filtools Documentation

Release 2019.04.1

Michael Keith

Oct 19, 2021

User Guide

CHAPTER 1

Introduction

Filtools is a collection of tools for working with pulsar filterbank data. The intent is to provide a collection of useful library routines and programs for solving tricky problems when working with filterbank data. Initially this will focus on sigproc format filtebank files, but may extend to presto and psrfits format files in future.

As of writing, this is only just starting and contains two useful components: the *filtools.FilterbankIO* class for reading and writing filtebank data, and a series of routines and software for injecting simulated pulsar signals into filterbank files.

CHAPTER 2

Installation

filtools can be installed using pip:

```
pip install git+https://bitbucket.org/mkeith/filtools
```

Or directly by downloading the git repository:

```
git clone https://bitbucket.org/mkeith/filtools
cd filtools
python setup.py install
```

The git repository can be viewed at https://bitbucket.org/mkeith/filtools

2.1 Requirements

- Python 2.7 or 3.6+
- numpy
- yaml

Some routines can also make use of tempo2 predictors by installing the t2pred python library from tempo2.

CHAPTER 3

Program Manuals

Under development

3.1 ft_inject_pulsar

```
usage: ft_inject_pulsar [-h] -o OUTPUT [-p PULSE] [-S FLUX] [-i SIDX]
                        [--sidx-reffreq SIDX_REFFREQ] [-E PULSE_SIGMA]
                        [--pred PRED] [--f0 F0] [--f1 F1] [--f2 F2]
                        [--accn ACCN] [--pepoch PEPOCH] [-D DM] [-x]
                        [-c SCATTER_TIME] [-X SCATTER_INDEX]
                        [--scatter-reffreq SCATTER_REFFREQ] [-T SITE]
                        [--gain GAIN] [--tsys TSYS] [--rms RMS]
                        [--offset OFFSET]
                        input

Inject a pulsar signal into a filterbank file

positional arguments:
  input                 Input filterbank file

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Output file name

Pulse profile settings:
  -p PULSE, --pulse PULSE
                        Pulse model type (sin, delta), or profile file to load
  -S FLUX, --flux FLUX  Phase averatge flux density (Jy)
  -i SIDX, --sidx SIDX  Spectral index
  --sidx-reffreq SIDX_REFFREQ
                        Spectral index reference frequency (1400 MHz)
  -E PULSE_SIGMA, --pulse-sigma PULSE_SIGMA
```

(continues on next page)

(continued from previous page)

```
                        'sigma' for log-normal pulse intensity distribution

Pulse period settings:
  --pred PRED           Predictor or polyco for pulsar
  --f0 F0               Pulse frequency for simple spin model
  --f1 F1               Pulse freq 1st derivative for simple spin model
  --f2 F2               Pulse freq 2nd derivative for simple spin model
  --accn ACCN           Pulse accelerationfor simple spin model (overrides F1)
  --pepoch PEPOCH       Epoch of somple spin model (default=centre)

Propagtion settings:
  -D DM, --dm DM        DM of injected pulsar
  -x, --no-intra-chan-dm
                        Disable intra-channel DM
  -c SCATTER_TIME, --scatter-time SCATTER_TIME
                        Scattering timescale
  -X SCATTER_INDEX, --scatter-index SCATTER_INDEX
                        Scattering index
  --scatter-reffreq SCATTER_REFFREQ
                        Scattering index reference frequency (default=1400MHz)

Telescope settings:
  -T SITE, --site SITE  Telescope description to use.
  --gain GAIN           Simulated telescope gain (K/Jy)
  --tsys TSYS           Simulated telescope tsys (K)
  --rms RMS             RMS in input data (default=auto)
  --offset OFFSET       Offset in input data default=auto)
usage: ft_inject_pulsar [-h] -o OUTPUT [-p PULSE] [-S FLUX] [-i SIDX]
                        [--sidx-reffreq SIDX_REFFREQ] [-E PULSE_SIGMA]
                        [--pred PRED] [--precision-pred] [--f0 F0] [--f1 F1]
                        [--f2 F2] [--accn ACCN] [--pepoch PEPOCH] [-D DM] [-x]
                        [-c SCATTER_TIME] [-X SCATTER_INDEX]
                        [--scatter-reffreq SCATTER_REFFREQ] [-T SITE]
                        [--gain GAIN] [--tsys TSYS] [--rms RMS]
                        [--offset OFFSET]
                        input

Inject a pulsar signal into a filterbank file

positional arguments:
  input                 Input filterbank file

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Output file name

Pulse profile settings:
  -p PULSE, --pulse PULSE
                        Pulse model type (sin, delta), or profile file to load
  -S FLUX, --flux FLUX  Phase averatge flux density (Jy)
  -i SIDX, --sidx SIDX  Spectral index
  --sidx-reffreq SIDX_REFFREQ
                        Spectral index reference frequency (1400 MHz)
  -E PULSE_SIGMA, --pulse-sigma PULSE_SIGMA
                        'sigma' for log-normal pulse intensity distribution
```

(continues on next page)

(continued from previous page)

```
Pulse period settings:
  --pred PRED           Predictor or polyco for pulsar
  --precision-pred      Call t2pred for every phase computation - slow
  --f0 F0               Pulse frequency for simple spin model
  --f1 F1               Pulse freq 1st derivative for simple spin model
  --f2 F2               Pulse freq 2nd derivative for simple spin model
  --accn ACCN           Pulse accelerationfor simple spin model (overrides F1)
  --pepoch PEPOCH       Epoch of somple spin model (default=centre)

Propagtion settings:
  -D DM, --dm DM        DM of injected pulsar
  -x, --no-intra-chan-dm
                        Disable intra-channel DM
  -c SCATTER_TIME, --scatter-time SCATTER_TIME
                        Scattering timescale
  -X SCATTER_INDEX, --scatter-index SCATTER_INDEX
                        Scattering index
  --scatter-reffreq SCATTER_REFFREQ
                        Scattering index reference frequency (default=1400MHz)

Telescope settings:
  -T SITE, --site SITE  Telescope description to use.
  --gain GAIN           Simulated telescope gain (K/Jy)
  --tsys TSYS           Simulated telescope tsys (K)
  --rms RMS             RMS in input data (default=auto)
  --offset OFFSET       Offset in input data default=auto)
```

3.2 ft_scrunch

```
usage: ft_scrunch [-h] [-b BITS] [-N NORMALISE] [-c CHANS] [-s SAMPLES]
                  [-F FACTOR] [-D DC] [-O OFFSET] [-d DM] [-B BLOCKSIZE]
                  input output

'Scrunch' channels and/or samples in the input filterbank.
Output is scaled acording to the normalisation method and the supplied
scale factor. If scrunching by 's' samples and 'c' channels, with a
scale factor 'F', the result is

out[i] = sum(in[i:i+s][i:i+c]) * F / g + o

where:

g = c*s       for 'mean' normalisation
g = sqrt(c*s) for 'sqrt' normalisation
g = 1.0       for 'sum'  normalisation

and 'o' is a term to remove the addition of the DC offset, defined by

o = D - c*s*D*F/g

Note that o=0 if D=0, or in the case of mean normalisation (with F=1).

Generally mean normalisation is best to prevent overflowing small data
types, but 'sqrt' may better preserve weaker signals.
```

(continues on next page)

(continued from previous page)

```
The values s,c,F,D can be set by the -s, -c, -F and -D options respectively.

This can also be useful to simply re-scale the data with c=1 and s=1.

If reducing number of bits, the output DC offset can also be set using the -O
option. E.g. to convert to 8-bit data to 2-bit data, one might try

 $ ft_scrunch -F 0.02 -b 2 -O 1.5 8bit.fil 2bit.fil

to write 2-bit data with the mean at 1.5 and scaling by a factor of 0.02

positional arguments:
  input                 Input filterbank file
  output                Output filterbank file

optional arguments:
  -h, --help            show this help message and exit
  -b BITS, --bits BITS  Output number of bits (default=same as input)
  -N NORMALISE, --normalise NORMALISE
                        Normalisation method (default=mean)
  -c CHANS, --chans CHANS
                        Number of adjacent channels to scrunch (default=1)
  -s SAMPLES, --samples SAMPLES
                        Number of adjacent samples to scrunch (default=1)
  -F FACTOR, --factor FACTOR
                        Scale factor (default=1)
  -D DC, --dc DC        DC offset (default=estimate)
  -O OFFSET, --offset OFFSET
                        Output DC offset (default=input)
  -d DM, --dm DM        Remove DM before scrunching channels (default=don't)
  -B BLOCKSIZE, --blocksize BLOCKSIZE
                        Set read block size
```

3.3 ft_scrunch_threads

```
usage: ft_scrunch_threads [-h] [-b BITS] [-N NORMALISE] [-c CHANS]
                          [-s SAMPLES] [-F FACTOR] [-D DC] [-O OFFSET] [-d DM]
                          [-B BLOCKSIZE] [-t THREADS]
                          [--thread-chunk-size THREAD_CHUNK_SIZE] [--no-refdm]
                          input output

'Scrunch' channels and/or samples in the input filterbank.
Output is scaled acording to the normalisation method and the supplied
scale factor. If scrunching by 's' samples and 'c' channels, with a
scale factor 'F', the result is

out[i] = sum(in[i:i+s][i:i+c]) * F / g + o

where:

g = c*s       for 'mean' normalisation
g = sqrt(c*s) for 'sqrt' normalisation
g = 1.0       for 'sum'  normalisation
```

(continues on next page)

(continued from previous page)

```
and 'o' is a term to remove the addition of the DC offset, defined by

o = D - c*s*D*F/g

Note that o=0 if D=0, or in the case of mean normalisation (with F=1).

Generally mean normalisation is best to prevent overflowing small data
types, but 'sqrt' may better preserve weaker signals.

The values s,c,F,D can be set by the -s, -c, -F and -D options respectively.

This can also be useful to simply re-scale the data with c=1 and s=1.

If reducing number of bits, the output DC offset can also be set using the -O
option. E.g. to convert to 8-bit data to 2-bit data, one might try

 $ ft_scrunch_threads -F 0.02 -b 2 -O 1.5 8bit.fil 2bit.fil

to write 2-bit data with the mean at 1.5 and scaling by a factor of 0.02

** This version is optimised for multi-threaded output **

positional arguments:
  input                 Input filterbank file
  output                Output filterbank file

optional arguments:
  -h, --help            show this help message and exit
  -b BITS, --bits BITS  Output number of bits (default=same as input)
  -N NORMALISE, --normalise NORMALISE
                        Normalisation method (default=mean)
  -c CHANS, --chans CHANS
                        Number of adjacent channels to scrunch (default=1)
  -s SAMPLES, --samples SAMPLES
                        Number of adjacent samples to scrunch (default=1)
  -F FACTOR, --factor FACTOR
                        Scale factor (default=1)
  -D DC, --dc DC        DC offset (default=estimate)
  -O OFFSET, --offset OFFSET
                        Output DC offset (default=input)
  -d DM, --dm DM        Remove DM before scrunching channels (default=don't)
  -B BLOCKSIZE, --blocksize BLOCKSIZE
                        Set read block size
  -t THREADS, --threads THREADS
                        Number of threads
  --thread-chunk-size THREAD_CHUNK_SIZE
                        Number of blocks to process per thread
  --no-refdm            Don't set the REFDM header field when dedispersing
```

3.4 ft_convert_signed_to_unsigned

```
usage: ft_convert_signed_to_unsigned [-h] [-S] input output
```

(continues on next page)

(continued from previous page)

```
Fix a filterbank style file that is encoded as signed data to unsigned data.
This program is intended to fix cases where a filterbank file has been written
by some external program as *signed* 8-bit values rather than the typical
unsigned 8-bit values. The new file is written as unsigned.

positional arguments:
  input            Input filterbank file, containing signed 8-bit numbers
  output           Output filterbank file, writen as unsigned 8-bit numbers

optional arguments:
  -h, --help       show this help message and exit
  -S, --no-signed  Don't write an UNSIGNED entry in the output file
```

3.5 ft_freqstack

```
usage: ft_freqstack [-h] [--output OUTPUT] [-B BLOCKSIZE] input [input ...]

Stack filterbank files in the frequency direction.

The output file will start at the latest start time of the input files and end at the␣
↪earliest end time.

The output file will be padded with zeros for channels not included in the input␣
↪files.

Note that behaviour is not defined if the input data overlaps or the channel␣
↪bandwidths vary between files!

positional arguments:
  input                 Input filterbank files

optional arguments:
  -h, --help            show this help message and exit
  --output OUTPUT, -o OUTPUT
                        Output filterbank file
  -B BLOCKSIZE, --blocksize BLOCKSIZE
                        Set read block size
```

CHAPTER 4

filtools package

class filtools.**FilterbankIO**

Bases: object

This class deals with reading from and writing to sigproc filterbank files.

To read a filterbank file, first set up a new *FilterbankIO* object, and use the *read_header()* method to read the header parameters. The FilterbankIO object is then ready to be used for reading. To read the file, we use the *read_block()* method to read samples from the file.

Example:

```
inputfile = filtools.FilterbankIO()
inputfile.read_header("example.fil")
data = inputfile.read_block(1024) # Reads 1024 samples from the file
```

Note:

Currently *FilterbankIO* supports sigproc data files with the following properties:

- Single IF (i.e. single polarisation)
- Fixed channel bandwidth (i.e. each channel has same bandwidth/offset)

And, data type is one of:

- 1-bit unsigned integer
- 2-bit unsigned integer
- 4-bit unsigned integer
- 8-bit unsigned integer
- 16-bit unsigned integer
- 32-bit floating point

---

header
The header dict can be used to provide direct access to the sigproc header parameters. Parameters changed here before calling *write_header()* can change the ouput header parameters. However, it is usually best to access information about the data using the other methods provided below, as the structure can be re-used for other data types.

bits_per_sample()
Number of bits per sample in data file

centre_frequency()
The centre frequency

channel_bandwidth()
The channel bandwidth.

clone_header()
Create a new *FilterbankIO* object with the header parameters copied from this object

Returns A new *FilterbankIO* object

close()
Close the input file, reset the read/write state of the *FilterbankIO* object

current_position()
Print the current position of the file pointer in samples.

The next call to *read_block()* or *write_block()* will read or write from this sample.

frequency_table()
Get an array of centre frequency of each channel

get_reader()
This method returns a function optimised to read the input data.‘

get_writer()
Returns an optimised write function. Used by *write_block()*.

mjd_at_sample(*sample*)
Get the MJD at given sample number

read_block(*nsamples*, *dtype=<class 'numpy.float32'>*)
Read a block of data.

This will read up to `nsamples` from the file, returning a 2-d array indexed first by sample and then by channel. i.e. the shape of the returned array will be (nsamples_read,nchannels).

If the file has reached the end of file, then less than nsamples will be read, and if past the end of the file, an array of size (0,nchannels) will be returned.

Note: On first call this method is dynamically replaced using *get_reader()* to define an optimised verison of the reader method. This should be invisiable to most users.

Parameters

- **nsamples** – The number of samples to read.
- **dtype** – A numpy compatible data type. Data will be converted to this type using the astype numpy method, and should be at least large enough to hold the read data types.

Returns A 2-d numpy array of requested type.

read_header(*filename*)
Read header from sigproc filterbank file

Parameters **filename** (*str*) – filename to read

Returns Number of bytes read (i.e. size of header)

This method will also put the *FilterbankIO* object in a state for reading the file, i.e. enabling the *read_block()* method.

Header parameters can be accessed by the methods of the *FilterbankIO* object, or by direct interface with the sigproc API using the *header* dict member.

reference_dm()

sample_interval()
The time between each sample

seconds_since(*sample*, *mjdref*)
Get the number of seconds since `mjdref` at sample number `sample`.

Parameters

- **sample** – Integer sample number
- **refmjd** – Reference mjd

Returns seconds since `mjdref` at sample `sample`

seconds_since_start(*sample*)
Get the number of seconds since start of observation at sample number `sample`.

seek_to_sample(*sample*)
Position the file pointer at given sample number (index from zero).

The next call to *read_block()* or *write_block()* will start at the given sample. E.g.

```
fil.seek_to_sample(100)
block = fil.read_block(128) # block[0] is sample number 100
```

set_bits_per_sample(**args*, ***kwargs*)

set_frequency_table(**args*, ***kwargs*)

set_reference_dm(**args*, ***kwargs*)

set_sample_interval(**args*, ***kwargs*)

set_total_channels(**args*, ***kwargs*)

total_bytes()
Compute the total number of bytes in the file

total_channels()
The total number of frequency channels in the file

total_samples()
The total number of samples in the file

write_block(*block*)
Write a block of data to a file opened for writing.

The data block should be a numpy array in the same format as would be obtained from *read_block()*. The data will be truncated and cast to the required type to write to the file using the ndarray.astype method from numpy.

The block should have shape (nsamples,nchannels), where nsamples is the number of samples to write.

Parameters **block** – A 2-d numpy array to write.

Note: On first call this this method is dynamically replaced using *get_writer()* to define an optimised verison of the writer method. This should be invisiable to most users.

---

write_header(*filename*)
Write the current state to a new sigproc filterbank file

Parameters **filename** – filename to write

Returns Number of bytes written (i.e. size of header)

This method will also put the *FilterbankIO* object in a state for writing data to the file, i.e. enabling the *write_block()* method.

class filtools.**FluxScale**(*scale=1*, *offset=0*)
Bases: object

Helps scaling flux density data to the data format.

This is used either by directly specifying a scale and offset to apply, or by setting a gain and Tsys value, or by specifying physical telescope parameters to define the gain and Tsys.

init(*tsamp*, *bw*, *npol_avg=2*)
Call this once you have set the gain and tsys to set up the scaling.

Parameters

- **tsamp** – sample time in seconds.
- **bw** – channel bandwidth in MHz.
- **npol_avg** – Number of polarisations that have been averaged already (1 or 2).

set_gain_from_file(*file*)
Read a yaml file with a telescope definition. Properties are: tsys, n_antenna, diameter, efficiency and/or gain.

set_gain_from_physical(*ndish*, *D*, *efficiency*)
Specify physical parameters to determine gain.

Parameters

- **ndish** – Number of antennas
- **D** – Diameter of each antenna (m).
- **efficiency** – Efficiency of the observing system (antenna efficiency).

to_file_units(*buffer*)
Convert from flux density units to file units

to_flux_units(*buffer*)
Convert from file units to flux density units

class filtools.**SimpleIntegerise**(*thresh=1e-08*)
Bases: object

Converts floats to integers assuming previously digitised data had a uniform underlying distribution.

Parameters **thresh** – Ignore values below this to improve performance (default 1e-8).

__call__(*block*)
Convert a block of floating point data to integers.

4.1 Subpackages

4.1.1 filtools.inject_pulsar package

This package contains classes designed to aid injection of pulsar and other similar signals into filterbank data

4.1.1.1 Submodules

4.1.1.2 filtools.inject_pulsar.ism_models module

class filtools.inject_pulsar.ism_models.**simple_propagation_model**(*freq*, *input*, *dm*, *intra_chan=True*, *dm_const=0.000241*)

Bases: object

Models simple cold-plasma dispersion delays.

Optionally includes smearing due to intra-channel delay. Disable to model coherent dedispersion.

The DM delay is modeled as

$$t_{\mathrm{DM}} = \frac{\nu^{-2}\mathrm{DM}}{k_{\mathrm{DM}}}$$

Parameters

- **freq** – array of centre frequency for each channel in MHz. The algorithm assumes equally spaced channels.
- **input** – Callable or function.
- **dm** – the simulated DM in cm$^-3$pc.
- **intra_chan** – if True, will model the smearing in the pulse due to the intra-channel delays
- **dm_const** – override to change the DM constant (k_{DM}) used to compute the DM delays.

__call__(*t1*, *t2*, *mjdref*)
Compute average flux density between t1 and t2 in each frequency channel.

Output is in flux density units.

Parameters

- **t1** – Start time in seconds, relative to mjdref
- **t2** – End time in seconds, relative to mjdref

Returns Average flux density between t1 and t2.

class filtools.inject_pulsar.ism_models.**simple_scattering_model**(*freq*, *input*, *t_scatt*, *dx*, *scatt_idx=-4.0*, *ref_freq=1400.0*)

Bases: object

Approximates thin-screen scattering in the ionised interstellar medium.

Effectively convolves the incoming signal with an exponential of the form

$$\exp\left(-\frac{t}{t_{\rm scatt}}\left(\frac{\nu_{\rm ref}}{\nu}\right)^{\alpha}\right)$$

Warning: The simple scattering model assumes that each sample to be generated is called in sequence. I.e. no calls to get_spectrum occur out of sequence. This will prevent multi-threaded operation of codes using this module.

Parameters

- **input** – the input signal generator
- **freq** – array of frequency channels in MHz.
- **t_scatt** – the scattering timescale.
- **dx** – the sampling interval in the data file.
- **scatt_idx** – the spectral index of the scattering timescale (α)

__call__(*t1*, *t2*, *mjdref*)

Compute average flux density between t1 and t2 in each frequency channel.

Output is in flux density units.

Warning: The simple scattering model assumes that each sample to be generated is called in sequence. I.e. no calls to get_spectrum occur out of sequence. This will prevent multi-threaded operation of codes using this module.

Parameters

- **t1** – Start time in seconds, relative to mjdref
- **t2** – End time in seconds, relative to mjdref

Returns Average flux density between t1 and t2.

4.1.1.3 filtools.inject_pulsar.phase_models module

class filtools.inject_pulsar.phase_models.**precision_predictor_phase_model**(*predictor*)

Bases: object

A phase model that uses a phase predictor (or polyco) from tempo or tempo2. It uses the t2pred library to convert between time and phase, so can model a wide range of pulsar behaviour. This model calls t2pred on every request so is slow, but very accurate.

Parameters **predictor** (*A t2pred.phase_predictor object*) – The phase predictor to use

Note: This module will only function if you have installed the t2pred python library in tempo2. This can be installed using setup.py in the python/t2pred directory in the tempo2 source code.

get_frequency(*time*, *mjdref*)
Get the pulse frequency at time in seconds relative to refmjd

get_phase(*time*, *mjdref*)
Get the pulse phase at time in seconds relative to refmjd

class filtools.inject_pulsar.phase_models.**predictor_phase_model**(*predictor*, *dt=0.1*)

Bases: object

A phase model that uses a phase predictor (or polyco) from tempo or tempo2. It uses the t2pred library to convert between time and phase, so can model a wide range of pulsar behaviour. This version keeps its own polynomial interpolation of the predictor over short spans to reduce the number of calls to t2pred and greatly increase the performance for a very minor precision reduction.

Parameters

- **predictor** (*t2pred.phase_predictor*) – The phase predictor to use
- **dt** – The interpolation window size in seconds

Note: This module will only function if you have installed the t2pred python library in tempo2. This can be installed using setup.py in the python/t2pred directory in the tempo2 source code.

---

get_frequency(*time*, *mjdref*)
Get the pulse frequency at time in seconds relative to refmjd

get_phase(*time*, *mjdref*)
Get the pulse phase at time in seconds relative to refmjd.

make_poly(*t*)
This routine is used internally to make the polynomial approimation of the timing model. If a new-ish version of scipy is installed it uses scipy.interp.CubicHermiteSpline, or otherwise uses a scipy.interp.BPoly.

class filtools.inject_pulsar.phase_models.**simple_phase_model**(*epoch*, *f0*, *f1=0.0*, *f2=0.0*, *accn=None*)

Bases: object

A very simple model of the rotational phase of a pulsar

This model is that the pulsar is in the rest frame of the observer and follows a polynomial spin where the phase is given by:

$$\phi(t) = f_0 * t + (f_1 * t^2)/2.0 + (f_2 * t^3)/6.0$$

phase is therefore defined as being from 0 to 1, rather than as an angle.

Parameters

- **epoch** – The reference epoch ($t = 0$)
- **f0** – Frequency at t=0
- **f1** – Frequency derivative at t=0
- **f2** – Frequency second derivative at t=0
- **accn** – Doppler acceleration at t=0 (overrides choice of f1).

get_frequency(*time*, *mjdref*)
Get the pulse frequency at time in seconds relative to refmjd

get_phase(*time*, *mjdref*)
Get the pulse phase at time in seconds relative to refmjd

4.1.1.4 filtools.inject_pulsar.pulse_models module

class filtools.inject_pulsar.pulse_models.**delta_pulse_model**(*phase_model*, *freq*)
Bases: object

Models a pulsar as a delta function.

Only has flux at phase zero. Phase average flux density is 1 unit.

Parameters

- **phase_model** – Object that implements the get_phase method.
- **freq** – An array of observing frequnecies (ignored)

__call__(*t1*, *t2*, *mjdref*)
Compute average flux density between t1 and t2 in each frequency channel.

Output is in flux density units.

Parameters

- **t1** – Start time in seconds, relative to mjdref
- **t2** – End time in seconds, relative to mjdref

Returns Average flux density between t1 and t2.

class filtools.inject_pulsar.pulse_models.**piecewise_pulse_model**(*phase_model*, *freq*, *profile*)
Bases: object

Models a pulsar as an arbitrary piecewise pulse shape

Parameters

- **phase_model** – Object that implements the get_phase method.
- **freq** – An array of observing frequnecies (ignored)
- **profile** – An array of pulse intensity as a function of time

__call__(*t1*, *t2*, *mjdref*)
Compute average flux density between t1 and t2 in each frequency channel.

Output is in flux density units.

Parameters

- **t1** – Start time in seconds, relative to mjdref
- **t2** – End time in seconds, relative to mjdref

Returns Average flux density between t1 and t2.

repad(*n*)
Pads the internal array with multiple copies of the pulse profile. Used internally to ensure that the integration can cope with profiles spanning multiple pulse periods.

Parameters **n** – The number of times to pad the profile.

class filtools.inject_pulsar.pulse_models.**sinusoid_pulse_model**(*phase_model*, *freq*)

Bases: object

Models a pulsar as a sinusoid function.

$$I(\phi) = \cos(2\pi\phi) + 1$$

Where ϕ is the phase in turns (i.e. between 0 and 1).

Parameters

- **phase_model** – Object that implements the get_phase method.
- **freq** – An array of observing frequnecies (ignored)

__call__(*t1*, *t2*, *mjdref*)

Compute average flux density between t1 and t2 in each frequency channel.

Output is in flux density units.

Parameters

- **t1** – Start time in seconds, relative to mjdref
- **t2** – End time in seconds, relative to mjdref

Returns Average flux density between t1 and t2.

4.2 Submodules

4.3 filtools.sigproc module

class filtools.sigproc.**FilterbankIO**

Bases: object

This class deals with reading from and writing to sigproc filterbank files.

To read a filterbank file, first set up a new *FilterbankIO* object, and use the *read_header()* method to read the header parameters. The FilterbankIO object is then ready to be used for reading. To read the file, we use the *read_block()* method to read samples from the file.

Example:

```
inputfile = filtools.FilterbankIO()
inputfile.read_header("example.fil")
data = inputfile.read_block(1024) # Reads 1024 samples from the file
```

Note:

Currently *FilterbankIO* supports sigproc data files with the following properties:

- Single IF (i.e. single polarisation)
- Fixed channel bandwidth (i.e. each channel has same bandwidth/offset)

And, data type is one of:

- 1-bit unsigned integer
- 2-bit unsigned integer

- 4-bit unsigned integer
- 8-bit unsigned integer
- 16-bit unsigned integer
- 32-bit floating point

---

header
The header dict can be used to provide direct access to the sigproc header parameters. Parameters changed here before calling *write_header()* can change the ouput header parameters. However, it is usually best to access information about the data using the other methods provided below, as the structure can be re-used for other data types.

bits_per_sample()
Number of bits per sample in data file

centre_frequency()
The centre frequency

channel_bandwidth()
The channel bandwidth.

clone_header()
Create a new *FilterbankIO* object with the header parameters copied from this object

Returns A new *FilterbankIO* object

close()
Close the input file, reset the read/write state of the *FilterbankIO* object

current_position()
Print the current position of the file pointer in samples.

The next call to *read_block()* or *write_block()* will read or write from this sample.

frequency_table()
Get an array of centre frequency of each channel

get_reader()
This method returns a function optimised to read the input data.‘

get_writer()
Returns an optimised write function. Used by *write_block()*.

mjd_at_sample(*sample*)
Get the MJD at given sample number

read_block(*nsamples*, *dtype=<class 'numpy.float32'>*)
Read a block of data.

This will read up to `nsamples` from the file, returning a 2-d array indexed first by sample and then by channel. i.e. the shape of the returned array will be (nsamples_read,nchannels).

If the file has reached the end of file, then less than nsamples will be read, and if past the end of the file, an array of size (0,nchannels) will be returned.

Note: On first call this method is dynamically replaced using *get_reader()* to define an optimised verison of the reader method. This should be invisiable to most users.

Parameters

- **nsamples** – The number of samples to read.
- **dtype** – A numpy compatible data type. Data will be converted to this type using the astype numpy method, and should be at least large enough to hold the read data types.

Returns A 2-d numpy array of requested type.

read_header(*filename*)
Read header from sigproc filterbank file

Parameters **filename** (*str*) – filename to read

Returns Number of bytes read (i.e. size of header)

This method will also put the *FilterbankIO* object in a state for reading the file, i.e. enabling the *read_block()* method.

Header parameters can be accessed by the methods of the *FilterbankIO* object, or by direct interface with the sigproc API using the *header* dict member.

reference_dm()

sample_interval()
The time between each sample

seconds_since(*sample*, *mjdref*)
Get the number of seconds since `mjdref` at sample number `sample`.

Parameters

- **sample** – Integer sample number
- **refmjd** – Reference mjd

Returns seconds since `mjdref` at sample `sample`

seconds_since_start(*sample*)
Get the number of seconds since start of observation at sample number `sample`.

seek_to_sample(*sample*)
Position the file pointer at given sample number (index from zero).

The next call to *read_block()* or *write_block()* will start at the given sample. E.g.

```
fil.seek_to_sample(100)
block = fil.read_block(128) # block[0] is sample number 100
```

set_bits_per_sample(**args*, ***kwargs*)

set_frequency_table(**args*, ***kwargs*)

set_reference_dm(**args*, ***kwargs*)

set_sample_interval(**args*, ***kwargs*)

set_total_channels(**args*, ***kwargs*)

total_bytes()
Compute the total number of bytes in the file

total_channels()
The total number of frequency channels in the file

total_samples()
The total number of samples in the file

write_block(*block*)
Write a block of data to a file opened for writing.

The data block should be a numpy array in the same format as would be obtained from *read_block()*. The data will be truncated and cast to the required type to write to the file using the ndarray.astype method from numpy.

The block should have shape (nsamples,nchannels), where nsamples is the number of samples to write.

Parameters **block** – A 2-d numpy array to write.

Note: On first call this this method is dynamically replaced using *get_writer()* to define an optimised verison of the writer method. This should be invisiable to most users.

---

write_header(*filename*)
Write the current state to a new sigproc filterbank file

Parameters **filename** – filename to write

Returns Number of bytes written (i.e. size of header)

This method will also put the *FilterbankIO* object in a state for writing data to the file, i.e. enabling the *write_block()* method.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

Symbols

B

C

D

F

G