Bonsu - The Interactive Phase Retrieval Suite Documentation

Release 3.7.0

Marcus C. Newton

CONTENTS

CHAPTER

ONE

INSTALLATION

1.1 Installation on Mac OS X

If a native Python environment is used, ensure that all dependencies are installed, ideally from the Python Package Index using pip. They are listed below.

- wxPython
- NumPy
- VTK
- Pillow
- H5Py
- Cython
- hdf5plugin (optional)

A stand alone python distribution such as Anaconda Python will likely include all of the above prerequisites.

It is then possible to install Bonsu from the Python Package Index using the following command:

```
$ pip install bonsu
```

If the optional hdf5plugin module is required, use the following command:

```
$ pip install bonsu[hdf5plugin]
```

To install in the conda environment, run the following in the terminal or an Anaconda Prompt:

```
$ conda install -c conda-forge bonsu
```

1.2 Installation on Windows

Download 64-bit Python $\geq$ 3.7 for Microsoft Windows from the Python website and follow the installation instructions. Ensure that pip is included in the installation. Prerequisite binaries are available for download in the Python Package Index. They are also listed below.

- wxPython
- NumPy
- VTK
- Pillow
- H5Py

- Cython
- hdf5plugin (optional)

Prerequisites are installed using the pip command found in the PythonScript folder. For example:

```
$ pip install wxpython
```

Note: Prebuilt VTK packages $\geq$ 7.0 in PyPI generally requre a graphics card with OpenGL 3 support.

Alternatively, it is possible to install a stand alone python distribution such as Anaconda Python . This will likely include all of the above prerequisites.

It is then possible to install Bonsu from the Python Package Index using the following command:

```
$ pip install bonsu
```

If the optional hdf5plugin module is required, use the following command:

```
$ pip install bonsu[hdf5plugin]
```

1.3 Installation on Linux/Unix

1.3.1 System Preparation

Before Bonsu can be installed from source, a number of software packages are required. They are namely:

- Python
- wxPython
- FFTW with threading
- NumPy (FFTW aware)
- VTK (compiled with python bindings)
- Python Imaging Library (Pillow)
- H5Py
- Cython
- hdf5plugin (optional)
- A reasonably new c^{++}compiler such as gcc

As most Linux systems and services require Python natively, it is likely that Python is already provided in your distribution. Also, many distributions package all of the above prerequisites. Please check your distributions repositories.

It is also possible to install a stand alone python distribution such as that provided by Anaconda. This will include most of the above prerequisites. In order to compile and use any additional packages (such as FFTW) into the stand alone distribution, a number of environmental variables should be set. In Anaconda Python, ensure that you create and activate your local environment:

```
$ conda create --name myenv
$ conda activate myenv
$ export PREFIX="/path/to/conda/myenv/"
```

1.3.2 Note on Building FFTW

To build FFTW into a stand alone python distribution such as that provided by Enthought , use the following sequence of commands:

```
$ ./configure --prefix=${PREFIX} --enable-threads --enable-shared
$ make
$ make install
```

1.3.3 Installing Bonsu

Source code for Bonsu is available for download here.

See also:

https://github.com/bonsudev/bonsu

Unpack the source code with the following:

```
$ tar -xvf Bonsu-{x}.tar.gz
```

where x is the current version.

Enter the newly created directory and first build the source:

```
$ python setup.py build
```

If this is successful you will now have a fully prepared built source tree for installation. To install the package type the following (super user privileges may be required):

```
$ python setup.py install
```

To install the package to a non-default location (perhaps where super user privileges are not required) use the following command:

```
$ python setup.py install --prefix=/my/custom/install/path
```

The `--prefix` option defines the installation base directory. To allow Python to find the distributions installed with this scheme, it may be necessary to modify Python's search path(s) to include the above.

To start the program, execute the following from the command line:

```
$ bonsu &
```

Note: For Linux users, a link to the program should be visible in the applications menu.

1.3.4 HDF5 Plugins

If HDF5 plugins are installed to a non-default location, h5py might require the following environmental variable:

```
$ export HDF5_PLUGIN_PATH=/path/to/plugins.so/folder
```

1.3.5 Remote Access

If Bonsu is used remotely via X11 fowarding or X2Go (NX technology), errors with rendering might occur dependent on the available graphics hardware. A possible workaround is to add the following environment variables to your .bashrc script:

```
$ export LIBGL_ALWAYS_SOFTWARE=true
$ export MESA_GL_VERSION_OVERRIDE=3.2
$ export MESA_GLSL_VERSION_OVERRIDE=150
```

CHAPTER

TWO

INTRODUCTION

2.1 Overview

Bonsu, the interactive phase retrieval suite, is the first phase retrieval software package for phase retrieval with real-time visualisation in both two and three dimensions. It is complete with an inventory of algorithms and routines for data manipulation and reconstruction.

Bonsu is open-source, is designed around the python language (with c bindings) and is largely platform independent. It is also able to handle data in formats such as:

1. SPE
2. HDF5
3. VTK
4. Numpy
5. Image formats including PNG, JPG, TIFF, PPM

2.2 Arrays

Bonsu uses NumPy array format (http://numpy.org) internally for manipulating and outputting raw data. Double-precision complex NumPy arrays are used for phase retrieval. NumPy is part of the 'Scientific Computing Tools For Python' or SciPy package (http://scipy.org) available in Python. There are a number of standard tools made available for array conversion, manipulation and viewing. Tools for cropping, padding, binning, support array creation and mask array creation are provided. Arrays can be exported in both NumPy array and VTK format.

For further details see *Array Creation and Storage*.

2.3 Algorithms

Bonsu comes complete with a number of *algorithms* for phase retrieval. They include Fienup's hybrid input-output (HIO) (Fienup, 1982), HIO with positivity constraint, phase-constrained HIO (Harder et al., 2010), hybrid projection reflection (Bauschke et al., 2003), relaxed average alternating reflection (Luke, 2005) and the error reduction algorithm (Gerchberg & Saxton, 1972). In addition, a shrink-wrap algorithm that is able to make use of either of the above algorithms is also provided (Marchesini et al., 2003; Newton et al., 2010). Phase retrieval algorithms are implemented in the c language for fast execution and seamlessly integrated into the application.

For a complete list see *Library Reference – Phasing Algorithms*.

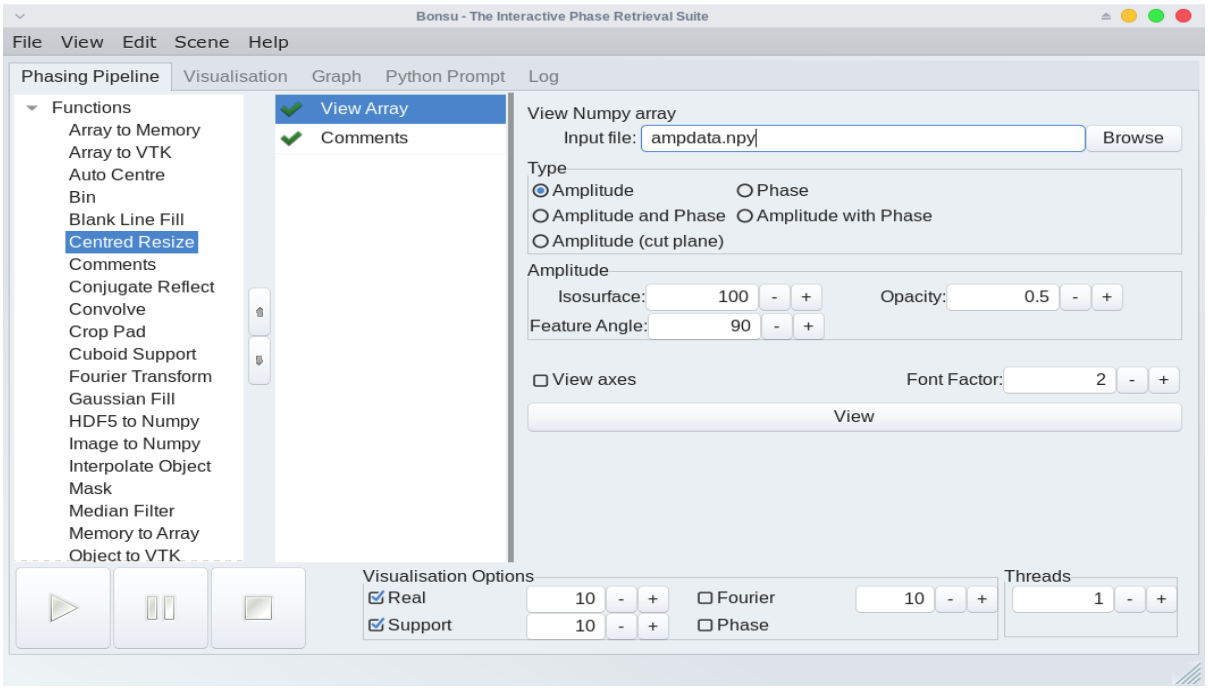
CHAPTER

THREE

GENERAL USAGE

The following describes the general usage procedure when interacting with the graphical user interface. If you would like to script with Bonsu, there are at least two options available. The first option is to use the *Python Script function* which permits arbitrary operations on arrays. This is useful when combining custom operations on arrays with native functions. The second option is to import Bonsu into a Python script. The procedure for doing so is described in detail *here*.

3.1 User Interface



The graphical user interface is divided into five tabs which are namely `Phasing Pipeline`, `Visualisation`, `Graph`, `Python Prompt` and `Log`. Algorithms and tools are manipulated and executed on the `Phasing Pipeline` tab. During algorithmic phase retrieval, real-time images of the object during reconstruction are shown on the `Visualisation` tab. Information regarding the statistical error during phase retrieval is displayed on the `Graph` tab. The `Log` tab provides the user with any information the various algorithms and tools return during execution. The `Python Prompt` is, as it suggests, a fully functioning interactive python shell.

When loading and saving files, the current working directory where the program was loaded is used as the default path. It is possible to change the working directory by selecting *Edit* → *Current Working Directory* from the main menu.

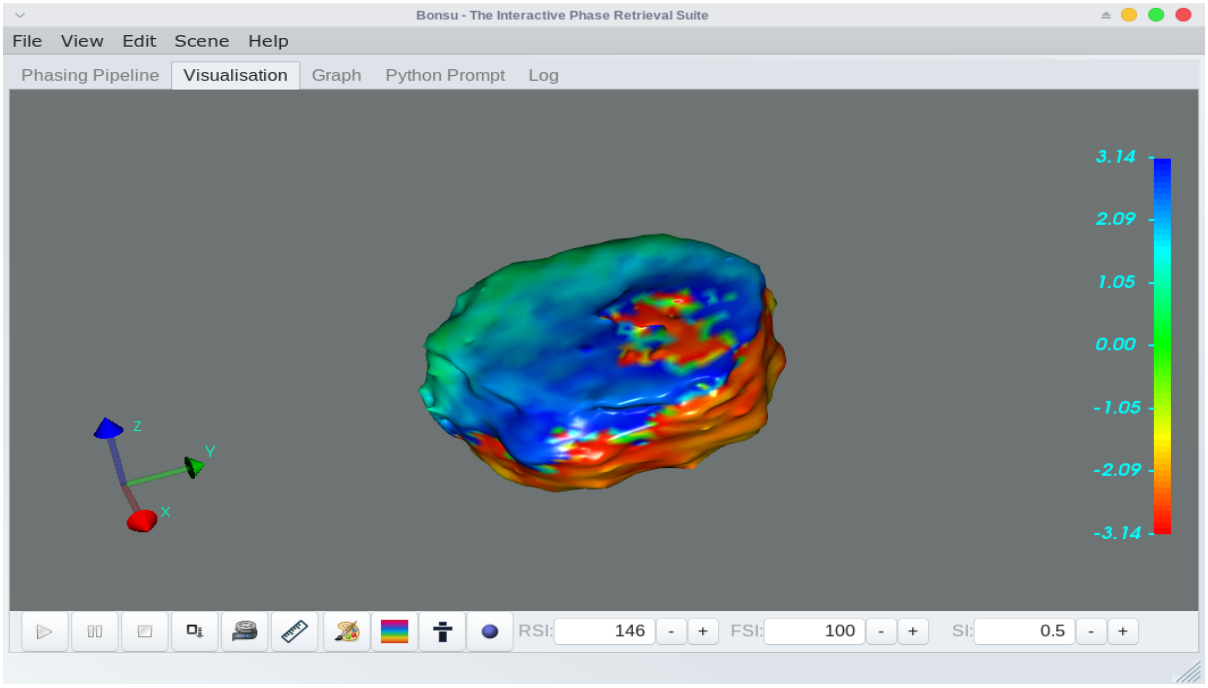
The remainder of this section describes in further detail the use of each graphical interface tab.

3.1.1 Phasing Pipeline

Most user interaction occurs in the `Phasing Pipeline` tab. This tab is divided into three columns. From left to right, the first shows a drop-down menu divided into three sections where various array functions, phasing algorithms and posterior phasing operations can be found. For a complete list of available operations and algorithms, please see the *Library Reference*. Once selected from the drop-down menu (with a carriage return or mouse double click), an instance of the selected item appears in the centre column (or `pipeline`), where it can be selected in the pipeline and properties modified for that instance. The properties appear in the right-hand column. Check boxes to the immediate left of each item in the pipeline provide the option to exempt that item from execution without removing the item from the pipeline. By simply selecting the check so that a cross is displayed, the item will be ignored during execution of the pipeline. This is often useful for testing purposes. Once items have been added to the pipeline, it is possible to save the contents of the pipeline and all custom settings by simply choosing `Save` from the menu. At this point, the user is prompted for a file name. See *Saving Sessions* for further details.

Below the three columns are the control buttons `Start`, `Pause` and `Stop`. These buttons will, respectively, execute all items in the pipeline sequentially in the order that they appear, pause the execution of an algorithm or stop the execution of an algorithm. The remaining checkboxes control the type of visualization that appears on the 'Visualisation' tab and the maximum number of threads used by the Fourier transform algorithm. The options include no visualization, real-space amplitude, real-space amplitude with phase, Fourier-space amplitude and Fourier-space amplitude with phase.

3.1.2 Visualisation



In order to view real-time visualisation, either `real-space` or `Fourier-space` must be selected from the `visualisation options` on the `Phasing Pipeline` tab. This is the default. The `Visualisation` tab is where a rendered scene of the reconstruction will appear. From here the user can interact fully with and manipulate an object generated with either a view function (e.g. *View Array* or *View Object*) or an algorithm during reconstructed (see *Library Reference – Phasing Algorithms*). Either a two-dimensional or three-dimensional object is displayed depending on the dimensions of the data. In both cases, pressing the "p" keyboard button will display the current co-ordinates of the mouse cursor and print these coordinates to the *Log*.

It is also possible to undock the rendered scene by selecting *View* → *Visualisation* → *Undock* from the main menu. This will open the current visualisation in a new window. Selecting *View* → *Visualisation* → *Dock* returns the rendered scene back to the `Visualisation` tab.

Below the rendered scene, there are a number of buttons for manipulating the visualisation. The control buttons `Start`, `Pause` and `Stop` are identical in operation to those on the `Phasing Pipeline` tab and are present for convenience. To save images of the object for further use, select the `save` button. After this, no prompt for a file name will appear. Instead, the file will be saved to the working directory with the present date and time used as a filename string. The `Measure scene` button can be used to perform measurements on the rendered object and to view the orientation widget. The `Scene background` button will open a window that can be used to modify the background colour of the rendered scene. The `Lookup table` button will open a window which can be used to change the colour map for the amplitude and phase of the rendered object. The `Data range` button will open a window that will allow the user to adjust the data range used for the colour map. The `Isosurface` button will display three number fields for the `Real space isosurface` (RSI), `Fourier space isosurface` (FSI) and `Support isosurface` (SI).

3.1.3 Visualisation: Animation

A feature for animating the rendered scene is made available by selecting the `Animate` button. From here a window appears with various options for scene animation. The axis about which the rotation occurs can be selected by choosing a value greater than zero. Either a single axis or a composite of the three rectilinear (x,y,z) axes can be selected. If multiple axis are chosen, their vector sum is used as the final axis of rotation. The rendered scene is generally rotated about the centre of mass.

Options for setting the angle of increment and total number of rotation steps are also available. It is also possible to save an image for each step by selecting `save scene` and providing a file name. Indices of the animation step will be appended to the filename automatically.

3.1.4 Visualisation: Measurement

Features for performing measurements on the rendered scene are made available by selecting the `Measure scene` button on the `Visualisation` tab. A window appears which shows a number of tabs for measurement. When enabled, the `Line Scan` tab provides a feature for arbitrary direction line scans measurements in both two and three dimensions. A line will appear in the render window with handles at either end for manipulation. As the line is stretched and moved, the corresponding line scan through the data appears on the window tab. Information on the line properties are also displayed on the *Log* tab. To save the data, select `Save Data` from the window tab. No prompt for a file name will appear. Instead, the file will be saved in csv format to the working directory with the present date and time used as a filename string.

When enabled, the `Angle` tab provides a feature for measuring arbitrary angle in both two and three dimensions. In order to begin using this feature, the user must first select three points using the mouse cursor in the render scene. These points will become the vertices of the angular measurement.

3.1.5 Graph

During phase reconstruction, the `Graph` tab displays the reconstruction residual error plotted on a graph with dynamic axes. From here the graph can be paused (independently of the reconstruction process) and the data saved to a text file. No prompt for a file name will appear. Instead, the file will be saved in csv format to the working directory with the present date and time used as a filename string. It is also possible to save the reconstruction residual error information automatically after the phase retrieval process, by appending *Save Residual* to the phasing pipeline.

3.1.6 Python Prompt

The 'Python Prompt' tab enables the user to perform additional tasks through the Python interpreter. This might be the execution of additional Python scripts or additional array operations with standard Numpy tools. Data arrays stored in memory using the `memoryn` keywords (see *Array Creation and Storage* below) are accessible as variables of the same name and can be manipulated as NumPy arrays. Changes made to memory variable arrays will affect subsequent use of that array in the `phasing pipeline`.

3.1.7 Log

The 'Log' tab allows the user to view additional information on the program's operation and any graph data values in text form. From here the log data can be saved to a text file. No prompt for a file name will appear. Instead, the file will be saved in txt format to the working directory with the present date and time used as a filename string.

3.2 Array Creation and Storage

NumPy array format is used internally for data manipulation. NumPy arrays loaded from file are converted to double-precision complex format in row-major order (c style). NumPy arrays are also saved in this format by default.

When a large number of array operations are performed, it is often desirable to store the result in memory at each intermediate step. This can help to reduce disk usage and improve overall performance at the expense of additional memory usage. Bonsu provides a way to store and access arrays in memory, that is accessible to all operations and algorithms. By using either of the keywords `memory0`, `memory1`, ... , `memoryn` (where `n` is a positive integer) as a string in place of a NumPy file path, it is possible to save to memory an array manipulated by any phasing algorithm or operation that uses NumPy arrays. The array is then accessible by using the corresponding keyword in place of the NumPy file path when attempting to load an array. It is also possible to directly visualise the array using *View Array*.

In addition to the above, keywords `memorysequence` and `memorycoords` can be used to view or use *sequence data* and *Co-ordinate Transformation* arrays respectively. This is useful for using *View Object* exclusively with arrays in memory.

3.3 Saving a Session

It is possible to save the contents of the pipeline and all customised settings by simply choosing `Save` from the program menu. Pipeline data is saved in '.fin' file format. Similarly, to recover a pipeline, simply select `open` from the program menu and locate the saved '.fin' file. Bonsu also allows pipeline entries to be appended. For example, opening a '.fin' file twice will append the list of tasks twice into the pipeline. To clear the pipeline, simply select `New` from the menu.

CHAPTER

FOUR

HISTORY AND LICENCE

4.1 History of the Interactive Phase Retrieval Suite

The Interactive Phase Retrieval Suite was first conceived in 2011 by Marcus Newton with the major design goal of providing a complete set of tools for phase reconstruction and visualisation. It was intended as the successor to previous *command line only* tools which were widely in use at the time. It aimed to be encompassing and placed emphasis on maintaining reproducibility on various platforms. It achieved this largely by providing the means to easily share encapsulated data analysis parameters.

The name *Bonsu* means `whale` or large marine mammals of the Cetacea group, in the Akan language of western African. This was chosen as a sentiment to the "eye-like" focus of the sound beam that certain cetacean mammals emit during sonar echo location.

4.2 Licence

All releases of the Interactive Phase Retrieval Suite are Open Source (see http://www.opensource.org/ for the Open Source Definition). Bonsu, the Interactive Phase Retrieval Suite, is distributed under the GNU General Public License. A copy of the GNU General Public Licence is provided with the program. Also see http://www.gnu.org/licenses/.

CHAPTER

FIVE

LIBRARY REFERENCE

5.1 Library Reference - Visualisation Tools

5.1.1 Comments

Input comments into the pipeline.

5.1.2 Nexus File Viewer for I16

Diamond Light Source Nexus File Viewer for Beamline I16

5.1.3 Laxarus Viewer

Laxarus XRD Imaging HDF5 Viewer.

5.1.4 View Array

Render an image of the two or three-dimensional input Numpy array to the *Visualisation tab*.

Parameters:

Input file
User provided path string to the input NumPy file.

Type

- **Amplitude** - An iso-surface of the array amplitude.
- **Phase** - An interactive scalar cut plane of the array phase.
- **Amplitude and Phase** - An iso-surface of the array amplitude and an interactive scalar cut plane of the array phase.
- **Amplitude with Phase** - An iso-surface of the array amplitude with the array phase mapped onto the iso-surface.
- **Amplitude (cut plane)** - An interactive scalar cut plane of the array amplitude.

Amplitude

- **Iso-surface** - Scalar value for the rendered iso-surface.
- **Opacity** - Opacity of the isosurface when viewing `Amplitude and Phase`.
- **Feature angle** - Minimum angle between two surface normals of the rendered object.

Phase

- **Max** - Maximum phase.
- **Min** - Minimum phase.

Origin
Origin point of the interactive scalar cut plane.

Normal
Normal vector to the interactive scalar cut plane.

Spacing
Spacing of the input array indices.

View Axes
Optionally view the axes of the array.

5.1.5 View Object

Render an object of the three-dimensional input Numpy array with it's corresponding co-ordinates array to the *Visualisation tab*.

Parameters:

Input file
User provided path string to the input NumPy file.

Co-ord's File
User provided path string to the corresponding co-ordinates NumPy array file. Alternatively, the *memory array* keyword `memorycoords` provides access to the internally stored coordinate array. Generating a coordinate array is achieved using the *Coordinate transformation* operation.

Type

- **Amplitude** - An iso-surface of the object amplitude.
- **Phase** - An interactive scalar cut plane of the object phase.
- **Amplitude and Phase** - An iso-surface of the object amplitude and an interactive scalar cut plane of the object phase.
- **Amplitude with Phase** - An iso-surface of the object amplitude with the object phase mapped onto the iso-surface.
- **Amplitude (cut plane)** - An interactive scalar cut plane of the object amplitude.

Amplitude

- **Iso-surface** - Scalar value for the rendered iso-surface.
- **Opacity** - Opacity of the isosurface when viewing `Amplitude and Phase`.
- **Feature angle** - Minimum angle between two surface normals of the rendered object.

Phase

- **Max** - Maximum phase.
- **Min** - Minimum phase.

Origin
Origin point of the interactive scalar cut plane.

Normal
Normal vector to the interactive scalar cut plane.

View Axes
Optionally view the axes of the object.

5.1.6 View Support

Render an image of a three-dimensional input Numpy data array and its Support array to the *Visualisation tab*.

Parameters:

Support file
User provided path string to the NumPy array file.

Data array file
User provided path string to the NumPy array file.

Isosurface options

- **Support iso-surface** - Scalar value for the rendered iso-surface on the interval [0,1].
- **Opacity** - Opacity of the support isosurface.
- **Data file iso-surface** - Scalar value for the rendered iso-surface.
- **Feature angle** - Minimum angle between two surface normals of the rendered object.

View Axes
Optionally view the axes of the array.

5.1.7 View VTK Array

Render an input VTK array to the *Visualisation tab*.

Parameters:

Input file
User provided path string to the input NumPy file.

Type

- **Amplitude** - An iso-surface of the array amplitude.
- **Phase** - An interactive scalar cut plane of the array phase.
- **Amplitude (cut plane)** - An interactive scalar cut plane of the array amplitude.

Amplitude

- **Iso-surface** - Scalar value for the rendered iso-surface.
- **Feature angle** - Minimum angle between two surface normals of the rendered object.

Phase

- **Max** - Maximum phase.
- **Min** - Minimum phase.

Origin
Origin point of the interactive scalar cut plane.

Normal
Normal vector to the interactive scalar cut plane.

View Axes
Optionally view the axes of the array.

5.2 Library Reference - Import Tools

5.2.1 Array to Memory

Provides a means to load a NumPy array from disk directly into a memory by using either of the 10 keywords `memory0`, `memory1`, `...` , `memory9` as a string in place of a NumPy file path. See *Array Creation and Storage* for furthr details.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided choice of memory keyword.

5.2.2 Empty Array

Creates an empty new array. If any array is supplied in the `(xyz) from array` field, the dimensions of this array will be used for the empty array. Otherwise, the specified dimensions are used.

Parameters:

Output file
User provided path string for the output NumPy file.

(xyz) from array
Optional user provided path string for NumPy file used to provide the `support` array dimensions.

(xyz) from dimensions {x,y,z}
Dimensions of the `support` array. This is overridden if the `(xyz) from array` field is used.

5.2.3 HDF5 to Numpy

Provides a means to browse and import data within a Hierarchical Data Format (HDF) file. HDF5 is supported via the python H5Py package.

Parameters:

Input HDF file
User provided path string to the input HDF file.

HDF key path
Comma separated list of HDF keys found within the `input HDF file`.

Output File
User provided path string for the output NumPy file.

5.2.4 Image to Numpy

Provides a means to import image data into NumPy format. Accepted formats include Portable Network Graphics (PNG), JPEG, Portable PixMap (PPM) and Tagged Image File (TIF) format. Output Numpy arrays are three dimensional with unit length in the third dimension.

Parameters:

Input Image file
User provided path string to the input image file.

Output File
User provided path string for the output NumPy file.

5.2.5 Load Co-ordinates

Initialises or replaces the internally stored `Co-ordinates data` with the supplied array.

Parameters:

Input file
User provided path string to the input NumPy file.

5.2.6 Load PSF

Initialises or replaces the internally stored `Point Spread Function data` with the supplied array. This is typically used before a *partial coherence* phase retrieval algorithm commences.

Parameters:

Input file
User provided path string to the input NumPy file.

5.2.7 SPE to Numpy

Provides a means to import data from SPE format into NumPy format. During conversion, additional information is provided and appears in the *Log*.

Parameters:

Input SPE file
User provided path string to the input SPE file.

Output File
User provided path string for the output NumPy file.

5.3 Library Reference - Export Tools

5.3.1 Array to VTK

Convert an input NumPy array into VTK file format. Uses either the amplitude or phase of the array.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output VTK file.

Type
Use either the amplitude or the phase of the complex `input file` in order to construct a VTK array.

5.3.2 Memory to Array

Provides a means to save to disk a NumPy array stored in memory. See *Array to Memory* for furthr details.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided choice of memory keyword.

5.3.3 Object to VTK

Convert an input NumPy array and it's corresponding co-ordinates array into VTK file format. Uses either the amplitude or phase of the array.

Parameters:

Input file
User provided path string to the input NumPy file.

Co-ord's File
User provided path string to the corresponding co-ordinates NumPy file. Generating coordinates files is achieved using the *Coordinate transformation* operation.

Output File
User provided path string for the output VTK file.

Type
Use either the amplitude or the phase of the complex `input file` in order to construct a VTK array.

5.4 Library Reference - Functions

5.4.1 Auto Centre

Centres the input NumPy array by placing the voxel with the largest amplitude at the centre of the array. The output NumPy array will necessarily increase in dimensions when compared to the input if centring is required.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

5.4.2 Bin

Creates an array in which each voxel results from the grouping and addition of neighbouring voxels of the `input file`. The dimensions used (in each {x,y,z} direction) for each voxel group is dictated by the `Bin dimensions` parameter.

Parameters:

Input file
User provided path string to the input NumPy file from which `bin` groups are defined.

Output File
User provided path string for the output `bin` NumPy file.

Bin dimensions {x,y,z}
Dimensions used for grouping voxels.

5.4.3 Scale Array Dims

Creates an array in which groupings of neighbouring voxels result from a single voxel of the `input file`. The dimensions used (in each {x,y,z} direction) for each voxel group is dictated by the `Scale dimensions` parameter.

Parameters:

Input file
User provided path string to the input NumPy file from which `Scale` groups are defined.

Output File
User provided path string for the output `Scaled` NumPy file.

Scale dimensions {x,y,z}
Dimensions used for grouping voxels.

5.4.4 Blank Line Fill Array

Applies an average filter to the amplitude of non-zero voxels of the input array within the specified region.

Parameters:

Input file
User provided path string to the input NumPy file.

ROI path
Region of interest within the `input HDF file`.

Output File
User provided path string for the output NumPy file.

Filter kernel dimensions
Dimensions of the averging kernel.

5.4.5 Centred Resize

Centres the input NumPy array by placing the voxel with the largest amplitude at the centre of the array and then resizes the cetered array to the specified dimensions.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

Dimensions {x,y,z}
Dimensions of the output array.

5.4.6 Conjugate Reflect

Takes the complex conjugate and reflects the coordinates of the input NumPy file.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

5.4.7 Convolve

Performs a discrete convolution of the two input NumPy files.

Parameters:

First Input file
User provided path string to the first input NumPy file.

Second Input file
User provided path string to the second input NumPy file.

Output File
User provided path string for the output NumPy file.

5.4.8 Crop Pad

Provides a means to crop and zero pad a NumPy array. The input array is first cropped and subsequently zero padded according to the specified parameters.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

Crop dimensions: Start: {i,j,k}
Number of indices by which to crop the array along each {i,j,k} direction. Cropping direction begins from the start of the array (and counts forward) for each {i,j,k} direction.

Crop dimensions: End: {i,j,k}
Number of indices by which to crop the array along each {i,j,k} direction. Cropping direction begins from the end of the array (and counts backward) for each {i,j,k} direction.

Pad dimensions: Start: {i,j,k}
Number of indices by which to pad the array along each {i,j,k} direction. Padding direction begins from the start of the array for each {i,j,k} direction.

Pad dimensions: End: {i,j,k}
Number of indices by which to pad the array along each {i,j,k} direction. Padding direction begins from the end of the array for each {i,j,k} direction.

5.4.9 Cuboid Support

Creates a *support* array for use in phase retrieval. Voxels in this array take a value of $1 + 0j$ inside the *support* and zero everywhere else. The support is automatically centred within the array.

If any array is supplied in the `(xyz) from array` field, the dimensions of this array will be used for the *support* array. Otherwise, the specified dimensions will be used.

Parameters:

Support file
User provided path string for the output `support` NumPy file.

(xyz) from array
Optional user provided path string for NumPy file used to provide the `support` array dimensions.

(xyz) from dimensions {x,y,z}
Dimensions of the `support` array. This is overridden if the `(xyz) from array` field is used.

Support size {x,y,z}
Dimensions of the support. This must be smaller than the size of the array.

5.4.10 Polyhedron Support

Creates a *support* array for use in phase retrieval using the specified planes to form the surface. The vector formed by each terminal and initial point coordinate pair is the normal vector to a surface that encloses the support. Voxels in this array take a value of $1 + 0j$ inside the *support* and zero everywhere else.

If any array is supplied in the `(xyz) from array` field, the dimensions of this array will be used for the *support* array. Otherwise, the specified dimensions will be used.

Parameters:

Support file
User provided path string for the output `support` NumPy file.

(xyz) from array
Optional user provided path string for NumPy file used to provide the `support` array dimensions.

(xyz) from dimensions {x,y,z}
Dimensions of the `support` array. This is overridden if the `(xyz) from array` field is used.

Initial Point Coordinates
List of initial point coordinates of vector normals to each surface. One per line with comma separated ordinates. Parentheses are ignored.

Terminal Point Coordinates
List of terminal point coordinates of vector normals to each surface. One per line with comma separated ordinates. Parentheses are ignored.

5.4.11 Fourier Transform

Performs a Fast Fourier transform (FFT) on the input NumPy file in the specified direction.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

To

- **Fourier Space** - Transform to Fourier space.
- **Real Space** - Transform to Real space.

5.4.12 Gaussian Fill

Fill the input NumPy file with a Gaussian function.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

Sigma
Standard Deviation of the Gaussian function.

5.4.13 Interpolate Object

Interpolates an input NumPy array and it's corresponding co-ordinates onto regular grid NumPy array with dimensions dictated by `Array grid size`. Shepards famous algorithm is used for the interpolation.

Parameters:

Input file
User provided path string to the input NumPy file.

Co-ord's File
User provided path string to the corresponding co-ordinates NumPy file. Generating coordinates files is achieved using the *Coordinate transformation* operation.

Output File
User provided path string for the output NumPy file.

Array grid size
Dimensions of the output NumPy array.

Array Bounds: Start: {x,y,z}
Start bounding coordinates. Each value must be smaller than the `End` bounding coordinates. If all values are zero, they are ignored and the digault bounds are used instead.

Array Bounds: End: {x,y,z}
End bounding coordinates. Each value must be larger than the `Start` bounding coordinates. If all values are zero, they are ignored and the digault bounds are used instead.

Interpolation range
Fraction of the input array used for interpolating each point. Larger values approaching unity will significantly slow down the interpolation process.

5.4.14 Affine Transform

Performs an affine (linear) transformation on the input coordinates file.

Parameters:

Input co-ord's file
User provided path string to the corresponding co-ordinates NumPy file. Generating coordinates files is achieved using the *Coordinate transformation* operation.

Output co-ord's file
User provided path string for the output co-ordinates NumPy file.

Translate: {x,y,z}
Translation of coordinates in each orthogonal direction. Units are equal to those of the original coordinates.

Scale: {x,y,z}
Scaling of each orthogonal axis.

Rotate: {x,y,z}
Rotation about each orthogonal axis. Units are in degrees.

5.4.15 Mask

Creates a `mask` array for use in phase retrieval. Voxels in this array take a value of $1 + 0j$ if the corresponding voxel of the `input file` is inside the range of `Minimum Value` to `Maximum Value`. Voxels outside the same range are set to zero.

Parameters:

Input file
User provided path string to the input NumPy file from which `Minimum Value` and `Maximum Value` are obtained.

Output File
User provided path string for the output `mask` NumPy file.

Maximum Value
Values above (equal or below) `Maximum Value` in the `input file` will result in a corresponding voxel value of zero ($1 + 0j$) in the `mask`.

Minimum Value
Values below (equal or above) `Minimum Value` in the `input file` will result in a corresponding voxel value of zero ($1 + 0j$) in the `mask`.

5.4.16 Median Filter

Applies a median filter to the amplitude of the input array. Voxels are only replaced in the array if they deviate from the median filtered equivalent by an amount greater than the `Normal deviation`.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

Filter kernel dimensions
Dimensions of the median filtering kernel.

Normal deviation
Median filter voxel replace cut-off value. This value is compared element-wise to the normalised difference in amplitude of voxels from the input array and from the median filtered equivalent, i.e. the magnitude of (`median filtered array` - `array`)/ `array`. Voxels that exceeed this cut-off are replaced with their median-filtered equivalent.

5.4.17 Python Script

Execute python script. *Memory arrays* are exposed to this function.

5.4.18 Rotate Support

Rotates a binary array (such as a support array) by the specified angle in degrees normal to the specified axis. This operation is only intended for binary arrays.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

Axis
Axis normal to the rotation. 1,2 and 3 correspond to axis x,y and z.

Angle
Angle of rotation in degrees.

5.4.19 Scale Array

Scales the input array by a given amount. Scaling is performed element-wise.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

Scale factor
Factor by which each voxel of the array is scaled.

5.4.20 Sum or Subtract Array

Takes the sum or difference of two input arrays and places the result in the output array.

Parameters:

Input file 1
User provided path string to the input NumPy file.

Input file 2
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

Add or Subtract
Choose whether to add or subtract array 2 from array 1.

5.4.21 Threshold Data

Creates an array using data from the `input file` for which voxels outside the range of `Minimum Value` to `Maximum Value` are set to zero.

Parameters:

Input file
User provided path string to the input NumPy file from which `Minimum Value` and `Maximum Value` are obtained.

Output File
User provided path string for the output `mask` NumPy file.

Maximum Value
Values above `Maximum Value` in the `input file` will result in a corresponding voxel value of zero in the `output file`.

Minimum Value
Values below `Minimum Value` in the `input file` will result in a corresponding voxel value of zero in the `output file`.

5.4.22 Transpose Array

Transposes an array from dimensions {x,y,z} to dimensions {z,y,x}.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided choice of memory keyword.

5.4.23 Voxel Replace

Replaces all voxels with indices in the range of `Start dimensions` and `End dimensions` with the specified `complex value`.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

Start dimensions: {i,j,k}
Starting indices from which the replacement will begin.

End dimensions: {i,j,k}
Final indices for which the replacement will end.

Complex Value
Real and imaginary parts of a complex number used repeatedly to replace voxel data in the `input file`.

5.4.24 Wrap Data

Reorders the data of the input NumPy file into wrap-around order.

Parameters:

Input file
User provided path string to the input NumPy file.

Output File
User provided path string for the output NumPy file.

Wrap Direction
If an array dimension has an odd number of elements, a Forward followed by a Reverse wrap is required to obtain the original array.

5.5 Library Reference – Phasing Algorithms

When coherent diffraction measurements are performed phase information is lost and in general we obtain the reciprocal space intensity distribution $I(\mathbf{k}) = |\hat{\rho}_0(\mathbf{k})|^2 = |\hat{\rho}_0(\mathbf{k})e^{i\phi(\mathbf{k})}|^2$, where $\hat{\rho}(\mathbf{k})$ is the complex density in Fourier-space. We wish to recover the complex real-space density $\rho(\mathbf{r})$. This is achieved using iterative reconstruction algorithms that traverse back and forth between direct and Fourier space while applying a constraint at each turn.

Generally, phase retrieval algorithms define a support region in real-space for which the amplitude of the object density is unrestricted. Outside this region the amplitude of the density $\rho(\mathbf{r})$ is minimised in some way. A Fourier-space constraint is also defined ($\boldsymbol{P}_{FS}$) and requires the objects amplitude to be proportional in some way to the original measurement on some set $\mathcal{M}$ such that $\boldsymbol{P}_{FS}|\hat{\rho}(\mathbf{k})|e^{i\phi(\mathbf{k})} = |\hat{\rho}_0(\mathbf{k})|\ e^{i\phi(\mathbf{k})}$, for all $\mathbf{k}$ in $\mathcal{M}$.

It is possible to execute more than one of algorithms in the *phasing pipeline* Algorithms executed in this way will share reconstruction data, hereafter referred to as sequence data. It is also possible, unless otherwise stated, for algorithms to share `support` data. This is achieved by leaving blank the path string field for the `support` NumPy file. Such a feature is desirable for algorithms that modify the support array such as the shrink-wrap method.

The following describes the available algorithms for phase retrieval.

5.5.1 Random Start

Randomises the phase of the internally stored `sequence data`. This is typically used before phase retrieval commences. If this is not used, the `sequence data` is filled with zeros.

Parameters:

Amp max
Specifies the amplitude of the array that is randomised.

5.5.2 Array Start

Replaces the internally stored `sequence data` with the supplied array. This is typically used before phase retrieval commences.

Parameters:

Input file
User provided path string to the input NumPy file.

5.5.3 Hybrid Input-Output (HIO)

Fienup's hybrid input-output (HIO) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Beta
Relaxation parameter for the HIO algorithm.

Iterations
Total number of iterations for which the algorithm will perform.

5.5.4 HIO Mask

Fienup's hybrid input-output (HIO) algorithm with the addition of a Fourier space constraint *mask*. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the HIO algorithm.

Iterations
Total number of iterations for which the algorithm will perform.

5.5.5 HIO Plus

Fienup's hybrid input-output (HIO) algorithm with non-negativity constraint and with the addition of a Fourier space constraint *mask*. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the HIO algorithm.

Iterations
Total number of iterations for which the algorithm will perform.

5.5.6 Phase Constrained HIO (PCHIO)

Fienup's hybrid input-output (HIO) algorithm with phase constraint and with the addition of a Fourier space constraint *mask*. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the HIO algorithm.

Iterations
Total number of iterations for which the algorithm will perform.

Phase Max
Voxels with phase greater than `Phase max` are penalised.

Phase Min
Voxels with phase lesser than `Phase min` are penalised.

5.5.7 Phase Gradient Constrained HIO (PGCHIO)

Fienup's hybrid input-output (HIO) algorithm with phase gradient constraint in the Q-vector direction with the addition of a Fourier space constraint *mask*. See here for more details.

> **Warning:** Co-ordinate transformation of the object or Q-vector may be required.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the HIO algorithm.

Iterations
Total number of iterations for which the algorithm will perform.

Phase Max
Voxels with phase gradient greater than `Phase max` are penalised, in the direction the Q-vector.

Phase Min
Voxels with phase gradient lesser than `Phase min` are penalised, in the direction the Q-vector.

Q {x,y,z}
Q-vector, in real space coordinates. Only the vectors direction is significant.

5.5.8 Error Reduction (ER)

Error Reduction (ER) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Iterations
Total number of iterations for which the algorithm will perform.

5.5.9 ER Mask

Error Reduction (ER) algorithm with the addition of a Fourier space constraint *mask*. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Iterations
Total number of iterations for which the algorithm will perform.

5.5.10 Phase Only ER (POER)

Phase Only Error Reduction (POER) algorithm with the addition of a Fourier space constraint *mask*. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Iterations
Total number of iterations for which the algorithm will perform.

5.5.11 Relaxed Average Alternating Reflection (RAAR)

Relaxed Average Alternating Reflection (RAAR) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the RAAR algorithm.

Iterations
Total number of iterations for which the algorithm will perform.

5.5.12 Hybrid Projection Reflection (HPR)

Hybrid Projection Reflection (HPR) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the HPR algorithm.

Iterations
Total number of iterations for which the algorithm will perform.

5.5.13 Compressed Sensing HIO (CSHIO)

Compressed Sensing HIO (CSHIO) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the HIO algorithm.

Iterations
Total number of iterations for which the algorithm will perform.

p-norm
p-norm of the Lebesgue space.

Epsilon
Positive relaxation parameter (ϵ) for the weighted p-norm.

Epsilon min
Minimum value that `epsilon` can take.

Divisor
Amount by which `epsilon` is divided when the following condition is met: $\left| \| \rho^{(n)}(\mathbf{r})\|_2 - \|\rho^{(n-1)}(\mathbf{r})\|_2 \right| < \frac{\sqrt{\epsilon}}{\eta}$

eta
Parameter (η) in the `divisor` condition above.

5.5.14 HIO Mask with Partial Coherence Optimisation (HIO Mask PC)

HIO Mask with Partial Coherence Optimisation (HIO Mask PC) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the HIO algorithm.

Iterations
Total number of iterations for which the algorithm will perform (not including R-L iterations).

Iterations preceding R-L optimisation
Interval of iterations after which Richardson-Lucy optimisation commences in successive cycles.

R-L iterations
Number of R-L iterations performed at each cycle.

Interval between R-L optimisation
Number of iterations between each Richardson-Lucy optimisation cycle.

Initial PSF HWHM The initial point spread function (PSF) optimised using the R-L algorithm is a normal Lorentzian function ($\frac{1}{\pi}\frac{a}{x^2+a^2}$). The user can specify the HWHM of its Fourier transform ($e^{-a|x|}$).

Zero fill end dimensions of PSF
After optimising the PSF with the R-L algorithm, voxels upto a distance {i,j,k} from the perimeter of the PSF array are set to zero. This can improve stability of the algorithm.

Reset PSF
Resets the PSF to a normal Lorentzian function before the next R-L optimisation cycle.

5.5.15 ER Mask with Partial Coherence Optimisation (ER Mask PC)

ER Mask with Partial Coherence Optimisation (ER Mask PC) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the ER algorithm.

Iterations
Total number of iterations for which the algorithm will perform (not including R-L iterations).

Iterations preceding R-L optimisation
Interval of iterations after which Richardson-Lucy optimisation commences in successive cycles.

R-L iterations
Number of R-L iterations performed at each cycle.

Interval between R-L optimisation
Number of iterations between each Richardson-Lucy optimisation cycle.

Initial PSF HWHM The initial point spread function (PSF) optimised using the R-L algorithm is a normal Lorentzian function ($\frac{1}{\pi}\frac{a}{x^2+a^2}$). The user can specify the HWHM of its Fourier transform ($e^{-a|x|}$).

Zero fill end dimensions of PSF
After optimising the PSF with the R-L algorithm, voxels upto a distance {i,j,k} from the perimeter of the PSF array are set to zero. This can improve stability of the algorithm.

Reset PSF
Resets the PSF to a normal Lorentzian function before the next R-L optimisation cycle.

5.5.16 HPR Mask with Partial Coherence Optimisation (HPR Mask PC)

HPR Mask with Partial Coherence Optimisation (HPR Mask PC) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the HPR algorithm.

Iterations
Total number of iterations for which the algorithm will perform (not including R-L iterations).

Iterations preceding R-L optimisation
Interval of iterations after which Richardson-Lucy optimisation commences in successive cycles.

R-L iterations
Number of R-L iterations performed at each cycle.

Interval between R-L optimisation
Number of iterations between each Richardson-Lucy optimisation cycle.

Initial PSF HWHM The initial point spread function (PSF) optimised using the R-L algorithm is a normal Lorentzian function ($\frac{1}{\pi}\frac{a}{x^2+a^2}$). The user can specify the HWHM of its Fourier transform ($e^{-a|x|}$).

Zero fill end dimensions of PSF
After optimising the PSF with the R-L algorithm, voxels upto a distance {i,j,k} from the perimeter of the PSF array are set to zero. This can improve stability of the algorithm.

Reset PSF
Resets the PSF to a normal Lorentzian function before the next R-L optimisation cycle.

5.5.17 RAAR Mask with Partial Coherence Optimisation (RAAR Mask PC)

RAAR Mask with Partial Coherence Optimisation (RAAR Mask PC) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the RAAR algorithm.

Iterations
Total number of iterations for which the algorithm will perform (not including R-L iterations).

Iterations preceding R-L optimisation
Interval of iterations after which Richardson-Lucy optimisation commences in successive cycles.

R-L iterations
Number of R-L iterations performed at each cycle.

Interval between R-L optimisation
Number of iterations between each Richardson-Lucy optimisation cycle.

Initial PSF HWHM The initial point spread function (PSF) optimised using the R-L algorithm is a normal Lorentzian function ($\frac{1}{\pi}\frac{a}{x^2+a^2}$). The user can specify the HWHM of its Fourier transform ($e^{-a|x|}$).

Zero fill end dimensions of PSF
After optimising the PSF with the R-L algorithm, voxels upto a distance {i,j,k} from the perimeter of the PSF array are set to zero. This can improve stability of the algorithm.

Reset PSF
Resets the PSF to a normal Lorentzian function before the next R-L optimisation cycle.

5.5.18 Reweighted 2D Saddle Point Optimisation (SO2D)

Reweighted 2D Saddle Point Optimisation (SO2D) algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Iterations
Total number of iterations for which the algorithm will perform.

Step optimisation iterations
Total number of iterations performed when optimising the step length.

Max step size
Maximum size for the total step length ($\sqrt{\alpha^2 + \beta^2}$).

Initial Beta
Relaxation parameter for the initial step (β).

Max step increment
Maximum amount by which the step can be incremented.

Min step increment
Minimum amount by which the step can be incremented.

Exit Ratio
Value below $|\psi|/|\psi_0|$ will halt step length optimisation.

Reset Ratio
Value above $|\psi|/|\psi_0|$ will reset step length to that of the initial value (initial β).

Exit Error
Value below $(\psi^{(n+1)} - \psi^{(n)})/\psi^{(n)}$ will halt step length optimisation.

5.5.19 Shrink Wrap

Shrink Wrap algorithm. See here for more details.

Parameters:

Exp amp
User provided path string to the input experimental data NumPy file.

Square Root Exp amp
Causes the `Exp amp` array to be square rooted before phase retrieval commences. This is necessary if the array contains intensity measurements.

Support
User provided path string to the *support* data NumPy file.

Mask
User provided path string to the `mask` data NumPy file.

Beta
Relaxation parameter for the HIO algorithm.

Iterations
Total number of iterations for which the algorithm will perform.

Cycle length
Interval of iterations after which the support is updated.

Sigma
Standard deviation of the Gaussian smoothing function for the support.

Threshold
Fractional value below which `sequence data` is not used when creating the new support.

Algorithm
Algorithm to use when performing the shrink wrapped phase retrieval.

5.6 Library Reference - Phasing Operations

5.6.1 Save Sequence

Save `sequence data` to a NumPy file.

Parameters:

Output file
User provided path string for the output NumPy file.

5.6.2 Save Support

Save `Support` data to a NumPy file.

Parameters:

Output file
User provided path string for the output NumPy file.

5.6.3 Save Residual

Save `Residual` data to a NumPy file.

Parameters:

Output file
User provided path string for the output CSV file.

5.6.4 Save PSF

Saves the internally stored `Point Spread Function data` to a NumPy file.

Parameters:

Output file
User provided path string for the ouput NumPy file.

5.6.5 Co-ordinate Transformation

Perform a co-ordinate transformation on either `sequence data` or an input array. The transformation is derived using the principles as demonstrated by Pfeifer, M.A. . The generated co-ordinate array is stored internally and is accessible for *viewing* with the *memory array* keyword `memorycoords`.

Parameters:

Transform from
Source from which to perform the transformation against.

Transform type
Transform in real or reciprocal space.

Input data
User provided path string to the input NumPy file.

Output amp file
User provided path string for the output amplitude VTK file.

Output phase file
User provided path string for the output phase VTK file.

Rocking curve type
Theta (θ) or Phi (ϕ) rocking curve measurement.

Array binning
Amount by which the array was `binned`. See the section on *binning* for further details.

2theta
2θ angle.

d theta
Increment angle for θ rocking curve measurement.

phi
ϕ angle.

d phi
Increment angle for ϕ rocking curve measurement.

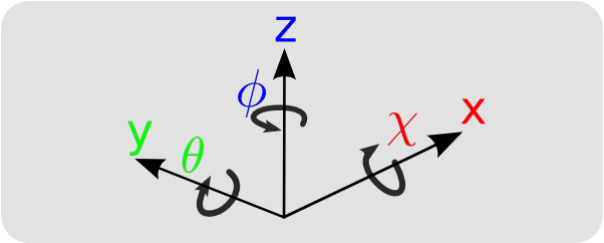
Pixel {x,y} (microns)
Detector pixel dimensions, in microns.

wavelength (nm)
Wavelength of light used in experiment. Units of nanometres.

Arm length (m)
Distance from the sample to the centre of diffraction pattern at the detector. Units of metres.

CCD x-axis flip
Reverse the direction of detector x-axis readout. Required for some PI CCD detectors.

Experimental geometry:



Curved arrows show the direction of rotation of object (as opposed to the axis). The beam is assumed to travel along the positive x direction.

5.6.6 Save Co-ordinates

Save `Co-ordinates` data to a NumPy file.

Output file
User provided path string for the output NumPy file.

Note: A co-ordinate transformation must occur prior to this operation.

CHAPTER

SIX

SCRIPTING WITH BONSU

6.1 Core concepts

Bonsu provides an object-oriented scripting interface to each algorithm for ease and interoperability with additional Python modules. To begin using this interface, import the phasing module as follows:

```
>>> from bonsu import phasing
```

Once imported, the phasing module provides access to all phase retrieval algorithms. For a complete list see *Library Reference – Phasing Algorithms*.

Each algorithm will take a number of NumPy arrays as input. Input data is loaded and prepared in wrap around order. It is therefore also necessary to utilise the *Wrap Data* function. Note that the data must be stored in a "C"-ordered double precision complex Numpy array:

```
>>> expdata = numpy.array(numpy.load("expdata.npy"), dtype=numpy.cdouble, copy=True,␣
↪order='C')
>>> numpy.sqrt(expdata, expdata)
>>> expdata[:] = phasing.WrapArray(expdata)
```

The data `mask` in wrap around order is also needed:

```
>>> mask = numpy.load("mask.npy")
>>> mask[:] = phasing.WrapArray(mask)
```

An array to contain the reconstructed object is also needed and can be created in a number of ways, with data initialised as needed, i.e. random data or initial starting data from a previous reconstruction.

```
>>> seqdata = numpy.empty_like(expdata)
>>> seqdata[:] = 1.0 # initialise as 1.0.
```

To utilise the *HIO Mask* algorithm, create an instance as follows:

```
>>> hio = phasing.HIOMask()
```

Then initialise the remaining parameters for 100 iterations:

```
>>> hio.SetStartiter(0)
>>> hio.SetNumiter(100)
>>> hio.SetNumthreads(3)
>>> hio.SetBeta(0.9)
>>> hio.SetExpdata(expdata)
>>> hio.SetSupport(support)
>>> hio.SetMask(mask)
>>> hio.SetSeqdata(seqdata)
```

To begin reconstruction we first prepare the algorithm as follows:

```
>>> hio.Prepare()
```

Execution then follows:

```
>>> hio.Start()
```

In a threaded or non-blocking use of the algorithm, it is possible the pause, resume and stop the reconstruction using the following methods:

```
>>> hio.Pause()
>>> hio.Resume()
>>> hio.Stop()
```

The output of the reconstruction is saved as follows:

```
>>> numpy.save("output.npy", seqdata)
```

Each algorithm can use `Shrink Wrap` to modify the support during reconstruction. Prefix the algorithm name with `SW` to utilise the `Shrink Wrap` version of the algorithm. For example, to use the `Shrink Wrap` version of the *HIO Mask* algorithm, import and instantiate the following:

```
>>> swhio = phasing.SWHIOMask()
```

Then initialise the remaining parameters as follows:

```
>>> swhio.SetStartiter(0)
>>> swhio.SetNumiter(100)
>>> swhio.SetNumthreads(3)
>>> swhio.SetExpdata(expdata)
>>> swhio.SetSupport(support)
>>> swhio.SetMask(mask)
>>> swhio.SetSeqdata(seqdata)
>>> swhio.Prepare()
```

Methods unqiue to the `Shrink Wrap` implementation that require initialisation are as follows:

```
>>> swhio.SetSWCyclelength(20)
>>> swhio.SetSWThreshold(0.2)
>>> swhio.SetSWSigma(3.0)
```

To begin reconstruction we also first prepare the `Shrink Wrap` algorithm as follows:

```
>>> swhio.SWPrepare()
```

Execution then follows:

```
>>> swhio.SWStart()
```

6.2 Custom Algorithms

Bonsu provides tools to build custom projection algorithms. To build a custom algorithm, create a script that imports an abstract algorithm class and subclasses the `DoIter` and `RSCons` methods. `DoIter` is called at each iteration of the algorithm. It will apply the real-space and Fourier-space constraints. `RSCons` is a method called by `DoIter` to apply the real-space constraint.

```
from bonsu.phasing.abstract import PhaseAbstract

class MyAlgorithm(PhaseAbstract):
        """
        My custom HIO algorithm
        """
        def DoIter(self):
                self.rho_m1[:] = self.seqdata[:]
                fftw_stride(self.seqdata,self.seqdata,self.plan,FFTW_TORECIP,1)
                self.SetRes()
                self.SetAmplitudes()
                fftw_stride(self.seqdata,self.seqdata,self.plan,FFTW_TOREAL,1)
                n = self.citer_flow[0]
                self.residual[n] = self.res/self.sos
                amp = numpy.absolute(self.seqdata)
                sos1 = numpy.sum(amp*amp)
                self.RSCons()
                amp = numpy.absolute(self.seqdata)
                sos2 = numpy.sum(amp*amp)
                norm = sqrt(sos1/sos2)
                self.seqdata[:] = norm*self.seqdata[:]
                self.citer_flow[0] += 1

        def RSCons(self):
                realsupport = numpy.real(self.support)
                nosupport = realsupport < 0.5
                self.seqdata[nosupport] = self.rho_m1[nosupport] - self.
↪seqdata[nosupport] * self.beta
```

The custom algorithm is then instantiated as before:

```
myalg = MyAlgorithm()
```

Initialisation of parameters proceeds as before. To begin reconstruction prepare the algorithm as follows:

```
myalg.PrepareCustom()
```

Execution then follows as before:

```
myalg.Start()
```

6.3 Concurrent Phase Retrieval

Bonsu provides tools to reconstruct atomic displacement field information (and subsequently strain field information) via concurrent phase reconstruction using diffraction pattern data that is obtained from multiple Bragg reflections of a single crystal. The method used is described in detail in the following publication: Newton, Phys. Rev. B 102, 014104 (2020) .

The concurrent phase retrieval method can in principle utilise any projection algorithm implemented using the *Custom Algorithm interface*. The following example illustrates how to use utilise the method using the standard Abstract Algorithm class (HIO). Please note that diffraction pattern data should undergo *co-ordinate transformation* and *interpolation* onto a regular grid prior to commencement of the reconstruction process. Please also note that no checking of the data type (numpy.cdouble) and consistency in the array dimensions is performed.

```
from bonsu.phasing.concurrent import PhaseConcurrent
from bonsu.phasing.abstract import PhaseAbstract
from bonsu.phasing.wrap import WrapArray

# Specify number of diffraction patterns / Q-vectors
NQ = 4

# Instantiate class object
d = PhaseConcurrent()
# Specify class object of phase retrieval algorithm used for reconstruction
# Define your own and use here, if preferred.
d.SetBaseClass(PhaseAbstract)
# Set number of concurrent datasets / Q-vectors
d.SetNQ(NQ)

# Set Q-vectors
d.SetQ(0, [1,0,0])
d.SetQ(1, [1,1,0])
d.SetQ(2, [0,1,0])
d.SetQ(3, [0,0,1])

# Define names of files
expnames = ["diffamp0.npy","diffamp1.npy","diffamp2.npy","diffamp3.npy"]
masknames = ["mask0.npy","mask1.npy","mask2.npy","mask3.npy"]
supportname = "support.npy"

# Use names above to load arrays into list of length NQ
amps = [WrapArray(numpy.load(name)) for name in expnames]
masks = [WrapArray(numpy.load(name)) for name in masknames]
supports = [numpy.load(supportname)]*NQ
seqdatas = [numpy.ones_like(amp) for amp in amps]

# Class methods below are the same as those in the phase retrieval algorithm
# class but will instead take a list of length NQ.
d.SetExpdata(amps)
d.SetSupport(supports)
d.SetMask(masks)
d.SetSeqdata(seqdatas)
d.SetBeta([0.9]*NQ)

# Set iterations
d.SetStartiter(0)
d.SetNumiter(100)
```

(continues on next page)

(continued from previous page)

```
# Number of FFTW threads for each concurrent reconstruction.
d.SetNumthreads(1)

# Prepare
d.Prepare()
# Start
d.Start()

# Save reconstructed phase for each Q-vector
# Filenames are automatically indexed.
d.SaveSequences("output.npy")

# Save displacement field
d.SaveDField("dfield.npy")
```

6.4 Class Reference

6.4.1 Abstract Classes

class bonsu.phasing.abstract.PhaseAbstract(*parent=None*)

Bases: object

Phasing Base Class

PrepareCustom()

Prepare algorithm using FFTW python interface.

SetSOS()

Set Sum of Squares.

GetSOS()

Get Sum of Squares.

SetStartiter(*startiter*)

Set the starting iteration number.

SetNumiter(*numiter*)

Set the number of iterations of the algorithm to perform.

SetNumthreads(*nthreads*)

Set the number of FFTW threads.

SetSeqdata(*seqdata*)

Set the reconstruction data array.

GetSeqdata()

Get the reconstruction data array.

SetExpdata(*expdata*)

Set the raw experimental amplitude data array.

GetExpdata()

Get the raw experimental amplitude data array.

SetSupport(*support*)

Set the support array.

`GetSupport()`
: Get the support array.

`SetMask(`*mask*`)`
: Set the mask data array.

`GetMask()`
: Set the mask data array.

`SetBeta(`*beta*`)`
: Set beta relaxation parameter.

`GetBeta()`
: Get beta relaxation parameter.

`Pause()`
: Pause the reconstruction process.

`Resume()`
: Resume the reconstruction process.

`Start()`
: Start the reconstruction process.

`Stop()`
: Stop the reconstruction process.

`class bonsu.phasing.abstract.PhaseAbstractPC(`*parent=None*`)`

Bases: *PhaseAbstract*

Phasing Partial Coherence Base Class

`SetPSF(`*psf*`)`
: Set point-spread function from data array.

`GetPSF()`
: Get point-spread function from data array.

`LorentzFillPSF()`
: Create a point-spread function with Lorentz distribution. This will use the HWHM specified using SetGammaHWHM.

`SetNumiterRLpre(`*niterrlpre*`)`
: Set the number of iterations before RL optimisation.

`GetNumiterRLpre()`
: Get the number of iterations before RL optimisation.

`SetNumiterRL(`*niterrl*`)`
: Set the number of RL iterations.

`GetNumiterRL()`
: Get the number of RL iterations.

`SetNumiterRLinterval(`*niterrlinterval*`)`
: Set number of iterations between each RL optimisation cycle.

`GetNumiterRLinterval()`
: Get number of iterations between each RL optimisation cycle.

`SetGammaHWHM(`*gammaHWHM*`)`
: Set the half-width half-maximum of the Lorentz distribution. This is utilised in the LorentzFillPSF method.

`GetGammaHWHM()`

Get the half-width half-maximum of the Lorentz distribution.

`SetPSFZeroEnd(ze)`

Voxels upto a distance of [i,j,k] from the perimeter of the PSF array are set to zero. This can improve stability of the algorithm.

`GetPSFZeroEnd()`

Get zero voxels perimeter size.

`class bonsu.phasing.ShrinkWrap.ShrinkWrap`

Shrink Wrap algorithm.

`SetSWCyclelength(cycle)`

Set interval of iterations after which the support is updated.

`GetSWCyclelength()`

Get interval of iterations after which the support is updated.

`SetSWSigma(sigma)`

Set standard deviation of the Gaussian smoothing function for the support.

`GetSWSigma()`

Get standard deviation of the Gaussian smoothing function for the support.

`SetSWThreshold(thresh)`

Set fractional value below which sequence data is not used when creating the new support.

`GetSWThreshold()`

Get fractional value below which sequence data is not used when creating the new support.

`SWPrepare()`

Prepare shrink wrap algorithm.

`SWStart()`

Start the shrink wrap reconstruction process.

6.4.2 Hybrid Input-Output Classes

`class bonsu.phasing.HIO.HIO(parent=None)`

Bases: *PhaseAbstract*

Fienup's hybrid input-output (HIO) algorithm.

`class bonsu.phasing.HIO.SWHIO`

Bases: *HIO*, *ShrinkWrap*

Fienup's hybrid input-output (HIO) algorithm. Shrink wrapped.

`class bonsu.phasing.HIO.HIOMask(parent=None)`

Bases: *PhaseAbstract*

Fienup's hybrid input-output (HIO) algorithm with the addition of a Fourier space constraint mask.

`class bonsu.phasing.HIO.SWHIOMask`

Bases: *HIOMask*, *ShrinkWrap*

Fienup's hybrid input-output (HIO) algorithm with the addition of a Fourier space constraint mask. Shrink wrapped.

class bonsu.phasing.HIO.HIOPlus(*parent=None*)

Bases: *PhaseAbstract*

Fienup's hybrid input-output (HIO) algorithm with non-negativity constraint and with the addition of a Fourier space constraint mask.

class bonsu.phasing.HIO.SWHIOPlus

Bases: *HIOPlus*, *ShrinkWrap*

Fienup's hybrid input-output (HIO) algorithm with non-negativity constraint and with the addition of a Fourier space constraint mask. Shrink wrapped.

class bonsu.phasing.HIO.PCHIO(*parent=None*)

Bases: *PhaseAbstract*

Fienup's hybrid input-output (HIO) algorithm with phase constraint and with the addition of a Fourier space constraint mask.

SetMaxphase(*max*)

Set phase maximum.

GetMaxphase()

Get phase maximum.

SetMinphase(*min*)

Set phase minimum.

GetMinphase()

Get phase minimum.

class bonsu.phasing.HIO.SWPCHIO

Bases: *PCHIO*, *ShrinkWrap*

Fienup's hybrid input-output (HIO) algorithm with phase constraint and with the addition of a Fourier space constraint mask. Shrink wrapped.

class bonsu.phasing.HIO.PGCHIO(*parent=None*)

Bases: *PhaseAbstract*

Fienup's hybrid input-output (HIO) algorithm with phase gradient constraint in the Q-vector direction with the addition of a Fourier space constraint mask.

SetMaxphase(*max*)

Set phase maximum.

GetMaxphase()

Get phase maximum.

SetMinphase(*min*)

Set phase minimum.

GetMinphase()

Get phase minimum.

SetQ(*q*)

Set Q-vector tuple

GetQ()

Get Q-vector tuple

class bonsu.phasing.HIO.SWPGCHIO

Bases: *PGCHIO*, *ShrinkWrap*

Fienup's hybrid input-output (HIO) algorithm with phase gradient constraint in the Q-vector direction with the addition of a Fourier space constraint mask. Shrink wrapped.

class bonsu.phasing.HIO.HIOMaskPC(*parent=None*)

Bases: *PhaseAbstractPC*

HIO Mask with Partial Coherence Optimisation.

class bonsu.phasing.HIO.SWHIOMaskPC

Bases: *HIOMaskPC*, *ShrinkWrap*

HIO Mask with Partial Coherence Optimisation. Shrink wrapped.

Start()

Start the reconstruction process.

6.4.3 Error Reduction Classes

class bonsu.phasing.ER.ER(*parent=None*)

Bases: *PhaseAbstract*

Error Reduction (ER) algorithm.

class bonsu.phasing.ER.SWER

Bases: *ER*, *ShrinkWrap*

Error Reduction (ER) algorithm. Shrink wrapped.

class bonsu.phasing.ER.ERMask(*parent=None*)

Bases: *PhaseAbstract*

Error Reduction (ER) algorithm with the addition of a Fourier space constraint mask.

class bonsu.phasing.ER.SWERMask

Bases: *ERMask*, *ShrinkWrap*

Error Reduction (ER) algorithm with the addition of a Fourier space constraint mask. Shrink wrapped.

class bonsu.phasing.ER.POER(*parent=None*)

Bases: *PhaseAbstract*

Phase Only Error Reduction (POER) algorithm with the addition of a Fourier space constraint mask.

class bonsu.phasing.ER.ERMaskPC(*parent=None*)

Bases: *PhaseAbstractPC*

ER Mask with Partial Coherence Optimisation algorithm.

class bonsu.phasing.ER.SWERMaskPC

Bases: *ERMaskPC*, *ShrinkWrap*

ER Mask with Partial Coherence Optimisation algorithm. Shrink wrapped.

Start()

Start the reconstruction process.

6.4.4 Hybrid Project-Reflection Classes

class bonsu.phasing.HPR.HPR(*parent=None*)

Bases: *PhaseAbstract*

Hybrid Projection Reflection (HPR) algorithm.

class bonsu.phasing.HPR.SWHPR

Bases: *HPR*, *ShrinkWrap*

Hybrid Projection Reflection (HPR) algorithm. Shrink wrapped.

class bonsu.phasing.HPR.HPRPC(*parent=None*)

Bases: *PhaseAbstractPC*

HPR Mask with Partial Coherence Optimisation algorithm.

class bonsu.phasing.HPR.SWHPRPC

Bases: *HPRPC*, *ShrinkWrap*

HPR Mask with Partial Coherence Optimisation algorithm. Shrink wrapped.

Start()

Start the reconstruction process.

6.4.5 Relaxed Average Alternating Reflection Classes

class bonsu.phasing.RAAR.RAAR(*parent=None*)

Bases: *PhaseAbstract*

Relaxed Average Alternating Reflection (RAAR) algorithm.

class bonsu.phasing.RAAR.SWRAAR

Bases: *RAAR*, *ShrinkWrap*

Relaxed Average Alternating Reflection (RAAR) algorithm. Shrink wrapped.

class bonsu.phasing.RAAR.RAARPC(*parent=None*)

Bases: *PhaseAbstractPC*

RAAR Mask with Partial Coherence Optimisation algorithm.

class bonsu.phasing.RAAR.SWRAARPC

Bases: *RAARPC*, *ShrinkWrap*

RAAR Mask with Partial Coherence Optimisation algorithm. Shrink wrapped.

Start()

Start the reconstruction process.

6.4.6 CSHIO Classes

class bonsu.phasing.CSHIO.CSHIO(*parent=None*)

Bases: *PhaseAbstract*

Compressed Sensing HIO (CSHIO) algorithm.

SetPnorm(*p*)

Set p-norm of the Lebesgue space.

`GetPnorm()`
Get p-norm of the Lebesgue space.

`SetEpsilon(`*epsilon*`)`
Set positive relaxation parameter (epsilon) for the weighted p-norm.

`GetEpsilon()`
Get positive relaxation parameter (epsilon) for the weighted p-norm.

`SetEpsilonmin(`*epsilonmin*`)`
Set minimum value that epsilon can take.

`GetEpsilonmin()`
Set minimum value that epsilon can take.

`SetDivisor(`*d*`)`
Set amount by which epsilon is divided when constraint is met.

`GetDivisor()`
Get amount by which epsilon is divided when constraint is met.

`SetEta(`*eta*`)`
Set parameter in the divisor condition.

`GetEta()`
Get parameter in the divisor condition.

`class bonsu.phasing.CSHIO.SWCSHIO`
Bases: *CSHIO*, *ShrinkWrap*

Compressed Sensing HIO (CSHIO) algorithm. Shrink wrapped.

6.4.7 SO2D Classes

`class bonsu.phasing.SO2D.SO2D(`*parent=None*`)`
Bases: *PhaseAbstract*

`SetNumsoiter(`*numsoiter*`)`
Set total number of iterations performed when optimising the step length.

`GetNumsoiter()`
Get total number of iterations performed when optimising the step length.

`SetReweightiter(`*reweightiter*`)`
Set iteration after which reweighting is applied. A negative value implies no reweighting.

`GetReweightiter()`
Get iteration after which reweighting is applied. A negative value implies no reweighting.

`SetDtaumax(`*dtaumax*`)`
Set maximum amount by which the step can be incremented.

`GetDtaumax()`
Get maximum amount by which the step can be incremented.

`SetDtaumin(`*dtaumin*`)`
Set minimum amount by which the step can be incremented.

`GetDtaumin()`
Get minimum amount by which the step can be incremented.

`SetTaumax(`*taumax*`)`
Set maximum size for the total step length.

`GetTaumax()`
Get maximum size for the total step length.

`SetPsiexitratio(`*psiexitratio*`)`
Set value below **|psi|**/**|psi_0|** that will halt step length optimisation.

`GetPsiexitratio()`
Get value below **|psi|**/**|psi_0|** that will halt step length optimisation.

`SetPsiexiterror(`*psiexiterror*`)`
Set value below (psi^(n+1) - psi^(n))/psi^(n) that will halt step length optimisation.

`GetPsiexiterror()`
Get value below (psi^(n+1) - psi^(n))/psi^(n) that will halt step length optimisation.

`SetPsiresetratio(`*psiresetratio*`)`
Set value above **|psi|**/**|psi_0|** that will reset the step length to that of the initial value.

`GetPsiresetratio()`
Get value above **|psi|**/**|psi_0|** that will reset the step length to that of the initial value.

`class bonsu.phasing.SO2D.SWSO2D`
Bases: *SO2D*, *ShrinkWrap*

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

b

INDEX

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W