GW Analysis Tools

Generated by Doxygen 1.8.15

Chapter 1

Gravitational Waves Analysis Tools

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

1.1 Compatibility

Known to work with gcc/g++-7

Known to work with gcc/g++-9

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

1.2 Required Software

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

Required non-standard Python packages: Cython

Required non-standard packages for documentation: Doxygen

1.3 Current Development

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

To do:

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

1.4 Installation

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

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

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

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

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

1.5 Supported Functionality

1.5.1 Waveform Generation

IMRPhenomD, IMRPhenomPv2

1.5.2 Modified Gravity

ppE_IMRPhenomD_Inspiral ppE_IMRPhenomD_IMR ppE_IMRPhenomPv2_Inspiral ppE_IMRPhenomPv2_IMR

1.5.3 Fisher Analysis

utilizes the above waveform templates

1.5.4 MCMC Routines

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

Includes log likelihood caclulation for implementation in other samplers.

1.6 Usage

1.6.1 Environment variables

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

1.6.2 Include

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

1.6.3 Link

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

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

For Cuda code: use -lcuda -lcudart

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

1.6.4 Python Importable Code

Two modules currently available:

1.6.4.1 gw_analysis_tools_py.mcmc_routines_ext

mcmc_routines_ext.pyx wraps the log_likelihood functions in mcmc_routines.cpp

1.6.4.2 gw_analysis_tools_py.waveform_generator_ext

waveform_generator_ext.pyx wraps the fourier_waveform function in waveform_generator.cpp

Also contains the SNR calculation function

1.6.4.3 Custom Waveforms

If adding waveforms and to have full accesibility:

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

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

For autodiff Fishers – write the class as a template with double and adouble types for all variables. Then write the necessary fisher subroutines (see fisher file to determine whats necessary)

For numerical Fishers – write finite difference method, following the template of the previous waveforms

For MCMC sampling – write mcmc_fisher_wrapper and mcmc_likelihood_wrapper options and write any necessary initialization in MCMC_MH_GW

Author

Scott Perkins

Contact: scottep3@illinois.edu

Chapter 2

gw_analysis_tools

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

Chapter 3

Namespace Index

3.1 Namespace List

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Namespace Documentation

6.1 waveform_generator_ext Namespace Reference

Python wrapper for the waveform generation in waveform_generator.cpp.

Classes

- class gen_params_py

  Python wrapper for the generation parameters structure, as defined in util.cpp.

Functions

- def **double** (self, double, mass1, double, mass2, double, DL, spin1, spin2, double, phic, double, tc, :1] bppe, double[::1] betappe, int Nmod, double theta, double phi, double incl_angle, double f_ref, double phiRef, bool NSflag):self.params.mass1=mass1 self.params.mass2=mass2 self.params.Luminosity↩_Distance=DL self.params.spin1=spin1 self.params.spin2=spin2 self.params.phic=phic self.params.tc=tc self.params.bppe=&bppe[0] self.params.betappe=&betappe[0] self.params.Nmod=Nmod self.params.incl↩_angle=incl_angle self.params.theta=theta self.params.phi=phi self.params.f_ref=f_ref self.params.phi↩Ref=phiRef self.params.NSflag=NSflag ##Computes the waveform in Fourier space # @param frequencies The array of frequencies to use # @param generation_method Method to use for the waveform generation # @param gen_params_py Parameters of the binary def fourier_waveform_py(double[::1] frequencies, string generation_method, gen_params_py parameters):cdef double[::1] waveform_real=np.↩ascontiguousarray(np.zeros((frequencies.size) int, dtype=np.float64)
- def **double** (:1] frequencies, string generation_method, gen_params_py parameters):cdef double[::1] amplitude=np.ascontiguousarray(np.zeros((frequencies.size) double, dtype=np.float64, frequencies, frequencies, size, amplitude, generation_method, parameters, params, :1] frequencies, string generation_method, gen_params_py parameters):cdef double[::1] phase=np.ascontiguousarray(np.zeros((frequencies.size) double, dtype=np.float64, frequencies, frequencies, size, phase, generation_method, parameters, params, :1] frequencies, string generation_method, gen_params_py parameters):cdef double[::1] waveform_plus_↩real=np.ascontiguousarray(np.zeros((frequencies.size) double, dtype=np.float64)

Variables

- **complex128_t**
- **ndim**
- **waveform**
- **dtype**
- **i** = i +1

6.1.1 Detailed Description

Python wrapper for the waveform generation in waveform_generator.cpp.

Chapter 7

Class Documentation

7.1 Comparator Class Reference

Class to facilitate the comparing of chains for priority.

Public Member Functions

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

7.1.1 Detailed Description

Class to facilitate the comparing of chains for priority.

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

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

- src/mcmc_sampler.cpp

7.2 Comparatorswap Class Reference

Public Member Functions

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

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

- src/mcmc_sampler.cpp

7.3 mcmc_routines_ext.fftw_outline_py Class Reference

Public Member Functions

- def **__init__** (self, N)
- def **__reduce__** (self)

Public Attributes

- **N**

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

- gw_analysis_tools_py/src/mcmc_routines_ext.pyx

7.4 waveform_generator_ext.gen_params_py Class Reference

Python wrapper for the generation parameters structure, as defined in util.cpp.

7.4.1 Detailed Description

Python wrapper for the generation parameters structure, as defined in util.cpp.

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

- gw_analysis_tools_py/src/waveform_generator_ext.pyx

7.5 ThreadPool Class Reference

Public Member Functions

- **ThreadPool** (std::size_t numThreads)
- void **enqueue** (int i)
- void **enqueue_swap** (int i)
- void **public_stop** ()

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

- src/mcmc_sampler.cpp

Chapter 8

File Documentation

8.1 gw_analysis_tools_py/src/mcmc_routines_ext.pyx File Reference

File that wraps the code in mcmc_gw.cpp, mcmc_sampler.cpp, mcmc_sampler_internals.cpp, autocorrelation.cpp.

Classes

- class mcmc_routines_ext.fftw_outline_py

Functions

- def **mcmc_routines_ext.write_auto_corr_file_from_data_file_py** (string, autocorr_filename, string, datafile, int, length, int, dimension, int, num_segments, double, target_corr, int, num_threads)
- def **mcmc_routines_ext.arange** (string, autocorr_filename, :1] data, int length, int dimension, int num_↩ segments, double target_corr, int num_threads):#Not ideal -- have to wrap the memview in a real c++array cdef double ∗∗temparr=< double ∗∗ >malloc(sizeof(double ∗double, length)
- def **mcmc_routines_ext.allocate_FFTW_mem_forward_py** (fftw_outline_py, plan, int, length)
- def **mcmc_routines_ext.deallocate_FFTW_mem_py** (fftw_outline_py, plan)

8.1.1 Detailed Description

File that wraps the code in mcmc_gw.cpp, mcmc_sampler.cpp, mcmc_sampler_internals.cpp, autocorrelation.cpp.

8.2 gw_analysis_tools_py/src/waveform_generator_ext.pyx File Reference

File that contains cython code to wrap the c++ library.

Classes

- class waveform_generator_ext.gen_params_py

  Python wrapper for the generation parameters structure, as defined in util.cpp.

Namespaces

- waveform_generator_ext

  Python wrapper for the waveform generation in waveform_generator.cpp.

Functions

- def **waveform_generator_ext.double** (self, double, mass1, double, mass2, double, DL, spin1, spin2, double, phic, double, tc, :1] bppe, double[::1] betappe, int Nmod, double theta, double phi, double incl_angle, double f_ref, double phiRef, bool NSflag):self.params.mass1=mass1 self.params.mass2=mass2 self.↩ params.Luminosity_Distance=DL self.params.spin1=spin1 self.params.spin2=spin2 self.params.phic=phic self.params.tc=tc self.params.bppe=&bppe[0] self.params.betappe=&betappe[0] self.params.Nmod=Nmod self.params.incl_angle=incl_angle self.params.theta=theta self.params.phi=phi self.params.f_ref=f_ref self.↩ params.phiRef=phiRef self.params.NSflag=NSflag ##Computes the waveform in Fourier space # @param frequencies The array of frequencies to use # @param generation_method Method to use for the waveform generation # @param gen_params_py Parameters of the binary def fourier_waveform_py(double[::1] frequencies, string generation_method, gen_params_py parameters):cdef double[::1] waveform_real=np.↩ ascontiguousarray(np.zeros((frequencies.size) int, dtype=np.float64)
- def **waveform_generator_ext.double** (:1] frequencies, string generation_method, gen_params↩ _py parameters):cdef double[::1] amplitude=np.ascontiguousarray(np.zeros((frequencies.size) double, dtype=np.float64, frequencies, frequencies, size, amplitude, generation_method, parameters, params, :1] frequencies, string generation_method, gen_params_py parameters):cdef double[::1] phase=np.↩ ascontiguousarray(np.zeros((frequencies.size) double, dtype=np.float64, frequencies, frequencies, size, phase, generation_method, parameters, params, :1] frequencies, string generation_method, gen_params_py parameters):cdef double[::1] waveform_plus_real=np.ascontiguousarray(np.zeros((frequencies.size) double, dtype=np.float64)

Variables

- **waveform_generator_ext.complex128_t**
- **waveform_generator_ext.ndim**
- **waveform_generator_ext.waveform**
- **waveform_generator_ext.dtype**
- **waveform_generator_ext.i** = i +1

8.2.1 Detailed Description

File that contains cython code to wrap the c++ library.

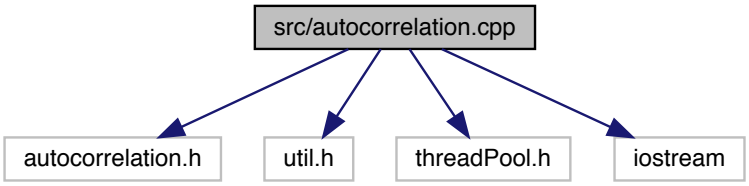
8.3 README.dox File Reference

8.4 src/autocorrelation.cpp File Reference

```
#include "autocorrelation.h"
#include "util.h"
#include "threadPool.h"
```

```
#include <iostream>
```

Include dependency graph for autocorrelation.cpp:



Macros

- #define MAX_SERIAL 200000

Functions

- void write_auto_corr_file_from_data_file (std::string autocorr_filename, std::string datafile, int length, int dimension, int num_segments, double target_corr, int num_threads)
- void write_auto_corr_file_from_data (std::string autocorr_filename, double **data, int length, int dimension, int num_segments, double target_corr, int num_threads)

  Writes the autocorrelation file from a data array.
- void auto_corr_from_data (double **data, int length, int dimension, int **output, int num_segments, double target_corr, int num_threads)

  Calculates the autocorrelation length for a set of data for a number of segments for each dimension – completely host code, utilitizes FFTW3 for longer chuncks of the chains.
- void threaded_ac_spectral (int thread, threaded_ac_jobs_fft job)

  Internal routine to calculate an spectral autocorrelation job.
- void threaded_ac_serial (int thread, threaded_ac_jobs_serial job)

  Internal routine to calculate an serial autocorrelation job.
- double auto_correlation_serial (double *arr, int length, int start, double target)

  Calculates the autocorrelation of a chain with the brute force method.
- void auto_correlation_spectral (double *chain, int length, double *autocorr, fftw_outline *plan_forw, fftw_outline *plan_rev)

  Wrapper function for convience – assumes the data array starts at 0.
- void auto_correlation_spectral (double *chain, int length, int start, double *autocorr, fftw_outline *plan_forw, fftw_outline *plan_rev)

  Faster approximation of the autocorrelation of a chain. Implements FFT/IFFT – accepts FFTW plan as argument for plan reuse and multi-threaded applications.
- void auto_correlation_spectral (double *chain, int length, double *autocorr)

  Faster approximation of the autocorrelation of a chain. Implements FFT/IFFT.
- double auto_correlation (double *arr, int length, double tolerance)

  OUTDATED – numerically finds autocorrelation length – not reliable.
- double auto_correlation_serial_old (double *arr, int length)

  OUTDATED Calculates the autocorrelation – less general version.
- double auto_correlation_grid_search (double *arr, int length, int box_num, int final_length, double target_length)

OUTDATED – Grid search method of computing the autocorrelation – unreliable.

- double auto_correlation_internal (double *arr, int length, int lag, double ave)

  Internal function to compute the auto correlation for a given lag.
- void auto_corr_intervals_outdated (double *data, int length, double *output, int num_segments, double accuracy)

  Function that computes the autocorrelation length on an array of data at set intervals to help determine convergence.
- void write_auto_corr_file_from_data (std::string auto_corr_filename, double **output, int intervals, int dimension, int N_steps)

  OUTDATED – writes autocorrelation lengths for a data array, but only with the serial method and only for a target correlation of .01.
- void write_auto_corr_file_from_data_file (std::string auto_corr_filename, std::string output_file, int intervals, int dimension, int N_steps)

  OUTDATED – writes autocorrelation lengths for a data file, but only with the serial method and only for a target correlation of .01.

8.4.1 Detailed Description

Turns out calculating the autocorrelation is more complicated if you want to do it fast, so it gets its own file now

8.4.2 Macro Definition Documentation

8.4.2.1 MAX_SERIAL

```
#define MAX_SERIAL 200000
```

Max length of array to use serial calculation

8.4.3 Function Documentation

8.4.3.1 auto_corr_from_data()

```
void auto_corr_from_data (
            double ** data,
            int length,
            int dimension,
            int ** output,
            int num_segments,
            double target_corr,
            int num_threads )
```

Calculates the autocorrelation length for a set of data for a number of segments for each dimension – completely host code, utilitizes FFTW3 for longer chuncks of the chains.

Takes in the data from a sampler, shape data[N_steps][dimension]

Outputs lags that correspond to the target_corr – shape output[dimension][num_segments]

Parameters

	data	Input data
	length	length of input data
	dimension	dimension of data
out	*output*	array that stores the auto-corr lengths – array[num_segments]
	num_segments	number of segements to compute the auto-corr length
	target_corr	Autocorrelation for which the autocorrelation length is defined (lag of autocorrelation for which it equals the target_corr)
	num_threads	Total number of threads to use

8.4.3.2 auto_corr_intervals_outdated()

```
void auto_corr_intervals_outdated (
            double * data,
            int length,
            double * output,
            int num_segments,
            double accuracy )
```

Function that computes the autocorrelation length on an array of data at set intervals to help determine convergence.

outdated version – new version uses FFTs

Parameters

	data	Input data
	length	length of input data
out	*output*	array that stores the auto-corr lengths – array[num_segments]
	num_segments	number of segements to compute the auto-corr length
	accuracy	longer chains are computed numerically, this specifies the tolerance

8.4.3.3 auto_correlation_grid_search()

```
double auto_correlation_grid_search (
            double * arr,
            int length,
            int box_num,
            int final_length,
            double target_length )
```

OUTDATED – Grid search method of computing the autocorrelation – unreliable.

Hopefully more reliable than the box-search method, which can sometimes get caught in a recursive loop when the stepsize isn't tuned, but also faster than the basic linear, serial search

Parameters

arr	Input array to use for autocorrelation
length	Length of input array
box_num	number of boxes to use for each iteration, default is 10
final_length	number of elements per box at which the grid search ends and the serial calculation begins
target_length	target correlation that corresponds to the returned lag

8.4.3.4 auto_correlation_internal()

```
double auto_correlation_internal (
            double * arr,
            int length,
            int lag,
            double ave )
```

Internal function to compute the auto correlation for a given lag.

8.4.3.5 auto_correlation_serial()

```
double auto_correlation_serial (
            double * arr,
            int length,
            int start,
            double target )
```

Calculates the autocorrelation of a chain with the brute force method.

Parameters

arr	input array
length	Length of input array
start	starting index (probably 0)
target	Target autocorrelation for which ``length'' is defined

8.4.3.6 auto_correlation_spectral() [1/2]

```
void auto_correlation_spectral (
            double * chain,
            int length,
            int start,
            double * autocorr,
```

```
            fftw_outline * plan_forw,
            fftw_outline * plan_rev )
```

Faster approximation of the autocorrelation of a chain. Implements FFT/IFFT – accepts FFTW plan as argument for plan reuse and multi-threaded applications.

Based on the Wiener-Khinchin Theorem.

Algorithm used from https://lingpipe-blog.com/2012/06/08/autocorrelation-fft-kiss-eigen/

NOTE the length used in initializing the fftw plans should be L = pow(2, std::ceil(std::log2(length))) – the plans are padded so the total length is a power of two

Option to provide starting index for multi-dimension arrays in collapsed to one dimension

length is the length of the segment to be analyzed, not necessarily the dimension of the chain

8.4.3.7 auto_correlation_spectral() [2/2]

```
void auto_correlation_spectral (
            double * chain,
            int length,
            double * autocorr )
```

Faster approximation of the autocorrelation of a chain. Implements FFT/IFFT.

Based on the Wiener-Khinchin Theorem.

Algorithm used from https://lingpipe-blog.com/2012/06/08/autocorrelation-fft-kiss-eigen/

8.4.3.8 threaded_ac_serial()

```
void threaded_ac_serial (
            int thread,
            threaded_ac_jobs_serial job )
```

Internal routine to calculate an serial autocorrelation job.

Allows for a more efficient use of the threadPool class

8.4.3.9 threaded_ac_spectral()

```
void threaded_ac_spectral (
            int thread,
            threaded_ac_jobs_fft job )
```

Internal routine to calculate an spectral autocorrelation job.

Allows for a more efficient use of the threadPool class

8.4.3.10 write_auto_corr_file_from_data()

```
void write_auto_corr_file_from_data (
            std::string autocorr_filename,
            double ** data,
            int length,
            int dimension,
            int num_segments,
            double target_corr,
            int num_threads )
```

Writes the autocorrelation file from a data array.

Parameters

autocorr_filename	Name of the file to write the autocorrelation to
data	Input chains
length	length of input data
dimension	dimension of data
num_segments	number of segements to compute the auto-corr length
target_corr	Autocorrelation for which the autocorrelation length is defined (lag of autocorrelation for which it equals the target_corr)
num_threads	Total number of threads to use

8.4.3.11 write_auto_corr_file_from_data_file()

```
void write_auto_corr_file_from_data_file (
            std::string autocorr_filename,
            std::string datafile,
            int length,
            int dimension,
            int num_segments,
            double target_corr,
            int num_threads )
```

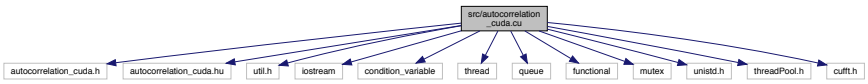
Parameters

length	length of input data
dimension	dimension of data
num_segments	number of segements to compute the auto-corr length
target_corr	Autocorrelation for which the autocorrelation length is defined (lag of autocorrelation for which it equals the target_corr)
num_threads	Total number of threads to use

8.5 src/autocorrelation_cuda.cu File Reference

```
#include "autocorrelation_cuda.h"
#include "autocorrelation_cuda.hu"
#include "util.h"
#include <iostream>
#include <condition_variable>
#include <thread>
#include <queue>
#include <functional>
#include <mutex>
#include <unistd.h>
#include <threadPool.h>
#include <cufft.h>
```

Include dependency graph for autocorrelation_cuda.cu:



Functions

- __device__ __host__ void auto_corr_internal (double *arr, int length, int lag, double average, double *corr, int start_id)

  Internal function to calculate the autocorrelation for a given lag Customized for the thread pool architecture, with extra arguments because of the way the memory is allocated.
- __global__ void auto_corr_internal_kernal (double *arr, int length, double average, int *rho_index, double target_corr, double var, int start_id)

  Internal function to launch the CUDA kernel for a range of autocorrelations.
- void write_file_auto_corr_from_data_file_accel (std::string acfile, std::string chains_file, int dimension, int N↩ _steps, int num_segments, double target_corr)

  Write data file for autocorrelation lengths of the data given a data file name, as written by the mcmc_sampler.
- void write_file_auto_corr_from_data_accel (std::string acfile, double **chains, int dimension, int N_steps, int num_segments, double target_corr)

  Write data file given output chains, as formatted by the mcmc_sampler.
- void auto_corr_from_data_accel (double **output, int dimension, int N_steps, int num_segments, double target_corr, double **autocorr)

  Find autocorrelation of data at different points in the chain length and output to autocorr.
- void ac_gpu_wrapper (int thread, int job_id)

  Wrapper function for the thread pool.
- void launch_ac_gpu (int device, int element, double **data, int length, int dimension, double target_corr, int num_segments)

  Launch the GPU kernel, formatted for the thread pool.
- void allocate_gpu_plan (GPUplan *plan, int data_length, int dimension, int num_segments)

  Allocates memory for autocorrelation–GPU structure.
- void deallocate_gpu_plan (GPUplan *plan, int data_length, int dimension, int num_segments)

  Deallocates memory for the autocorrelation–GPU structure.
- void copy_data_to_device (GPUplan *plan, double **input_data, int data_length, int dimension, int num_↩ segments)

  Copy data to device before starting kernels.

Variables

- GPUplan * **plans_global**

8.5.1 Function Documentation

8.5.1.1 ac_gpu_wrapper()

```
void ac_gpu_wrapper (
            int thread,
            int job_id )
```

Wrapper function for the thread pool.

Parameters

thread	Host thread
job↩ _id	Job ID

8.5.1.2 allocate_gpu_plan()

```
void allocate_gpu_plan (
            GPUplan * plan,
            int data_length,
            int dimension,
            int num_segments )
```

Allocates memory for autocorrelation–GPU structure.

Parameters

plan	Structure for GPU plan
data_length	Length of data
dimension	Dimension of the data
num_segments	Number of segments to calculate the autocorrelation length

8.5.1.3 auto_corr_from_data_accel()

```
void auto_corr_from_data_accel (
            double ** output,
            int dimension,
            int N_steps,
            int num_segments,
            double target_corr,
            double ** autocorr )
```

Find autocorrelation of data at different points in the chain length and output to autocorr.

Parameters

	output	Chain data input
	dimension	Dimension of the data
	N_steps	Number of steps in the data
	num_segments	number of segments to calculate the autocorrelation length
	target_corr	Target correlation ratio
out	*autocorr*	Autocorrelation lengths for the different segments

8.5.1.4 auto_corr_internal()

```
__device__ __host__ void auto_corr_internal (
            double * arr,
            int length,
            int lag,
            double average,
            double * corr,
            int start_id )
```

Internal function to calculate the autocorrelation for a given lag Customized for the thread pool architecture, with extra arguments because of the way the memory is allocated.

Parameters

	arr	Input array of data
	length	Length of input array
	lag	Lag to be used to calculate the correlation
	average	Average of the array arr
out	*corr*	output correlation
	start_id	ID of location to start calculation – input arrary arr is assumed to be contiguous for multiple dimensions

8.5.1.5 auto_corr_internal_kernal()

```
__global__ void auto_corr_internal_kernal (
            double * arr,
            int length,
            double average,
            int * rho_index,
            double target_corr,
            double var,
            int start_id )
```

Internal function to launch the CUDA kernel for a range of autocorrelations.

Correlation function used:

rho(lag) = 1 / (length - lag) \sum (arr[i+lag]-average) (arr[i]- average)

target_corr = rho(rho_index)/rho(0) = rho(rho_index)/var

Parameters

	arr	Input array of data
	length	Length of data array
	average	Average of input data
out	*rho_index*	Index of the lag that results ina correlation ratio target_corr
	target_corr	Target correlation ratio rho(lag)/rho(0) = target_corr
	var	Variance rho(0)
	start_id	Starting index to use for the data array arr

8.5.1.6 copy_data_to_device()

```
void copy_data_to_device (
            GPUplan * plan,
            double ** input_data,
            int data_length,
            int dimension,
            int num_segments )
```

Copy data to device before starting kernels.

Parameters

plan	GPU plan
input_data	Input chain data
data_length	Length of data
dimension	Dimension of the data
num_segments	Number of segments to calculate the autocorrelation length

8.5.1.7 deallocate_gpu_plan()

```
void deallocate_gpu_plan (
            GPUplan * plan,
            int data_length,
            int dimension,
            int num_segments )
```

Deallocates memory for the autocorrelation–GPU structure.

Parameters

plan	Structure for the GPU plan
data_length	Length of data
dimension	Dimension of the data
num_segments	Number of segments to calculate the autocorrelation length

8.5.1.8 write_file_auto_corr_from_data_accel()

```
void write_file_auto_corr_from_data_accel (
            std::string acfile,
            double ** chains,
            int dimension,
            int N_steps,
```

```
            int num_segments,
            double target_corr )
```

Write data file given output chains, as formatted by the mcmc_sampler.

Parameters

acfile	Output autocorrelation filename
chains	Chain data from MCMC_sampler
dimension	Dimension of the data
N_steps	Number of steps in the chain
num_segments	Number of segments to check the autocorrelation length for each dimension
target_corr	Target correlation ratio to use for the correlation length calculation

8.5.1.9 write_file_auto_corr_from_data_file_accel()

```
void write_file_auto_corr_from_data_file_accel (
            std::string acfile,
            std::string chains_file,
            int dimension,
            int N_steps,
            int num_segments,
            double target_corr )
```

Write data file for autocorrelation lengths of the data given a data file name, as written by the mcmc_sampler.

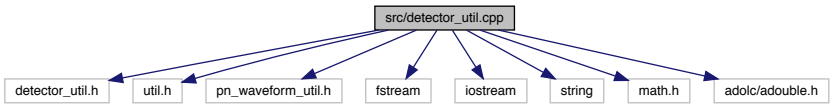
Parameters

acfile	Filename of the autocorrelation data
chains_file	Filename of the data file for the chains
dimension	Dimension of the data
N_steps	Number of steps in the chain
num_segments	Number of segments to check the autocorrelation length for each dimension
target_corr	Target correlation ratio to use for the correlation length calculation

8.6 src/detector_util.cpp File Reference

```
#include "detector_util.h"
#include "util.h"
#include "pn_waveform_util.h"
#include <fstream>
#include <iostream>
#include <string>
#include <math.h>
#include <adolc/adouble.h>
```

Include dependency graph for detector_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)

  Analytic function approximating the PSD for aLIGO.
- double Hanford_O1_fitted (double f)

  Numerically fit PSD to the Hanford Detector's O1.
- std::complex< double > Q (double theta, double phi, double iota, double psi)

  Utility for the overall amplitude and phase shift for spin-aligned systems.
- std::complex< double > Q (double theta, double phi, double iota)

  Utility for the overall amplitude and phase shift for spin-aligned systems.
- double right_interferometer_plus (double theta, double phi)

  Response function of a 90 deg interferometer for plus polarization.
- double right_interferometer_cross (double theta, double phi)

  Response function of a 90 deg interferometer for cross polarization.
- void celestial_horizon_transform (double RA, double DEC, double gps_time, std::string detector, double *phi, double *theta)

  Transform from celestial coordinates to local horizontal coords.
- void derivative_celestial_horizon_transform (double RA, double DEC, double gps_time, std::string detector, double *dphi_dRA, double *dtheta_dRA, double *dphi_dDEC, double *dtheta_dDEC)

  Numerical derivative of the transformation.
- double DTOA (double theta1, double theta2, std::string detector1, std::string detector2)

  calculate difference in time of arrival (DTOA) for a given source location and 2 different detectors
- double radius_at_lat (double latitude, double elevation)
- void detector_response_functions_equatorial (double D[3][3], double ra, double dec, double psi, double gmst, double *Fplus, double *Fcross)

  Calculates the response coefficients for a detector with response tensor D for a source at RA, Dec, and psi.
- void detector_response_functions_equatorial (std::string detector, double ra, double dec, double psi, double gmst, double *Fplus, double *Fcross)

  Same as the other function, but for active detectors.
- template<class T >
  T **LISA_response_plus** (source_parameters< T > *params, T theta_s, T phi_s, T theta_l, T phi_l, T alpha_0, T phi_0, T f)
- template<class T >
  T **LISA_response_cross** (source_parameters< T > *params, T theta_s, T phi_s, T theta_l, T phi_l, T alpha_0, T phi_0, T f)
- template<class T >
  T LISA_response_plus_time (T theta_s, T phi_s, T theta_l, T phi_l, T alpha_0, T phi_0, T t)

  Time dependent detector response of LISA for non-precessing waveforms.
- template<class T >
  T **LISA_response_cross_time** (T theta_s, T phi_s, T theta_l, T phi_l, T alpha_0, T phi_0, T t)

- template double **LISA_response_plus_time**< **double** > (double, double, double, double, double, double, double)
- template adouble **LISA_response_plus_time**< **adouble** > (adouble, adouble, adouble, adouble, adouble, adouble, adouble)
- template double **LISA_response_cross_time**< **double** > (double, double, double, double, double, double, double)
- template adouble **LISA_response_cross_time**< **adouble** > (adouble, adouble, adouble, adouble, adouble, adouble, adouble)
- template double **LISA_response_plus**< **double** > (source_parameters< double > *params, double, double, double, double, double, double)
- template adouble **LISA_response_plus**< **adouble** > (source_parameters< adouble > *params, adouble, adouble, adouble, adouble, adouble, adouble)
- template double **LISA_response_cross**< **double** > (source_parameters< double > *params, double, double, double, double, double, double)
- template adouble **LISA_response_cross**< **adouble** > (source_parameters< adouble > *params, adouble, adouble, adouble, adouble, adouble, adouble)

8.6.1 Detailed Description

Routines to construct noise curves for various detectors and for detector specific utilities for response functions and coordinate transformations

8.6.2 Function Documentation

8.6.2.1 aLIGO_analytic()

```
double aLIGO_analytic (
            double f )
```

Analytic function approximating the PSD for aLIGO.

CITE (Will?)

8.6.2.2 celestial_horizon_transform()

```
void celestial_horizon_transform (
            double RA,
            double DEC,
            double gps_time,
            std::string detector,
            double * phi,
            double * theta )
```

Transform from celestial coordinates to local horizontal coords.

(RA,DEC) -> (altitude, azimuth)

Need gps_time of transformation, as the horizontal coords change in time

detector is used to specify the lat and long of the local frame

Parameters

RA	in RAD
DEC	in RAD
phi	in RAD
theta	in RAD

8.6.2.3 derivative_celestial_horizon_transform()

```
void derivative_celestial_horizon_transform (
             double RA,
             double DEC,
             double gps_time,
             std::string detector,
             double * dphi_dRA,
             double * dtheta_dRA,
             double * dphi_dDEC,
             double * dtheta_dDEC )
```

Numerical derivative of the transformation.

Planned for use in Fisher calculations, but not currently implemented anywhere

Parameters

RA	in RAD
DEC	in RAD

8.6.2.4 detector_response_functions_equatorial() [1/2]

```
void detector_response_functions_equatorial (
             double D[3][3],
             double ra,
             double dec,
             double psi,
             double gmst,
             double * Fplus,
             double * Fcross )
```

Calculates the response coefficients for a detector with response tensor D for a source at RA, Dec, and psi.

Taken from LALSuite

The response tensor for each of the operational detectors is precomputed in detector_util.h, but to create a new tensor, follow the outline in Anderson et al 36 PRD 63 042003 (2001) Appendix B

Parameters

	D	Detector Response tensor (3x3)

Parameters

	ra	Right ascension in rad
	dec	Declination in rad
	psi	polarization angle in rad
	gmst	Greenwich mean sidereal time (rad)
out	*Fplus*	Fplus response coefficient
out	*Fcross*	Fcross response coefficient

8.6.2.5 detector_response_functions_equatorial() [2/2]

```
void detector_response_functions_equatorial (
            std::string detector,
            double ra,
            double dec,
            double psi,
            double gmst,
            double * Fplus,
            double * Fcross )
```

Same as the other function, but for active detectors.

Parameters

	detector	Detector
	ra	Right ascension in rad
	dec	Declination in rad
	psi	polarization angle in rad
	gmst	Greenwich mean sidereal time (rad)
out	*Fplus*	Fplus response coefficient
out	*Fcross*	Fcross response coefficient

8.6.2.6 DTOA()

```
double DTOA (
            double theta1,
            double theta2,
            std::string detector1,
            std::string detector2 )
```

calculate difference in time of arrival (DTOA) for a given source location and 2 different detectors

Parameters

theta1	spherical polar angle for detector 1 in RAD
theta2	spherical polar angle for detector 2 in RAD
detector1	name of detector one
detector2	name of detector two

8.6.2.7 Hanford_O1_fitted()

```
double Hanford_O1_fitted (
            double f )
```

Numerically fit PSD to the Hanford Detector's O1.

CITE (Yunes?)

8.6.2.8 LISA_response_plus_time()

```
template<class T >
T LISA_response_plus_time (
            T theta_s,
            T phi_s,
            T theta_l,
            T phi_l,
            T alpha_0,
            T phi_0,
            T t )
```

Time dependent detector response of LISA for non-precessing waveforms.

See https://arxiv.org/abs/gr-qc/0411129 or https://arxiv.org/abs/gr-qc/9703068

All the arguments are ``barred'', using the notation in these two works. That is, they are relative to the solar system barycenter.

To get the second interferometer's response, evaluate with phi_l - pi/4.

8.6.2.9 populate_noise()

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

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

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

Detector names must be spelled exactly

Detectors include: aLIGO_analytic, Hanford_O1_fitted

Parameters

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

8.6.2.10 Q() [1/2]

```
std::complex<double> Q (
            double theta,
            double phi,
            double iota,
            double psi )
```

Utility for the overall amplitude and phase shift for spin-aligned systems.

For spin aligned, all the extrinsic parameters have the effect of an overall amplitude modulation and phase shift

8.6.2.11 Q() [2/2]

```
std::complex<double> Q (
            double theta,
            double phi,
            double iota )
```

Utility for the overall amplitude and phase shift for spin-aligned systems.

For spin aligned, all the extrinsic parameters have the effect of an overall amplitude modulation and phase shift

8.6.2.12 radius_at_lat()

```
double radius_at_lat (
            double latitude,
            double elevation )
```

/brief Analytic approximation of the radius from the center of earth to a given location

Just the raidus as a function of angles, modelling an oblate spheroid

Parameters

latitude	latitude in degrees
elevation	elevation in meters

8.6.2.13 right_interferometer_cross()

```
double right_interferometer_cross (
            double theta,
            double phi )
```

Response function of a 90 deg interferometer for cross polarization.

Theta and phi are local, horizontal coordinates relative to the detector

8.6.2.14 right_interferometer_plus()

```
double right_interferometer_plus (
            double theta,
            double phi )
```

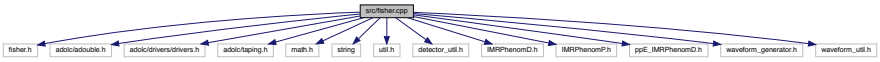
Response function of a 90 deg interferometer for plus polarization.

Theta and phi are local, horizontal coordinates relative to the detector

8.7 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 "detector_util.h"
#include "IMRPhenomD.h"
#include "IMRPhenomP.h"
#include "ppE_IMRPhenomD.h"
#include "waveform_generator.h"
#include "waveform_util.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.
- void calculate_derivatives (double **amplitude_deriv, double **phase_deriv, double *amplitude, double *frequencies, int length, string detector, string gen_method, gen_params *parameters)

  Abstraction layer for handling the case separation for the different waveforms.
- void fisher_autodiff (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 to within numerical error using automatic differention - slower than the numerical version.

8.7.1 Detailed Description

All subroutines associated with waveform differentiation and Fisher analysis

8.7.2 Function Documentation

8.7.2.1 calculate_derivatives()

```
void calculate_derivatives (
             double ** amplitude_deriv,
             double ** phase_deriv,
             double * amplitude,
             double * frequencies,
             int length,
             string detector,
             string gen_method,
             gen_params * parameters )
```

Abstraction layer for handling the case separation for the different waveforms.

8.7.2.2 fisher()

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

Calculates the fisher matrix for the given arguments.

Parameters

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

8.7.2.3 fisher_autodiff()

```
void fisher_autodiff (
```

```
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 to within numerical error using automatic differention - slower than the numerical version.

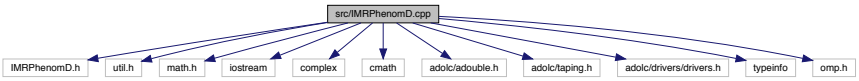
Parameters

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

8.8 src/IMRPhenomD.cpp File Reference

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

Include dependency graph for IMRPhenomD.cpp:



Macros

- #define **omp** ignore

Variables

- double **log_64** = 4.15888308336

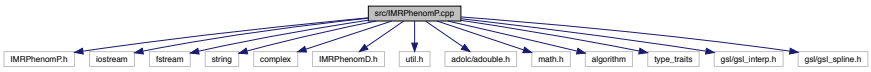
8.8.1 Detailed Description

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

8.9 src/IMRPhenomP.cpp File Reference

```
#include "IMRPhenomP.h"
#include <iostream>
#include <fstream>
#include <string>
#include <complex>
#include "IMRPhenomD.h"
#include "util.h"
#include <adolc/adouble.h>
#include <math.h>
#include <algorithm>
#include <type_traits>
#include <gsl/gsl_interp.h>
#include <gsl/gsl_spline.h>
```

Include dependency graph for IMRPhenomP.cpp:



Macros

- #define **ROTATEZ**(angle, vx, vy, vz)
- #define **ROTATEY**(angle, vx, vy, vz)

Variables

- const double **sqrt_6** = 2.44948974278317788

8.9.1 Detailed Description

Source code for IMRPhenomP

8.9.2 Macro Definition Documentation

8.9.2.1 ROTATEY

```
#define ROTATEY(
              angle,
              vx,
              vy,
              vz )
```

Value:

```
tmp1 = vx*cos(angle) + vz*sin(angle);\
tmp2 = - vx*sin(angle) + vz*cos(angle);\
vx = tmp1;\
vz = tmp2
```

8.9.2.2 ROTATEZ

```
#define ROTATEZ(
              angle,
              vx,
              vy,
              vz )
```

Value:

```
tmp1 = vx*cos(angle) - vy*sin(angle);\
tmp2 = vx*sin(angle) + vy*cos(angle);\
vx = tmp1;\
vy = tmp2
```

8.10 src/mcmc_gw.cpp File Reference

```
#include "mcmc_gw.h"
#include "waveform_generator.h"
#include "util.h"
#include "detector_util.h"
#include "waveform_util.h"
#include "fisher.h"
#include "mcmc_sampler.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <complex>
#include <fftw3.h>
#include <algorithm>
#include <gsl/gsl_interp.h>
#include <gsl/gsl_spline.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_rng.h>
```

Include dependency graph for mcmc_gw.cpp:

Functions

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

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

  routine to maximize over all extrinsic quantities and return the log likelihood
- double **maximized_Log_Likelihood** (double *data_real, double *data_imag, double *psd, double *frequencies, size_t length, gen_params *params, std::string detector, std::string generation_method, fftw_outline *plan)
- double maximized_coal_Log_Likelihood (std::complex< double > *data, double *psd, double *frequencies, size_t length, gen_params *params, std::string detector, std::string generation_method, fftw_outline *plan, double *tc, double *phic)

  Function to maximize only over coalescence variables tc and phic, returns the maximum values used.
- double **maximized_coal_Log_Likelihood_internal** (std::complex< double > *data, double *psd, double *frequencies, std::complex< double > *detector_response, size_t length, fftw_outline *plan, double *tc, double *phic)
- double Log_Likelihood (std::complex< double > *data, double *psd, double *frequencies, size_t length, gen_params *params, std::string detector, std::string generation_method, fftw_outline *plan)

  Unmarginalized log of the likelihood.
- double maximized_Log_Likelihood_aligned_spin_internal (std::complex< double > *data, double *psd, double *frequencies, std::complex< double > *detector_response, size_t length, fftw_outline *plan)

  Maximized match over coalescence variables - returns log likelihood NOT NORMALIZED for aligned spins.
- double maximized_Log_Likelihood_unaligned_spin_internal (std::complex< double > *data, double *psd, double *frequencies, std::complex< double > *hplus, std::complex< double > *hcross, size_t length, fftw_outline *plan)

  log likelihood function that maximizes over extrinsic parameters tc, phic, D, and phiRef, the reference frequency - for unaligned spins
- double Log_Likelihood_internal (std::complex< double > *data, double *psd, double *frequencies, std::complex< double > *detector_response, int length, fftw_outline *plan)

  Internal function for the unmarginalized log of the likelihood.
- void continue_RJPTMCMC_MH_GW (std::string start_checkpoint_file, double ***output, int ***status, int max_dim, int min_dim, int N_steps, int swp_freq, double(*log_prior)(double *param, int *status, int dimension, int chain_id), void(*RJ_proposal)(double *current_param, double *proposed_param, int *current_status, int *proposed_status, int max_dim, int chain_id, double step_width), int numThreads, bool pool, bool show_prog, int num_detectors, std::complex< double > **data, double **noise_psd, double **frequencies,

int *data_length, double gps_time, std::string *detectors, int Nmod, int *bppe, std::string generation_↩ method, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string final_checkpoint_filename)

Takes in an MCMC checkpoint file and continues the chain.

- void RJPTMCMC_MH_GW (double ***output, int ***status, int max_dim, int min_dim, int N_steps, int chain_N, double *initial_pos, int *initial_status, double *seeding_var, double *chain_temps, int swp_↩ freq, double(*log_prior)(double *param, int *status, int dimension, int chain_id), void(*RJ_proposal)(double *current_param, double *proposed_param, int *current_status, int *proposed_status, int max_dim, int chain_id, double step_width), int numThreads, bool pool, bool show_prog, int num_detectors, std::complex< double > **data, double **noise_psd, double **frequencies, int *data_length, double gps_time, std↩ ::string *detectors, int Nmod_max, int *bppe, std::string generation_method, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string checkpoint_file)

  Wrapper for the RJPTMCMC_MH function, specifically for GW analysis.
- void PTMCMC_MH_GW (double ***output, int dimension, int N_steps, int chain_N, double *initial_pos, double *seeding_var, double *chain_temps, int swp_freq, double(*log_prior)(double *param, int dimension, int chain_id), int numThreads, bool pool, bool show_prog, int num_detectors, std::complex< double > **data, double **noise_psd, double **frequencies, int *data_length, double gps_time, std::string *detectors, int Nmod, int *bppe, std::string generation_method, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string checkpoint_file)

  Wrapper for the MCMC_MH function, specifically for GW analysis.
- void PTMCMC_MH_dynamic_PT_alloc_GW (double ***output, int dimension, int N_steps, int chain_N, int max_chain_N_thermo_ensemble, double *initial_pos, double *seeding_var, double *chain_temps, int swp↩ _freq, int t0, int nu, std::string chain_distribution_scheme, double(*log_prior)(double *param, int dimension, int chain_id), int numThreads, bool pool, bool show_prog, int num_detectors, std::complex< double > **data, double **noise_psd, double **frequencies, int *data_length, double gps_time, std::string *detectors, int Nmod, int *bppe, std::string generation_method, std::string statistics_filename, std::string chain_filename, std::string likelihood_log_filename, std::string checkpoint_filename)

  Takes in an MCMC checkpoint file and continues the chain.
- void continue_PTMCMC_MH_GW (std::string start_checkpoint_file, double ***output, int dimension, int N↩ _steps, int swp_freq, double(*log_prior)(double *param, int dimension, int chain_id), int numThreads, bool pool, bool show_prog, int num_detectors, std::complex< double > **data, double **noise_psd, double **frequencies, int *data_length, double gps_time, std::string *detectors, int Nmod, int *bppe, std::string generation_method, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string final_checkpoint_filename)

  Takes in an MCMC checkpoint file and continues the chain.
- void RJPTMCMC_method_specific_prep (std::string generation_method, int max_dim, int min_dim, double *seeding_var, bool local_seeding)

  Unpacks MCMC parameters for method specific initiation (RJ version)
- void PTMCMC_method_specific_prep (std::string generation_method, int dimension, double *seeding_var, bool local_seeding)

  Unpacks MCMC parameters for method specific initiation.
- void MCMC_fisher_wrapper (double *param, int dimension, double **output, int chain_id)

  Fisher function for MCMC for GW.
- double **MCMC_likelihood_extrinsic** (bool save_waveform, gen_params *parameters, std::string generation_method, int *data_length, double **frequencies, std::complex< double > **data, double **psd, std::string *detectors, fftw_outline *fftw_plans, int num_detectors, double RA, double DEC, double gps_time)
- double MCMC_likelihood_wrapper (double *param, int dimension, int chain_id)

  log likelihood function for MCMC for GW
- double **RJPTMCMC_likelihood_wrapper** (double *param, int *status, int max_dim, int chain_id)
- void **RJPTMCMC_RJ_proposal** (double *current_params, double *proposed_params, int *current_status, int *proposed_status, int max_dim, int chain_id, double step_width)
- void **RJPTMCMC_fisher_wrapper** (double *param, int *status, int min_dim, double **output, int chain_id)

8.10.1 Detailed Description

Routines for implementation in MCMC algorithms specific to GW CBC analysis

8.10.2 Function Documentation

8.10.2.1 continue_PTMCMC_MH_GW()

```
void continue_PTMCMC_MH_GW (
            std::string start_checkpoint_file,
            double *** output,
            int dimension,
            int N_steps,
            int swp_freq,
            double(*)(double *param, int dimension, int chain_id) log_prior,
            int numThreads,
            bool pool,
            bool show_prog,
            int num_detectors,
            std::complex< double > ** data,
            double ** noise_psd,
            double ** frequencies,
            int * data_length,
            double gps_time,
            std::string * detectors,
            int Nmod,
            int * bppe,
            std::string generation_method,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string final_checkpoint_filename )
```

Takes in an MCMC checkpoint file and continues the chain.

Obviously, the user must be sure to correctly match the dimension, number of chains, the generation_method, the prior function, the data, psds, freqs, and the detectors (number and name), and the gps_time to the previous run, otherwise the behavior of the sampler is undefined.

numThreads and pool do not necessarily have to be the same

8.10.2.2 continue_RJPTMCMC_MH_GW()

```
void continue_RJPTMCMC_MH_GW (
            std::string start_checkpoint_file,
            double *** output,
            int *** status,
            int max_dim,
            int min_dim,
```

```
            int N_steps,
            int swp_freq,
            double(*)(double *param, int *status, int dimension, int chain_id) log_prior,
            void(*)(double *current_param, double *proposed_param, int *current_status, int
*proposed_status, int max_dim, int chain_id, double step_width) RJ_proposal,
            int numThreads,
            bool pool,
            bool show_prog,
            int num_detectors,
            std::complex< double > ** data,
            double ** noise_psd,
            double ** frequencies,
            int * data_length,
            double gps_time,
            std::string * detectors,
            int Nmod,
            int * bppe,
            std::string generation_method,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string final_checkpoint_filename )
```

Takes in an MCMC checkpoint file and continues the chain.

Obviously, the user must be sure to correctly match the dimension, number of chains, the generation_method, the prior function, the data, psds, freqs, and the detectors (number and name), and the gps_time to the previous run, otherwise the behavior of the sampler is undefined.

numThreads and pool do not necessarily have to be the same

8.10.2.3 Log_Likelihood()

```
double Log_Likelihood (
            std::complex< double > * data,
            double * psd,
            double * frequencies,
            size_t length,
            gen_params * params,
            std::string detector,
            std::string generation_method,
            fftw_outline * plan )
```

Unmarginalized log of the likelihood.

8.10.2.4 Log_Likelihood_internal()

```
double Log_Likelihood_internal (
            std::complex< double > * data,
            double * psd,
            double * frequencies,
            std::complex< double > * detector_response,
            int length,
            fftw_outline * plan )
```

Internal function for the unmarginalized log of the likelihood.

.5 ∗ ((h | h) - 2 (D | h))

8.10.2.5 maximized_coal_Log_Likelihood()

```
double maximized_coal_Log_Likelihood (
            std::complex< double > * data,
            double * psd,
            double * frequencies,
            size_t length,
            gen_params * params,
            std::string detector,
            std::string generation_method,
            fftw_outline * plan,
            double * tc,
            double * phic )
```

Function to maximize only over coalescence variables tc and phic, returns the maximum values used.

8.10.2.6 maximized_coal_log_likelihood_IMRPhenomD() [1/3]

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

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

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

Parameters

chirpmass	in solar masses

8.10.2.7 maximized_coal_log_likelihood_IMRPhenomD() [2/3]

```
double maximized_coal_log_likelihood_IMRPhenomD (
            double * frequencies,
            size_t length,
            double * real_data,
            double * imag_data,
            double * noise,
```

```
            double SNR,
            double chirpmass,
            double symmetric_mass_ratio,
            double spin1,
            double spin2,
            bool NSflag )
```

Parameters

chirpmass	in solar masses

8.10.2.8 maximized_coal_log_likelihood_IMRPhenomD() [3/3]

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

Parameters

chirpmass	in solar masses

8.10.2.9 maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [1/3]

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

Parameters

chirpmass	in solar masses

8.10.2.10 maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [2/3]

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

Parameters

chirpmass	in solar masses

8.10.2.11 maximized_coal_log_likelihood_IMRPhenomD_Full_Param() [3/3]

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

Parameters

chirpmass	in solar masses

8.10.2.12 maximized_Log_Likelihood()

```
double maximized_Log_Likelihood (
             std::complex< double > * data,
             double * psd,
             double * frequencies,
             size_t length,
             gen_params * params,
             std::string detector,
             std::string generation_method,
             fftw_outline * plan )
```

routine to maximize over all extrinsic quantities and return the log likelihood

IMRPhenomD – maximizes over DL, phic, tc, \iota, \phi, \theta IMRPhenomP – maximizes over DL, phic,tc, \psi, \phi , \theta

8.10.2.13 maximized_Log_Likelihood_aligned_spin_internal()

```
double maximized_Log_Likelihood_aligned_spin_internal (
             std::complex< double > * data,
             double * psd,
             double * frequencies,
             std::complex< double > * detector_response,
             size_t length,
             fftw_outline * plan )
```

Maximized match over coalescence variables - returns log likelihood NOT NORMALIZED for aligned spins.

Note: this function is not properly normalized for an absolute comparison. This is made for MCMC sampling, so to minimize time, constant terms like (Data|Data), which would cancel in the Metropolis-Hasting ratio, are left out for efficiency

8.10.2.14 maximized_Log_Likelihood_unaligned_spin_internal()

```
double maximized_Log_Likelihood_unaligned_spin_internal (
             std::complex< double > * data,
             double * psd,
             double * frequencies,
             std::complex< double > * hplus,
             std::complex< double > * hcross,
             size_t length,
             fftw_outline * plan )
```

log likelihood function that maximizes over extrinsic parameters tc, phic, D, and phiRef, the reference frequency - for unaligned spins

Ref: arXiv 1603.02444v2

8.10.2.15 MCMC_fisher_wrapper()

```
void MCMC_fisher_wrapper (
            double * param,
            int dimension,
            double ** output,
            int chain_id )
```

Fisher function for MCMC for GW.

Wraps the fisher calculation in src/fisher.cpp and unpacks parameters correctly for common GW analysis

Supports all the method/parameter combinations found in MCMC_MH_GW

8.10.2.16 MCMC_likelihood_wrapper()

```
double MCMC_likelihood_wrapper (
            double * param,
            int dimension,
            int chain_id )
```

log likelihood function for MCMC for GW

Wraps the above likelihood functions and unpacks parameters correctly for common GW analysis

Supports all the method/parameter combinations found in MCMC_MH_GW

8.10.2.17 PTMCMC_method_specific_prep()

```
void PTMCMC_method_specific_prep (
            std::string generation_method,
            int dimension,
            double * seeding_var,
            bool local_seeding )
```

Unpacks MCMC parameters for method specific initiation.

Populates seeding vector if non supplied, populates mcmc_Nmod, populates mcmc_log_beta, populates mcmc_↩ intrinsic

8.10.2.18 PTMCMC_MH_dynamic_PT_alloc_GW()

```
void PTMCMC_MH_dynamic_PT_alloc_GW (
             double *** output,
             int dimension,
             int N_steps,
             int chain_N,
             int max_chain_N_thermo_ensemble,
             double * initial_pos,
             double * seeding_var,
             double * chain_temps,
             int swp_freq,
             int t0,
             int nu,
             std::string chain_distribution_scheme,
             double(*)(double *param, int dimension, int chain_id) log_prior,
             int numThreads,
             bool pool,
             bool show_prog,
             int num_detectors,
             std::complex< double > ** data,
             double ** noise_psd,
             double ** frequencies,
             int * data_length,
             double gps_time,
             std::string * detectors,
             int Nmod,
             int * bppe,
             std::string generation_method,
             std::string statistics_filename,
             std::string chain_filename,
             std::string likelihood_log_filename,
             std::string checkpoint_filename )
```

Takes in an MCMC checkpoint file and continues the chain.

Obviously, the user must be sure to correctly match the dimension, number of chains, the generation_method, the prior function, the data, psds, freqs, and the detectors (number and name), and the gps_time to the previous run, otherwise the behavior of the sampler is undefined.

numThreads and pool do not necessarily have to be the same

8.10.2.19 PTMCMC_MH_GW()

```
void PTMCMC_MH_GW (
             double *** output,
             int dimension,
             int N_steps,
             int chain_N,
             double * initial_pos,
             double * seeding_var,
             double * chain_temps,
             int swp_freq,
             double(*)(double *param, int dimension, int chain_id) log_prior,
             int numThreads,
             bool pool,
```

```
bool show_prog,
int num_detectors,
std::complex< double > ** data,
double ** noise_psd,
double ** frequencies,
int * data_length,
double gps_time,
std::string * detectors,
int Nmod,
int * bppe,
std::string generation_method,
std::string statistics_filename,
std::string chain_filename,
std::string auto_corr_filename,
std::string likelihood_log_filename,
std::string checkpoint_file )
```

Wrapper for the MCMC_MH function, specifically for GW analysis.

Handles the details of setting up the MCMC sampler and wraps the fisher and log likelihood to conform to the format of the sampler

NOTE – This sampler is NOT thread safe. There is global memory declared for each call to MCMC_MH_GW, so separate samplers should not be run in the same process space

Supported parameter combinations:

IMRPhenomD - 4 dimensions – ln chirpmass, eta, chi1, chi2

IMRPhenomD - 7 dimensions – ln D_L, tc, phic, ln chirpmass, eta, chi1, chi2

IMRPhenomD - 9 dimensions – cos inclination, RA, DEC, ln D_L, ln chirpmass, eta, chi1, chi2, psi

dCS_IMRPhenomD_log - 8 dimensions – cos inclination, RA, DEC, ln D_L, ln chirpmass, eta, chi1, chi2, ln \alpha^2 (the coupling parameter)

dCS_IMRPhenomD- 8 dimensions – cos inclination, RA, DEC, ln D_L, ln chirpmass, eta, chi1, chi2, \alpha^2 (the coupling parameter)

dCS_IMRPhenomD_root_alpha- 8 dimensions – cos inclination, RA, DEC, ln D_L, ln chirpmass, eta, chi1, chi2, \sqrt \alpha (in km) (the coupling parameter)

IMRPhenomPv2 - 9 dimensions – cos J_N, ln chirpmass, eta, |chi1|, |chi1|, theta_1, theta_2, phi_1, phi_2

Parameters

statistics_filename	Filename to output sampling statistics, if empty string, not output
chain_filename	Filename to output data (chain 0 only), if empty string, not output
auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.10.2.20 RJPTMCMC_method_specific_prep()

```
void RJPTMCMC_method_specific_prep (
```

```
            std::string generation_method,
            int max_dim,
            int min_dim,
            double * seeding_var,
            bool local_seeding )
```

Unpacks MCMC parameters for method specific initiation (RJ version)

Populates seeding vector if non supplied, populates mcmc_Nmod, populates mcmc_log_beta, populates mcmc_↩ intrinsic

8.10.2.21 RJPTMCMC_MH_GW()

```
void RJPTMCMC_MH_GW (
            double *** output,
            int *** status,
            int max_dim,
            int min_dim,
            int N_steps,
            int chain_N,
            double * initial_pos,
            int * initial_status,
            double * seeding_var,
            double * chain_temps,
            int swp_freq,
            double(*)(double *param, int *status, int dimension, int chain_id) log_prior,
            void(*)(double *current_param, double *proposed_param, int *current_status, int
*proposed_status, int max_dim, int chain_id, double step_width) RJ_proposal,
            int numThreads,
            bool pool,
            bool show_prog,
            int num_detectors,
            std::complex< double > ** data,
            double ** noise_psd,
            double ** frequencies,
            int * data_length,
            double gps_time,
            std::string * detectors,
            int Nmod_max,
            int * bppe,
            std::string generation_method,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string checkpoint_file )
```

Wrapper for the RJPTMCMC_MH function, specifically for GW analysis.

Handles the details of setting up the MCMC sampler and wraps the fisher and log likelihood to conform to the format of the sampler

NOTE – This sampler as a whole is NOT thread safe. There is global memory declared for each call to MCMC_↩ MH_GW, so separate samplers should not be run in the same process space.

Supported parameter combinations:

IMRPhenomD - 8 dimensions –

MIN DIMENSIONS – cos inclination, RA, DEC, ln D_L, ln chirpmass, eta, chi1, chi2, TRANSDIMENSIONAL DIM↩ ENSIONS – ppE parameters for the bppe array specified

If RJ_proposal is NULL, a default proposal is used.

Parameters

statistics_filename	Filename to output sampling statistics, if empty string, not output
chain_filename	Filename to output data (chain 0 only), if empty string, not output
auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.11 src/mcmc_sampler.cpp File Reference

```
#include "mcmc_sampler.h"
#include "autocorrelation.h"
#include "util.h"
#include "mcmc_sampler_internals.h"
#include "threadPool.h"
#include <iostream>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <time.h>
#include <condition_variable>
#include <mutex>
#include <thread>
#include <vector>
#include <queue>
#include <functional>
#include <unistd.h>
#include <fstream>
#include "adolc/adolc.h"
```

Include dependency graph for mcmc_sampler.cpp:

Classes

- class Comparator

  Class to facilitate the comparing of chains for priority.
- class Comparatorswap
- class ThreadPool

Macros

- #define **omp** ignore

Functions

- void continue_RJPTMCMC_MH_internal (std::string start_checkpoint_file, double ***output, int ***status, int N_steps, int swp_freq, std::function< double(double *, int *, int, int)> log_prior, std::function< double(double *, int *, int, int)> log_likelihood, std::function< void(double *, int *, int, double **, int)>fisher, std::function< void(double *, double *, int *, int *, int, int, int)> RJ_proposal, int numThreads, bool pool, bool show_prog, bool update_RJ_width, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string end_checkpoint_file)

  Routine to take a checkpoint file and begin a new chain at said checkpoint.
- void RJPTMCMC_MH_internal (double ***output, int ***parameter_status, int max_dimension, int min_dimension, int N_steps, int chain_N, double *initial_pos, int *initial_status, double *seeding_var, double *chain_temps, int swp_freq, std::function< double(double *, int *, int, int)> log_prior, std::function< double(double *, int *, int, int)> log_likelihood, std::function< void(double *, int *, int, double **, int)>fisher, std::function< void(double *, double *, int *, int *, int, int, int)> RJ_proposal, int numThreads, bool pool, bool show_prog, bool update_RJ_width, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string checkpoint_file)

  Generic reversable jump sampler, where the likelihood, prior, and reversable jump proposal are parameters supplied by the user.
- void PTMCMC_MH_dynamic_PT_alloc_internal (double ***output, int dimension, int N_steps, int chain_N, int max_chain_N_thermo_ensemble, double *initial_pos, double *seeding_var, double *chain_temps, int swp_freq, int t0, int nu, std::string chain_distribution_scheme, std::function< double(double *, int *, int, int)> log_prior, std::function< double(double *, int *, int, int)> log_likelihood, std::function< void(double *, int *, int, double **, int)>fisher, int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string likelihood_log_filename, std::string checkpoint_file)

  Dyanmically tunes an MCMC for optimal spacing. step width, and chain number.
- void PTMCMC_MH_internal (double ***output, int dimension, int N_steps, int chain_N, double *initial_pos, double *seeding_var, double *chain_temps, int swp_freq, std::function< double(double *, int *, int, int)> log_prior, std::function< double(double *, int *, int, int)> log_likelihood, std::function< void(double *, int *, int, double **, int)>fisher, int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string checkpoint_file)

  Generic sampler, where the likelihood, prior are parameters supplied by the user.
- void continue_PTMCMC_MH_internal (std::string start_checkpoint_file, double ***output, int N_steps, int swp_freq, std::function< double(double *, int *, int, int)> log_prior, std::function< double(double *, int *, int, int)> log_likelihood, std::function< void(double *, int *, int, double **, int)>fisher, int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string end_checkpoint_file)

  Routine to take a checkpoint file and begin a new chain at said checkpoint.
- void PTMCMC_MH_step_incremental (sampler *sampler, int increment)

  Internal function that runs the actual loop for the sampler – increment version.
- void PTMCMC_MH_loop (sampler *sampler)

  Internal function that runs the actual loop for the sampler.
- void **mcmc_step_threaded** (int j)
- void **mcmc_swap_threaded** (int i, int j)
- void PTMCMC_MH (double ***output, int dimension, int N_steps, int chain_N, double *initial_pos, double *seeding_var, double *chain_temps, int swp_freq, double(*log_prior)(double *param, int dimension), double(*log_likelihood)(double *param, int dimension), void(*fisher)(double *param, int dimension, double **fisher), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string checkpoint_file)
- void PTMCMC_MH (double ***output, int dimension, int N_steps, int chain_N, double *initial_pos, double *seeding_var, double *chain_temps, int swp_freq, double(*log_prior)(double *param, int dimension, int chain_id), double(*log_likelihood)(double *param, int dimension, int chain_id), void(*fisher)(double *param, int dimension, double **fisher, int chain_id), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string checkpoint_file)

- void continue_PTMCMC_MH (std::string start_checkpoint_file, double ***output, int N_steps, int swp_freq, double(*log_prior)(double *param, int dimension, int chain_id), double(*log_likelihood)(double *param, int dimension, int chain_id), void(*fisher)(double *param, int dimension, double **fisher, int chain_id), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string end_checkpoint_file)
- void continue_PTMCMC_MH (std::string start_checkpoint_file, double ***output, int N_steps, int swp_freq, double(*log_prior)(double *param, int dimension), double(*log_likelihood)(double *param, int dimension), void(*fisher)(double *param, int dimension, double **fisher), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string end_checkpoint_file)
- void PTMCMC_MH_dynamic_PT_alloc (double ***output, int dimension, int N_steps, int chain_N, int max_chain_N_thermo_ensemble, double *initial_pos, double *seeding_var, double *chain_temps, int swp_freq, int t0, int nu, std::string chain_distribution_scheme, double(*log_prior)(double *param, int dimension), double(*log_likelihood)(double *param, int dimension), void(*fisher)(double *param, int dimension, double **fisher), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string likelihood_log_filename, std::string checkpoint_file)
- void PTMCMC_MH_dynamic_PT_alloc (double ***output, int dimension, int N_steps, int chain_N, int max_chain_N_thermo_ensemble, double *initial_pos, double *seeding_var, double *chain_temps, int swp_freq, int t0, int nu, std::string chain_distribution_scheme, double(*log_prior)(double *param, int dimension, int chain_id), double(*log_likelihood)(double *param, int dimension, int chain_id), void(*fisher)(double *param, int dimension, double **fisher, int chain_id), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string likelihood_log_filename, std::string checkpoint_file)
- void RJPTMCMC_MH (double ***output, int ***parameter_status, int max_dimension, int min_dimension, int N_steps, int chain_N, double *initial_pos, int *initial_status, double *seeding_var, double *chain_temps, int swp_freq, double(*log_prior)(double *param, int *status, int max_dimension, int chain_id), double(*log_likelihood)(double *param, int *status, int max_dimension, int chain_id), void(*fisher)(double *param, int *status, int max_dimension, double **fisher, int chain_id), void(*RJ_proposal)(double *current_param, double *proposed_param, int *current_status, int *proposed_status, int max_dimension, int chain_id), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string checkpoint_file)
- void RJPTMCMC_MH (double ***output, int ***parameter_status, int max_dimension, int min_dimension, int N_steps, int chain_N, double *initial_pos, int *initial_status, double *seeding_var, double *chain_temps, int swp_freq, double(*log_prior)(double *param, int *status, int max_dimension, int chain_id), double(*log_likelihood)(double *param, int *status, int max_dimension, int chain_id), void(*fisher)(double *param, int *status, int max_dimension, double **fisher, int chain_id), void(*RJ_proposal)(double *current_param, double *proposed_param, int *current_status, int *proposed_status, int max_dimension, int chain_id, double gaussian_width), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string checkpoint_file)
- void continue_RJPTMCMC_MH (std::string start_checkpoint_file, double ***output, int ***status, int N_steps, int swp_freq, double(*log_prior)(double *param, int *status, int max_dimension, int chain_id), double(*log_likelihood)(double *param, int *status, int max_dimension, int chain_id), void(*fisher)(double *param, int *status, int max_dimension, double **fisher, int chain_id), void(*RJ_proposal)(double *current_param, double *proposed_param, int *current_status, int *proposed_status, int max_dimension, int chain_id, double gaussian_width), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string end_checkpoint_file)
- void continue_RJPTMCMC_MH (std::string start_checkpoint_file, double ***output, int ***status, int N_steps, int swp_freq, double(*log_prior)(double *param, int *status, int max_dimension, int chain_id), double(*log_likelihood)(double *param, int *status, int max_dimension, int chain_id), void(*fisher)(double *param, int *status, int max_dimension, double **fisher, int chain_id), void(*RJ_proposal)(double *current_param, double *proposed_param, int *current_status, int *proposed_status, int max_dimension, int chain_id), int numThreads, bool pool, bool show_prog, std::string statistics_filename, std::string chain_filename, std::string auto_corr_filename, std::string likelihood_log_filename, std::string end_checkpoint_file)

Variables

- const gsl_rng_type * **T**

- gsl_rng * **r**
- sampler * **samplerptr**
- ThreadPool * **poolptr**

8.11.1 Detailed Description

Source file for the sampler foundation

Source file for generic MCMC sampler. Sub routines that are application agnostic are housed in mcmc_sampler↩ _internals

8.11.2 Function Documentation

8.11.2.1 continue_PTMCMC_MH() [1/2]

```
void continue_PTMCMC_MH (
            std::string start_checkpoint_file,
            double *** output,
            int N_steps,
            int swp_freq,
            double(*)(double *param, int dimension, int chain_id) log_prior,
            double(*)(double *param, int dimension, int chain_id) log_likelihood,
            void(*)(double *param, int dimension, double **fisher, int chain_id) fisher,
            int numThreads,
            bool pool,
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string end_checkpoint_file )
```

Parameters

	start_checkpoint_file	File for starting checkpoint
out	*output*	output array, dimensions: output[chain_N][N_steps][dimension]
	N_steps	Number of new steps to take
	swp_freq	frequency of swap attempts between temperatures
	log_prior	Funcion pointer for the log_prior
	log_likelihood	Function pointer for the log_likelihood
	fisher	Function pointer for the fisher - if NULL, fisher steps are not used
	numThreads	Number of threads to use
	pool	Boolean for whether to use deterministic'' vsstochastic" sampling
	show_prog	Boolean for whether to show progress or not (turn off for cluster runs
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	end_checkpoint_file	Filename to output data for checkpoint at the end of the continued run, if empty string, not saved

8.11.2.2 continue_PTMCMC_MH() [2/2]

```
void continue_PTMCMC_MH (
            std::string start_checkpoint_file,
            double *** output,
            int N_steps,
            int swp_freq,
            double(*)(double *param, int dimension) log_prior,
            double(*)(double *param, int dimension) log_likelihood,
            void(*)(double *param, int dimension, double **fisher) fisher,
            int numThreads,
            bool pool,
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string end_checkpoint_file )
```

Parameters

	start_checkpoint_file	File for starting checkpoint
out	*output*	output array, dimensions: output[chain_N][N_steps][dimension]
	N_steps	Number of new steps to take
	swp_freq	frequency of swap attempts between temperatures
	log_prior	Funcion pointer for the log_prior
	log_likelihood	Function pointer for the log_likelihood
	fisher	Function pointer for the fisher - if NULL, fisher steps are not used
	numThreads	Number of threads to use
	pool	Boolean for whether to use deterministic'' vsstochastic" sampling
	show_prog	Boolean for whether to show progress or not (turn off for cluster runs
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	end_checkpoint_file	Filename to output data for checkpoint at the end of the continued run, if empty string, not saved

8.11.2.3 continue_PTMCMC_MH_internal()

```
void continue_PTMCMC_MH_internal (
            std::string start_checkpoint_file,
            double *** output,
            int N_steps,
            int swp_freq,
```

```
            std::function< double(double *, int *, int, int)> log_prior,
            std::function< double(double *, int *, int, int)> log_likelihood,
            std::function< void(double *, int *, int, double **, int)> fisher,
            int numThreads,
            bool pool,
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string end_checkpoint_file )
```

Routine to take a checkpoint file and begin a new chain at said checkpoint.

See MCMC_MH_internal for more details of parameters (pretty much all the same)

Parameters

	start_checkpoint_file	File for starting checkpoint
out	*output*	output array, dimensions: output[chain_N][N_steps][dimension]
	N_steps	Number of new steps to take
	swp_freq	frequency of swap attempts between temperatures
	log_prior	std::function for the log_prior function – takes double *position, int dimension, int chain_id
	log_likelihood	std::function for the log_likelihood function – takes double *position, int dimension, int chain_id
	fisher	std::function for the fisher function – takes double *position, int dimension, double **output_fisher, int chain_id
	numThreads	Number of threads to use
	pool	Boolean for whether to use deterministic'' vsstochastic" sampling
	show_prog	Boolean for whether to show progress or not (turn off for cluster runs
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	end_checkpoint_file	Filename to output data for checkpoint at the end of the continued run, if empty string, not saved

8.11.2.4 continue_RJPTMCMC_MH() [1/2]

```
void continue_RJPTMCMC_MH (
            std::string start_checkpoint_file,
            double *** output,
            int *** status,
            int N_steps,
            int swp_freq,
            double(*)(double *param, int *status, int max_dimension, int chain_id) log_prior,
            double(*)(double *param, int *status, int max_dimension, int chain_id) log_↩
likelihood,
```

```
            void(*)(double *param, int *status, int max_dimension, double **fisher, int chain↩
_id) fisher,
            void(*)(double *current_param, double *proposed_param, int *current_status, int
*proposed_status, int max_dimension, int chain_id, double gaussian_width) RJ_proposal,
            int numThreads,
            bool pool,
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string end_checkpoint_file )
```

Parameters

	start_checkpoint_file	File for starting checkpoint
out	*output*	output array, dimensions: output[chain_N][N_steps][dimension]
out	*status*	output parameter status array, dimensions: status[chain_N][N_steps][dimension]
	N_steps	Number of new steps to take
	swp_freq	frequency of swap attempts between temperatures
	RJ_proposal	std::function for the log_likelihood function – takes double *position, int *param_status,int dimension, int chain_id
	numThreads	Number of threads to use
	pool	Boolean for whether to use deterministic'' vsstochastic" sampling
	show_prog	Boolean for whether to show progress or not (turn off for cluster runs
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	end_checkpoint_file	Filename to output data for checkpoint at the end of the continued run, if empty string, not saved

8.11.2.5 continue_RJPTMCMC_MH() [2/2]

```
void continue_RJPTMCMC_MH (
            std::string start_checkpoint_file,
            double *** output,
            int *** status,
            int N_steps,
            int swp_freq,
            double(*)(double *param, int *status, int max_dimension, int chain_id) log_prior,
            double(*)(double *param, int *status, int max_dimension, int chain_id) log_↩
likelihood,
            void(*)(double *param, int *status, int max_dimension, double **fisher, int chain↩
_id) fisher,
            void(*)(double *current_param, double *proposed_param, int *current_status, int
*proposed_status, int max_dimension, int chain_id) RJ_proposal,
            int numThreads,
```

```
            bool pool,
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string end_checkpoint_file )
```

Parameters

	start_checkpoint_file	File for starting checkpoint
out	*output*	output array, dimensions: output[chain_N][N_steps][dimension]
out	*status*	output parameter status array, dimensions: status[chain_N][N_steps][dimension]
	N_steps	Number of new steps to take
	swp_freq	frequency of swap attempts between temperatures
	RJ_proposal	std::function for the log_likelihood function – takes double ∗position, int ∗param_status,int dimension, int chain_id
	numThreads	Number of threads to use
	pool	Boolean for whether to use deterministic'' vsstochastic" sampling
	show_prog	Boolean for whether to show progress or not (turn off for cluster runs
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	end_checkpoint_file	Filename to output data for checkpoint at the end of the continued run, if empty string, not saved

8.11.2.6 continue_RJPTMCMC_MH_internal()

```
void continue_RJPTMCMC_MH_internal (
            std::string start_checkpoint_file,
            double *** output,
            int *** status,
            int N_steps,
            int swp_freq,
            std::function< double(double *, int *, int, int)> log_prior,
            std::function< double(double *, int *, int, int)> log_likelihood,
            std::function< void(double *, int *, int, double **, int)> fisher,
            std::function< void(double *, double *, int *, int *, int, int, int)> RJ_proposal,
            int numThreads,
            bool pool,
            bool show_prog,
            bool update_RJ_width,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string end_checkpoint_file )
```

Routine to take a checkpoint file and begin a new chain at said checkpoint.

See MCMC_MH_internal for more details of parameters (pretty much all the same)

Parameters

	start_checkpoint_file	File for starting checkpoint
out	*output*	output array, dimensions: output[chain_N][N_steps][dimension]
out	*status*	output parameter status array, dimensions: status[chain_N][N_steps][dimension]
	N_steps	Number of new steps to take
	swp_freq	frequency of swap attempts between temperatures
	log_prior	std::function for the log_prior function – takes double *position, int *param_status, int dimension, int chain_id
	log_likelihood	std::function for the log_likelihood function – takes double *position, int *param_status,int dimension, int chain_id
	fisher	std::function for the fisher function – takes double *position, int *param_status,int dimension, double **output_fisher, int chain_id
	RJ_proposal	std::function for the log_likelihood function – takes double *position, int *param_status,int dimension, int chain_id
	numThreads	Number of threads to use
	pool	Boolean for whether to use deterministic'' vsstochastic" sampling
	show_prog	Boolean for whether to show progress or not (turn off for cluster runs
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output – if multiple cold chains, it will append each output to the other, and write out the total
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	end_checkpoint_file	Filename to output data for checkpoint at the end of the continued run, if empty string, not saved

8.11.2.7 PTMCMC_MH() [1/2]

```
void PTMCMC_MH (
            double *** output,
            int dimension,
            int N_steps,
            int chain_N,
            double * initial_pos,
            double * seeding_var,
            double * chain_temps,
            int swp_freq,
            double(*)(double *param, int dimension) log_prior,
            double(*)(double *param, int dimension) log_likelihood,
            void(*)(double *param, int dimension, double **fisher) fisher,
            int numThreads,
            bool pool,
```

```
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string checkpoint_file )
```

Parameters

out	*output*	Output chains, shape is double[chain_N, N_steps,dimension]
	dimension	dimension of the parameter space being explored
	N_steps	Number of total steps to be taken, per chain
	chain_N	Number of chains
	initial_pos	Initial position in parameter space - shape double[dimension]
	seeding_var	Variance of the normal distribution used to seed each chain higher than 0 - shape double[dimension]
	chain_temps	Double array of temperatures for the chains
	swp_freq	the frequency with which chains are swapped
	log_prior	Funcion pointer for the log_prior
	log_likelihood	Function pointer for the log_likelihood
	fisher	Function pointer for the fisher - if NULL, fisher steps are not used
	numThreads	Number of threads to use (=1 is single threaded)
	pool	boolean to use stochastic chain swapping (MUST have $>$2 threads)
	show_prog	boolean whether to print out progress (for example, should be set to `‘false'' if submitting to a cluster)
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.11.2.8 PTMCMC_MH() [2/2]

```
void PTMCMC_MH (
            double *** output,
            int dimension,
            int N_steps,
            int chain_N,
            double * initial_pos,
            double * seeding_var,
            double * chain_temps,
            int swp_freq,
            double(*)(double *param, int dimension, int chain_id) log_prior,
            double(*)(double *param, int dimension, int chain_id) log_likelihood,
            void(*)(double *param, int dimension, double **fisher, int chain_id) fisher,
            int numThreads,
            bool pool,
            bool show_prog,
```

```
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string checkpoint_file )
```

Parameters

out	*output*	Output chains, shape is double[chain_N, N_steps,dimension]
	dimension	dimension of the parameter space being explored
	N_steps	Number of total steps to be taken, per chain
	chain_N	Number of chains
	initial_pos	Initial position in parameter space - shape double[dimension]
	seeding_var	Variance of the normal distribution used to seed each chain higher than 0 - shape double[dimension]
	chain_temps	Double array of temperatures for the chains
	swp_freq	the frequency with which chains are swapped
	log_prior	Funcion pointer for the log_prior
	log_likelihood	Function pointer for the log_likelihood
	fisher	Function pointer for the fisher - if NULL, fisher steps are not used
	numThreads	Number of threads to use (=1 is single threaded)
	pool	boolean to use stochastic chain swapping (MUST have $>$2 threads)
	show_prog	boolean whether to print out progress (for example, should be set to ``false'' if submitting to a cluster)
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.11.2.9 PTMCMC_MH_dynamic_PT_alloc() [1/2]

```
void PTMCMC_MH_dynamic_PT_alloc (
            double *** output,
            int dimension,
            int N_steps,
            int chain_N,
            int max_chain_N_thermo_ensemble,
            double * initial_pos,
            double * seeding_var,
            double * chain_temps,
            int swp_freq,
            int t0,
            int nu,
            std::string chain_distribution_scheme,
            double(*)(double *param, int dimension) log_prior,
            double(*)(double *param, int dimension) log_likelihood,
            void(*)(double *param, int dimension, double **fisher) fisher,
```

```
            int numThreads,
            bool pool,
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string likelihood_log_filename,
            std::string checkpoint_file )
```

Parameters

out	*output*	Output chains, shape is double[max_chain_N, N_steps,dimension]
	dimension	dimension of the parameter space being explored
	N_steps	Number of total steps to be taken, per chain AFTER chain allocation
	chain_N	Maximum number of chains to use
	max_chain_N_thermo_ensemble	Maximum number of chains to use in the thermodynamic ensemble (may use less)
	initial_pos	Initial position in parameter space - shape double[dimension]
	seeding_var	Variance of the normal distribution used to seed each chain higher than 0 - shape double[dimension]
	chain_temps	Final chain temperatures used – should be shape double[chain_N]
	swp_freq	the frequency with which chains are swapped
	t0	Time constant of the decay of the chain dynamics ($\sim$1000)
	nu	Initial amplitude of the dynamics ($\sim$100)
	log_prior	Funcion pointer for the log_prior
	log_likelihood	Function pointer for the log_likelihood
	fisher	Function pointer for the fisher - if NULL, fisher steps are not used
	numThreads	Number of threads to use (=1 is single threaded)
	pool	boolean to use stochastic chain swapping (MUST have $>$2 threads)
	show_prog	boolean whether to print out progress (for example, should be set to `‘false" if submitting to a cluster)
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.11.2.10 PTMCMC_MH_dynamic_PT_alloc() [2/2]

```
void PTMCMC_MH_dynamic_PT_alloc (
            double *** output,
            int dimension,
            int N_steps,
            int chain_N,
            int max_chain_N_thermo_ensemble,
            double * initial_pos,
            double * seeding_var,
```

```
            double * chain_temps,
            int swp_freq,
            int t0,
            int nu,
            std::string chain_distribution_scheme,
            double(*)(double *param, int dimension, int chain_id) log_prior,
            double(*)(double *param, int dimension, int chain_id) log_likelihood,
            void(*)(double *param, int dimension, double **fisher, int chain_id) fisher,
            int numThreads,
            bool pool,
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string likelihood_log_filename,
            std::string checkpoint_file )
```

Parameters

out	*output*	Output chains, shape is double[max_chain_N, N_steps,dimension]
	dimension	dimension of the parameter space being explored
	N_steps	Number of total steps to be taken, per chain AFTER chain allocation
	chain_N	Maximum number of chains to use
	max_chain_N_thermo_ensemble	Maximum number of chains to use in the thermodynamic ensemble (may use less)
	initial_pos	Initial position in parameter space - shape double[dimension]
	seeding_var	Variance of the normal distribution used to seed each chain higher than 0 - shape double[dimension]
	chain_temps	Final chain temperatures used – should be shape double[chain_N]
	swp_freq	the frequency with which chains are swapped
	t0	Time constant of the decay of the chain dynamics ($\sim$1000)
	nu	Initial amplitude of the dynamics ($\sim$100)
	log_prior	Funcion pointer for the log_prior
	log_likelihood	Function pointer for the log_likelihood
	fisher	Function pointer for the fisher - if NULL, fisher steps are not used
	numThreads	Number of threads to use (=1 is single threaded)
	pool	boolean to use stochastic chain swapping (MUST have $>$2 threads)
	show_prog	boolean whether to print out progress (for example, should be set to `‘false" if submitting to a cluster)
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.11.2.11 PTMCMC_MH_dynamic_PT_alloc_internal()

```
void PTMCMC_MH_dynamic_PT_alloc_internal (
            double *** output,
```

```
int dimension,
int N_steps,
int chain_N,
int max_chain_N_thermo_ensemble,
double * initial_pos,
double * seeding_var,
double * chain_temps,
int swp_freq,
int t0,
int nu,
std::string chain_distribution_scheme,
std::function< double(double *, int *, int, int)> log_prior,
std::function< double(double *, int *, int, int)> log_likelihood,
std::function< void(double *, int *, int, double **, int)> fisher,
int numThreads,
bool pool,
bool show_prog,
std::string statistics_filename,
std::string chain_filename,
std::string likelihood_log_filename,
std::string checkpoint_file )
```

Dyanmically tunes an MCMC for optimal spacing. step width, and chain number.

NOTE: nu, and t0 parameters determine the dynamics, so these are important quantities. nu is related to how many swap attempts it takes to substantially change the temperature ladder, why t0 determines the length of the total dyanimcally period. Moderate initial choices would be 10 and 1000, respectively.

Based on arXiv:1501.05823v3

Currently, Chain number is fixed

max_chain_N_thermo_ensemble sets the maximium number of chains to use to in successively hotter chains to cover the likelihood surface while targeting an optimal swap acceptance target_swp_acc.

max_chain_N determines the total number of chains to run once thermodynamic equilibrium has been reached. This results in chains being added after the initial PT dynamics have finished according to chain_distribution_↩ scheme.

If no preference, set max_chain_N_thermo_ensemble = max_chain_N = numThreads = (number of cores (number of threads if hyperthreaded))– this will most likely be the most optimal configuration. If the number of cores on the system is low, you may want to use n∗numThreads for some integer n instead, depending on the system.

chain_distribution_scheme:

"cold": All chains are added at T=1 (untempered)

"refine": Chains are added between the optimal temps geometrically – this may be a good option as it will be a good approximation of the ideal distribution of chains, while keeping the initial dynamical time low

"double": Chains are added in order of rising temperature that mimic the distribution achieved by the earier PT dynamics

"half_ensemble": For every cold chain added, half of the ensemble is added again. Effectively, two cold chains for every ensemble

Parameters

out	*output*	Output chains, shape is double[max_chain_N, N_steps,dimension]
	dimension	dimension of the parameter space being explored
	N_steps	Number of total steps to be taken, per chain AFTER chain allocation
	chain_N	Maximum number of chains to use
	max_chain_N_thermo_ensemble	Maximum number of chains to use in the thermodynamic ensemble (may use less)
	initial_pos	Initial position in parameter space - shape double[dimension]
	seeding_var	Variance of the normal distribution used to seed each chain higher than 0 - shape double[dimension]
out	*chain_temps*	Final chain temperatures used – should be shape double[chain_N]
	swp_freq	the frequency with which chains are swapped
	t0	Time constant of the decay of the chain dynamics ($\sim$1000)
	nu	Initial amplitude of the dynamics ($\sim$100)
	log_prior	std::function for the log_prior function – takes double $*$position, int dimension, int chain_id
	log_likelihood	std::function for the log_likelihood function – takes double $*$position, int dimension, int chain_id
	fisher	std::function for the fisher function – takes double $*$position, int dimension, double $**$output_fisher, int chain_id
	numThreads	Number of threads to use (=1 is single threaded)
	pool	boolean to use stochastic chain swapping (MUST have $>$2 threads)
	show_prog	boolean whether to print out progress (for example, should be set to ``false'' if submitting to a cluster)
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.11.2.12 PTMCMC_MH_internal()

```
void PTMCMC_MH_internal (
            double *** output,
            int dimension,
            int N_steps,
            int chain_N,
            double * initial_pos,
            double * seeding_var,
            double * chain_temps,
            int swp_freq,
            std::function< double(double *, int *, int, int)> log_prior,
            std::function< double(double *, int *, int, int)> log_likelihood,
            std::function< void(double *, int *, int, double **, int)> fisher,
            int numThreads,
```

```
bool pool,
bool show_prog,
std::string statistics_filename,
std::string chain_filename,
std::string auto_corr_filename,
std::string likelihood_log_filename,
std::string checkpoint_file )
```

Generic sampler, where the likelihood, prior are parameters supplied by the user.

Base of the sampler, generic, with user supplied quantities for most of the samplers properties

Uses the Metropolis-Hastings method, with the option for Fisher/MALA steps if the Fisher routine is supplied.

3 modes to use -

single threaded (numThreads = 1) runs single threaded

multi-threaded `‘deterministic'' (numThreads>1 ; pool = false) progresses each chain in parallel for swp_freq steps, then waits for all threads to complete before swapping temperatures in sequenctial order (j, j+1) then (j+1, j+2) etc (sequenctially)

multi-threaded `‘stochastic'' (numThreads>2 ; pool = true) progresses each chain in parallel by queueing each temperature and evaluating them in the order they were submitted. Once finished, the threads are queued to swap, where they swapped in the order they are submitted. This means the chains are swapped randomly, and the chains do NOT finish at the same time. The sampler runs until the the 0th chain reaches the step number

Note on limits: In the prior function, if a set of parameters should be disallowed, return -std::numeric_↩ limits<double>::infinity() – (this is in the <limits> file in std)

Format for the auto_corr file (compatable with csv, dat, txt extensions): each row is a dimension of the cold chain, with the first row being the lengths used for the auto-corr calculation:

lengths: length1 , length2 ...

dim1: length1 , length2 ...

Format for the chain file (compatable with csv, dat, txt extensions): each row is a step, each column a dimension:

Step1: dim1 , dim2 , ...

Step2: dim1 , dim2 , ...

Statistics_filename : should be txt extension

checkpoint_file : This file saves the final position of all the chains, as well as other metadata, and can be loaded by the function <FUNCTION> to continue the chain from the point it left off. Not meant to be read by humans, the data order is custom to this software library. An empty string ("") means no checkpoint will be saved. For developers, the contents are:

dimension, # of chains

temps of chains

Stepping widths of all chains

Final position of all chains

Parameters

out	*output*	Output chains, shape is double[chain_N, N_steps,dimension]
	dimension	dimension of the parameter space being explored
	N_steps	Number of total steps to be taken, per chain
	chain_N	Number of chains
	initial_pos	Initial position in parameter space - shape double[dimension]
	seeding_var	Variance of the normal distribution used to seed each chain higher than 0 - shape double[dimension]
	chain_temps	Double array of temperatures for the chains
	swp_freq	the frequency with which chains are swapped
	log_prior	std::function for the log_prior function – takes double ∗position, int dimension, int chain_id
	log_likelihood	std::function for the log_likelihood function – takes double ∗position, int dimension, int chain_id
	fisher	std::function for the fisher function – takes double ∗position, int dimension, double ∗∗output_fisher, int chain_id
	numThreads	Number of threads to use (=1 is single threaded)
	pool	boolean to use stochastic chain swapping (MUST have >2 threads)
	show_prog	boolean whether to print out progress (for example, should be set to ``false'' if submitting to a cluster)
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.11.2.13 PTMCMC_MH_loop()

```
void PTMCMC_MH_loop (
            sampler * sampler )
```

Internal function that runs the actual loop for the sampler.

8.11.2.14 PTMCMC_MH_step_incremental()

```
void PTMCMC_MH_step_incremental (
            sampler * sampler,
            int increment )
```

Internal function that runs the actual loop for the sampler – increment version.

The regular loop function runs for the entire range, this increment version will only step ``increment'' steps – asynchronous: steps are measured by the cold chains NEEDS TO CHANGE

8.11.2.15 RJPTMCMC_MH() [1/2]

```
void RJPTMCMC_MH (
            double *** output,
            int *** parameter_status,
            int max_dimension,
            int min_dimension,
            int N_steps,
            int chain_N,
            double * initial_pos,
            int * initial_status,
            double * seeding_var,
            double * chain_temps,
            int swp_freq,
            double(*)(double *param, int *status, int max_dimension, int chain_id) log_prior,
            double(*)(double *param, int *status, int max_dimension, int chain_id) log_↩
likelihood,
            void(*)(double *param, int *status, int max_dimension, double **fisher, int chain↩
_id) fisher,
            void(*)(double *current_param, double *proposed_param, int *current_status, int
*proposed_status, int max_dimension, int chain_id) RJ_proposal,
            int numThreads,
            bool pool,
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string checkpoint_file )
```

Parameters

out	*output*	Output chains, shape is double[chain_N, N_steps,dimension]
out	*parameter_status*	Parameter status for each step corresponding to the output chains, shape is double[chain_N, N_steps,dimension]
	max_dimension	maximum dimension of the parameter space being explored – only consideration is memory, as memory scales with dimension. Keep this reasonable, unless memory is REALLY not an issue
	min_dimension	minimum dimension of the parameter space being explored >=1
	N_steps	Number of total steps to be taken, per chain
	chain_N	Number of chains
	initial_pos	Initial position in parameter space - shape double[dimension]
	initial_status	Initial status of the parameters in the initial position in parameter space - shape int[max_dim]
	seeding_var	Variance of the normal distribution used to seed each chain higher than 0 - shape double[max_dimension] – initial seeding of zero corresponds to the dimension turned off initially
	chain_temps	Double array of temperatures for the chains
	swp_freq	the frequency with which chains are swapped
	RJ_proposal	std::function for the log_likelihood function – takes double ∗position, int ∗param_status,int dimension, int chain_id
	numThreads	Number of threads to use (=1 is single threaded)
	pool	boolean to use stochastic chain swapping (MUST have >2 threads)
	show_prog	boolean whether to print out progress (for example, should be set to `‘false'' if submitting to a cluster)
	statistics_filename	Filename to output sampling statistics, if empty string, not output

Parameters

	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.11.2.16 RJPTMCMC_MH() [2/2]

```
void RJPTMCMC_MH (
            double *** output,
            int *** parameter_status,
            int max_dimension,
            int min_dimension,
            int N_steps,
            int chain_N,
            double * initial_pos,
            int * initial_status,
            double * seeding_var,
            double * chain_temps,
            int swp_freq,
            double(*)(double *param, int *status, int max_dimension, int chain_id) log_prior,
            double(*)(double *param, int *status, int max_dimension, int chain_id) log_↩
likelihood,
            void(*)(double *param, int *status, int max_dimension, double **fisher, int chain↩
_id) fisher,
            void(*)(double *current_param, double *proposed_param, int *current_status, int
*proposed_status, int max_dimension, int chain_id, double gaussian_width) RJ_proposal,
            int numThreads,
            bool pool,
            bool show_prog,
            std::string statistics_filename,
            std::string chain_filename,
            std::string auto_corr_filename,
            std::string likelihood_log_filename,
            std::string checkpoint_file )
```

Parameters

out	*output*	Output chains, shape is double[chain_N, N_steps,dimension]
out	*parameter_status*	Parameter status for each step corresponding to the output chains, shape is double[chain_N, N_steps,dimension]
	max_dimension	maximum dimension of the parameter space being explored – only consideration is memory, as memory scales with dimension. Keep this reasonable, unless memory is REALLY not an issue
	min_dimension	minimum dimension of the parameter space being explored >=1
	N_steps	Number of total steps to be taken, per chain
	chain_N	Number of chains
	initial_pos	Initial position in parameter space - shape double[dimension]
	initial_status	Initial status of the parameters in the initial position in parameter space - shape int[max_dim]

Parameters

	seeding_var	Variance of the normal distribution used to seed each chain higher than 0 - shape double[max_dimension] – initial seeding of zero corresponds to the dimension turned off initially
	chain_temps	Double array of temperatures for the chains
	swp_freq	the frequency with which chains are swapped
	RJ_proposal	std::function for the log_likelihood function – takes double ∗position, int ∗param_status,int dimension, int chain_id
	numThreads	Number of threads to use (=1 is single threaded)
	pool	boolean to use stochastic chain swapping (MUST have >2 threads)
	show_prog	boolean whether to print out progress (for example, should be set to ``false'' if submitting to a cluster)
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.11.2.17 RJPTMCMC_MH_internal()

```
void RJPTMCMC_MH_internal (
             double *** output,
             int *** parameter_status,
             int max_dimension,
             int min_dimension,
             int N_steps,
             int chain_N,
             double * initial_pos,
             int * initial_status,
             double * seeding_var,
             double * chain_temps,
             int swp_freq,
             std::function< double(double *, int *, int, int)> log_prior,
             std::function< double(double *, int *, int, int)> log_likelihood,
             std::function< void(double *, int *, int, double **, int)> fisher,
             std::function< void(double *, double *, int *, int *, int, int, int)> RJ_proposal,
             int numThreads,
             bool pool,
             bool show_prog,
             bool update_RJ_width,
             std::string statistics_filename,
             std::string chain_filename,
             std::string auto_corr_filename,
             std::string likelihood_log_filename,
             std::string checkpoint_file )
```

Generic reversable jump sampler, where the likelihood, prior, and reversable jump proposal are parameters supplied by the user.

Note: Using a min_dimension tells the sampler that there is a base model'', and that the dimensions from min_dim to max_dim aresmall'' corrections to that model. This helps inform some of the proposal algorithms and speeds up computation. If using discrete models with no overlap of variables (ie model A or model B), set min_dim to 0. Even if reusing certain parameters, if the extra dimensions don't describe small'' deviations, it's probably best to set min_dim to 0.Since the RJ proposal is user specified, even if there are parameters that should never be removed, it's up to the user to dictate that. Using min_dim will not affect that aspect of the sampler. If there's abase-model'', the fisher function should produce a fisher matrix for the base model only. The modifications are then normally distributed around the last parameter value. Then the fisher function should take the minimum dimension instead of the maximum, like the other functions.

Currently, no dynamic PT option, as it would be too many free parameters for the sampler to converge to a reasonable temperature distribution in a reasonable amount of time. Best use case, use the PTMCMC_MH_dynamic for the ``base'' dimension space, and use that temperature ladder.

Base of the sampler, generic, with user supplied quantities for most of the samplers properties

Uses the Metropolis-Hastings method, with the option for Fisher/MALA steps if the Fisher routine is supplied.

3 modes to use -

single threaded (numThreads = 1) runs single threaded

multi-threaded ``deterministic'' (numThreads>1 ; pool = false) progresses each chain in parallel for swp_freq steps, then waits for all threads to complete before swapping temperatures in sequenctial order (j, j+1) then (j+1, j+2) etc (sequenctially)

multi-threaded ``stochastic'' (numThreads>2 ; pool = true) progresses each chain in parallel by queueing each temperature and evaluating them in the order they were submitted. Once finished, the threads are queued to swap, where they swapped in the order they are submitted. This means the chains are swapped randomly, and the chains do NOT finish at the same time. The sampler runs until the the 0th chain reaches the step number

Note on limits: In the prior function, if a set of parameters should be disallowed, return -std::numeric_↩ limits<double>::infinity() – (this is in the <limits> file in std)

The parameter array uses the dimensions [0,min_dim] always, and [min_dim, max_dim] in RJPTMCMC fashion

Format for the auto_corr file (compatable with csv, dat, txt extensions): each row is a dimension of the cold chain, with the first row being the lengths used for the auto-corr calculation:

lengths: length1 , length2 ...

dim1: length1 , length2 ...

Format for the chain file (compatable with csv, dat, txt extensions): each row is a step, each column a dimension:

Step1: dim1 , dim2 , ..., max_dim, param_status1, param_status2, ...

Step2: dim1 , dim2 , ..., max_dim, param_status1, param_status2, ...

Statistics_filename : should be txt extension

checkpoint_file : This file saves the final position of all the chains, as well as other metadata, and can be loaded by the function <FUNCTION> to continue the chain from the point it left off. Not meant to be read by humans, the data order is custom to this software library. An empty string ("") means no checkpoint will be saved. For developers, the contents are:

dimension, # of chains

temps of chains

Stepping widths of all chains

Final position of all chains

Parameters

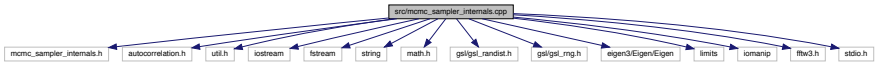
out	*output*	Output chains, shape is double[chain_N, N_steps,dimension]
out	*parameter_status*	Parameter status for each step corresponding to the output chains, shape is double[chain_N, N_steps,dimension]
	max_dimension	maximum dimension of the parameter space being explored – only consideration is memory, as memory scales with dimension. Keep this reasonable, unless memory is REALLY not an issue
	min_dimension	minimum dimension of the parameter space being explored >=1
	N_steps	Number of total steps to be taken, per chain
	chain_N	Number of chains
	initial_pos	Initial position in parameter space - shape double[dimension]
	initial_status	Initial status of the parameters in the initial position in parameter space - shape int[max_dim]
	seeding_var	Variance of the normal distribution used to seed each chain higher than 0 - shape double[max_dimension] – initial seeding of zero corresponds to the dimension turned off initially
	chain_temps	Double array of temperatures for the chains
	swp_freq	the frequency with which chains are swapped
	log_prior	std::function for the log_prior function – takes double *position, int *param_status, int dimension, int chain_id
	log_likelihood	std::function for the log_likelihood function – takes double *position, int *param_status,int dimension, int chain_id
	fisher	std::function for the fisher function – takes double *position, int *param_status,int dimension, double **output_fisher, int chain_id
	RJ_proposal	std::function for the log_likelihood function – takes double *position, int *param_status,int dimension, int chain_id
	numThreads	Number of threads to use (=1 is single threaded)
	pool	boolean to use stochastic chain swapping (MUST have >2 threads)
	show_prog	boolean whether to print out progress (for example, should be set to ``false'' if submitting to a cluster)
	update_RJ_width	boolean whether to print out progress (for example, should be set to ``false'' if submitting to a cluster)
	statistics_filename	Filename to output sampling statistics, if empty string, not output
	chain_filename	Filename to output data (chain 0 only), if empty string, not output – if multiple cold chains, it will append each output to the other, and write out the total
	auto_corr_filename	Filename to output auto correlation in some interval, if empty string, not output
	likelihood_log_filename	Filename to write the log_likelihood and log_prior at each step – use empty string to skip
	checkpoint_file	Filename to output data for checkpoint, if empty string, not saved

8.12 src/mcmc_sampler_internals.cpp File Reference

```
#include "mcmc_sampler_internals.h"
#include "autocorrelation.h"
#include "util.h"
#include <iostream>
#include <fstream>
```

```
#include <string>
#include <math.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_rng.h>
#include <eigen3/Eigen/Eigen>
#include <limits>
#include <iomanip>
#include <fftw3.h>
#include <stdio.h>
```

Include dependency graph for mcmc_sampler_internals.cpp:



Functions

- int mcmc_step (sampler *sampler, double *current_param, double *next_param, int *current_status, int *next_status, int chain_number)

  interface function between the sampler and the internal step functions
- void gaussian_step (sampler *sampler, double *current_param, double *proposed_param, int *current_↩ status, int *proposed_status, int chain_id)

  Straight gaussian step.
- void fisher_step (sampler *sampler, double *current_param, double *proposed_param, int *current_status, int *proposed_status, int chain_index)

  Fisher informed gaussian step.
- void **update_fisher** (sampler *sampler, double *current_param, int *param_status, int chain_index)
- void mmala_step (sampler *sampler, double *current_param, double *proposed_param, int *current_status, int *proposed_status)

  MMALA informed step – Currently not supported.
- void diff_ev_step (sampler *sampler, double *current_param, double *proposed_param, int *current_status, int *proposed_status, int chain_id)

  differential evolution informed step
- void RJ_smooth_history (sampler *sampler, double *current_param, int *current_param_status, int base_↩ history_id, double *eff_history_coord, int *eff_history_status, int chain_id)
- void RJ_step (sampler *sampler, double *current_param, double *proposed_param, int *current_status, int *proposed_status, int chain_number)

  RJ-proposal step for trans-dimensional MCMCs.
- void chain_swap (sampler *sampler, double ***output, int ***param_status, int step_num, int *swp_↩ accepted, int *swp_rejected)

  subroutine to perform chain comparison for parallel tempering
- int single_chain_swap (sampler *sampler, double *chain1, double *chain2, int *chain1_status, int *chain2↩ _status, int T1_index, int T2_index)

  subroutine to actually swap two chains
- void assign_probabilities (sampler *sampler, int chain_index)

  update and initiate probabilities for each variety of step
- void transfer_chain (sampler *samplerptr_dest, sampler *samplerptr_source, int id_dest, int id_source, bool transfer_output)

  Copies contents of one chain to another.
- bool check_sampler_status (sampler *samplerptr)

  Checks the status of a sampler for the stochastic sampling.
- void update_step_widths (sampler *samplerptr, int chain_id)

Updates the step widths, shooting for 20% acceptance ratios for each type of step.

- void **allocate_sampler_mem** (sampler ∗sampler)
- void **deallocate_sampler_mem** (sampler ∗sampler)
- void **update_history** (sampler ∗sampler, double ∗new_params, int ∗new_param_status, int chain_index)
- void **write_stat_file** (sampler ∗sampler, std::string filename)
- void write_checkpoint_file (sampler ∗sampler, std::string filename)

  Routine that writes metadata and final positions of a sampler to a checkpoint file.
- void load_temps_checkpoint_file (std::string check_file, double ∗temps, int chain_N)

  load temperatures from checkpoint file
- void load_checkpoint_file (std::string check_file, sampler ∗sampler)

  load checkpoint file into sampler struct
- void **assign_ct_p** (sampler ∗sampler, int step, int chain_index)
- void **assign_ct_m** (sampler ∗sampler, int step, int chain_index)
- void **assign_initial_pos** (sampler ∗samplerptr, double ∗initial_pos, int ∗initial_status, double ∗seeding_var)
- double PT_dynamical_timescale (int t0, int nu, int t)

  Timescale of the PT dynamics.
- void update_temperatures (sampler ∗samplerptr, int t0, int nu, int t)

  updates the temperatures for a sampler such that all acceptance rates are equal
- void initiate_full_sampler (sampler ∗sampler_new, sampler ∗sampler_old, int chain_N_thermo_ensemble, int chain_N, std::string chain_allocation_scheme)

  For the dynamic PT sampler, this is the function that starts the full sampler with the max number of chains.
- void write_output_file (std::string file, int step_num, int max_dimension, double ∗∗output, int ∗∗status)

  Utility to write out the parameters and status of a sampler to a file.

8.12.1 Detailed Description

File containing definitions for all the internal, generic mcmc subroutines

8.12.2 Function Documentation

8.12.2.1 assign_probabilities()

```
void assign_probabilities (
            sampler * sampler,
            int chain_index )
```

update and initiate probabilities for each variety of step

Type 0: Gaussian step

Type 1: Differential Evolution step

Type 2: MMALA step (currently not supported)

Type 3: Fisher step

8.12.2.2 chain_swap()

```
void chain_swap (
             sampler * sampler,
             double *** output,
             int *** param_status,
             int step_num,
             int * swp_accepted,
             int * swp_rejected )
```

subroutine to perform chain comparison for parallel tempering

The total output file is passed, and the chains are swapped sequentially

This is the routine for ``Deterministic'' sampling (parallel or sequential, but not pooled)

Parameters

sampler	sampler struct
output	output vector containing chains
param_status	Parameter status
step_num	current step number

8.12.2.3 check_sampler_status()

```
bool check_sampler_status (
             sampler * samplerptr )
```

Checks the status of a sampler for the stochastic sampling.

Just loops through the ref_chain_status variables

8.12.2.4 diff_ev_step()

```
void diff_ev_step (
             sampler * sampler,
             double * current_param,
             double * proposed_param,
             int * current_status,
             int * proposed_status,
             int chain_id )
```

differential evolution informed step

Differential evolution uses the past history of the chain to inform the proposed step:

Take the difference of two random, accepted previous steps and step along that with some step size, determined by a gaussian

Parameters

	sampler	Sampler struct
	current_param	current position in parameter space
out	*proposed_param*	Proposed position in parameter space

8.12.2.5 fisher_step()

```
void fisher_step (
            sampler * sampler,
            double * current_param,
            double * proposed_param,
            int * current_status,
            int * proposed_status,
            int chain_index )
```

Fisher informed gaussian step.

Parameters

	sampler	Sampler struct
	current_param	current position in parameter space
out	*proposed_param*	Proposed position in parameter space

8.12.2.6 gaussian_step()

```
void gaussian_step (
            sampler * sampler,
            double * current_param,
            double * proposed_param,
            int * current_status,
            int * proposed_status,
            int chain_id )
```

Straight gaussian step.

Parameters

	sampler	Sampler struct
	current_param	current position in parameter space
out	*proposed_param*	Proposed position in parameter space

8.12.2.7 initiate_full_sampler()

```
void initiate_full_sampler (
            sampler * sampler_new,
            sampler * sampler_old,
            int chain_N_thermo_ensemble,
            int chain_N,
            std::string chain_allocation_scheme )
```

For the dynamic PT sampler, this is the function that starts the full sampler with the max number of chains.

The output file will be reused, but the positions are set back to zero (copying the current position to position zero)

Assumes the output, chain_temps have been allocated in memory for the final number of chains chain_N and steps N_steps

Allocates memory for the new sampler sampler_new -> it's the user's responsibility to deallocate with deallocate↩ _sampler_mem

Parameters

sampler_old	Dynamic sampler
chain_N_thermo_ensemble	Number of chains used in the thermodynamic ensemble
chain_N	Number of chains to use in the static sampler
chain_allocation_scheme	Scheme to use to allocate any remaining chains

8.12.2.8 load_checkpoint_file()

```
void load_checkpoint_file (
            std::string check_file,
            sampler * sampler )
```

load checkpoint file into sampler struct

NOTE – allocate_sampler called in function – MUST deallocate manually

NOTE – sampler->chain_temps allocated internally – MUST free manually

8.12.2.9 load_temps_checkpoint_file()

```
void load_temps_checkpoint_file (
            std::string check_file,
            double * temps,
            int chain_N )
```

load temperatures from checkpoint file

Assumed the temps array is already allocated in memory for the correct number of chains

Just a utility routine to read temperatures from checkpoint file

It would be easy to read in the chain number and allocate memory in the function, but I prefer to leave allocation/deallocation up to the client

8.12.2.10 mmala_step()

```
void mmala_step (
            sampler * sampler,
            double * current_param,
            double * proposed_param,
            int * current_status,
            int * proposed_status )
```

MMALA informed step – Currently not supported.

Parameters

	sampler	Sampler struct
	current_param	current position in parameter space
out	*proposed_param*	Proposed position in parameter space

8.12.2.11 PT_dynamical_timescale()

```
double PT_dynamical_timescale (
            int t0,
            int nu,
            int t )
```

Timescale of the PT dynamics.

kappa in the the language of arXiv:1501.05823v3

Parameters

t0	Timescale of the dyanmics
nu	Initial amplitude (number of steps to base dynamics on)
t	current time

8.12.2.12 RJ_smooth_history()

```
void RJ_smooth_history (
            sampler * sampler,
            double * current_param,
            int * current_param_status,
            int base_history_id,
            double * eff_history_coord,
            int * eff_history_status,
            int chain_id )
```

Parameters

	sampler	Current sampler

Parameters

	current_param	Current parameters to match
	current_param_status	Current parameters to match
	base_history_id	Original history element
out	*eff_history_coord*	Modified history coord
out	*eff_history_status*	Modified History status
	chain_id	Chain ID of the current chain

8.12.2.13 RJ_step()

```
void RJ_step (
            sampler * sampler,
            double * current_param,
            double * proposed_param,
            int * current_status,
            int * proposed_status,
            int chain_number )
```

RJ-proposal step for trans-dimensional MCMCs.

This extra step may seem unnecessary, I'm just adding it in in case the extra flexibility is useful in the future for preprocessing of the chain before sending it to the user's RJ_proposal

Parameters

	sampler	sampler
	current_param	current coordinates in parameter space
out	*proposed_param*	Proposed coordinates in parameter space
	current_status	Current status of parameters
out	*proposed_status*	Proposed status of parameters
	chain_number	chain mumber

8.12.2.14 single_chain_swap()

```
int single_chain_swap (
            sampler * sampler,
            double * chain1,
            double * chain2,
            int * chain1_status,
            int * chain2_status,
            int T1_index,
            int T2_index )
```

subroutine to actually swap two chains

This is the more general subroutine, which just swaps the two chains passed to the function

Parameters

sampler	sampler structure
chain1	parameter position of chain that could be changed
chain2	chain that is not swapped, but provides parameters to be swapped by the other chain
chain1_status	Parameter status array for chain1
chain2_status	Parameter status array for chain2
T1_index	number of chain swappe in chain_temps
T2_index	number of chain swapper in chain_temps

8.12.2.15 transfer_chain()

```
void transfer_chain (
             sampler * samplerptr_dest,
             sampler * samplerptr_source,
             int id_dest,
             int id_source,
             bool transfer_output )
```

Copies contents of one chain to another.

Transfers id_source in samplerptr_source to id_dest samplerptr_dest

NOTE: This copies the VALUE, not the reference. This could be expensive, so use with caution

id_dest is ERASED

samplerptr_dest and samplerptr_source MUST have the same dimension, the same sampling details (like having or not having a fisher) etc

samplerptr_dest must be previously allocated properly

As output is the largest transfer by far, the transfer_output flag can be used to allow the user to handle that manually.

8.12.2.16 update_temperatures()

```
void update_temperatures (
             sampler * samplerptr,
             int t0,
             int nu,
             int t )
```

updates the temperatures for a sampler such that all acceptance rates are equal

Follows the algorithm outlined in arXiv:1501.05823v3

Fixed temperatures for the first and last chain

used in MCMC_MH_dynamic_PT_alloc_internal

For defined results, this should be used while the sampler is using non-pooling methods

8.12.2.17 write_checkpoint_file()

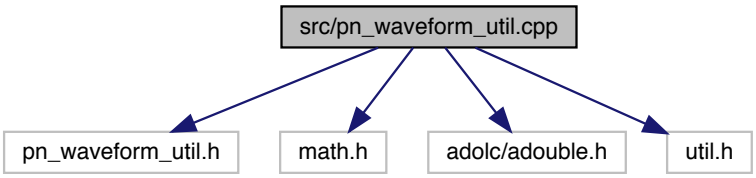
```
void write_checkpoint_file (
            sampler * sampler,
            std::string filename )
```

Routine that writes metadata and final positions of a sampler to a checkpoint file.

8.13 src/pn_waveform_util.cpp File Reference

```
#include "pn_waveform_util.h"
#include <math.h>
#include <adolc/adouble.h>
#include "util.h"
```

Include dependency graph for pn_waveform_util.cpp:



Functions

- template<class T >
  T t_2PN (T f, T eta, T chirpmass, T chi1, T chi2, T tc)

  Time t as a function of f up to 2nd PN order.
- template double **t_2PN**< **double** > (double, double, double, double, double, double)
- template adouble **t_2PN**< **adouble** > (adouble, adouble, adouble, adouble, adouble, adouble)

8.13.1 Detailed Description

PN waveform utilities

8.13.2 Function Documentation

8.13.2.1 t_2PN()

```
template<class T >
T t_2PN (
             T f,
             T eta,
             T chirpmass,
             T chi1,
             T chi2,
             T tc )
```

Time t as a function of f up to 2nd PN order.

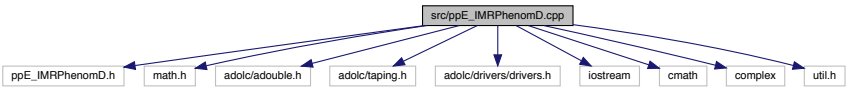
Taken from https://arxiv.org/pdf/gr-qc/0411129.pdf

Non-precessing for now

8.14 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 <iostream>
#include <cmath>
#include <complex>
#include "util.h"
```

Include dependency graph for ppE_IMRPhenomD.cpp:



8.14.1 Detailed Description

File for the implementation of the ppE formalism for testing GR

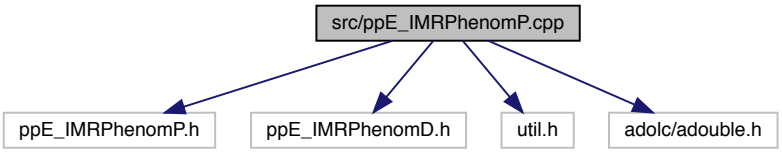
Extends the IMRPhenomD template to include non-GR phase terms

Supported waveforms: ppE Inspiral, ppE IMR, dCS, EdGB

8.15 src/ppE_IMRPhenomP.cpp File Reference

```
#include "ppE_IMRPhenomP.h"
#include "ppE_IMRPhenomD.h"
#include "util.h"
#include <adolc/adouble.h>
```

Include dependency graph for ppE_IMRPhenomP.cpp:



Macros

- #define **ROTATEZ**(angle, vx, vy, vz)
- #define **ROTATEY**(angle, vx, vy, vz)

Variables

- const double **sqrt_6** = 2.44948974278317788

8.15.1 Detailed Description

Source code file for parameterized post Einsteinian Modifications to the precessing waveform model IMRPhenomP

8.15.2 Macro Definition Documentation

8.15.2.1 ROTATEY

```
#define ROTATEY(
                angle,
                vx,
                vy,
                vz )
```

Value:

```
tmp1 = vx*cos(angle) + vz*sin(angle);\
tmp2 = - vx*sin(angle) + vz*cos(angle);\
vx = tmp1;\
vz = tmp2
```

8.15.2.2 ROTATEZ

```
#define ROTATEZ(
              angle,
              vx,
              vy,
              vz )
```

Value:

```
tmp1 = vx*cos(angle) - vy*sin(angle);\
tmp2 = vx*sin(angle) + vy*cos(angle);\
vx = tmp1;\
vy = tmp2
```

8.16 src/util.cpp File Reference

```
#include "util.h"
#include "GWATConfig.h"
#include "D_Z_Config.h"
#include <math.h>
#include <string>
#include <string.h>
#include <complex>
#include <iostream>
#include <stdio.h>
#include <fstream>
#include <adolc/adouble.h>
#include <gsl/gsl_interp.h>
#include <gsl/gsl_spline.h>
#include <gsl/gsl_errno.h>
```

Include dependency graph for util.cpp:

Functions

- void initiate_LumD_Z_interp (gsl_interp_accel **Z_DL_accel_ptr, gsl_spline **Z_DL_spline_ptr)

  Function that uses the GSL libraries to interpolate pre-calculated Z-D_L data.
- void free_LumD_Z_interp (gsl_interp_accel **Z_DL_accel_ptr, gsl_spline **Z_DL_spline_ptr)

  Frees the allocated interpolation function.
- adouble Z_from_DL_interp (adouble DL, gsl_interp_accel *Z_DL_accel_ptr, gsl_spline *Z_DL_spline_ptr)
- double Z_from_DL_interp (double DL, gsl_interp_accel *Z_DL_accel_ptr, gsl_spline *Z_DL_spline_ptr)
- double Z_from_DL (double DL, std::string cosmology)

  Calculates the redshift given the luminosity distance.
- adouble Z_from_DL (adouble DL, std::string cosmology)

  Calculates the redshift given the luminosity distance adouble version for ADOL-C implementation.
- double DL_from_Z (double Z, std::string cosmology)

  Calculates the luminosity distance given the redshift.
- adouble DL_from_Z (adouble Z, std::string cosmology)

Calculates the luminosity distance given the redshift adouble version for ADOL-C implementation.

- double cosmology_interpolation_function (double x, double ∗coeffs, int interp_degree)

  Custom interpolation function used in the cosmology calculations.
- adouble cosmology_interpolation_function (adouble x, double ∗coeffs, int interp_degree)

  Custom interpolation function used in the cosmology calculations adouble version for ADOL-C.
- double cosmology_lookup (std::string cosmology)

  Helper function for mapping cosmology name to an internal index.
- void printProgress (double percentage)

  routine to print the progress of a process to the terminal as a progress bar
- void allocate_FFTW_mem_forward (fftw_outline ∗plan, int length)

  Allocate memory for FFTW3 methods used in a lot of inner products input is a locally defined structure that houses all the pertinent data.
- void allocate_FFTW_mem_reverse (fftw_outline ∗plan, int length)

  Allocate memory for FFTW3 methods used in a lot of inner products –INVERSE input is a locally defined structure that houses all the pertinent data.
- void deallocate_FFTW_mem (fftw_outline ∗plan)

  deallocates the memory used for FFTW routines
- 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)

  Local function to calculate a factorial.
- double pow_int (double base, int power)

  Local power function, specifically for integer powers.
- adouble **pow_int** (adouble base, int power)
- double cbrt_internal (double base)

  Fucntion that just returns the cuberoot.
- adouble cbrt_internal (adouble base)

  Fucntion that just returns the cuberoot ADOL-C doesn't have the cbrt function (which is faster), so have to use the power function.
- double ∗∗ allocate_2D_array (int dim1, int dim2)

  Utility to malloc 2D array.
- int ∗∗ **allocate_2D_array_int** (int dim1, int dim2)
- void deallocate_2D_array (double ∗∗array, int dim1, int dim2)

  Utility to free malloc'd 2D array.
- void **deallocate_2D_array** (int ∗∗array, int dim1, int dim2)
- double ∗∗∗ allocate_3D_array (int dim1, int dim2, int dim3)

  Utility to malloc 3D array.
- int ∗∗∗ **allocate_3D_array_int** (int dim1, int dim2, int dim3)
- void deallocate_3D_array (double ∗∗∗array, int dim1, int dim2, int dim3)

  Utility to free malloc'd 2D array.
- void deallocate_3D_array (int ∗∗∗array, int dim1, int dim2, int dim3)

  Utility to free malloc'd 2D array.

- void read_file (std::string filename, double **output, int rows, int cols)

  Utility to read in data.
- void read_file (std::string filename, double *output)

  Utility to read in data (single dimension vector)
- void read_LOSC_data_file (std::string filename, double *output, double *data_start_time, double *duration, double *fs)

  Read data file from LIGO Open Science Center.
- void read_LOSC_PSD_file (std::string filename, double **output, int rows, int cols)

  Read PSD file from LIGO Open Science Center.
- void allocate_LOSC_data (std::string *data_files, std::string psd_file, int num_detectors, int psd_length, int data_file_length, double trigger_time, std::complex< double > **data, double **psds, double **freqs)

  Prepare data for MCMC directly from LIGO Open Science Center.
- void free_LOSC_data (std::complex< double > **data, double **psds, double **freqs, int num_detectors, int length)
- void tukey_window (double *window, int length, double alpha)

  Tukey window function for FFTs.
- void write_file (std::string filename, double **input, int rows, int cols)

  Utility to write 2D array to file.
- void write_file (std::string filename, double *input, int length)

  Utility to write 1D array to file.
- void celestial_horizon_transform (double RA, double DEC, double gps_time, double LONG, double LAT, double *phi, double *theta)

  Utility to transform from celestial coord RA and DEC to local horizon coord for detector response functions.
- double gps_to_GMST (double gps_time)

  Utility to transform from gps time to GMST `https://aa.usno.navy.mil/faq/docs/GAST.php`.
- double gps_to_JD (double gps_time)

  Utility to transform from gps to JD.
- void transform_cart_sph (double *cartvec, double *sphvec)

  utility to transform a vector from cartesian to spherical (radian)
- void transform_sph_cart (double *sphvec, double *cartvec)

  utility to transform a vector from spherical (radian) to cartesian
- template<class T >
  std::complex< T > **cpolar** (T mag, T phase)
- template<class T >
  std::complex< T > XLALSpinWeightedSphericalHarmonic (T theta, T phi, int s, int l, int m)
- template std::complex< double > **XLALSpinWeightedSphericalHarmonic**< **double** > (double, double, int, int, int)
- template std::complex< adouble > **XLALSpinWeightedSphericalHarmonic**< **adouble** > (adouble, adouble, int, int, int)
- template std::complex< double > **cpolar**< **double** > (double, double)
- template std::complex< adouble > **cpolar**< **adouble** > (adouble, adouble)

8.16.1 Detailed Description

General utilities that are not necessarily specific to any part of the project at large

8.16.2 Function Documentation

8.16.2.1 allocate_2D_array()

```
double** allocate_2D_array (
             int dim1,
             int dim2 )
```

Utility to malloc 2D array.

8.16.2.2 allocate_3D_array()

```
double*** allocate_3D_array (
             int dim1,
             int dim2,
             int dim3 )
```

Utility to malloc 3D array.

8.16.2.3 allocate_LOSC_data()

```
void allocate_LOSC_data (
             std::string * data_files,
             std::string psd_file,
             int num_detectors,
             int psd_length,
             int data_file_length,
             double trigger_time,
             std::complex< double > ** data,
             double ** psds,
             double ** freqs )
```

Prepare data for MCMC directly from LIGO Open Science Center.

Trims data for Tobs (determined by PSD file) 3/4∗Tobs in front of trigger, and 1/4∗Tobs behind

Currently, default to sampling frequency and observation time set by PSD – cannot be customized

Output is in order of PSD columns – string vector of detectos MUST match order of PSD cols

Output shapes– psds = [num_detectors][psd_length] data = [num_detectors][psd_length]

freqs = [num_detectors][psd_length]

Total observation time = 1/(freq[i] - freq[i-1]) (from PSD file)

Sampling frequency fs = max frequency from PSD file

ALLOCATES MEMORY – must be freed to prevent memory leak

Parameters

	data_files	Vector of strings for each detector file from LOSC
	psd_file	String of psd file from LOSC
	num_detectors	Number of detectors to use
	psd_length	Length of the PSD file (number of rows of DATA)
	data_file_length	Length of the data file (number of rows of DATA)
	trigger_time	Time for the signal trigger (GPS)
out	*data*	Output array of data for each detector
out	*psds*	Output array of psds for each detector
out	*freqs*	Output array of freqs for each detector

8.16.2.4 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

8.16.2.5 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

8.16.2.6 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

8.16.2.7 celestial_horizon_transform()

```
void celestial_horizon_transform (
            double RA,
            double DEC,
            double gps_time,
            double LONG,
            double LAT,
            double * phi,
            double * theta )
```

Utility to transform from celestial coord RA and DEC to local horizon coord for detector response functions.

Outputs are the spherical polar angles defined by North as 0 degrees azimuth and the normal to the earth as 0 degree polar

Parameters

	RA	Right acsension (rad)
	DEC	Declination (rad)
	gps_time	GPS time
	LONG	Longitude (rad)
	LAT	Latitude (rad)
out	*phi*	horizon azimuthal angle (rad)
out	*theta*	horizon polar angle (rad)

8.16.2.8 cosmology_interpolation_function()

```
double cosmology_interpolation_function (
            double x,
            double * coeffs,
            int interp_degree )
```

Custom interpolation function used in the cosmology calculations.

Power series in half power increments of x, up to 11/2. powers of x

8.16.2.9 deallocate_2D_array()

```
void deallocate_2D_array (
            double ** array,
            int dim1,
            int dim2 )
```

Utility to free malloc'd 2D array.

8.16.2.10 deallocate_3D_array() [1/2]

```
void deallocate_3D_array (
            double *** array,
            int dim1,
            int dim2,
            int dim3 )
```

Utility to free malloc'd 2D array.

8.16.2.11 deallocate_3D_array() [2/2]

```
void deallocate_3D_array (
            int *** array,
            int dim1,
            int dim2,
            int dim3 )
```

Utility to free malloc'd 2D array.

8.16.2.12 DL_from_Z()

```
double DL_from_Z (
            double Z,
            std::string cosmology )
```

Calculates the luminosity distance given the redshift.

Based on Astropy.cosmology calculations – see python script in the ./data folder of the project – numerically calculated given astropy.cosmology's definitions (http://docs.astropy.org/en/stable/cosmology/) and used scipy.optimize to fit to a power series, stepping in half powers of Z. These coefficients are then output to a header file (D_Z_config.h) which are used here to calculate distance. Custom cosmologies etc can easily be acheived by editing the python script D_Z_config.py, the c++ functions do not need modification. They use whatever data is available in the header file. If the functional form of the fitting function changes, these functions DO need to change.

5 cosmological models are available (this argument must be spelled exactly):

PLANCK15, PLANCK13, WMAP9, WMAP7, WMAP5

8.16.2.13 free_LOSC_data()

```
void free_LOSC_data (
            std::complex< double > ** data,
            double ** psds,
            double ** freqs,
            int num_detectors,
            int length )
```

/brief Free data allocated by prep_LOSC_data function

8.16.2.14 initiate_LumD_Z_interp()

```
void initiate_LumD_Z_interp (
            gsl_interp_accel ** Z_DL_accel_ptr,
            gsl_spline ** Z_DL_spline_ptr )
```

Function that uses the GSL libraries to interpolate pre-calculated Z-D_L data.

Initiates the requried functions – GSL interpolation requires allocating memory before hand

8.16.2.15 pow_int()

```
double pow_int (
            double base,
            int power )
```

Local power function, specifically for integer powers.

Much faster than the std version, because this is only for integer powers

8.16.2.16 printProgress()

```
void printProgress (
            double percentage )
```

routine to print the progress of a process to the terminal as a progress bar

Call everytime you want the progress printed

8.16.2.17 read_file() [1/2]

```
void read_file (
            std::string filename,
            double ** output,
            int rows,
            int cols )
```

Utility to read in data.

Takes filename, and assigns to output[rows][cols]

File must be comma separated doubles

Parameters

	filename	input filename, relative to execution directory
out	*output*	array to store output, dimensions rowsXcols
	rows	first dimension
	cols	second dimension

8.16.2.18 read_file() [2/2]

```
void read_file (
            std::string filename,
            double * output )
```

Utility to read in data (single dimension vector)

Takes filename, and assigns to output[i∗rows + cols]

Output vector must be long enough, no check is done for the length

File must be comma separated doubles

Parameters

	filename	input filename, relative to execution directory
out	*output*	output array, assumed to have the proper length of total items

8.16.2.19 read_LOSC_data_file()

```
void read_LOSC_data_file (
            std::string filename,
            double * output,
            double * data_start_time,
            double * duration,
            double * fs )
```

Read data file from LIGO Open Science Center.

Convenience function for cutting off the first few lines of text

Parameters

	filename	input filename
out	*output*	Output data
out	*data_start_time*	GPS start time of the data in file
out	*duration*	Duration of the signal
out	*fs*	Sampling frequency of the data

8.16.2.20 read_LOSC_PSD_file()

```
void read_LOSC_PSD_file (
            std::string filename,
            double ** output,
            int rows,
            int cols )
```

Read PSD file from LIGO Open Science Center.

Convenience function for cutting off the first few lines of text

8.16.2.21 transform_cart_sph()

```
void transform_cart_sph (
            double * cartvec,
            double * sphvec )
```

utility to transform a vector from cartesian to spherical (radian)

order:

cart: x, y, z

spherical: r, polar, azimuthal

8.16.2.22 transform_sph_cart()

```
void transform_sph_cart (
            double * sphvec,
            double * cartvec )
```

utility to transform a vector from spherical (radian) to cartesian

order:

cart: x, y, z

spherical: r, polar, azimuthal

8.16.2.23 tukey_window()

```
void tukey_window (
            double * window,
            int length,
            double alpha )
```

Tukey window function for FFTs.

As defined by https://en.wikipedia.org/wiki/Window_function

8.16.2.24 write_file() [1/2]

```
void write_file (
            std::string filename,
            double ** input,
            int rows,
            int cols )
```

Utility to write 2D array to file.

Grid of data, comma separated

Grid has rows rows and cols columns

Parameters

filename	Filename of output file, relative to execution directory
input	Input 2D array pointer array[rows][cols]
rows	First dimension of array
cols	second dimension of array

8.16.2.25 write_file() [2/2]

```
void write_file (
            std::string filename,
```

```
            double * input,
            int length )
```

Utility to write 1D array to file.

Single column of data

Parameters

filename	Filename of output file, relative to execution directory
input	input 1D array pointer array[length]
length	length of array

8.16.2.26 XLALSpinWeightedSphericalHarmonic()

```
template<class T >
std::complex<T> XLALSpinWeightedSphericalHarmonic (
            T theta,
            T phi,
            int s,
            int l,
            int m )
```

Shamelessly stolen from LALsuite

Parameters

theta	polar angle (rad)
phi	azimuthal angle (rad)
s	spin weight
l	mode number l
m	mode number m

8.16.2.27 Z_from_DL()

```
double Z_from_DL (
            double DL,
            std::string cosmology )
```

Calculates the redshift given the luminosity distance.

Based on Astropy.cosmology calculations – see python script in the ./data folder of the project – numerically calculated given astropy.cosmology's definitions (http://docs.astropy.org/en/stable/cosmology/) and used scipy.optimize to fit to a power series, stepping in half powers of DL. These coefficients are then output to a header file (D_Z_config.h) which are used here to calculate redshift. Custom cosmologies etc can easily be acheived by editing the python script D_Z_config.py, the c++ functions do not need modification. They use whatever data is available in the header file.

5 cosmological models are available (this argument must be spelled exactly, although case insensitive):

PLANCK15, PLANCK13, WMAP9, WMAP7, WMAP5

8.16.2.28 Z_from_DL_interp() [1/2]

```
adouble Z_from_DL_interp (
            adouble DL,
            gsl_interp_accel * Z_DL_accel_ptr,
            gsl_spline * Z_DL_spline_ptr )
```

Function that returns Z from a given luminosity Distance – only Planck15

adouble version for ADOL-C calculations

8.16.2.29 Z_from_DL_interp() [2/2]

```
double Z_from_DL_interp (
            double DL,
            gsl_interp_accel * Z_DL_accel_ptr,
            gsl_spline * Z_DL_spline_ptr )
```

Function that returns Z from a given luminosity Distance – only Planck15

8.17 src/waveform_generator.cpp File Reference

```
#include <iostream>
#include "waveform_generator.h"
#include "IMRPhenomD.h"
#include "IMRPhenomP.h"
#include "ppE_IMRPhenomD.h"
#include "ppE_IMRPhenomP.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_plus, std::complex< double > *waveform_cross, string generation_method, gen_params *parameters)

  Function to produce the plus/cross polarizations of an quasi-circular binary.
- int fourier_waveform (double *frequencies, int length, double *waveform_plus_real, double *waveform_plus_imag, double *waveform_cross_real, double *waveform_cross_imag, string generation_method, gen_params *parameters)
- 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.

8.17.1 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

8.17.2 Function Documentation

8.17.2.1 fourier_amplitude()

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

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

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

Parameters

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

8.17.2.2 fourier_phase()

```
int fourier_phase (
            double * frequencies,
            int length,
            double * phase,
```

```
            string generation_method,
            gen_params * parameters )
```

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

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

Parameters

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

8.17.2.3 fourier_waveform() [1/4]

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

Function to produce the plus/cross polarizations 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

This puts the responsibility on the user to pass the necessary parameters

NEED TO OUTLINE OPTIONS FOR EACH METHOD IN DEPTH

NEW PHASE OPTIONS for

PHENOMD ONLY:

If phic is assigned, the reference frequency and reference phase are IGNORED.

If Phic is unassigned, a reference phase AND a reference frequency are looked for.If no options are found, both are set to 0.

If tc is assigned, it is used.

If tc is unassigned, the waveform is shifted so the merger happens at 0.

PhenomPv2:

PhiRef and f_ref are required, phic is not an option.

tc, if specified, is used with the use of interpolation. If not, tc is set such that coalescence happens at t=0

Parameters

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

8.17.2.4 fourier_waveform() [2/4]

```
int fourier_waveform (
            double * frequencies,
            int length,
            double * waveform_plus_real,
            double * waveform_plus_imag,
            double * waveform_cross_real,
            double * waveform_cross_imag,
            string generation_method,
            gen_params * parameters )
```

Parameters

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

8.17.2.5 fourier_waveform() [3/4]

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

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

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

Parameters

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

8.17.2.6 fourier_waveform() [4/4]

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

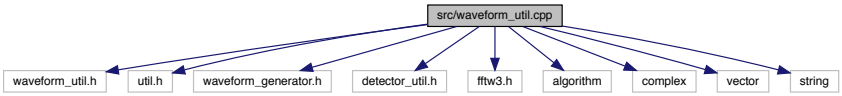
Parameters

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

8.18 src/waveform_util.cpp File Reference

```
#include "waveform_util.h"
#include "util.h"
#include "waveform_generator.h"
#include "detector_util.h"
#include <fftw3.h>
#include <algorithm>
#include <complex>
#include <vector>
#include <string>
```

Include dependency graph for waveform_util.cpp:



Functions

- double data_snr_maximized_extrinsic (double *frequencies, int length, std::complex< double > *data, double *psd, std::string detector, std::string generation_method, gen_params *param)

  Utility to calculate the snr of a fourier transformed data stream while maximizing over the coalescence parameters phic and tc.
- double data_snr_maximized_extrinsic (double *frequencies, int length, double *data_real, double *data_imag, double *psd, std::string detector, std::string generation_method, gen_params *param)

  Light wrapper for the data_snr_maximized_extrinsic method.
- double calculate_snr (std::string detector, std::complex< double > *waveform, double *frequencies, int length)

  Caclulates the snr given a detector and waveform (complex) and frequencies.
- int fourier_detector_response (double *frequencies, int length, std::complex< double > *hplus, std::complex< double > *hcross, std::complex< double > *detector_response, double theta, double phi, std::string detector)
- int fourier_detector_response (double *frequencies, int length, std::complex< double > *hplus, std::complex< double > *hcross, std::complex< double > *detector_response, double theta, double phi, double psi, std::string detector)
- int fourier_detector_response_equatorial (double *frequencies, int length, std::complex< double > *hplus, std::complex< double > *hcross, std::complex< double > *detector_response, double ra, double dec, double psi, double gmst, std::string detector)
- int fourier_detector_response (double *frequencies, int length, std::complex< double > *response, std::string detector, std::string generation_method, gen_params *parameters)

  Function to produce the detector response caused by impinging gravitational waves from a quasi-circular binary.
- int fourier_detector_response_equatorial (double *frequencies, int length, std::complex< double > *response, std::string detector, std::string generation_method, gen_params *parameters)

  Function to produce the detector response caused by impinging gravitational waves from a quasi-circular binary for equatorial coordinates.
- int fourier_detector_amplitude_phase (double *frequencies, int length, double *amplitude, double *phase, std::string detector, std::string generation_method, gen_params *parameters)

  Calculates the amplitude (magnitude) and phase (argument) of the response of a given detector.

8.18.1 Detailed Description

Utilities for waveforms - SNR calculation and detector response

includes snr and detector response

8.18.2 Function Documentation

8.18.2.1 calculate_snr()

```
double calculate_snr (
            std::string detector,
            std::complex< double > * waveform,
            double * frequencies,
            int length )
```

Caclulates the snr given a detector and waveform (complex) and frequencies.

This function computes the un-normalized snr: \sqrt((H | H))

Parameters

detector	detector name - must match the string of populate_noise precisely
waveform	complex waveform
frequencies	double array of frequencies that the waveform is evaluated at
length	length of the above two arrays

8.18.2.2 data_snr_maximized_extrinsic() [1/2]

```
double data_snr_maximized_extrinsic (
            double * frequencies,
            int length,
            std::complex< double > * data,
            double * psd,
            std::string detector,
            std::string generation_method,
            gen_params * param )
```

Utility to calculate the snr of a fourier transformed data stream while maximizing over the coalescence parameters phic and tc.

The gen_params structure holds the parameters for the template to be used (the maximimum likelihood parameters)

Parameters

frequencies	Frequencies used by data
length	length of the data
data	input data in the fourier domain
psd	PSD for the detector that created the data
detector	Name of the detector –See noise_util for options
generation_method	Generation method for the template – See waveform_generation.cpp for options
param	gen_params structure for the template

8.18.2.3 data_snr_maximized_extrinsic() [2/2]

```
double data_snr_maximized_extrinsic (
            double * frequencies,
            int length,
            double * data_real,
            double * data_imag,
            double * psd,
            std::string detector,
            std::string generation_method,
            gen_params * param )
```

Light wrapper for the data_snr_maximized_extrinsic method.

Splits the data into real and imaginary, so all the arguments are C-safe

Parameters

frequencies	Frequencies used by data
length	length of the data
data_real	input data in the fourier domain – real part
data_imag	input data in the fourier domain – imaginary part
psd	PSD for the detector that created the data
detector	Name of the detector –See noise_util for options
generation_method	Generation method for the template – See waveform_generation.cpp for options
param	gen_params structure for the template

8.18.2.4 fourier_detector_amplitude_phase()

```
int fourier_detector_amplitude_phase (
            double * frequencies,
            int length,
            double * amplitude,
            double * phase,
            std::string detector,
            std::string generation_method,
            gen_params * parameters )
```

Calculates the amplitude (magnitude) and phase (argument) of the response of a given detector.

This is for general waveforms, and will work for precessing waveforms

Not as fast as non-precessing, but that can't be helped. MUST include plus/cross polarizations

8.18.2.5 fourier_detector_response() [1/3]

```
int fourier_detector_response (
            double * frequencies,
            int length,
            std::complex< double > * hplus,
            std::complex< double > * hcross,
            std::complex< double > * detector_response,
            double theta,
            double phi,
            std::string detector )
```

Parameters

	frequencies	array of frequencies corresponding to waveform
	length	length of frequency/waveform arrays
	hcross	precomputed cross polarization of the waveform
out	*detector_response*	detector response
	theta	polar angle (rad) theta in detector frame
	phi	azimuthal angle (rad) phi in detector frame
	detector	detector - list of supported detectors in noise_util

8.18.2.6 fourier_detector_response() [2/3]

```
int fourier_detector_response (
            double * frequencies,
            int length,
            std::complex< double > * hplus,
            std::complex< double > * hcross,
            std::complex< double > * detector_response,
            double theta,
            double phi,
            double psi,
            std::string detector )
```

Parameters

	frequencies	array of frequencies corresponding to waveform
	length	length of frequency/waveform arrays
	hcross	precomputed cross polarization of the waveform
out	*detector_response*	detector response
	theta	polar angle (rad) theta in detector frame
	phi	azimuthal angle (rad) phi in detector frame
	psi	polarization angle (rad) phi in detector frame
	detector	detector - list of supported detectors in noise_util

8.18.2.7 fourier_detector_response() [3/3]

```
int fourier_detector_response (
            double * frequencies,
            int length,
            std::complex< double > * response,
            std::string detector,
            std::string generation_method,
            gen_params * parameters )
```

Function to produce the detector response caused by impinging gravitational waves from a 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

This puts the responsibility on the user to pass the necessary parameters

Detector options include classic interferometers like LIGO/VIRGO (coming soon: ET and LISA)

This is a wrapper that combines generation with response functions: if producing mulitple responses for one waveform (ie stacking Hanford, Livingston, and VIRGO), it will be considerably more efficient to calculate the waveform once, then combine each response manually

Parameters

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

8.18.2.8 fourier_detector_response_equatorial() [1/2]

```
int fourier_detector_response_equatorial (
            double * frequencies,
            int length,
            std::complex< double > * hplus,
            std::complex< double > * hcross,
            std::complex< double > * detector_response,
            double ra,
            double dec,
            double psi,
            double gmst,
            std::string detector )
```

Parameters

	frequencies	array of frequencies corresponding to waveform
	length	length of frequency/waveform arrays
	hcross	precomputed cross polarization of the waveform
out	*detector_response*	detector response
	ra	Right Ascension in rad
	dec	Declination in rad
	psi	polarization angle (rad)
	gmst	greenwich mean sidereal time
	detector	detector - list of supported detectors in noise_util

8.18.2.9 fourier_detector_response_equatorial() [2/2]

```
int fourier_detector_response_equatorial (
            double * frequencies,
            int length,
            std::complex< double > * response,
            std::string detector,
            std::string generation_method,
            gen_params * parameters )
```

Function to produce the detector response caused by impinging gravitational waves from a quasi-circular binary for equatorial coordinates.

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

This puts the responsibility on the user to pass the necessary parameters

Detector options include classic interferometers like LIGO/VIRGO (coming soon: ET and LISA)

This is a wrapper that combines generation with response functions: if producing mulitple responses for one waveform (ie stacking Hanford, Livingston, and VIRGO), it will be considerably more efficient to calculate the waveform once, then combine each response manually

Parameters

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

Index