blimpy

Nov 18, 2022

Contents

CHAPTER 1

License

Documentation Status codecov JOSS status

CHAPTER 2

Breakthrough Listen I/O Methods for Python.

2.1 Filterbank + Raw file readers

This repository contains Python readers for interacting with Sigproc filterbank (.fil), HDF5 (.h5) and guppi raw (.raw) files, as used in the Breakthrough Listen search for intelligent life.

2.2 Installation

The latest release can be installed via pip:

```
pip install blimpy
```

Or, the latest version of the development code can be installed from the github repo and then run `python setup.py install` or `pip install .` (with sudo if required), or by using the following terminal command:

```
pip install https://github.com/UCBerkeleySETI/blimpy/tarball/master
```

To install everything required to run the unit tests, run:

```
pip install -e .[full]
```

You will need `numpy`, `h5py`, `astropy`, `scipy`, and `matplotlib` as dependencies. A `pip install` should pull in numpy, h5py, and astropy, but you may still need to install scipy and matplotlib separately. To interact with files compressed with bitshuffle, you'll need the `bitshuffle` package too.

Note that h5py generally needs to be installed in this way:

```
$ pip install --no-binary=h5py h5py
```

2.3 Command line utilities

After installation, some command line utilities will be installed:

- `watutil`, Read/write/plot an .h5 file or a .fil file.
- `rawutil`, Plot data in a guppi raw file.
- `fil2h5`, Convert a .fil file into .h5 format.
- `h52fil`, Convert an .h5 file into .fil format.
- `bldice`, Dice a smaller frequency region from (either from/to .h5 or .fil).
- `matchfils`, Check if two .fil files are the same.
- `calcload`, Calculate the Waterfall max_load value needed to load the data array for a given file.
- `rawhdr`, Display the header fields of a raw guppi file.
- `stax`, For a collection of .h5 or .fil files sharing the same frequency range, create a vertical stack of waterfall plots as a single PNG file.
- `stix`, For a single very large .h5 or .fil file, create a horizontal or vertical stack of waterfall plots as a single PNG file.

Use the `-h` flag to any of the above command line utilities to display their available arguments.

2.4 Reading blimpy filterbank files in .fil or .h5 format

The `blimpy.Waterfall` provides a Python API for interacting with filterbank data. It supports all BL filterbank data products; see this example Jupyter notebook for an overview.

From the python, ipython or jupiter notebook environments.

```
from blimpy import Waterfall
fb = Waterfall('/path/to/filterbank.fil')
#fb = Waterfall('/path/to/filterbank.h5') #works the same way
fb.info()
data = fb.data
```

2.5 Reading guppi raw files

The Guppi Raw format can be read using the `GuppiRaw` class from `guppi.py`:

```
from blimpy import GuppiRaw
gr = GuppiRaw('/path/to/guppirawfile.raw')

header, data = gr.read_next_data_block()
```

or

```
from blimpy import GuppiRaw
gr = GuppiRaw('/path/to/guppirawfile.raw')

for header, data_x, data_y in gr.get_data():
    # process data
```

Note: Most users should start analysis with filterbank files, which are smaller in size and have been generated from the guppi raw files.

2.6 Further reading

A detailed overview of the data formats used in Breakthrough Listen can be found in our data format paper. An archive of data files from the Breakthrough Listen program is provided at seti.berkeley.edu/opendata.

2.7 If you have any requests or questions, please lets us know!

CHAPTER 3

Writing Docs

This is a rough guide to contributing to blimpy documentation.

3.1 TLDR

- Write docs in `.rst` or `.md`
- Add the name of the new docs file to `index.rst`
- Preview docs by installing `sphinx` via `pip` and run `make html` in the `docs/` directory
- Site will update automatically after pushing to the blimpy repo

3.2 Creating a Page

Currently, `readthedocs` is able to process two kinds of files: `reStructuredText (.rst)` and `Markdown (.md)`. You can find a brief guide for reStructuredText here and a guide for Markdown here.

The two file types are rendered equally by sphinx, so feel free to use whichever one you're more comfortable with.

To create a new page, you can create a new file in the same directory as `index.rst`. After creating the file, add the filename of the new file to `index.rst` to add a link it on the index page. The new file will also show up in the sidebar after this.

3.3 Previewing Docs

The docs are rendered using `sphinx`, a python package. Use `pip install sphinx` to install the package.

After `sphinx` is installed, you can preview your changes by running `make html` in the `docs/` directory. The rendered html files will be stored in `docs/_build/html`. The actual site will look exactly like the rendered files when built.

3.4 Automatic Documentation

You can run `sphinx-apidoc -o . ../blimpy/ -f` in `blimpy/docs` to generate autodoc pages from all the python modules in blimpy. Make sure to run this command every time a new file is added to blimpy.

3.5 Updating the Site

The blimpy Github repo is connected to *readthedocs.org* with a webhook. *readthedocs* will automatically update the site whenever a new commit is added to the repo.

CHAPTER 4

blimpy

4.1 blimpy package

4.1.1 Subpackages

blimpy.calib_utils package

Submodules

blimpy.calib_utils.calib_plots module

`blimpy.calib_utils.calib_plots.`**`get_diff`**(*dio_cross*, *feedtype*, ***kwargs*)

Returns ON-OFF for all Stokes parameters given a cross_pols noise diode measurement

`blimpy.calib_utils.calib_plots.`**`plot_Stokes_diode`**(*dio_cross*, *diff=True*, *feedtype='l'*, ***kwargs*)

Plots the uncalibrated full stokes spectrum of the noise diode. Use diff=False to plot both ON and OFF, or diff=True for ON-OFF

`blimpy.calib_utils.calib_plots.`**`plot_calibrated_diode`**(*dio_cross*, *chan_per_coarse=8*, *feedtype='l'*, ***kwargs*)

Plots the corrected noise diode spectrum for a given noise diode measurement after application of the inverse Mueller matrix for the electronics chain.

`blimpy.calib_utils.calib_plots.`**`plot_diode_fold`**(*dio_cross*, *bothfeeds=True*, *feedtype='l'*, *min_samp=-500*, *max_samp=7000*, *legend=True*, ***kwargs*)

Plots the calculated average power and time sampling of ON (red) and OFF (blue) for a noise diode measurement over the observation time series

`blimpy.calib_utils.calib_plots.`**`plot_diodespec`**(*ON_obs*, *OFF_obs*, *calflux*, *calfreq*, *spec_in*, *units='mJy'*, ***kwargs*)

Plots the full-band Stokes I spectrum of the noise diode (ON-OFF)

`blimpy.calib_utils.calib_plots.`**`plot_fullcalib`**(*dio_cross*, *feedtype='l'*, ***kwargs*)

Generates and shows five plots: Uncalibrated diode, calibrated diode, fold information, phase offsets, and gain offsets for a noise diode measurement. Most useful diagnostic plot to make sure calibration proceeds correctly.

`blimpy.calib_utils.calib_plots.`**`plot_gain_offsets`**(*dio_cross*, *dio_chan_per_coarse=8*, *feedtype='l'*, *ax1=None*, *ax2=None*, *legend=True*, ***kwargs*)

Plots the calculated gain offsets of each coarse channel along with the time averaged power spectra of the X and Y feeds

`blimpy.calib_utils.calib_plots.`**`plot_phase_offsets`**(*dio_cross*, *chan_per_coarse=8*, *feedtype='l'*, *ax1=None*, *ax2=None*, *legend=True*, ***kwargs*)

Plots the calculated phase offsets of each coarse channel along with the UV (or QU) noise diode spectrum for comparison

blimpy.calib_utils.fluxcal module

`blimpy.calib_utils.fluxcal.`**`Jy_to_Kelvin`**(*flux*, *freqs*)

Convert flux spectrum in Jy/beam to temperature units. Frequency inputs in MHz.

Parameters

- **flux** (*1D Array (float)*) – Spectrum over frequency band in Jy
- **freqs** (*1D Array (float)*) – Frequencies (in MHz)

`blimpy.calib_utils.fluxcal.`**`calibrate_fluxes`**(*main_obs_name*, *dio_name*, *dspec*, *Tsys*, *fullstokes=False*, ***kwargs*)

Produce calibrated Stokes I for an observation given a noise diode measurement on the source and a diode spectrum with the same number of coarse channels

Parameters

- **main_obs_name** (*str*) – Path to filterbank file containing final data to be calibrated
- **dio_name** (*str*) – Path to filterbank file for observation on the target source with flickering noise diode
- **dspec** (*1D Array (float) or float*) – Coarse channel spectrum (or average) of the noise diode in Jy (obtained from diode_spec())
- **Tsys** (*1D Array (float) or float*) – Coarse channel spectrum (or average) of the system temperature in Jy
- **fullstokes** (*boolean*) – Use fullstokes=True if data is in IQUV format or just Stokes I, use fullstokes=False if it is in cross_pols format

`blimpy.calib_utils.fluxcal.`**`diode_spec`**(*calON_obs*, *calOFF_obs*, *calflux*, *calfreq*, *spec_in*, *average=True*, *oneflux=False*, ***kwargs*)

Calculate the coarse channel spectrum and system temperature of the noise diode in Jy given two noise diode measurements ON and OFF the calibrator source with the same frequency and time resolution

Parameters

- **calON_obs** (*str*) – (see f_ratios() above)
- **calOFF_obs** (*str*) – (see f_ratios() above)
- **calflux** (*float*) – Known flux of calibrator source at a particular frequency
- **calfreq** (*float*) – Frequency where calibrator source has flux calflux (see above)

- **spec_in** (*float*) – Known power-law spectral index of calibrator source. Use convention flux(frequency) = constant * frequency^(spec_in)
- **average** (*boolean*) – Use average=True to return noise diode and Tsys spectra averaged over frequencies

`blimpy.calib_utils.fluxcal.`**f_ratios**(*calON_obs*, *calOFF_obs*, *chan_per_coarse*, ***kwargs*)

Calculate f_ON, and f_OFF as defined in van Straten et al. 2012 equations 2 and 3

Parameters

- **calON_obs** (*str*) – Path to filterbank file (any format) for observation ON the calibrator source
- **calOFF_obs** (*str*) – Path to filterbank file (any format) for observation OFF the calibrator source

`blimpy.calib_utils.fluxcal.`**foldcal**(*data*, *tsamp*, *diode_p=0.04*, *numsamps=1000*, *switch=False*, *inds=False*)

Returns time-averaged spectra of the ON and OFF measurements in a calibrator measurement with flickering noise diode

Parameters

- **data** (*2D Array object (float)*) – 2D dynamic spectrum for data (any Stokes parameter) with flickering noise diode.
- **tsamp** (*float*) – Sampling time of data in seconds
- **diode_p** (*float*) – Period of the flickering noise diode in seconds
- **numsamps** (*int*) – Number of samples over which to average noise diode ON and OFF
- **switch** (*boolean*) – Use switch=True if the noise diode "skips" turning from OFF to ON once or vice versa
- **inds** (*boolean*) – Use inds=True to also return the indexes of the time series where the ND is ON and OFF

`blimpy.calib_utils.fluxcal.`**get_Tsys**(*calON_obs*, *calOFF_obs*, *calflux*, *calfreq*, *spec_in*, ***kwargs*)

Returns frequency dependent system temperature given observations on and off a calibrator source

Parameters **diode_spec()) ((*See*) –**

`blimpy.calib_utils.fluxcal.`**get_Tsys_nodiode**(*calON_obs_name*, *calOFF_obs_name*, *calflux*, *calfreq*, *spec_in*)

Calculates system temperature from two flux calibrator scans taken without noise diode flickering. CURRENTLY ONLY IMPLEMENTED FOR STOKES I DATA.

Parameters

- **calON_obs_name** (*str*) – Path to filterbank file for scan ON calibrator target
- **calOFF_obs_name** (*str*) – Path to filterbank file for scan OFF calibrator
- **calflux** (*float*) – Flux in Jy of the calibrator source
- **calfreq** (*float or int*) – Frequency at which calflux was taken (MHz)
- **spec_in** (*float*) – Spectral index of this calibrator

`blimpy.calib_utils.fluxcal.`**get_calfluxes**(*calflux*, *calfreq*, *spec_in*, *centerfreqs*, *oneflux*)

Given properties of the calibrator source, calculate fluxes of the source in a particular frequency range

Parameters

- **calflux** (*float*) – Known flux of calibrator source at a particular frequency
- **calfreq** (*float*) – Frequency where calibrator source has flux calflux (MHz) (see above)
- **spec_in** (*float*) – Known power-law spectral index of calibrator source. Use convention flux(frequency) = constant * frequency^(spec_in)
- **centerfreqs** (*1D Array (float)*) – Central frequency values of each coarse channel
- **oneflux** (*boolean*) – Use oneflux to choose between calculating the flux for each core channel (False) or using one value for the entire frequency range (True)

`blimpy.calib_utils.fluxcal.`**`get_centerfreqs`**(*freqs*, *chan_per_coarse*)

Returns central frequency of each coarse channel

Parameters

- **freqs** (*1D Array (float)*) – Frequency values for each bin of the spectrum
- **chan_per_coarse** (*int*) – Number of frequency bins per coarse channel

`blimpy.calib_utils.fluxcal.`**`integrate_calib`**(*name*, *chan_per_coarse*, *fullstokes=False*, ***kwargs*)

Folds Stokes I noise diode data and integrates along coarse channels

Parameters

- **name** (*str*) – Path to noise diode filterbank file
- **chan_per_coarse** (*int*) – Number of frequency bins per coarse channel
- **fullstokes** (*boolean*) – Use fullstokes=True if data is in IQUV format or just Stokes I, use fullstokes=False if it is in cross_pols format

`blimpy.calib_utils.fluxcal.`**`integrate_chans`**(*spec*, *chan_per_coarse*)

Integrates over each core channel of a given spectrum. Important for calibrating data with frequency/time resolution different from noise diode data

Parameters

- **spec** (*1D Array (float)*) – Spectrum (any Stokes parameter) to be integrated
- **chan_per_coarse** (*int*) – Number of frequency bins per coarse channel

blimpy.calib_utils.stokescal module

`blimpy.calib_utils.stokescal.`**`apply_Mueller`**(*I*, *Q*, *U*, *V*, *gain_offsets*, *phase_offsets*, *chan_per_coarse*, *feedtype='l'*)

Returns calibrated Stokes parameters for an observation given an array of differential gains and phase differences.

`blimpy.calib_utils.stokescal.`**`calibrate_pols`**(*cross_pols*, *diode_cross*, *obsI=None*, *onefile=True*, *feedtype='l'*, ***kwargs*)

Write Stokes-calibrated filterbank file for a given observation with a calibrator noise diode measurement on the source

Parameters

- **cross_pols** (*string*) – Path to cross polarization filterbank file (rawspec output) for observation to be calibrated
- **diode_cross** (*string*) – Path to cross polarization filterbank file of noise diode measurement ON the target

- **obsI** (*string*) – Path to Stokes I filterbank file of main observation (only needed if one-file=False)
- **onefile** (*boolean*) – True writes all calibrated Stokes parameters to a single filterbank file, False writes four separate files
- **feedtype** (*'l' or 'c'*) – Basis of antenna dipoles. 'c' for circular, 'l' for linear

`blimpy.calib_utils.stokescal.`**`convert_to_coarse`**(*data*, *chan_per_coarse*)
Converts a data array with length n_chans to an array of length n_coarse_chans by averaging over the coarse channels

`blimpy.calib_utils.stokescal.`**`fracpols`**(*cross_dat*, ***kwargs*)
Output fractional linear and circular polarizations for a rawspec cross polarization .fil file. NOT STANDARD USE

`blimpy.calib_utils.stokescal.`**`gain_offsets`**(*Idat*, *Qdat*, *Udat*, *Vdat*, *tsamp*, *chan_per_coarse*, *feedtype='l'*, ***kwargs*)
Determines relative gain error in the X and Y feeds for an observation given I and Q (I and V for circular basis) noise diode data.

`blimpy.calib_utils.stokescal.`**`get_stokes`**(*cross_dat*, *feedtype='l'*)
Output stokes parameters (I,Q,U,V) for a rawspec cross polarization filterbank file

`blimpy.calib_utils.stokescal.`**`phase_offsets`**(*Idat*, *Qdat*, *Udat*, *Vdat*, *tsamp*, *chan_per_coarse*, *feedtype='l'*, ***kwargs*)
Calculates phase difference between X and Y feeds given U and V (U and Q for circular basis) data from a noise diode measurement on the target

`blimpy.calib_utils.stokescal.`**`write_polfils`**(*cross_dat*, *str_I*, ***kwargs*)
Writes two new filterbank files containing fractional linear and circular polarization data

`blimpy.calib_utils.stokescal.`**`write_stokefils`**(*cross_dat*, *str_I*, *Ifil=False*, *Qfil=False*, *Ufil=False*, *Vfil=False*, *Lfil=False*, ***kwargs*)
Writes up to 5 new filterbank files corresponding to each Stokes parameter (and total linear polarization L) for a given cross polarization .fil file

Module contents

4.1.2 Submodules

4.1.3 blimpy.calcload module

calcload.py - Calculate the Waterfall max_load value needed to load the data array for a given file.

`blimpy.calcload.`**`calc_max_load`**(*arg_path*, *verbose=False*)
Calculate the max_load parameter value for a subsequent Waterfall instantiation.

Algorithm:

- A = minimum Waterfall object size.
- B = data array size within one polarisation.
- Return ceil(A + B in GB)

`blimpy.calcload.`**`cmd_tool`**(*args=None*)
Command line entrypoint for "calcload"

4.1.4 blimpy.dice module

Script to dice data to course channel level. From BL FIL of HDF5 files, and outputs HDF5 with '_diced' appended to the file name.

..author: Greg Hellbourg (gregory.hellbourg@berkeley.edu)

March 2018

`blimpy.dice.`**`cmd_tool`**(*args=None*)

Dices (extracts frequency range) hdf5 or fil files to new file.

optional arguments:

-h, --help show this help message and exit

-f IN_FNAME, --input_filename IN_FNAME Name of file to write from (HDF5 or FIL)

-b F_START Start frequency in MHz

-e F_STOP Stop frequency in MHz

-x OUT_FORMAT, --output_file OUT_FORMAT Output file format [.h5 or .fil].

-o OUT_FNAME, --output_filename OUT_FNAME Ouput file name to write (to HDF5 or FIL).

-l MAX_LOAD Maximum data limit to load.

4.1.5 blimpy.fil2h5 module

Simple script for making an h5 file from a .fil.

..author: Emilio Enriquez (jeenriquez@gmail.com)

July 28th 2017

`blimpy.fil2h5.`**`cmd_tool`**(*args=None*)

Command line utility for converting Sigproc filterbank (.fil) to HDF5 (.h5) format

Usage: fil2h5 <FULL_PATH_TO_FIL_FILE> [options]

Options:

-h, --help show this help message and exit

-o OUT_DIR, --out_dir=OUT_DIR Location for output files. Default: local dir.

-n NEW_FILENAME, --new_filename=NEW_FILENAME New name. Default: replaces extention to .h5

-d, --delete_input This option deletes the input file after conversion.

-l MAX_LOAD Maximum data limit to load. Default:1GB

`blimpy.fil2h5.`**`make_h5_file`**(*filename*, *out_dir='./'*, *new_filename=None*, *t_start=None*, *t_stop=None*)

Converts file to HDF5 (.h5) format. Default saves output in current dir.

Parameters

- **`filename`** (*`str`*) – Name of filterbank file to read
- **`out_dir`** (*`str`*) – Output directory path. Defaults to cwd

- **new_filename** (*None or str*) – Name of output filename. If not set, will default to same as input, but with .h5 instead of .fil
- **t_start** (*int*) – Start integration ID to be extracted from file
- **t_stop** (*int*) – Stop integration ID to be extracted from file

4.1.6 blimpy.guppi module

guppi.py

A python file handler for guppi RAW files from the GBT.

The guppi raw format consists of a FITS-like header, followed by a block of data, and repeated over and over until the end of the file.

exception blimpy.guppi.**EndOfFileError**

Bases: Exception

class blimpy.guppi.**GuppiRaw**(*filename*, *n_blocks=None*)

Bases: object

Python class for reading Guppi raw files

Parameters **filename** (*str*) – name of the .raw file to open

Optional args:

n_blocks (int): if number of blocks to read is known, set it here. This saves seeking through the file to check how many integrations there are in the file.

find_n_data_blocks()

Seek through the file to find how many data blocks there are in the file

Returns number of data blocks in the file

Return type n_blocks (int)

generate_filterbank_header(*nchans=1*)

Generate a blimpy header dictionary

This function is useful for generating a default header so the raw data can be saved into a filterbank file.

Parameters **nchans** (*int*) – Number of fine channels in filterbank header.

TODO: Deprecate or move to sigproc.py?

generator_read_next_data_block_int8()

Read the next block of data and its header

Returns: (header, data) header (dict): dictionary of header metadata data (np.array): Numpy array of data, converted into to complex64.

get_data()

returns a generator object that reads data a block at a time; the generator prints "File depleted" and returns nothing when all data in the file has been read. :return:

plot_histogram(*filename=None*, *flag_show=True*)

Plot a histogram of data values

Parameters **filename** (*str*) – Name out output filename. If not set, file will not be saved to disk.

TODO: Move into plotting/

plot_spectrum(*filename=None*, *plot_db=True*, *flag_show=True*)
Do a (slow) numpy FFT and take power of data

Parameters

- **filename** (*str*) – Name of output filename. If not set, file will not be saved to disk.
- **plot_db** (*bool*) – If True, will plot in dB scale, otherwise linear.

TODO: Move into plotting/

print_stats()
Compute some basic stats on the next block of data

read_first_header()
Read first header in file

Returns keyword:value pairs of header metadata

Return type header (dict)

read_header()
Read next header (multiple headers in file)

Returns value header data and also the byte index of where the corresponding data block resides.

Return type (header, data_idx) - a dictionary of keyword

read_next_data_block()
Read the next block of data and its header

Returns: (header, data) header (dict): dictionary of header metadata data (np.array): Numpy array of data, converted into to complex64.

read_next_data_block_int8()
Instantiates a new generator as self.data_gen if there wasn't one already Calls next() on the generator once and returns the value Handles generator depletion :return: header, data_x, data_y

read_next_data_block_int8_2x()
Read the next block of data and its header

Returns: (header, data) header (dict): dictionary of header metadata data (np.array): Numpy array of data, converted into to complex64.

TODO: Deprecate?

read_next_data_block_shape(*header=None*)
Calculate the shape of the next data block. Use provided header instead of reading header if provided.

Parameters

- **header** (*dict*) – value pairs of header metadata read from
- **block** (*current*) –

Returns dshape (tuple) - shape of the corresponding data block

reset_index()
Return file_obj seek to start of file

blimpy.guppi.**cmd_tool**(*args=None*)
Command line tool for plotting and viewing info on GUPPI Raw files

4.1.7 blimpy.h5diag module

h5diag

`blimpy.h5diag.`**`cmd_tool`**(*args=None*)
Command line tool h5diag

`blimpy.h5diag.`**`examine`**(*filename*)
Diagnose the given HDF5 file

`blimpy.h5diag.`**`oops`**(*msg*)

`blimpy.h5diag.`**`read_header`**(*h5*)
Read header and return a Python dictionary of key:value pairs

4.1.8 blimpy.h52fil module

Simple script for making a .fil file from a .h5.

..author: Emilio Enriquez (jeenriquez@gmail.com)

July 28th 2017

`blimpy.h52fil.`**`cmd_tool`**(*args=None*)
Command line utility for converting HDF5 (.h5) to Sigproc filterbank (.fil) format

Usage: h52fil <FULL_PATH_TO_FIL_FILE> [options]

Options:

-h, --help	show this help message and exit
-o OUT_DIR, --out_dir=OUT_DIR	Location for output files. Default: local dir.
-n NEW_FILENAME, --new_filename=NEW_FILENAME	New filename. Default: replaces extension to .fil
-d, --delete_input	This option deletes the input file after conversion.
-l MAX_LOAD	Maximum data limit to load. Default:1GB

`blimpy.h52fil.`**`make_fil_file`**(*filename*, *out_dir='./'*, *new_filename=None*, *max_load=None*)
Converts file to Sigproc filterbank (.fil) format. Default saves output in current dir.

4.1.9 blimpy.match_fils module

4.1.10 blimpy.rawhdr module

Read the specified raw file. Examine & print the required fields. If verbose, print every header field value.

`blimpy.rawhdr.`**`check_float_field`**(*header*, *key*)
Check a float header field for validity.

Parameters

- **`header`** (*`dict`*) – Header of the .raw file.
- **`key`** (*`str`*) – Field's key value.

Returns 0 : valid value; 1 : invalid.

Return type int

`blimpy.rawhdr.`**`check_int_field`**(*header*, *key*, *valid_values*, *required=True*)
Check an integer header field for validity.

Parameters

- **header** (*dict*) – Header of the .raw file.
- **key** (*str*) – Field's key value.
- **valid_values** (*tuple*) – The list of valid values or None.
- **required** (*boolean, optional*) – Required? The default is True.

Returns 0 : valid value; 1 : invalid or missing (and required).

Return type int

`blimpy.rawhdr.`**`cmd_tool`**(*args=None*)
rawhdr command line entry point

Parameters **args** (*ArgParse, optional*) – Command line arguments. The default is None.

Returns **rc** – 0 : no errors; n>0 : at least one error.

Return type int

`blimpy.rawhdr.`**`examine_header`**(*filepath*)
Examine the critical .raw file header fields.

Parameters **filepath** (*str*) – Input .raw file path.

Returns **rc** – 0 : no errors; n>0 : at least one error.

Return type int

4.1.11 blimpy.stax module

Make waterfall plots of a file set, view from top to bottom.

`blimpy.stax.`**`ck_gt_bdry`**(*x*, *bdry*)

`blimpy.stax.`**`ck_lt_bdry`**(*x*, *bdry*)

`blimpy.stax.`**`cmd_tool`**(*args=None*)
Coomand line parser

`blimpy.stax.`**`make_waterfall_plots`**(*file_list*, *plot_dir*, *plot_dpi*, *height_ratios*, *f_start=None*, *f_stop=None*, ***kwargs*)
Make waterfall plots of a file set, view from top to bottom.

Parameters

- **file_list** (*list*) – List of filterbank file paths to plot in a stacked mode.
- **plot_dir** (*str*) – Path of where to store the output plot file (png).
- **plot_dpi** (*int*) – Number of dots per inch for the plots.
- **height_ratios** (*list*) – A list whose elements are the observation length for each file in order indicated by parameter file_list.
- **f_start** (*float*) – Start frequency, in MHz.
- **f_stop** (*float*) – Stop frequency, in MHz.
- **kwargs** (*dict*) – Keyword args to be passed to matplotlib imshow().

`blimpy.stax.`**`plot_waterfall`**(*wf*, *f_start=None*, *f_stop=None*, ***kwargs*)

Plot waterfall of data in a .fil or .h5 file.

Parameters

- **wf** (*blimpy.Waterfall object*) – Waterfall object of an H5 or Filterbank file containing the dynamic spectrum data.
- **f_start** (*float*) – Start frequency, in MHz.
- **f_stop** (*float*) – Stop frequency, in MHz.
- **kwargs** (*dict*) – Keyword args to be passed to matplotlib imshow().

Notes

Plot a single-panel waterfall plot (frequency vs. time vs. intensity) for one of the files in the set of interest, at the frequency of the expected event.

`blimpy.stax.`**`sort2`**(*x*, *y*)

Return lowest value, highest value

4.1.12 blimpy.stix module

Make waterfall plots of a large file.

`blimpy.stix.`**`cmd_tool`**(*args=None*)

Coomand line parser

`blimpy.stix.`**`image_stitch`**(*orientation*, *chunk_count*, *png_collection*, *path_saved_png*)

Stitch together multiple PNGs into one

Parameters

- **orientation** (*str*) – Assembling images horizontally (h) or vertically (v)?
- **chunk_count** (*int*) – Number of chunks in the file.
- **png_collection** (*list*) – The set of PNG file paths whose images are to be stitched together.
- **path_saved_png** (*str*) – The path of where to save the final PNG file.

`blimpy.stix.`**`make_waterfall_plots`**(*input_file*, *chunk_count*, *plot_dir*, *width*, *height*, *dpi*, *source_name=None*)

Make waterfall plots of a given huge-ish file.

input_file [str] Path of Filterbank or HDF5 input file to plot in a stacked mode.

chunk_count [int] The number of chunks to divide the entire bandwidth into.

plot_dir [str] Directory for storing the PNG files.

width [float] Plot width in inches.

height [float] Plot height in inches.

dpi [int] Plot dots per inch.

source_name [str] Source name from the file header.

`blimpy.stix.sort2`(*x*, *y*)
Return lowest value, highest value

4.1.13 blimpy.utils module

utils.py useful helper functions for common data manipulation tasks

`blimpy.utils.change_the_ext`(*path*, *old_ext*, *new_ext*)
Change the file extension of the given path to new_ext.

If the file path's current extension matches the old_ext, then the new_ext will replace the old_ext. Else, the new_ext will be appended to the argument path.

In either case, the resulting string is returned to caller.

E.g. /a/b/fil/d/foo.fil.bar.fil –> /a/b/fil/d/foo.fil.bar.h5 E.g. /a/fil/b/foo.bar –> /a/fil/b/foo.bar.h5 E.g. /a/fil/b/foo –> /a/fil/b/foo.h5

Parameters

- **path** (*str*) – Path of file to change the file extension..
- **old_ext** (*str*) – Old file extension (E.g. h5, fil, dat, log).
- **new_ext** (*str*) – New file extension (E.g. h5, fil, dat, log).

Returns

Return type New file path, amended as described.

`blimpy.utils.closest`(*xarr*, *val*)
Return the index of the closest in xarr to value val

`blimpy.utils.db`(*x*, *offset=0*)
Convert linear to dB

`blimpy.utils.lin`(*x*)
Convert dB to linear

`blimpy.utils.rebin`(*d*, *n_x=None*, *n_y=None*, *n_z=None*)
Rebin data by averaging bins together

Args: d (np.array): data n_x (int): number of bins in x dir to rebin into one n_y (int): number of bins in y dir to rebin into one

Returns: d: rebinned data with shape (n_x, n_y)

`blimpy.utils.unpack`(*data*, *nbit*)
upgrade data from nbits to 8bits

Notes: Pretty sure this function is a little broken!

`blimpy.utils.unpack_1to8`(*data*)
Promote 1-bit unisgned data into 8-bit unsigned data.

Parameters **data** – Numpy array with dtype == uint8

`blimpy.utils.unpack_2to8`(*data*)
Promote 2-bit unisgned data into 8-bit unsigned data.

Parameters **data** – Numpy array with dtype == uint8

Notes

DATA MUST BE LOADED as np.array() with dtype='uint8'.

This works with some clever shifting and AND / OR operations. Data is LOADED as 8-bit, then promoted to 32-bits: /ABCD EFGH/ (8 bits of data) /0000 0000/0000 0000/0000 0000/ABCD EFGH/ (8 bits of data as a 32-bit word)

Once promoted, we can do some shifting, AND and OR operations: /0000 0000/0000 ABCD/EFGH 0000/0000 0000/ (shifted << 12) /0000 0000/0000 ABCD/EFGH 0000/ABCD EFGH/ (bitwise OR of previous two lines) /0000 0000/0000 ABCD/0000 0000/0000 EFGH/ (bitwise AND with mask 0xF000F) /0000 00AB/CD00 0000/0000 00EF/GH00 0000/ (prev. line shifted << 6) /0000 00AB/CD00 ABCD/0000 00EF/GH00 EFGH/ (bitwise OR of previous two lines) /0000 00AB/0000 00CD/0000 00EF/0000 00GH/ (bitwise AND with 0x3030303)

Then we change the view of the data to interpret it as 4x8 bit: [000000AB, 000000CD, 000000EF, 000000GH] (change view from 32-bit to 4x8-bit)

The converted bits are then mapped to values in the range [-40, 40] according to a lookup chart. The mapping is based on specifications in the breakthough docs: https://github.com/UCBerkeleySETI/breakthrough/blob/master/doc/RAW-File-Format.md

`blimpy.utils.`**`unpack_4to8`**(*data*)

Promote 2-bit unisgned data into 8-bit unsigned data.

Parameters **`data`** – Numpy array with dtype == uint8

Notes

The process is this: # ABCDEFGH [Bits of one 4+4-bit value] # 00000000ABCDEFGH [astype(uint16)] # 0000ABCDEFGH0000 [<< 4] # 0000ABCDXXXXEFGH [bitwise 'or' of previous two lines] # 0000111100001111 [0x0F0F] # 0000ABCD0000EFGH [bitwise 'and' of previous two lines] # ABCD0000EFGH0000 [<< 4] # which effectively pads the two 4-bit values with zeros on the right # Note: This technique assumes LSB-first ordering

4.1.14 blimpy.waterfall module

waterfall.py

Python class and command line utility for reading and plotting waterfall files.

This provides a class, Waterfall(), which can be used to read a blimpy file (.fil or .h5):

fil = Waterfall("test_psr.fil") print(fil.header) print(fil.data.shape) print(fil.freqs)

plt.figure() fil.plot_spectrum(t=0) plt.show()

`class` `blimpy.waterfall.`**`Waterfall`**(*filename=None*, *f_start=None*, *f_stop=None*, *t_start=None*, *t_stop=None*, *load_data=True*, *max_load=None*, *header_dict=None*, *data_array=None*)

Bases: `object`

Class for loading and writing blimpy data (.fil, .h5)

`blank_dc`(*n_coarse_chan*)

Blank DC bins in coarse channels.

Removes the DC spike in centre of coarse channel bins.

Note: currently only works if entire file is read

calc_n_coarse_chan (*chan_bw=None*)
This makes an attempt to calculate the number of coarse channels in a given freq selection.

Note: This is unlikely to work on non-Breakthrough Listen data, as a-priori knowledge of the digitizer system is required.

Returns n_coarse_chan (int), number of coarse channels

calibrate_band_pass_N1 ()
One way to calibrate the band pass is to take the median value for every frequency fine channel, and divide by it.

sets data = data / band_pass

get_freqs ()
Get the frequency array for this Waterfall object.

Returns Values for all of the fine frequency channels.

Return type numpy array

grab_data (*f_start=None*, *f_stop=None*, *t_start=None*, *t_stop=None*, *if_id=0*)
Extract a portion of data by frequency range.

Parameters

- **f_start** (*float*) – start frequency in MHz
- **f_stop** (*float*) – stop frequency in MHz
- **if_id** (*int*) – IF input identification (req. when multiple IFs in file)

Returns frequency axis in MHz and data subset

Return type (freqs, data) (np.arrays)

info ()
Print header information and other derived information.

read_data (*f_start=None*, *f_stop=None*, *t_start=None*, *t_stop=None*)
Reads data selection if small enough.

Parameters

- **f_start** (*float*) – Start frequency in MHz
- **f_stop** (*float*) – Stop frequency in MHz
- **t_start** (*int*) – Integer time index to start at
- **t_stop** (*int*) – Integer time index to stop at

Data is loaded into self.data (nothing is returned)

write_to_fil (*filename_out*, **args*, ***kwargs*)

write_to_hdf5 (*filename_out*, **args*, ***kwargs*)

blimpy.waterfall.**cmd_tool** (*args=None*)
Command line tool for plotting and viewing info on blimpy files

4.1.15 blimpy.io.sigproc module

`blimpy.io.sigproc.`**`calc_n_ints_in_file`**(*filename*)
Calculate number of integrations in a given file

`blimpy.io.sigproc.`**`fil_double_to_angle`**(*angle*)
Reads a little-endian double in ddmmss.s (or hhmmss.s) format and then converts to Float degrees (or hours). This is primarily used to read src_raj and src_dej header values.

`blimpy.io.sigproc.`**`fix_header`**(*filename*, *keyword*, *new_value*)
Apply a quick patch-up to a Filterbank header by overwriting a header value

Parameters

- **`filename`** (*`str`*) – name of file to open and fix. WILL BE MODIFIED.
- **`keyword`** (*`stt`*) – header keyword to update
- **`new_value`** (*`long, double, angle or string`*) – New value to write.

Notes

This will overwrite the current value of the blimpy with a desired 'fixed' version. Note that this has limited support for patching string-type values - if the length of the string changes, all hell will break loose.

`blimpy.io.sigproc.`**`generate_sigproc_header`**(*f*)
Generate a serialzed sigproc header which can be written to disk.

Parameters **`f`** (*`Filterbank object`*) – Filterbank object for which to generate header

Returns Serialized string corresponding to header

Return type header_str (str)

`blimpy.io.sigproc.`**`is_filterbank`**(*filename*)
Open file and confirm if it is a filterbank file or not.

`blimpy.io.sigproc.`**`len_header`**(*filename*)
Return the length of the blimpy header, in bytes

Parameters **`filename`** (*`str`*) – name of file to open

Returns length of header, in bytes

Return type idx_end (int)

`blimpy.io.sigproc.`**`read_header`**(*filename*, *return_idxs=False*)
Read blimpy header and return a Python dictionary of key:value pairs

Parameters **`filename`** (*`str`*) – name of file to open

Optional args:

return_idxs (bool): Default False. If true, returns the file offset indexes for values

returns

`blimpy.io.sigproc.`**`read_next_header_keyword`**(*fh*)

Parameters **`fh`** (*`file`*) – file handler

Returns:

`blimpy.io.sigproc.`**`to_sigproc_angle`**(*angle_val*)

Convert an astropy.Angle to the ridiculous sigproc angle format string.

`blimpy.io.sigproc.`**`to_sigproc_keyword`**(*keyword*, *value=None*)

Generate a serialized string for a sigproc keyword:value pair

If value=None, just the keyword will be written with no payload. Data type is inferred by keyword name (via a lookup table)

Parameters

- **`keyword`** (*`str`*) – Keyword to write
- **`value`** (*`None, float, str, double or angle`*) – value to write to file

Returns serialized string to write to file.

Return type value_str (str)

4.1.16 Module contents

Python Module Index

b

Index

G

I

J

L

M

O

P

R

S

T

U

W