude, int *amp_tapes, int *phase_tapes)
- 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 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.

6.7.1 Detailed Description

template<class T>
class ppE_IMRPhenomD_Inspiral< T >

Class that extends the IMRPhenomD waveform to include non-GR terms in the inspiral portion of the phase. This is an appropriate waveform choice for generation effects, but not necessarily for propagation effects

6.7.2 Member Function Documentation

6.7.2.1 amplitude_tape()

```
template<class T >
void ppE_IMRPhenomD_Inspiral< 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 from IMRPhenomD< T >.

Reimplemented in ppE_IMRPhenomD_IMR< T >.

6.7.2.2 construct_amplitude_derivative()

```
template<class T >
void ppE_IMRPhenomD_Inspiral< 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 : {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 from IMRPhenomD< T >.

Reimplemented in ppE_IMRPhenomD_IMR< T >.

6.7.2.3 construct_phase_derivative()

```
template<class T >
void ppE_IMRPhenomD_Inspiral< 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 : {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 from IMRPhenomD< T >.

Reimplemented in ppE_IMRPhenomD_IMR< T >.

6.7.2.4 Dphase_ins()

```
template<class T >
T ppE_IMRPhenomD_Inspiral< 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 >.

6.7.2.5 phase_tape()

```
template<class T >
void ppE_IMRPhenomD_Inspiral< 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 from IMRPhenomD< T >.

Reimplemented in ppE_IMRPhenomD_IMR< T >.

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

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

6.8 source_parameters< T > Struct Template Reference

Static Public Member Functions

- static source_parameters< T > populate_source_parameters (T mass1, T mass2, T Luminosity_Distance, T *spin1, T *spin2, T phi_c, T t_c)

  Builds the structure that shuttles source parameters between functions.

Public Attributes

- T mass1
- T mass2
- T M
- T spin1z
- T spin2z
- T spin1x
- T spin2x
- T spin1y
- T spin2y
- T chirpmass
- T eta
- T chi_s
- T chi_a
- T chi_eff
- T chi_pn
- T DL
- T delta_mass
- T fRD
- T fdamp
- T f1
- T f3
- T f1_phase
- T f2_phase
- T phic
- T tc
- T **A0**
- T **betappe**
- int **bppe**

6.8.1 Member Function Documentation

6.8.1.1 populate_source_parameters()

```
template<class T >
source_parameters< T > source_parameters< T >::populate_source_parameters (
            T mass1,
            T mass2,
            T Luminosity_Distance,
            T * spin1,
            T * spin2,
            T phi_c,
            T t_c )  [static]
```

Builds the structure that shuttles source parameters between functions.

Populates the structure that is passed to all generation methods - contains all relavent source parameters

Parameters

mass1	mass of the larger body - in Solar Masses
mass2	mass of the smaller body - in Solar Masses
Luminosity_Distance	Luminosity Distance in Mpc
spin2	spin vector of the larger body {sx,sy,sz}
phi_c	spin vector of the smaller body {sx,sy,sz}
t_c	coalescence phase coalescence time

6.8.2 Member Data Documentation

6.8.2.1 chi_a

```
template<class T>
T source_parameters< T >::chi_a
```

Antisymmetric spin combination

6.8.2.2 chi_eff

```
template<class T>
T source_parameters< T >::chi_eff
```

Effective spin

6.8.2.3 chi_pn

```
template<class T>
T source_parameters< T >::chi_pn
```

PN spin

6.8.2.4 chi_s

```
template<class T>
T source_parameters< T >::chi_s
```

Symmetric spin combination

6.8.2.5 chirpmass

```
template<class T>
T source_parameters< T >::chirpmass
```

Chirp mass of the binary

6.8.2.6 delta_mass

```
template<class T>
T source_parameters< T >::delta_mass
```

Delta mass comibination

6.8.2.7 DL

```
template<class T>
T source_parameters< T >::DL
```

Luminoisity Distance

6.8.2.8 eta

```
template<class T>
T source_parameters< T >::eta
```

Symmetric mass ratio

6.8.2.9 f1

```
template<class T>
T source_parameters< T >::f1
```

Transition Frequency 1 for the amplitude

6.8.2.10 f1_phase

```
template<class T>
T source_parameters< T >::f1_phase
```

Transition frequency 1 for the phase

6.8.2.11 f2_phase

```
template<class T>
T source_parameters< T >::f2_phase
```

Transition frequency 2 for the phase

6.8.2.12 f3

```
template<class T>
T source_parameters< T >::f3
```

Transition Frequency 2 for the amplitude

6.8.2.13 fdamp

```
template<class T>
T source_parameters< T >::fdamp
```

Dampening frequency after merger

6.8.2.14 fRD

```
template<class T>
T source_parameters< T >::fRD
```

Ringdown frequency after merger

6.8.2.15 M

```
template<class T>
T source_parameters< T >::M
```

Total mass

6.8.2.16 mass1

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

mass of the larger component

6.8.2.17 mass2

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

mass of the smaller component

6.8.2.18 phic

```
template<class T>
T source_parameters< T >::phic
```

Coalescence phase

6.8.2.19 spin1x

```
template<class T>
T source_parameters< T >::spin1x
```

x-Spin component of the larger body

6.8.2.20 spin1y

```
template<class T>
T source_parameters< T >::spin1y
```

y-Spin component of the larger body

6.8.2.21 spin1z

```
template<class T>
T source_parameters< T >::spin1z
```

z-Spin component of the larger body

6.8.2.22 spin2x

```
template<class T>
T source_parameters< T >::spin2x
```

x-Spin component of the smaller body

6.8.2.23 spin2y

```
template<class T>
T source_parameters< T >::spin2y
```

y-Spin component of the smaller body

6.8.2.24 spin2z

```
template<class T>
T source_parameters< T >::spin2z
```

z-Spin component of the smaller body

6.8.2.25 tc

```
template<class T>
T source_parameters< T >::tc
```

Coalescence time

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

- include/util.h
- src/util.cpp

6.9 useful_powers< T > Struct Template Reference

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.

`#include <util.h>`

Public Attributes

- T **MFthird**
- T **MF2third**
- T **MF4third**
- T **MF5third**
- T **MFsquare**
- T **MF7third**
- T **MF8third**
- T **MFcube**
- T **MFminus_5third**
- T **MF3fourth**
- double **PIsquare**
- double **PIcube**
- double **PIthird**
- double **PI2third**
- double **PI4third**
- double **PI5third**
- double **PI7third**
- double **PIminus_5third**

6.9.1 Detailed Description

template<class T>
struct 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.

Powers of PI are initialized once, and powers of MF need to be calculated once per for loop (if in the inspiral portion).

use the functions precalc_powers_ins_amp, precalc_powers_ins_phase, precalc_powers_pi to initialize

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

- include/util.h

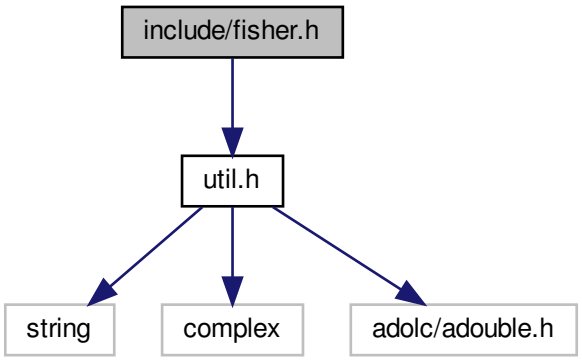
Chapter 7

File Documentation

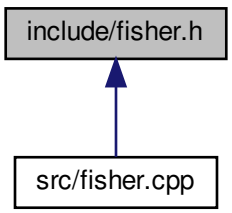
7.1 include/fisher.h File Reference

```
#include "util.h"
```

Include dependency graph for fisher.h:



This graph shows which files directly or indirectly include this file:



Functions

- void fisher (double *frequency, int length, string generation_method, string detector, double **output, int dimension, gen_params *parameters, int *amp_tapes=NULL, int *phase_tapes=NULL, double *noise=NULL)

  Calculates the fisher matrix for the given arguments.

7.1.1 Function Documentation

7.1.1.1 fisher()

```
void fisher (
             double * frequency,
             int length,
             string generation_method,
             string detector,
             double ** output,
             int dimension,
             gen_params * parameters,
             int * amp_tapes = NULL,
             int * phase_tapes = NULL,
             double * noise = NULL )
```

Calculates the fisher matrix for the given arguments.

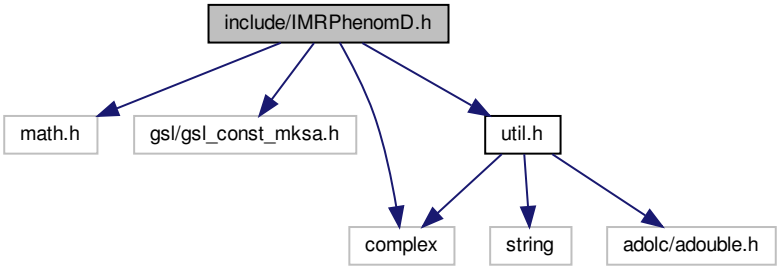
Parameters

length	if 0, standard frequency range for the detector is used
output	double [dimension][dimension]
amp_tapes	if speed is required, precomputed tapes can be used - assumed the user knows what they're doing, no checks done here to make sure that the number of tapes matches the requirement by the generation_method
phase_tapes	if speed is required, precomputed tapes can be used - assumed the user knows what they're doing, no checks done here to make sure that the number of tapes matches the requirement by the generation_method

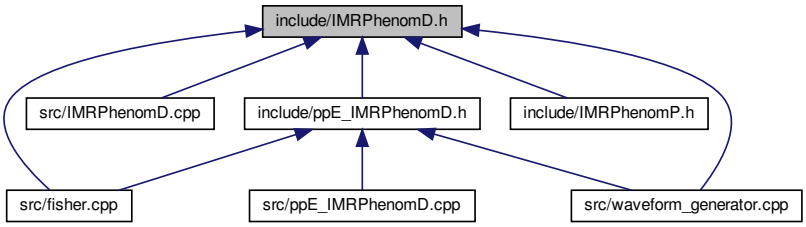
7.2 include/IMRPhenomD.h File Reference

```
#include <math.h>
#include <gsl/gsl_const_mksa.h>
#include <complex>
#include "util.h"
```

Include dependency graph for IMRPhenomD.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct lambda_parameters< T >
- class IMRPhenomD< T >

Variables

- const double lambda_num_params [19][11]

7.2.1 Detailed Description

Header file for utilities

7.2.2 Variable Documentation

7.2.2.1 lambda_num_params

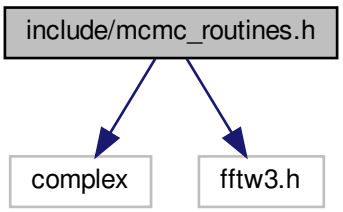
```
const double lambda_num_params[19][11]
```

Numerically calibrated parameters from arXiv:1508.07253 see the table in the data directory for labeled version

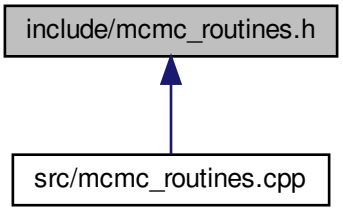
7.3 include/mcmc_routines.h File Reference

```
#include <complex>
#include <fftw3.h>
```

Include dependency graph for mcmc_routines.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct fftw_outline

Functions

- double maximized_coal_log_likelihood_IMRPhenomD (double *frequencies, int length, std::complex< double > *data, double *noise, double SNR, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, bool NSflag, fftw_outline *plan)

  Function to calculate the log Likelihood as defined by -1/2 (d-h|d-h) maximized over the extrinsic parameters phic and tc.
- double maximized_coal_log_likelihood_IMRPhenomD (double *frequencies, size_t length, double *real_data, double *imag_data, double *noise, double SNR, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, bool NSflag)
- double maximized_coal_log_likelihood_IMRPhenomD (double *frequencies, size_t length, double *real_data, double *imag_data, double *noise, double SNR, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, bool NSflag, fftw_outline *plan)
- double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (double *frequencies, int length, std::complex< double > *data, double *noise, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, double Luminosity_Distance, double theta, double phi, double iota, bool NSflag, fftw_outline *plan)
- double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (double *frequencies, size_t length, double *real_data, double *imag_data, double *noise, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, double Luminosity_Distance, double theta, double phi, double iota, bool NSflag)
- double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (double *frequencies, size_t length, double *real_data, double *imag_data, double *noise, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, double Luminosity_Distance, double theta, double phi, double iota, bool NSflag, fftw_outline *plan)
- void **initiate_likelihood_function** (fftw_outline *plan, int length)
- void **deactivate_likelihood_function** (fftw_outline *plan)

7.3.1 Function Documentation

7.3.1.1 maximized_coal_log_likelihood_IMRPhenomD() [1/3]

```
double maximized_coal_log_likelihood_IMRPhenomD (
            double * frequencies,
            int length,
            std::complex< double > * data,
            double * noise,
            double SNR,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            bool NSflag,
            fftw_outline * plan )
```

Function to calculate the log Likelihood as defined by -1/2 (d-h|d-h) maximized over the extrinsic parameters phic and tc.

frequency array must be uniform spacing - this shouldn't be a problem when working with real data as DFT return uniform spacing

Parameters

chirpmass	in solar masses

7.3.1.2 maximized_coal_log_likelihood_IMRPhenomD() [2/3]

```
double maximized_coal_log_likelihood_IMRPhenomD (
            double * frequencies,
            size_t length,
            double * real_data,
            double * imag_data,
            double * noise,
            double SNR,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            bool NSflag )
```

Parameters

chirpmass	in solar masses

7.3.1.3 maximized_coal_log_likelihood_IMRPhenomD() [3/3]

```
double maximized_coal_log_likelihood_IMRPhenomD (
            double * frequencies,
            size_t length,
            double * real_data,
            double * imag_data,
            double * noise,
            double SNR,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            bool NSflag,
            fftw_outline * plan )
```

Parameters

chirpmass	in solar masses

7.3.1.4 maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [1/3]

```
double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (
            double * frequencies,
            int length,
            std::complex< double > * data,
            double * noise,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            double Luminosity_Distance,
            double theta,
            double phi,
            double iota,
            bool NSflag,
            fftw_outline * plan )
```

Parameters

chirpmass	in solar masses

7.3.1.5 maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [2/3]

```
double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (
            double * frequencies,
            size_t length,
            double * real_data,
            double * imag_data,
            double * noise,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            double Luminosity_Distance,
            double theta,
            double phi,
            double iota,
            bool NSflag )
```

Parameters

chirpmass	in solar masses

7.3.1.6 maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [3/3]

```
double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (
            double * frequencies,
```

```
size_t length,
double * real_data,
double * imag_data,
double * noise,
double chirpmass,
double symmetric_mass_ratio,
double spin1,
double spin2,
double Luminosity_Distance,
double theta,
double phi,
double iota,
bool NSflag,
fftw_outline * plan )
```

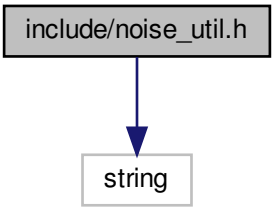
Parameters

chirpmass	in solar masses

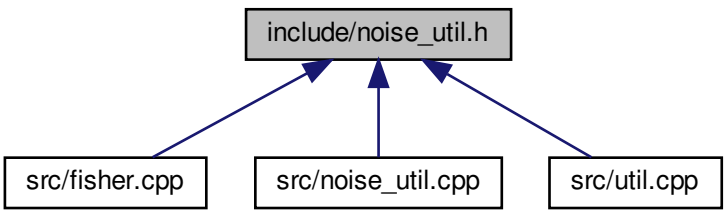
7.4 include/noise_util.h File Reference

```
#include <string>
```

Include dependency graph for noise_util.h:



This graph shows which files directly or indirectly include this file:



Functions

- void populate_noise (double ∗frequencies, std::string detector, double ∗noise_root, int length=0)

  Function to populate the squareroot of the noise curve for various detectors.
- double **aLIGO_analytic** (double f)
- double **Hanford_O1_fitted** (double f)

7.4.1 Function Documentation

7.4.1.1 populate_noise()

```
void populate_noise (
             double * frequencies,
             std::string detector,
             double * noise_root,
             int length )
```

Function to populate the squareroot of the noise curve for various detectors.

If frequencies are left as NULL, standard frequency spacing is applied and the frequencies are returned, in which case the frequencies argument becomes an output array

Detector names must be spelled exactly

Detectors include: aLIGO_analytic, Hanford_O1_fitted

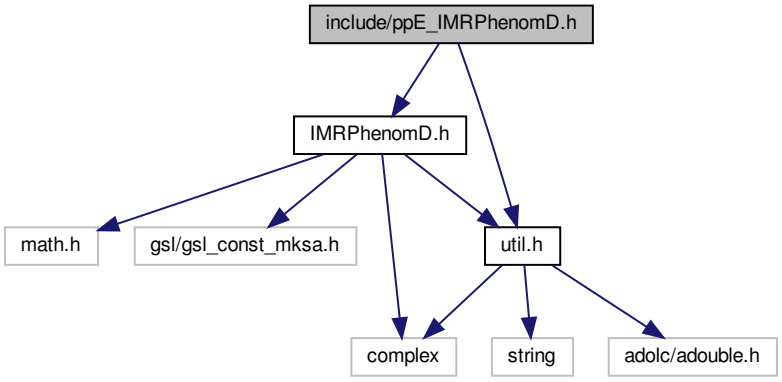
Parameters

frequencies	double array of frquencies (NULL)
detector	String to designate the detector noise curve to be used
noise_root	ouptput double array for the square root of the PSD of the noise of the specified detector
length	integer length of the output and input arrays

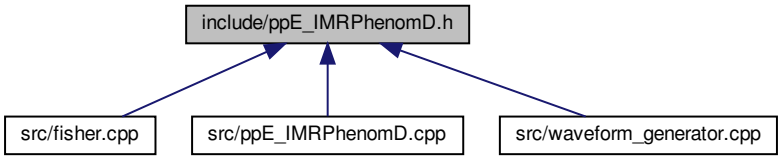
7.5 include/ppE_IMRPhenomD.h File Reference

```
#include "IMRPhenomD.h"
#include "util.h"
```

Include dependency graph for ppE_IMRPhenomD.h:



This graph shows which files directly or indirectly include this file:



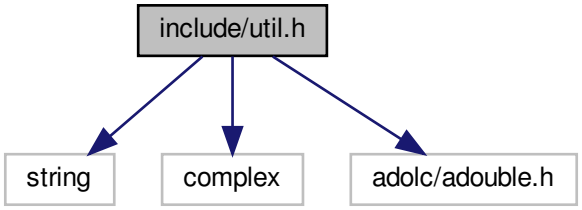
Classes

- class ppE_IMRPhenomD_Inspiral< T >
- class ppE_IMRPhenomD_IMR< T >

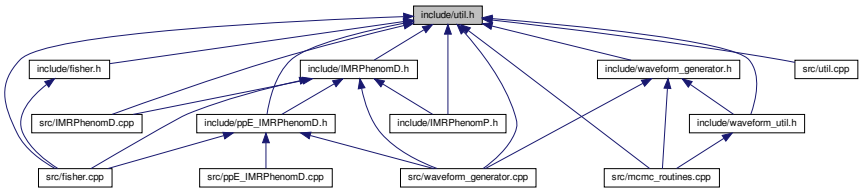
7.6 include/util.h File Reference

```
#include <string>
#include <complex>
#include <adolc/adouble.h>
```

Include dependency graph for util.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct gen_params
- struct 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.
- struct source_parameters< T >

Functions

- double calculate_eta (double mass1, double mass2)

  Calculates the symmetric mass ration from the two component masses.
- adouble **calculate_eta** (adouble mass1, adouble mass2)
- double calculate_chirpmass (double mass1, double mass2)

  Calculates the chirp mass from the two component masses.
- adouble **calculate_chirpmass** (adouble mass1, adouble mass2)
- double calculate_mass1 (double chirpmass, double eta)

  Calculates the larger mass given a chirp mass and symmetric mass ratio.
- adouble **calculate_mass1** (adouble chirpmass, adouble eta)
- double calculate_mass2 (double chirpmass, double eta)

  Calculates the smaller mass given a chirp mass and symmetric mass ratio.
- adouble **calculate_mass2** (adouble chirpmass, adouble eta)
- template<class T >
  T trapezoidal_sum_uniform (double delta_x, int length, T ∗integrand)

Trapezoidal sum rule to approximate discrete integral - Uniform spacing.

- template<class T >
  T trapezoidal_sum (double *delta_x, int length, T *integrand)

  Trapezoidal sum rule to approximate discrete integral - Non-Uniform spacing.
- template<class T >
  T simpsons_sum (double delta_x, int length, T *integrand)

  Simpsons sum rule to approximate discrete integral - Uniform spacing.
- long **factorial** (long num)

Variables

- const double gamma_E = 0.57721566490153286060651209008240243104 21
- const double c = 299792458.
- const double G =6.674e-11*(1.98855e30)
- const double MSOL_SEC =492549095.e-14
- const double MPC_SEC = 3085677581.e13/c

7.6.1 Detailed Description

General utilities (functions and structures) independent of modelling method

7.6.2 Function Documentation

7.6.2.1 calculate_chirpmass()

```
double calculate_chirpmass (
            double mass1,
            double mass2 )
```

Calculates the chirp mass from the two component masses.

The output units are whatever units the input masses are

7.6.2.2 calculate_mass1()

```
double calculate_mass1 (
            double chirpmass,
            double eta )
```

Calculates the larger mass given a chirp mass and symmetric mass ratio.

Units of the output match the units of the input chirp mass

7.6.2.3 calculate_mass2()

```
double calculate_mass2 (
            double chirpmass,
            double eta )
```

Calculates the smaller mass given a chirp mass and symmetric mass ratio.

Units of the output match the units of the input chirp mass

7.6.2.4 simpsons_sum()

```
template<class T >
T simpsons_sum (
            double delta_x,
            int length,
            T * integrand )
```

Simpsons sum rule to approximate discrete integral - Uniform spacing.

More accurate than the trapezoidal rule, but must be uniform

7.6.2.5 trapezoidal_sum()

```
template<class T >
T trapezoidal_sum (
            double * delta_x,
            int length,
            T * integrand )
```

Trapezoidal sum rule to approximate discrete integral - Non-Uniform spacing.

This version is slower than the uniform version, but will handle non-uniform spacing

7.6.2.6 trapezoidal_sum_uniform()

```
template<class T >
T trapezoidal_sum_uniform (
            double delta_x,
            int length,
            T * integrand )
```

Trapezoidal sum rule to approximate discrete integral - Uniform spacing.

This version is faster than the general version, as it has half the function calls

Something may be wrong with this function - had an overall offset for real data that was fixed by using the simpsons rule - not sure if this was because of a boost in accuracy or because something is off with the trapezoidal sum

7.6.3 Variable Documentation

7.6.3.1 c

```
const double c = 299792458.
```

Speed of light m/s

7.6.3.2 G

```
const double G =6.674e-11*(1.98855e30)
```

Gravitational constant in m**3/(s**2 SolMass)

7.6.3.3 gamma_E

```
const double gamma_E = 0.57721566490153286060651209008240243104 21
```

Euler number

7.6.3.4 MPC_SEC

```
const double MPC_SEC = 3085677581.e13/c
```

consts.kpc.to('m')*1000/c Mpc in sec

7.6.3.5 MSOL_SEC

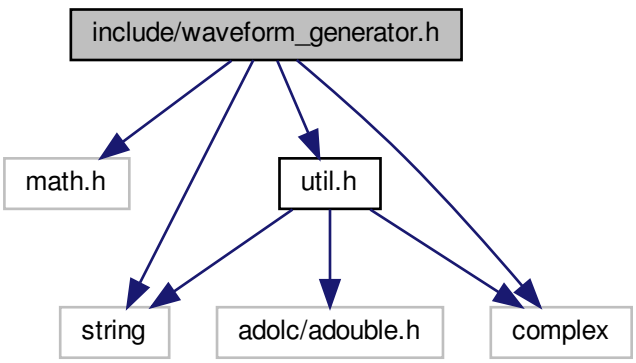
```
const double MSOL_SEC =492549095.e-14
```

G/c**3 seconds per solar mass

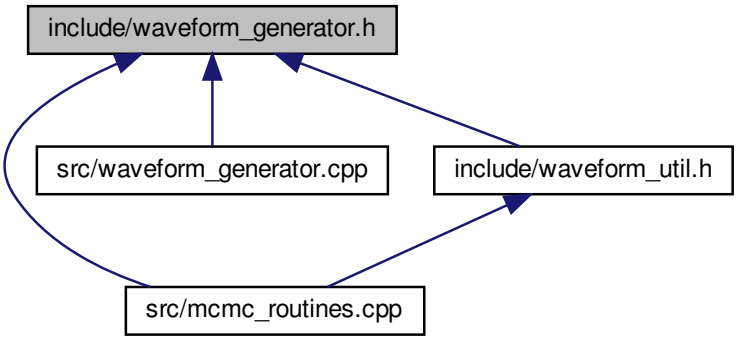
7.7 include/waveform_generator.h File Reference

```
#include <math.h>
#include "util.h"
#include <complex>
#include <string>
```

Include dependency graph for waveform_generator.h:



This graph shows which files directly or indirectly include this file:



Functions

- int **fourier_waveform** (double *frequencies, int length, std::complex< double > *waveform, std::string generation_method, gen_params *parameters)
- int **fourier_waveform** (double *frequencies, int length, double *waveform_real, double *waveform_imag, std::string generation_method, gen_params *parameters)
- int **fourier_amplitude** (double *frequencies, int length, double *amplitude, std::string generation_method, gen_params *parameters)
- int **fourier_phase** (double *frequencies, int length, double *phase, std::string generation_method, gen_↩ params *parameters)

7.8 README.dox File Reference

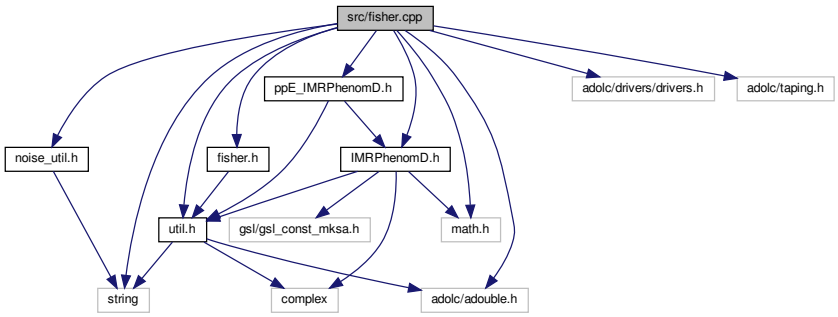
7.8.1 Detailed Description

hello

7.9 src/fisher.cpp File Reference

```
#include <fisher.h>
#include <adolc/adouble.h>
#include <adolc/drivers/drivers.h>
#include <adolc/taping.h>
#include <math.h>
#include <string>
#include "util.h"
#include "noise_util.h"
#include "IMRPhenomD.h"
```

```
#include "ppE_IMRPhenomD.h"
```
Include dependency graph for fisher.cpp:



Functions

- void fisher (double ∗frequency, int length, string generation_method, string detector, double ∗∗output, int dimension, gen_params ∗parameters, int ∗amp_tapes, int ∗phase_tapes, double ∗noise)

  Calculates the fisher matrix for the given arguments.

7.9.1 Detailed Description

All subroutines associated with waveform differentiation and Fisher analysis

7.9.2 Function Documentation

7.9.2.1 fisher()

```
void fisher (
            double * frequency,
            int length,
            string generation_method,
            string detector,
            double ** output,
            int dimension,
            gen_params * parameters,
            int * amp_tapes,
            int * phase_tapes,
            double * noise )
```

Calculates the fisher matrix for the given arguments.

Parameters

length	if 0, standard frequency range for the detector is used

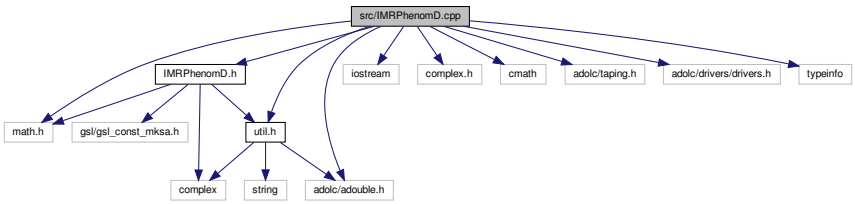
Parameters

output	double [dimension][dimension]
amp_tapes	if speed is required, precomputed tapes can be used - assumed the user knows what they're doing, no checks done here to make sure that the number of tapes matches the requirement by the generation_method
phase_tapes	if speed is required, precomputed tapes can be used - assumed the user knows what they're doing, no checks done here to make sure that the number of tapes matches the requirement by the generation_method

7.10 src/IMRPhenomD.cpp File Reference

```
#include "IMRPhenomD.h"
#include "util.h"
#include <math.h>
#include <iostream>
#include <complex.h>
#include <cmath>
#include <adolc/adouble.h>
#include <adolc/taping.h>
#include <adolc/drivers/drivers.h>
#include <typeinfo>
```

Include dependency graph for IMRPhenomD.cpp:



7.10.1 Detailed Description

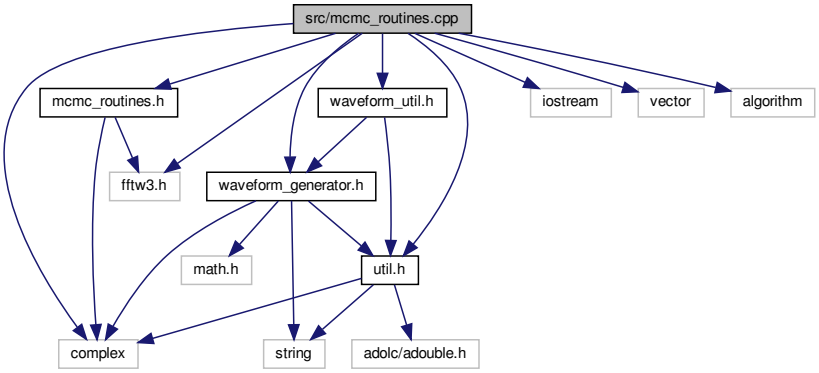
File that includes all the low level functions that go into constructing the waveform

7.11 src/mcmc_routines.cpp File Reference

```
#include "mcmc_routines.h"
#include "waveform_generator.h"
#include "util.h"
#include "waveform_util.h"
#include <iostream>
#include <vector>
#include <complex>
#include <fftw3.h>
```

```
#include <algorithm>
```

Include dependency graph for mcmc_routines.cpp:



Functions

- double maximized_coal_log_likelihood_IMRPhenomD (double *frequencies, int length, std::complex< double > *data, double *noise, double SNR, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, bool NSflag, fftw_outline *plan)

  Function to calculate the log Likelihood as defined by -1/2 (d-h|d-h) maximized over the extrinsic parameters phic and tc.
- double maximized_coal_log_likelihood_IMRPhenomD (double *frequencies, size_t length, double *real_data, double *imag_data, double *noise, double SNR, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, bool NSflag)
- double maximized_coal_log_likelihood_IMRPhenomD (double *frequencies, size_t length, double *real_data, double *imag_data, double *noise, double SNR, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, bool NSflag, fftw_outline *plan)
- double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (double *frequencies, int length, std::complex< double > *data, double *noise, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, double Luminosity_Distance, double theta, double phi, double iota, bool NSflag, fftw_outline *plan)
- double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (double *frequencies, size_t length, double *real_data, double *imag_data, double *noise, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, double Luminosity_Distance, double theta, double phi, double iota, bool NSflag)
- double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (double *frequencies, size_t length, double *real_data, double *imag_data, double *noise, double chirpmass, double symmetric_mass_ratio, double spin1, double spin2, double Luminosity_Distance, double theta, double phi, double iota, bool NSflag, fftw_outline *plan)
- void **initiate_likelihood_function** (fftw_outline *plan, int length)
- void **deactivate_likelihood_function** (fftw_outline *plan)

7.11.1 Detailed Description

Routines for implementation in MCMC algorithms

7.11.2 Function Documentation

7.11.2.1 maximized_coal_log_likelihood_IMRPhenomD() [1/3]

```
double maximized_coal_log_likelihood_IMRPhenomD (
            double * frequencies,
            int length,
            std::complex< double > * data,
            double * noise,
            double SNR,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            bool NSflag,
            fftw_outline * plan )
```

Function to calculate the log Likelihood as defined by -1/2 (d-h|d-h) maximized over the extrinsic parameters phic and tc.

frequency array must be uniform spacing - this shouldn't be a problem when working with real data as DFT return uniform spacing

Parameters

chirpmass	in solar masses

7.11.2.2 maximized_coal_log_likelihood_IMRPhenomD() [2/3]

```
double maximized_coal_log_likelihood_IMRPhenomD (
            double * frequencies,
            size_t length,
            double * real_data,
            double * imag_data,
            double * noise,
            double SNR,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            bool NSflag )
```

Parameters

chirpmass	in solar masses

7.11.2.3 maximized_coal_log_likelihood_IMRPhenomD() [3/3]

```
double maximized_coal_log_likelihood_IMRPhenomD (
            double * frequencies,
```

```
            size_t length,
            double * real_data,
            double * imag_data,
            double * noise,
            double SNR,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            bool NSflag,
            fftw_outline * plan )
```

Parameters

chirpmass	in solar masses

7.11.2.4 maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [1/3]

```
double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (
            double * frequencies,
            int length,
            std::complex< double > * data,
            double * noise,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            double Luminosity_Distance,
            double theta,
            double phi,
            double iota,
            bool NSflag,
            fftw_outline * plan )
```

Parameters

chirpmass	in solar masses

7.11.2.5 maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [2/3]

```
double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (
            double * frequencies,
            size_t length,
            double * real_data,
            double * imag_data,
            double * noise,
            double chirpmass,
            double symmetric_mass_ratio,
```

```
            double spin1,
            double spin2,
            double Luminosity_Distance,
            double theta,
            double phi,
            double iota,
            bool NSflag )
```

Parameters

chirpmass	in solar masses

7.11.2.6 maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [3/3]

```
double maximized_coal_log_likelihood_IMRPhenomD_Full_Param (
            double * frequencies,
            size_t length,
            double * real_data,
            double * imag_data,
            double * noise,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            double Luminosity_Distance,
            double theta,
            double phi,
            double iota,
            bool NSflag,
            fftw_outline * plan )
```

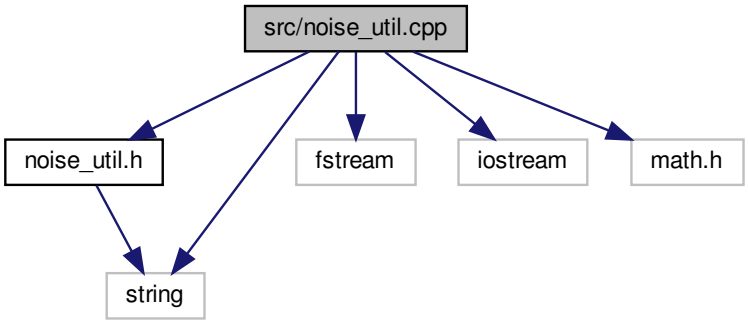
Parameters

chirpmass	in solar masses

7.12 src/noise_util.cpp File Reference

```
#include "noise_util.h"
#include <fstream>
#include <iostream>
#include <string>
#include <math.h>
```

Include dependency graph for noise_util.cpp:



Functions

- void populate_noise (double ∗frequencies, std::string detector, double ∗noise_root, int length)

  Function to populate the squareroot of the noise curve for various detectors.
- double **aLIGO_analytic** (double f)
- double **Hanford_O1_fitted** (double f)

7.12.1 Detailed Description

Routines to construct noise curves for various detectors

7.12.2 Function Documentation

7.12.2.1 populate_noise()

```
void populate_noise (
            double * frequencies,
            std::string detector,
            double * noise_root,
            int length )
```

Function to populate the squareroot of the noise curve for various detectors.

If frequencies are left as NULL, standard frequency spacing is applied and the frequencies are returned, in which case the frequencies argument becomes an output array

Detector names must be spelled exactly

Detectors include: aLIGO_analytic, Hanford_O1_fitted

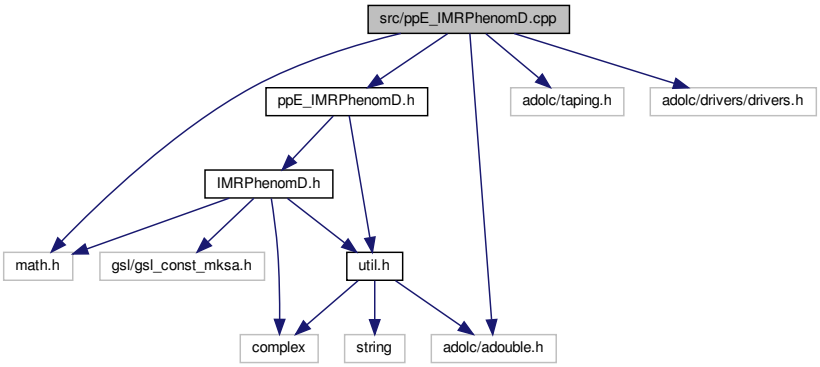
Parameters

frequencies	double array of frquencies (NULL)
detector	String to designate the detector noise curve to be used
noise_root	ouptput double array for the square root of the PSD of the noise of the specified detector
length	integer length of the output and input arrays

7.13 src/ppE_IMRPhenomD.cpp File Reference

```
#include "ppE_IMRPhenomD.h"
#include <math.h>
#include <adolc/adouble.h>
#include <adolc/taping.h>
#include <adolc/drivers/drivers.h>
```

Include dependency graph for ppE_IMRPhenomD.cpp:



7.13.1 Detailed Description

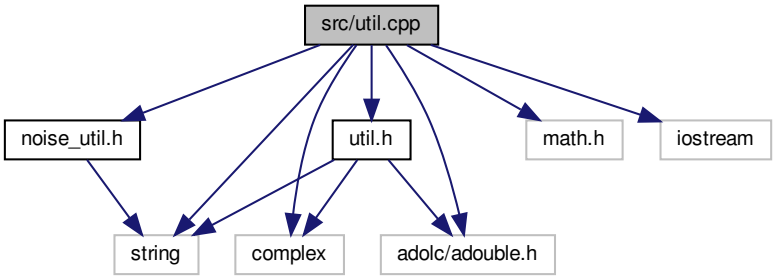
File for the implementation of the ppE formalism for testing GR

Extends the IMRPhenomD template to include non-GR phase terms

7.14 src/util.cpp File Reference

```
#include "util.h"
#include "noise_util.h"
#include <math.h>
#include <string>
#include <complex>
#include <iostream>
```

```
#include <adolc/adouble.h>
```
Include dependency graph for util.cpp:



Functions

- double calculate_chirpmass (double mass1, double mass2)

  Calculates the chirp mass from the two component masses.
- adouble **calculate_chirpmass** (adouble mass1, adouble mass2)
- double calculate_eta (double mass1, double mass2)

  Calculates the symmetric mass ration from the two component masses.
- adouble **calculate_eta** (adouble mass1, adouble mass2)
- double calculate_mass1 (double chirpmass, double eta)

  Calculates the larger mass given a chirp mass and symmetric mass ratio.
- adouble **calculate_mass1** (adouble chirpmass, adouble eta)
- double calculate_mass2 (double chirpmass, double eta)

  Calculates the smaller mass given a chirp mass and symmetric mass ratio.
- adouble **calculate_mass2** (adouble chirpmass, adouble eta)
- long **factorial** (long num)

7.14.1 Detailed Description

General utilities that are method independent

7.14.2 Function Documentation

7.14.2.1 calculate_chirpmass()

```
double calculate_chirpmass (
            double mass1,
            double mass2 )
```

Calculates the chirp mass from the two component masses.

The output units are whatever units the input masses are

7.14.2.2 calculate_mass1()

```
double calculate_mass1 (
            double chirpmass,
            double eta )
```

Calculates the larger mass given a chirp mass and symmetric mass ratio.

Units of the output match the units of the input chirp mass

7.14.2.3 calculate_mass2()

```
double calculate_mass2 (
            double chirpmass,
            double eta )
```

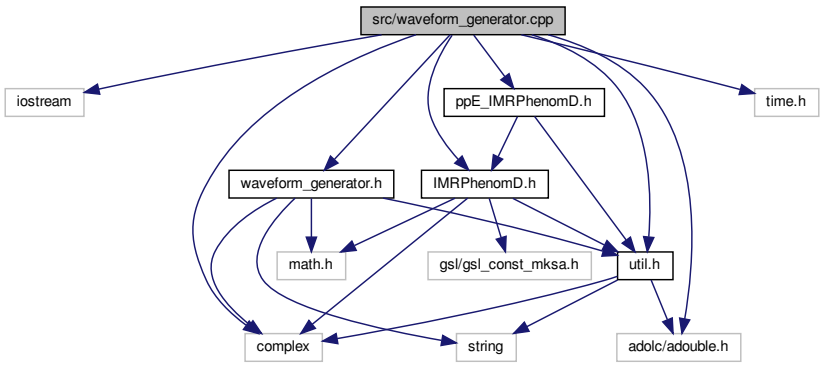
Calculates the smaller mass given a chirp mass and symmetric mass ratio.

Units of the output match the units of the input chirp mass

7.15 src/waveform_generator.cpp File Reference

```
#include <iostream>
#include "waveform_generator.h"
#include "IMRPhenomD.h"
#include "ppE_IMRPhenomD.h"
#include "util.h"
#include <complex>
#include <time.h>
#include <adolc/adouble.h>
```

Include dependency graph for waveform_generator.cpp:



Functions

- int fourier_waveform (double *frequencies, int length, std::complex< double > *waveform, string generation_method, gen_params *parameters)

  Function to produce the (2,2) mode of an quasi-circular binary.
- int fourier_waveform (double *frequencies, int length, double *waveform_real, double *waveform_imag, string generation_method, gen_params *parameters)
- int fourier_amplitude (double *frequencies, int length, double *amplitude, string generation_method, gen_params *parameters)

  Function to produce the amplitude of the (2,2) mode of an quasi-circular binary.
- int fourier_phase (double *frequencies, int length, double *phase, string generation_method, gen_params *parameters)

  Function to produce the phase of the (2,2) mode of an quasi-circular binary.

7.15.1 Detailed Description

File that handles the construction of the (2,2) waveform as described by IMRPhenomD by Khan et. al.

Builds a waveform for given DETECTOR FRAME parameters

7.15.2 Function Documentation

7.15.2.1 fourier_amplitude()

```
int fourier_amplitude (
            double * frequencies,
            int length,
            double * amplitude,
            string generation_method,
            gen_params * parameters )
```

Function to produce the amplitude of the (2,2) mode of an quasi-circular binary.

By using the structure parameter, the function is allowed to be more flexible in using different method of waveform generation - not all methods use the same parameters

Parameters

frequencies	double array of frequencies for the waveform to be evaluated at
length	integer length of all the arrays
amplitude	output array for the amplitude
generation_method	String that corresponds to the generation method - MUST BE SPELLED EXACTLY

7.15.2.2 fourier_phase()

```
int fourier_phase (
              double * frequencies,
              int length,
              double * phase,
              string generation_method,
              gen_params * parameters )
```

Function to produce the phase of the (2,2) mode of an quasi-circular binary.

By using the structure parameter, the function is allowed to be more flexible in using different method of waveform generation - not all methods use the same parameters

Parameters

frequencies	double array of frequencies for the waveform to be evaluated at
length	integer length of all the arrays
phase	output array for the phase
generation_method	String that corresponds to the generation method - MUST BE SPELLED EXACTLY

7.15.2.3 fourier_waveform() [1/2]

```
int fourier_waveform (
              double * frequencies,
              int length,
              std::complex< double > * waveform,
              string generation_method,
              gen_params * parameters )
```

Function to produce the (2,2) mode of an quasi-circular binary.

By using the structure parameter, the function is allowed to be more flexible in using different method of waveform generation - not all methods use the same parameters

Parameters

frequencies	double array of frequencies for the waveform to be evaluated at
length	integer length of all the arrays
waveform	complex array for the output waveform
generation_method	String that corresponds to the generation method - MUST BE SPELLED EXACTLY
parameters	structure containing all the source parameters

7.15.2.4 fourier_waveform() [2/2]

```
int fourier_waveform (
              double * frequencies,
```

```
int length,
double * waveform_real,
double * waveform_imag,
string generation_method,
gen_params * parameters )
```

Parameters

frequencies	double array of frequencies for the waveform to be evaluated at
length	integer length of all the arrays
waveform_real	complex array for the output waveform
waveform_imag	complex array for the output waveform
generation_method	String that corresponds to the generation method - MUST BE SPELLED EXACTLY
parameters	structure containing all the source parameters

Index