

## GW Analysis Tools

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Gravitational Waves Analysis Tools</b>	<b>1</b>
1.1	Software . . . . .	1
1.2	Installation . . . . .	1
1.3	Functionality . . . . .	1
1.3.1	Generation . . . . .	1
1.3.2	Gravity . . . . .	1
1.3.3	Analysis . . . . .	1
1.3.4	Routines . . . . .	1
<b>2</b>	<b>gw_analysis_tools</b>	<b>3</b>
<b>3</b>	<b>Hierarchical Index</b>	<b>5</b>
3.1	Class Hierarchy . . . . .	5
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>9</b>
5.1	File List . . . . .	9

<b>6</b>	<b>Class Documentation</b>	<b>11</b>
6.1	fftw_outline Struct Reference	11
6.2	gen_params Struct Reference	11
6.2.1	Member Data Documentation	11
6.2.1.1	betappe	12
6.2.1.2	bppe	12
6.2.1.3	incl_angle	12
6.2.1.4	Luminosity_Distance	12
6.2.1.5	mass1	12
6.2.1.6	mass2	12
6.2.1.7	NSflag	12
6.2.1.8	phic	12
6.2.1.9	spin1	13
6.2.1.10	spin2	13
6.2.1.11	tc	13
6.2.1.12	theta	13
6.3	IMRPhenomD< T > Class Template Reference	13
6.3.1	Member Function Documentation	16
6.3.1.1	amp_ins()	16
6.3.1.2	amp_int()	16
6.3.1.3	amp_mr()	16
6.3.1.4	amplitude_tape()	16
6.3.1.5	assign_nonstatic_pn_phase_coeff()	17
6.3.1.6	assign_nonstatic_pn_phase_coeff_deriv()	17
6.3.1.7	build_amp()	17
6.3.1.8	build_phase()	18
6.3.1.9	calculate_delta_parameter_0()	18
6.3.1.10	calculate_delta_parameter_1()	18
6.3.1.11	calculate_delta_parameter_2()	19
6.3.1.12	calculate_delta_parameter_3()	19

6.3.1.13	<a href="#">calculate_delta_parameter_4()</a>	19
6.3.1.14	<a href="#">change_parameter_basis()</a>	20
6.3.1.15	<a href="#">construct_amplitude()</a>	20
6.3.1.16	<a href="#">construct_amplitude_derivative()</a>	20
6.3.1.17	<a href="#">construct_phase()</a>	21
6.3.1.18	<a href="#">construct_phase_derivative()</a>	21
6.3.1.19	<a href="#">construct_waveform()</a> [1/2]	22
6.3.1.20	<a href="#">construct_waveform()</a> [2/2]	22
6.3.1.21	<a href="#">Damp_ins()</a>	24
6.3.1.22	<a href="#">Damp_mr()</a>	24
6.3.1.23	<a href="#">Dphase_ins()</a>	24
6.3.1.24	<a href="#">Dphase_int()</a>	25
6.3.1.25	<a href="#">Dphase_mr()</a>	25
6.3.1.26	<a href="#">fpeak()</a>	25
6.3.1.27	<a href="#">phase_connection_coefficients()</a>	25
6.3.1.28	<a href="#">phase_ins()</a>	26
6.3.1.29	<a href="#">phase_int()</a>	26
6.3.1.30	<a href="#">phase_mr()</a>	26
6.3.1.31	<a href="#">phase_tape()</a>	26
6.3.1.32	<a href="#">post_merger_variables()</a>	27
6.3.1.33	<a href="#">precalc_powers_ins()</a>	27
6.3.1.34	<a href="#">precalc_powers_ins_amp()</a>	27
6.3.1.35	<a href="#">precalc_powers_ins_phase()</a>	28
6.3.1.36	<a href="#">precalc_powers_PI()</a>	28
6.4	<a href="#">IMRPhenomPv2&lt; T &gt; Class Template Reference</a>	28
6.5	<a href="#">lambda_parameters&lt; T &gt; Struct Template Reference</a>	29
6.6	<a href="#">ppE_IMRPhenomD_IMR&lt; T &gt; Class Template Reference</a>	30
6.6.1	<a href="#">Detailed Description</a>	31
6.6.2	<a href="#">Member Function Documentation</a>	31
6.6.2.1	<a href="#">amplitude_tape()</a>	31

6.6.2.2	<a href="#">construct_amplitude_derivative()</a>	32
6.6.2.3	<a href="#">construct_phase_derivative()</a>	32
6.6.2.4	<a href="#">Dphase_int()</a>	33
6.6.2.5	<a href="#">Dphase_mr()</a>	33
6.6.2.6	<a href="#">phase_int()</a>	33
6.6.2.7	<a href="#">phase_mr()</a>	34
6.6.2.8	<a href="#">phase_tape()</a>	34
6.7	<a href="#">ppE_IMRPhenomD_Inspiral&lt; T &gt; Class Template Reference</a>	34
6.7.1	<a href="#">Detailed Description</a>	36
6.7.2	<a href="#">Member Function Documentation</a>	36
6.7.2.1	<a href="#">amplitude_tape()</a>	36
6.7.2.2	<a href="#">construct_amplitude_derivative()</a>	36
6.7.2.3	<a href="#">construct_phase_derivative()</a>	37
6.7.2.4	<a href="#">Dphase_ins()</a>	38
6.7.2.5	<a href="#">phase_tape()</a>	38
6.8	<a href="#">source_parameters&lt; T &gt; Struct Template Reference</a>	38
6.8.1	<a href="#">Member Function Documentation</a>	39
6.8.1.1	<a href="#">populate_source_parameters()</a>	39
6.8.2	<a href="#">Member Data Documentation</a>	40
6.8.2.1	<a href="#">chi_a</a>	40
6.8.2.2	<a href="#">chi_eff</a>	40
6.8.2.3	<a href="#">chi_pn</a>	40
6.8.2.4	<a href="#">chi_s</a>	40
6.8.2.5	<a href="#">chirpmass</a>	40
6.8.2.6	<a href="#">delta_mass</a>	41
6.8.2.7	<a href="#">DL</a>	41
6.8.2.8	<a href="#">eta</a>	41
6.8.2.9	<a href="#">f1</a>	41
6.8.2.10	<a href="#">f1_phase</a>	41
6.8.2.11	<a href="#">f2_phase</a>	41

6.8.2.12	f3	41
6.8.2.13	fdamp	42
6.8.2.14	fRD	42
6.8.2.15	M	42
6.8.2.16	mass1	42
6.8.2.17	mass2	42
6.8.2.18	phic	42
6.8.2.19	spin1x	42
6.8.2.20	spin1y	43
6.8.2.21	spin1z	43
6.8.2.22	spin2x	43
6.8.2.23	spin2y	43
6.8.2.24	spin2z	43
6.8.2.25	tc	43
6.9	useful_powers< T > Struct Template Reference	44
6.9.1	Detailed Description	44
<b>7</b>	<b>File Documentation</b>	<b>45</b>
7.1	include/fisher.h File Reference	45
7.1.1	Function Documentation	46
7.1.1.1	fisher()	46
7.2	include/IMRPhenomD.h File Reference	46
7.2.1	Detailed Description	47
7.2.2	Variable Documentation	47
7.2.2.1	lambda_num_params	48
7.3	include/mcmc_routines.h File Reference	48
7.3.1	Function Documentation	49
7.3.1.1	maximized_coal_log_likelihood_IMRPhenomD() [1/3]	49
7.3.1.2	maximized_coal_log_likelihood_IMRPhenomD() [2/3]	50
7.3.1.3	maximized_coal_log_likelihood_IMRPhenomD() [3/3]	50
7.3.1.4	maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [1/3]	51

7.3.1.5	maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [2/3]	51
7.3.1.6	maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [3/3]	51
7.4	include/noise_util.h File Reference	52
7.4.1	Function Documentation	53
7.4.1.1	populate_noise()	53
7.5	include/ppE_IMRPhenomD.h File Reference	53
7.6	include/util.h File Reference	54
7.6.1	Detailed Description	56
7.6.2	Function Documentation	56
7.6.2.1	calculate_chirpmass()	56
7.6.2.2	calculate_mass1()	56
7.6.2.3	calculate_mass2()	57
7.6.2.4	simpsons_sum()	57
7.6.2.5	trapezoidal_sum()	57
7.6.2.6	trapezoidal_sum_uniform()	57
7.6.3	Variable Documentation	57
7.6.3.1	c	58
7.6.3.2	G	58
7.6.3.3	gamma_E	58
7.6.3.4	MPC_SEC	58
7.6.3.5	MSOL_SEC	58
7.7	include/waveform_generator.h File Reference	58
7.8	README.dox File Reference	59
7.8.1	Detailed Description	59
7.9	src/fisher.cpp File Reference	59
7.9.1	Detailed Description	60
7.9.2	Function Documentation	60
7.9.2.1	fisher()	60
7.10	src/IMRPhenomD.cpp File Reference	61
7.10.1	Detailed Description	61



7.11	src/mcmc_routines.cpp File Reference	61
7.11.1	Detailed Description	62
7.11.2	Function Documentation	62
7.11.2.1	maximized_coal_log_likelihood_IMRPhenomD() [1/3]	63
7.11.2.2	maximized_coal_log_likelihood_IMRPhenomD() [2/3]	63
7.11.2.3	maximized_coal_log_likelihood_IMRPhenomD() [3/3]	63
7.11.2.4	maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [1/3]	64
7.11.2.5	maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [2/3]	64
7.11.2.6	maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [3/3]	65
7.12	src/noise_util.cpp File Reference	65
7.12.1	Detailed Description	66
7.12.2	Function Documentation	66
7.12.2.1	populate_noise()	66
7.13	src/ppE_IMRPhenomD.cpp File Reference	67
7.13.1	Detailed Description	67
7.14	src/util.cpp File Reference	67
7.14.1	Detailed Description	68
7.14.2	Function Documentation	68
7.14.2.1	calculate_chirpmass()	68
7.14.2.2	calculate_mass1()	69
7.14.2.3	calculate_mass2()	69
7.15	src/waveform_generator.cpp File Reference	69
7.15.1	Detailed Description	70
7.15.2	Function Documentation	70
7.15.2.1	fourier_amplitude()	70
7.15.2.2	fourier_phase()	71
7.15.2.3	fourier_waveform() [1/2]	71
7.15.2.4	fourier_waveform() [2/2]	71



# 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 environment 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 calculation

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:

fftw_outline . . . . .	11
gen_params . . . . .	11
IMRPhenomD< T > . . . . .	13
IMRPhenomPv2< T > . . . . .	28
ppE_IMRPhenomD_Inspiral< T > . . . . .	34
ppE_IMRPhenomD_IMR< T > . . . . .	30
lambda_parameters< T > . . . . .	29
source_parameters< T > . . . . .	38
useful_powers< T > . . . . .	44





## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">fftw_outline</a>	11
<a href="#">gen_params</a>	11
<a href="#">IMRPhenomD&lt; T &gt;</a>	13
<a href="#">IMRPhenomPv2&lt; T &gt;</a>	28
<a href="#">lambda_parameters&lt; T &gt;</a>	29
<a href="#">ppE_IMRPhenomD_IMR&lt; T &gt;</a>	30
<a href="#">ppE_IMRPhenomD_Inspiral&lt; T &gt;</a>	34
<a href="#">source_parameters&lt; T &gt;</a>	38
<a href="#">useful_powers&lt; T &gt;</a>	
To speed up calculations within the for loops, we pre-calculate reoccurring powers of M*F and Pi, since the pow() function is prohibitively slow	44



## Chapter 5

# File Index

### 5.1 File List

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

include/fisher.h . . . . .	45
include/IMRPhenomD.h . . . . .	46
include/IMRPhenomP.h . . . . .	??
include/mcmc_routines.h . . . . .	48
include/noise_util.h . . . . .	52
include/ppE_IMRPhenomD.h . . . . .	53
include/util.h . . . . .	54
include/waveform_generator.h . . . . .	58
include/waveform_util.h . . . . .	??
src/fisher.cpp . . . . .	59
src/IMRPhenomD.cpp . . . . .	61
src/mcmc_routines.cpp . . . . .	61
src/noise_util.cpp . . . . .	65
src/ppE_IMRPhenomD.cpp . . . . .	67
src/util.cpp . . . . .	67
src/waveform_generator.cpp . . . . .	69



## 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
```

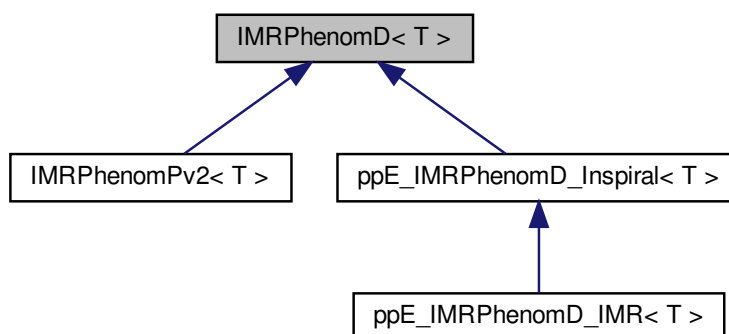
spherical angles for the source location relative to the detector

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

- include/[util.h](#)

## 6.3 IMRPhenomD&lt; T &gt; 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)
 

*Convenience 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 coefficients for the amplitude.*
- virtual void **assign\_static\_pn\_phase\_coeff** ([source\\_parameters](#)< T > \*source\_param, T \*coeff)
 

*Calculates the static PN coefficients for the phase - coefficients 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  $A/A_0$  for frequency  $f$  with precomputed powers of MF and PI.

return a T

additional argument contains useful powers of MF and PI in structure useful\_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  $A/A_0$  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  $A/A_0$  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

<i>input_params</i>	source parameters structure of the desired source
<i>tape</i>	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]
```

Convenience method to change parameter basis between common Fisher parameters and the intrinsic parameters of [IMRPhenomD](#).

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

##### Parameters

<i>old_param</i>	array of old params, order {A0, tc, phic, chirpmass, eta, spin1, spin2}
<i>new_param</i>	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

<i>frequencies</i>	T array of frequencies the waveform is to be evaluated at
<i>length</i>	integer length of the input array of frequencies and the output array
<i>amplitude</i>	output T array for the amplitude
<i>params</i>	Structure of source parameters to be initialized 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

<i>frequencies</i>	input array of frequency
<i>length</i>	length of the frequency array
<i>amplitude_derivative</i>	< dimension of the fisher output array for all the derivatives double[dimension][length]
<i>input_params</i>	Source parameters structure for the source
<i>tapes</i>	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

<i>frequencies</i>	T array of frequencies the waveform is to be evaluated at
<i>length</i>	integer length of the input and output arrays
<i>phase</i>	output T array for the phasee
<i>params</i>	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

<i>frequencies</i>	input array of frequency
<i>length</i>	length of the frequency array
<i>phase_derivative</i>	< dimension of the fisher output array for all the derivatives double[dimension][length]
<i>input_params</i>	Source parameters structure for the source
<i>tapes</i>	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

<i>frequencies</i>	T array of frequencies the waveform is to be evaluated at
<i>length</i>	integer length of the array of frequencies and the waveform
<i>waveform</i>	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

<i>frequency</i>	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  $A/A_0$  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  $A/A_0$  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

<i>input_params</i>	source parameters structure of the desired source
<i>tape</i>	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.

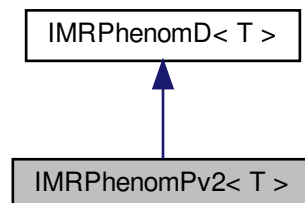
It seems the pow() function is very slow, so to speed things up, powers of PI will be precomputed and passed to the functions within the frequency loops

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

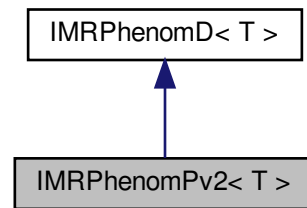
- include/IMRPhenomD.h
- src/IMRPhenomD.cpp

## 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]

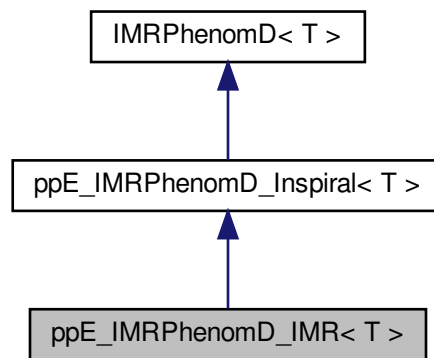
The documentation for this struct was generated from the following file:

- `include/IMRPhenomD.h`

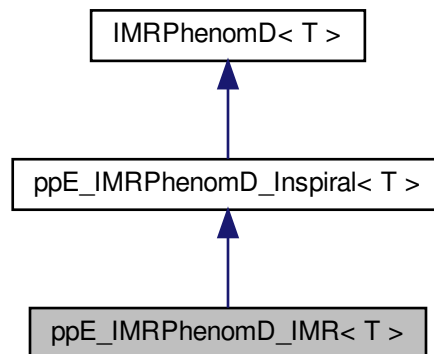
## 6.6 ppE\_IMRPhenomD\_IMR< T > Class Template Reference

```
#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

<i>input_params</i>	source parameters structure of the desired source
<i>tape</i>	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

<i>frequencies</i>	input array of frequency
<i>length</i>	length of the frequency array
<i>amplitude_derivative</i>	< dimension of the fisher output array for all the derivatives double[dimension][length]
<i>input_params</i>	Source parameters structure for the source
<i>tapes</i>	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

<i>frequencies</i>	input array of frequency
<i>length</i>	length of the frequency array
<i>phase_derivative</i>	< dimension of the fisher output array for all the derivatives double[dimension][length]
<i>input_params</i>	Source parameters structure for the source
<i>tapes</i>	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

<i>input_params</i>	source parameters structure of the desired source
<i>tape</i>	tape ids

Reimplemented from [ppE\\_IMRPhenomD\\_Inspiral< T >](#).

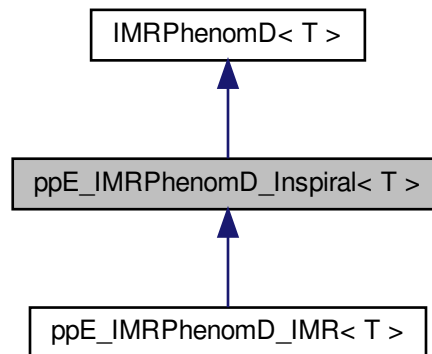
The documentation for this class was generated from the following files:

- [include/ppE\\_IMRPhenomD.h](#)
- [src/ppE\\_IMRPhenomD.cpp](#)

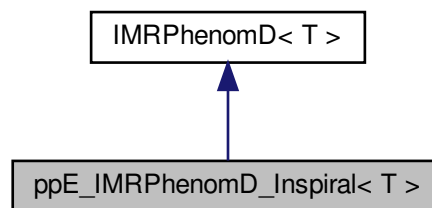
## 6.7 ppE\_IMRPhenomD\_Inspiral< T > Class Template Reference

```
#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

<i>input_params</i>	source parameters structure of the desired source
<i>tape</i>	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

<i>frequencies</i>	input array of frequency
<i>length</i>	length of the frequency array
<i>amplitude_derivative</i>	< dimension of the fisher output array for all the derivatives double[dimension][length]
<i>input_params</i>	Source parameters structure for the source
<i>tapes</i>	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

<i>frequencies</i>	input array of frequency
<i>length</i>	length of the frequency array
<i>phase_derivative</i>	< dimension of the fisher output array for all the derivatives double[dimension][length]
<i>input_params</i>	Source parameters structure for the source
<i>tapes</i>	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

<i>input_params</i>	source parameters structure of the desired source
<i>tape</i>	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 relevant source parameters

## Parameters

<i>mass1</i>	mass of the larger body - in Solar Masses
<i>mass2</i>	mass of the smaller body - in Solar Masses
<i>Luminosity_Distance</i>	Luminosity Distance in Mpc
<i>spin2</i>	spin vector of the larger body {sx,sy,sz}
<i>phi_c</i>	spin vector of the smaller body {sx,sy,sz}
<i>t_c</i>	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 combination

#### 6.8.2.7 DL

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

Luminosity 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 reoccurring powers of  $M \cdot F$  and  $\pi$ , since the `pow()` function is prohibitively 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 reoccurring powers of  $M \cdot F$  and  $\pi$ , since the `pow()` function is prohibitively 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](#)

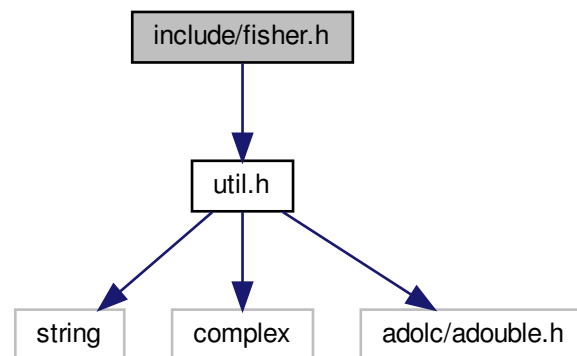
## Chapter 7

# File Documentation

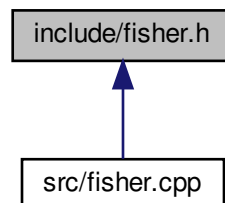
### 7.1 include/fisher.h File Reference

```
#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

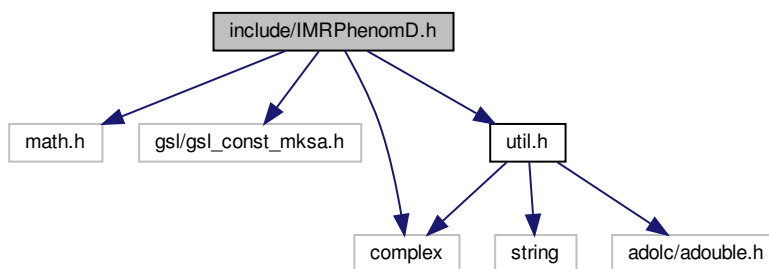
<i>length</i>	if 0, standard frequency range for the detector is used
<i>output</i>	double [dimension][dimension]
<i>amp_tapes</i>	if speed is required, precomputed tapes can be used - assumed the user knows what they're doing, no checks done here to make sure that the number of tapes matches the requirement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

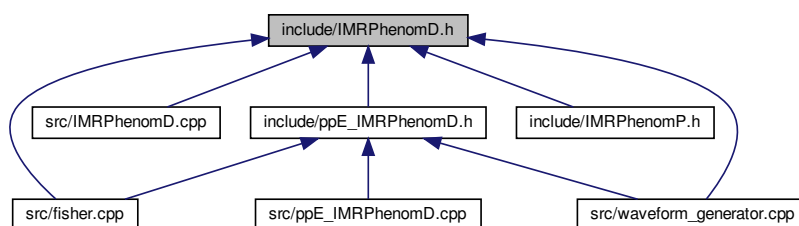
```
#include <math.h>
#include <gsl/gsl_const_mksa.h>
#include <complex>
#include "util.h"
```



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]
```

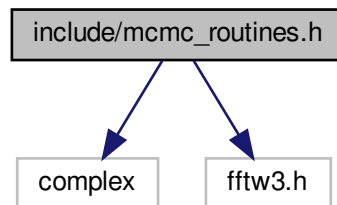
Numerically calibrated parameters from arXiv:1508.07253 see the table in the data directory for labeled version

## 7.3 include/mcmc\_routines.h File Reference

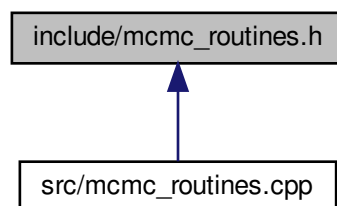
```
#include <complex>
```

```
#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  $\phi_{hc}$  and  $t_c$ .*
- 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  $\phi_{hc}$  and  $t_c$ .

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

## Parameters

<i>chirp</i> mass	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

<i>chirp</i> mass	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

<i>chirp</i> mass	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

<i>chirpmass</i>	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

<i>chirpmass</i>	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 )

```

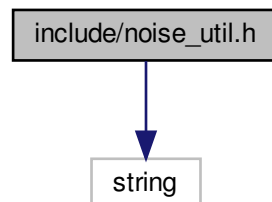
#### Parameters

<i>chirpmass</i>	in solar masses
------------------	-----------------

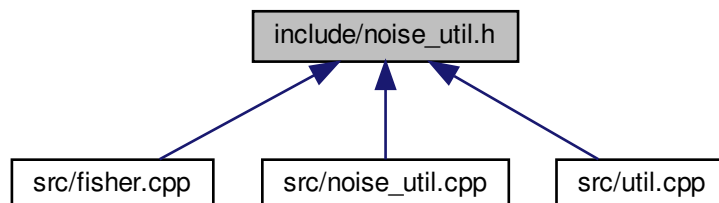
## 7.4 include/noise\_util.h File Reference

```
#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

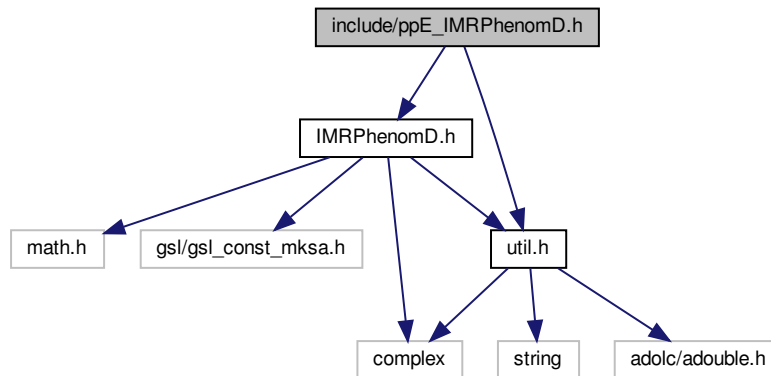
#### Parameters

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

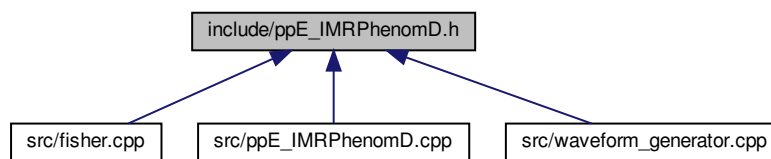
## 7.5 include/ppE\_IMRPhenomD.h File Reference

```
#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

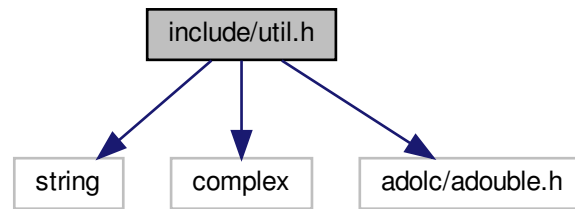
```

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

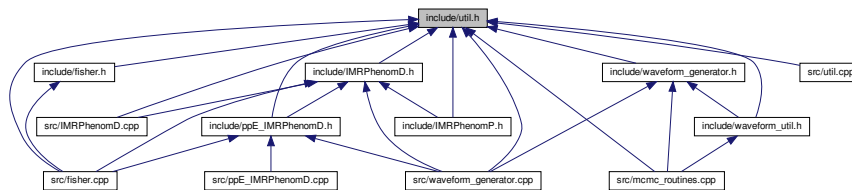
```



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 reoccurring powers of  $M \cdot F$  and  $P_i$ , since the `pow()` function is prohibitively 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 = 0.5772156649015328606065120900824024310421`
- `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  $\text{m}^3/(\text{s}^2 \text{ SolMass})$

### 7.6.3.3 gamma\_E

```
const double gamma_E = 0.5772156649015328606065120900824024310421
```

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

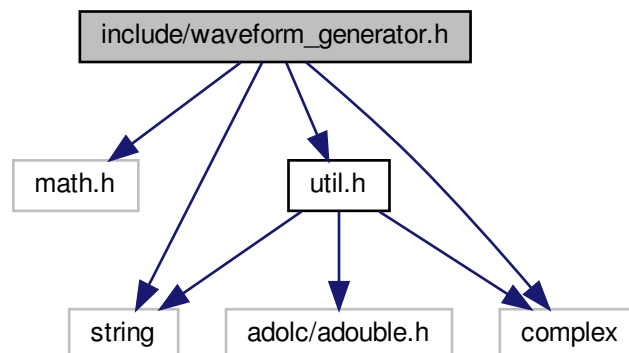
```
const double MSOL_SEC = 492549095.e-14
```

$G/c^3$  seconds per solar mass

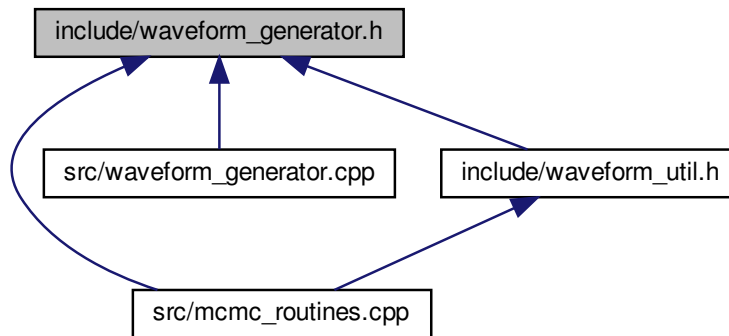
## 7.7 include/waveform\_generator.h File Reference

```
#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

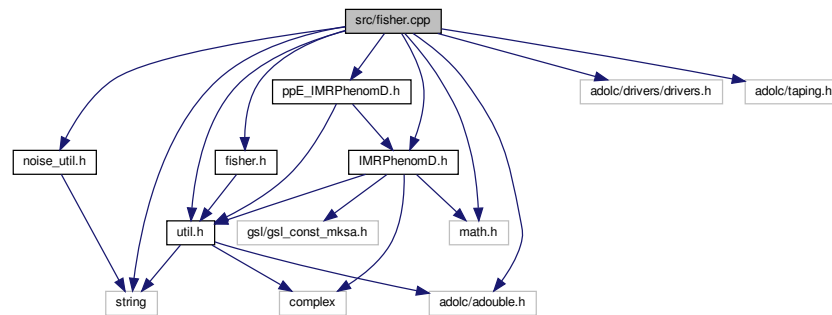
```

#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

<i>length</i>	if 0, standard frequency range for the detector is used
---------------	---

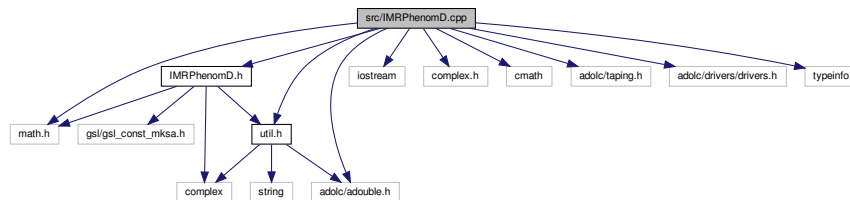
## Parameters

<i>output</i>	double [dimension][dimension]
<i>amp_tapes</i>	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
<i>phase_tapes</i>	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

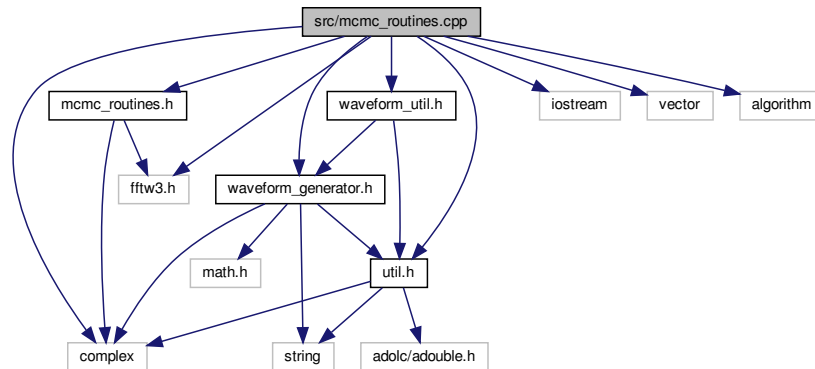
File that includes all the low level functions that go into constructing the waveform

## 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  $\phi_{\text{hc}}$  and  $t_c$ .*
- 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  $\phi_{\text{hc}}$  and  $t_c$ .

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

## Parameters

<i>chirpmass</i>	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

<i>chirpmass</i>	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

<i>chirpmass</i>	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

<i>chirpmass</i>	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**

<i>chirpmass</i>	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**

<i>chirpmass</i>	in solar masses
------------------	-----------------

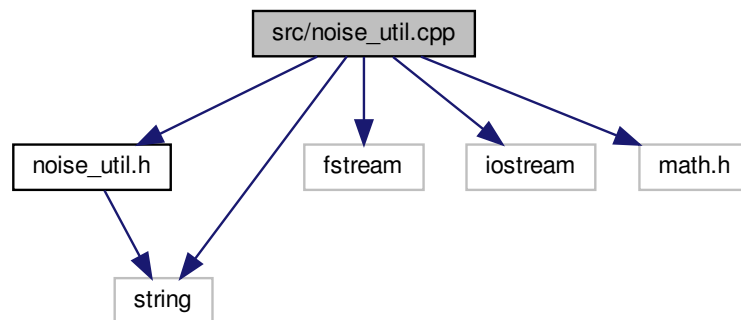
**7.12 src/noise\_util.cpp File Reference**

```

#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

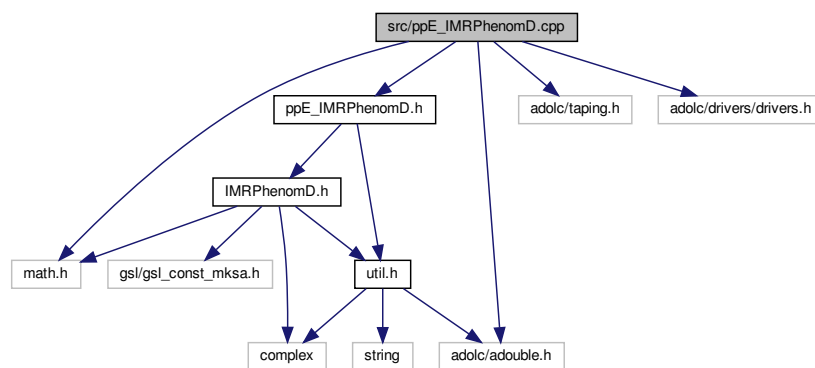
Detectors include: aLIGO\_analytic, Hanford\_O1\_fitted

## Parameters

<i>frequencies</i>	double array of frquencies (NULL)
<i>detector</i>	String to designate the detector noise curve to be used
<i>noise_root</i>	ouptput double array for the square root of the PSD of the noise of the specified detector
<i>length</i>	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

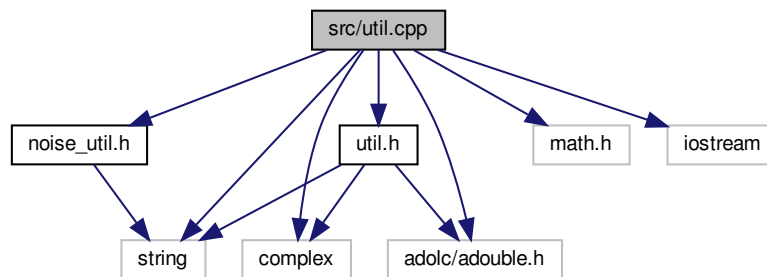
File for the implementation of the ppE formalism for testing GR

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 )
```

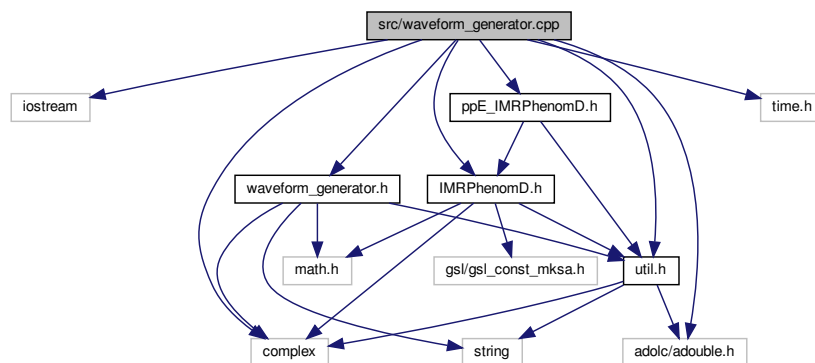
Calculates the smaller mass given a chirp mass and symmetric mass ratio.

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

<i>frequencies</i>	double array of frequencies for the waveform to be evaluated at
<i>length</i>	integer length of all the arrays
<i>amplitude</i>	output array for the amplitude
<i>generation_method</i>	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

<i>frequencies</i>	double array of frequencies for the waveform to be evaluated at
<i>length</i>	integer length of all the arrays
<i>phase</i>	output array for the phase
<i>generation_method</i>	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

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

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

# Index

amp\_ins  
    IMRPhenomD, [16](#)

amp\_int  
    IMRPhenomD, [16](#)

amp\_mr  
    IMRPhenomD, [16](#)

amplitude\_tape  
    IMRPhenomD, [16](#)  
    ppE\_IMRPhenomD\_IMR, [31](#)  
    ppE\_IMRPhenomD\_Inspiral, [36](#)

assign\_nonstatic\_pn\_phase\_coeff  
    IMRPhenomD, [17](#)

assign\_nonstatic\_pn\_phase\_coeff\_deriv  
    IMRPhenomD, [17](#)

betappe  
    gen\_params, [11](#)

bppe  
    gen\_params, [12](#)

build\_amp  
    IMRPhenomD, [17](#)

build\_phase  
    IMRPhenomD, [17](#)

c  
    util.h, [57](#)

calculate\_chirpmass  
    util.cpp, [68](#)  
    util.h, [56](#)

calculate\_delta\_parameter\_0  
    IMRPhenomD, [18](#)

calculate\_delta\_parameter\_1  
    IMRPhenomD, [18](#)

calculate\_delta\_parameter\_2  
    IMRPhenomD, [18](#)

calculate\_delta\_parameter\_3  
    IMRPhenomD, [19](#)

calculate\_delta\_parameter\_4  
    IMRPhenomD, [19](#)

calculate\_mass1  
    util.cpp, [68](#)  
    util.h, [56](#)

calculate\_mass2  
    util.cpp, [69](#)  
    util.h, [56](#)

change\_parameter\_basis  
    IMRPhenomD, [19](#)

chi\_a  
    source\_parameters, [40](#)

chi\_eff  
    source\_parameters, [40](#)

chi\_pn  
    source\_parameters, [40](#)

chi\_s  
    source\_parameters, [40](#)

chirpmass  
    source\_parameters, [40](#)

construct\_amplitude  
    IMRPhenomD, [20](#)

construct\_amplitude\_derivative  
    IMRPhenomD, [20](#)  
    ppE\_IMRPhenomD\_IMR, [31](#)  
    ppE\_IMRPhenomD\_Inspiral, [36](#)

construct\_phase  
    IMRPhenomD, [21](#)

construct\_phase\_derivative  
    IMRPhenomD, [21](#)  
    ppE\_IMRPhenomD\_IMR, [32](#)  
    ppE\_IMRPhenomD\_Inspiral, [37](#)

construct\_waveform  
    IMRPhenomD, [22](#)

Damp\_ins  
    IMRPhenomD, [24](#)

Damp\_mr  
    IMRPhenomD, [24](#)

delta\_mass  
    source\_parameters, [40](#)

DL  
    source\_parameters, [41](#)

Dphase\_ins  
    IMRPhenomD, [24](#)  
    ppE\_IMRPhenomD\_Inspiral, [37](#)

Dphase\_int  
    IMRPhenomD, [24](#)  
    ppE\_IMRPhenomD\_IMR, [33](#)

Dphase\_mr  
    IMRPhenomD, [25](#)  
    ppE\_IMRPhenomD\_IMR, [33](#)

eta  
    source\_parameters, [41](#)

f1  
    source\_parameters, [41](#)

f1\_phase  
    source\_parameters, [41](#)

f2\_phase  
    source\_parameters, [41](#)

f3

- source\_parameters, 41
- fRD
  - source\_parameters, 42
- fdamp
  - source\_parameters, 41
- fftw\_outline, 11
- fisher
  - fisher.cpp, 60
  - fisher.h, 46
- fisher.cpp
  - fisher, 60
- fisher.h
  - fisher, 46
- fourier\_amplitude
  - waveform\_generator.cpp, 70
- fourier\_phase
  - waveform\_generator.cpp, 70
- fourier\_waveform
  - waveform\_generator.cpp, 71
- fpeak
  - IMRPhenomD, 25
- G
  - util.h, 58
- gamma\_E
  - util.h, 58
- gen\_params, 11
  - betappe, 11
  - bppe, 12
  - incl\_angle, 12
  - Luminosity\_Distance, 12
  - mass1, 12
  - mass2, 12
  - NSflag, 12
  - phic, 12
  - spin1, 12
  - spin2, 13
  - tc, 13
  - theta, 13
- IMRPhenomD< T >, 13
- IMRPhenomD.h
  - lambda\_num\_params, 47
- IMRPhenomPv2< T >, 28
- IMRPhenomD
  - amp\_ins, 16
  - amp\_int, 16
  - amp\_mr, 16
  - amplitude\_tape, 16
  - assign\_nonstatic\_pn\_phase\_coeff, 17
  - assign\_nonstatic\_pn\_phase\_coeff\_deriv, 17
  - build\_amp, 17
  - build\_phase, 17
  - calculate\_delta\_parameter\_0, 18
  - calculate\_delta\_parameter\_1, 18
  - calculate\_delta\_parameter\_2, 18
  - calculate\_delta\_parameter\_3, 19
  - calculate\_delta\_parameter\_4, 19
  - change\_parameter\_basis, 19
  - construct\_amplitude, 20
  - construct\_amplitude\_derivative, 20
  - construct\_phase, 21
  - construct\_phase\_derivative, 21
  - construct\_waveform, 22
  - Damp\_ins, 24
  - Damp\_mr, 24
  - Dphase\_ins, 24
  - Dphase\_int, 24
  - Dphase\_mr, 25
  - fpeak, 25
  - phase\_connection\_coefficients, 25
  - phase\_ins, 25
  - phase\_int, 26
  - phase\_mr, 26
  - phase\_tape, 26
  - post\_merger\_variables, 27
  - precalc\_powers\_PI, 28
  - precalc\_powers\_ins, 27
  - precalc\_powers\_ins\_amp, 27
  - precalc\_powers\_ins\_phase, 27
- incl\_angle
  - gen\_params, 12
- include/IMRPhenomD.h, 46
- include/fisher.h, 45
- include/mcmc\_routines.h, 48
- include/noise\_util.h, 52
- include/ppE\_IMRPhenomD.h, 53
- include/util.h, 54
- include/waveform\_generator.h, 58
- lambda\_num\_params
  - IMRPhenomD.h, 47
- lambda\_parameters< T >, 29
- Luminosity\_Distance
  - gen\_params, 12
- M
  - source\_parameters, 42
- MPC\_SEC
  - util.h, 58
- MSOL\_SEC
  - util.h, 58
- mass1
  - gen\_params, 12
  - source\_parameters, 42
- mass2
  - gen\_params, 12
  - source\_parameters, 42
- maximized\_coal\_log\_likelihood\_IMRPhenomD\_Full\_↔
  - Param
    - mcmc\_routines.cpp, 64, 65
    - mcmc\_routines.h, 50, 51
- maximized\_coal\_log\_likelihood\_IMRPhenomD
  - mcmc\_routines.cpp, 62, 63
  - mcmc\_routines.h, 49, 50
- mcmc\_routines.cpp
  - maximized\_coal\_log\_likelihood\_IMRPhenomD\_↔
    - Full\_Param, 64, 65

- maximized\_coal\_log\_likelihood\_IMRPhenomD, 62, 63
- mcmc\_routines.h
  - maximized\_coal\_log\_likelihood\_IMRPhenomD\_↔ Full\_Param, 50, 51
  - maximized\_coal\_log\_likelihood\_IMRPhenomD, 49, 50
- NSflag
  - gen\_params, 12
- noise\_util.cpp
  - populate\_noise, 66
- noise\_util.h
  - populate\_noise, 53
- phase\_connection\_coefficients
  - IMRPhenomD, 25
- phase\_ins
  - IMRPhenomD, 25
- phase\_int
  - IMRPhenomD, 26
  - ppE\_IMRPhenomD\_IMR, 33
- phase\_mr
  - IMRPhenomD, 26
  - ppE\_IMRPhenomD\_IMR, 33
- phase\_tape
  - IMRPhenomD, 26
  - ppE\_IMRPhenomD\_IMR, 34
  - ppE\_IMRPhenomD\_Inspiral, 38
- phic
  - gen\_params, 12
  - source\_parameters, 42
- populate\_noise
  - noise\_util.cpp, 66
  - noise\_util.h, 53
- populate\_source\_parameters
  - source\_parameters, 39
- post\_merger\_variables
  - IMRPhenomD, 27
- ppE\_IMRPhenomD\_IMR< T >, 30
- ppE\_IMRPhenomD\_IMR
  - amplitude\_tape, 31
  - construct\_amplitude\_derivative, 31
  - construct\_phase\_derivative, 32
  - Dphase\_int, 33
  - Dphase\_mr, 33
  - phase\_int, 33
  - phase\_mr, 33
  - phase\_tape, 34
- ppE\_IMRPhenomD\_Inspiral
  - amplitude\_tape, 36
  - construct\_amplitude\_derivative, 36
  - construct\_phase\_derivative, 37
  - Dphase\_ins, 37
  - phase\_tape, 38
- ppE\_IMRPhenomD\_Inspiral< T >, 34
- precalc\_powers\_PI
  - IMRPhenomD, 28
- precalc\_powers\_ins
  - IMRPhenomD, 27
- precalc\_powers\_ins\_amp
  - IMRPhenomD, 27
- precalc\_powers\_ins\_phase
  - IMRPhenomD, 27
- README.dox, 59
- simpsons\_sum
  - util.h, 57
- source\_parameters
  - chi\_a, 40
  - chi\_eff, 40
  - chi\_pn, 40
  - chi\_s, 40
  - chirpmass, 40
  - delta\_mass, 40
  - DL, 41
  - eta, 41
  - f1, 41
  - f1\_phase, 41
  - f2\_phase, 41
  - f3, 41
  - fRD, 42
  - fdamp, 41
  - M, 42
  - mass1, 42
  - mass2, 42
  - phic, 42
  - populate\_source\_parameters, 39
  - spin1x, 42
  - spin1y, 42
  - spin1z, 43
  - spin2x, 43
  - spin2y, 43
  - spin2z, 43
  - tc, 43
- source\_parameters< T >, 38
- spin1
  - gen\_params, 12
- spin1x
  - source\_parameters, 42
- spin1y
  - source\_parameters, 42
- spin1z
  - source\_parameters, 43
- spin2
  - gen\_params, 13
- spin2x
  - source\_parameters, 43
- spin2y
  - source\_parameters, 43
- spin2z
  - source\_parameters, 43
- src/IMRPhenomD.cpp, 61
- src/fisher.cpp, 59
- src/mcmc\_routines.cpp, 61
- src/noise\_util.cpp, 65
- src/ppE\_IMRPhenomD.cpp, 67

- src/util.cpp, [67](#)
- src/waveform\_generator.cpp, [69](#)
- tc
  - gen\_params, [13](#)
  - source\_parameters, [43](#)
- theta
  - gen\_params, [13](#)
- trapezoidal\_sum
  - util.h, [57](#)
- trapezoidal\_sum\_uniform
  - util.h, [57](#)
- useful\_powers< T >, [44](#)
- util.cpp
  - calculate\_chirpmass, [68](#)
  - calculate\_mass1, [68](#)
  - calculate\_mass2, [69](#)
- util.h
  - c, [57](#)
  - calculate\_chirpmass, [56](#)
  - calculate\_mass1, [56](#)
  - calculate\_mass2, [56](#)
  - G, [58](#)
  - gamma\_E, [58](#)
  - MPC\_SEC, [58](#)
  - MSOL\_SEC, [58](#)
  - simpsons\_sum, [57](#)
  - trapezoidal\_sum, [57](#)
  - trapezoidal\_sum\_uniform, [57](#)
- waveform\_generator.cpp
  - fourier\_amplitude, [70](#)
  - fourier\_phase, [70](#)
  - fourier\_waveform, [71](#)