1. Getting started

1.1 Example filterbank files

Use one of the following filterbank files as an example:

- "https://git.dev.ti-more.net/uploads/-/system/personal_snippet/2/bc063035797e978034adfb6f2da75e70/p
- "https://git.dev.ti-more.net/uploads/-/system/personal_snippet/2/3da35656df8f722441579847974a03cb/p
- "https://git.dev.ti-more.net/uploads/-/system/personal_snippet/2/e6015ec024ad1f53d4c2f39511620db1/p

1.2 Import

```
from filterbank.filterbank import *
```

2. Filterbank Tutorial

2.1 Create a filterbank object

```
filterbank = Filterbank(<PATH TO FILTERBANK FILE>)
```

This is an example without parameters, see 2.3 for an example with the parameters. ## 2.2 Read the header from filterbank data

```
filterbank.header
```

Header data contains the following:

Variable	Description
source_name	name of filterbank file
P	period of pulsar in ms
DM	dispersion measure of pulsar
machine_id	id of machine used to receive signal data
telescope_id	id of telescope used to receive signal data
data_type	type of file `filterbank, time series`
fch1	center frequency of first filterbank channel (MHz)
foff	filterbank channel bandwidth (MHz)
nchans	number of filterbank channels
nbits	number of bits per time sample `8, 16 or 32`
tstart	timestamp of first sample (MJD)
nifs	number of seperate intermediate-frequency channels

freq_range = fch1 + (foff * nchans + fch1)
time_range = tstart + (tsamp/24/60/60)

freq_range is a tuple with a frequency start and a frequency stop
The same applies to time_range

The header data, including the center frequency, can be retrieved by calling the header attribute from the Filterbank object.

2.3 Read filterbank file

The attributes time_range and freq_range can be passed as parameters to select a specific portion of the filterbank file. To make the Filterbank object read the filterbank file at once, set the `read_all` parameter to `True`.

```
filterbank = Filterbank(<PATH TO FILTERBANK FILE>, freq_range, time_range, read_all)
```

2.4 Select a range of data from the filterbank file

The select_data method can be used to retrieve data from the Filterbank object. The user has the option to give a `time` and/or `frequency` range to select a subset from the entire dataset. The time-range can either be an index of a sample or a float that is represents a moment in time (in seconds).

```
filterbank.select_data(freq_range, time_range)
```

The `select_data` method returns an array of all different channels/frequencies and a large matrix with all the received radio signals.

The matrix contains for each time sample an array which has the intensity per channel/frequency.

2.5 Read filterbank as stream

When reading the filterbank file as a stream, the user should let the `read_all`-parameter stay `False` when initializing the filterbank object.

Each time the user calls the `next_row` method, it will retrieve an array with intensitiy per frequency for a new time sample from the filterbank file. When the last iteration of the filterbank is reached, the new_row method will return `False`.
The same goes for the `next_n_rows` method, where the user is able to define the amount of rows that should be returned.

```
filterbank.next_row()
```

```
filterbank.next_n_rows(n_rows=10)
```

3. Fourier Transformations

3.1 DFT

The `fourier.dft_slow` function is a plain implementation of discrete Fourier transformation.

The `fourier.fft_vectorized` function depends on this function.

3.1.1 Parameters

Parameter	Description
input_data	Array containing the values to be transformed.

Returns an array containing the transformed values.

3.1.2 Example usage

```
>>> from Asteria import fourier
>>> fourier.dft_slow([1,2,3,4])
array([10.+0.00000000e+00j, -2.+2.00000000e+00j, -2.-9.79717439e-16j, -2.-2.00000000e+00j])
```

3.2 FFT

The `fourier.fft_vectorized` function is a vectorized, non-recursive version of the Cooley-Tukey FFT

Gives the same result as `fourier.dft_slow` but is many times faster.

3.2.1 Parameters

Parameter	Description
input_data	Array containing the values to be transformed.

Returns an array containing the transformed values.

3.2.2 Example usage

```
>>> from Asteria import fourier
>>> fourier.fft_vectorized([1,2,3,4])
array([10.+0.00000000e+00j, -2.+2.00000000e+00j, -2.-9.79717439e-16j, -2.-2.00000000e+00j])
```

3.3 IFFT

This function computes the inverse of the one-dimensional n-point discrete Fourier transform computed by fft

3.3.1 Parameters

Parameter	Description
input_data	Input array, can be complex.

3.3.2 Example usage

```
>>> from Asteria import fourier
>>> fourier.ifft([0,4,0,0])
array([ 1.+0.j,  0.+1.j, -1.+0.j,  0.-1.j])
```

4. Plotting diagrams

4.1 PSD

The `plot.psd` function can be used to generate the data for a Power Spectral Density plot.

4.1.1 Parameters

Parameter	Description
samples	1-D array or sequence. Array or sequence containing the data to be plotted.
nfft	Integer, optional. The number of bins to be used. Defaults to 256.
sample_rate	Integer, optional. The sample rate of the input data in `samples`. Defaults to 2.

Parameter	Description
window	Callable, optional. The window function to be used. Defaults to `plot.window_hanning`.
sides	{'default', 'onesided', 'twosided'}. Specifies which sides of the spectrum to return. Default gives the default behavior, which returns one-sided for real data and both for complex data. 'onesided' forces the return of a one-sided spectrum, while 'twosided' forces two-sided.

4.1.2 Returns

Variable	Description
Pxx	1-D array. The values for the power spectrum before scaling (real valued).
freqs	1-D array. The frequencies corresponding to the elements in `Pxx`.

4.1.3 Example Usage

```
from filterbank.filterbank import Filterbank
import matplotlib.pyplot as plt
import numpy as np
from plot import psd
from filterbank.header import read_header

# Instatiate the filterbank reader and point to the filterbank file
fb = Filterbank(filename='examples/pspm32.fil')

# read the data in the filterbank file
_, samples = fb.select_data()

# Convert 2D Array to 1D Array with complex numbers
samples = samples[0] + (samples[1] * 1j)

# Read the header of the filterbank file
header = read_header('examples/pspm32.fil')

# Calculate the center frequency with the data in the header
center_freq = header[b'fch1'] + float(header[b'nchans']) * header[b'foff'] / 2.0

print(center_freq)
# Get the powerlevels and the frequencies
```

```
PXX, freqs, _ = psd(samples, nfft=1024, sample_rate=80, sides='twosided')

# Calculate the powerlevel dB's
power_levels =  10 * np.log10(PXX/(80))

# Add the center frequency to the frequencies so they match the actual frequencies
freqs = freqs + center_freq

# Plot the PSD
plt.grid(True)
plt.xlabel('Frequency (MHz)')
plt.ylabel('Intensity (dB)')
plt.plot(freqs, power_levels)
plt.show()
```

4.2 Waterfall

The `plot.waterfall.Waterfall` class can be used to generate waterfall plots.

4.2.1 Construction

Parameter	Description
filter_bank	A `filterbank` object.
center_freq	The center frequency of the signal in the filterbank object
sample_freq	The sample frequency of the signal in the filterbank object
fig	An imaging object, like `pyplot.figure()`
mode	String `{discrete, stream}`. The mode to operate on. Use discrete for discrete datasets, and stream for stream data. Defaults to `stream`.

4.2.2 Methods

Method	Description
init_plot(self)	Initialize the plot
update_plot_labes(self)	Generate the plot labels
get_next(self)	Returns the next row of data in the filterbank object
get_image(self)	Returns the image data of the full dataset, if using a discrete dataset.
update(self, i)	Updates the image with the next row of data, when using a continuous datastream.
animated_plotter(self)	Returns the figure and update function for matplotlib animation

Method	Description
get_center_freq(self)	Returns the center frequency stored in the filterbank header

4.2.3 Example Usage

4.2.3.1 With discrete data

```
import matplotlib.animation as animation
from filterbank.header import read_header
from filterbank.filterbank import Filterbank
from plot import waterfall
import pylab as pyl
from plot.plot import next_power_of_2

fb = Filterbank(filename='./pspm32.fil', read_all=True)

wf = waterfall.Waterfall(filter_bank=fb, fig=pyl.figure(), mode="discrete")

img = wf.get_image()

pyl.show(img)
```

4.2.3.2 With stream data

```
import matplotlib.animation as animation
from filterbank.header import read_header
from filterbank.filterbank import Filterbank
from plot import waterfall
import pylab as pyl
from plot.plot import next_power_of_2


fb = Filterbank(filename='./pspm32.fil')

wf = waterfall.Waterfall(fb=fb, fig=pyl.figure(), mode="stream")

fig, update, frames, repeat = wf.animated_plotter()

ani = animation.FuncAnimation(fig, update, frames=frames,repeat=repeat)
pyl.show()
```

5. Timeseries module

Learn how to use the timeseries object. The timeseries object is used for all timeseries related operations both filterbank and non-filterbank (general array) types.

5.1 Initialize the timeseries object

The `Timeseries` module can be used with both the filterbank file and a general `numpy` array.

5.1.1 Initialize using a filterbank file

- Create an filterbank object using the standard filterbank module.
- Read the filterbank file in memory (do not use streams)
- Initialize the timeseries object using the filterbank object.

```
import filterbank.filterbank as Filterbank
import timeseries.timeseries as Timeseries

# Read the filterbank file from a file.
filterbank_obj = Filterbank('./pspm32.fil')

# Read the filterbank as a whole instead of as a stream.
filterbank_obj =  filterbank_obj.read_filterbank()

# Initialize the timeseries object.
ts = Timeseries().from_filterbank(filterbank_obj)
```

5.1.2 Initialize using a numpy array

```
import numpy as np
import timeseries.timeseries as Timeseries

input_array = np.array([1, 2, 3, 4, 5, 7, 9, 10, 11])

ts = Timeseries(input_array)
```

5.2 Retrieve the timeseries array from timeseries object

5.2.1 Retrieve timeseries object

```
# Assumed that your timeseries object has been initialized.

timeseries_array = timeseries.get()
```

5.3 Downsample the timeseries

The first implemented feature for the timeseries object is the downsample/decimate function. This enables you to downsample your timeseries by `q` scale. This will make your input array smaller and basically 'cuts' the other parts off.

5.3.1 Downsample/Decimate

Called downsample in the current release because no anti-alliasing is used, might be renamed to decimate once more advanced operations (such as antialiasing) are used.

```
# Downsampled array shall be 3 times smaller than the current timeseries (as initialized)
scale = 3
# Returns an array with the downsampled timeseries, can also be retreived lated user timser
downsampled_array = timeseries.downsample(scale)
```

6. Clipping

6.1 Clipping

The `clipping.clipping` function can be used to remove noise from the data.

It combines all the individual methods of the clipping module to do this. The individual methods are described below in order.

```
clipping(<FREQUENCY_CHANNELS>, <TIME_SAMPLES>)
```

6.2 Filter samples

The `clipping.filter_samples` function can be used to remove entire time samples, that have noise in them.

The noise for a time sample is calculated by summing all the individual frequencies for one sample, and then comparing it with the average(mean) of all samples. If the intensity of a sample is lower than `average intensity per sample * factor`, the sample will be added back to the array. Otherwise, it's removed.

It is recommended to give only the first 2000 samples to this method, because adding more will only hurt the performance.

```
filter_samples(<TIME_SAMPLES>)
```

6.3 Filter channels

The `clipping.filter_channels` function can be used to remove entire channels/frequencies with noise from the data.

The noise for a frequency channel is identified by calculating the mean and standard deviation for each column in the data. After that, samples are removed if the mean or standard deviation of the intensity is higher than the `average intensity per channel * factor`. The channels with noise are removed from both the list with channels, as well as the entire dataset.

```
filter_channels(<FREQUENCY_CHANNELS>, <TIME_SAMPLES>)
```

6.4 Filter individual channels

The `clipping.filter_indv_channels` function can be used to replace all the remaining samples with noise.

The noise for each individual sample is identified by calculating the mean intensity per channel. After that, samples are replaced with the median of a channel if their intensity is higher than the `average intensity per channel * factor`.

```
filter_indv_channels(<TIME_SAMPLES>)
```

7. Dedispersion

7.1 Introduction

- Pulsars produce a narrow beam of electromagnetic radiation which rotates like a lighthouse beam, so a pulse is seen as it sweeps over a radiotelescope. The signal is spread over a wide frequency range.
- If space was an empty vacuum, all the signals would travel at the same speed, but due to free electrons different frequencies travel at slightly different speeds (dispersion).

7.2 Dedisperse

This method performs dedispersion on the filterbank data.

7.2.1 Parameters

Parameters	Description
Samples	Array or sequence containing the data to be plotted.
DM	Dispersion measure (cm-3 pc)

7.2.2 Returns

Variable	Description
Samples	Dedispersed samples

7.3 find_dm

This method finds the dispersion measure.

7.3.1 Parameters

Parameters	Description
Samples	1-D array or sequence. Array or sequence containing the data to be plotted.

7.3.2 Returns

Variable	Description
dm	Dispersion measure

7.4 find_line

This method finds a line starting from the sample index given in the parameters. This will stop if there isn't a intensity within the max_delay higher than the average_intensity.

7.4.1 Parameters

Parameters	Description
Samples	1-D array or sequence. Array or sequence containing the data to be plotted.
start_sample_index	The start sample index
max_delay	The max delay of the recieved signal
average_intensity	The average intensity

7.4.2 Returns

Variable	Description
start_sample_index	The first frequency
previous_sample_index	The higher frequency

7.5 find_estimation_intensity

This method finds the average intensity for top x intensities. The `average_intensity` is considered a requirement for intensities to be considered a pulsar.

7.5.1 Parameters

Parameters	Description
Samples	Array or sequence containing the data to be plotted.

7.5.2 Returns

Variable	Description
average_intensity	The avarage intesity

8. Pipeline

8.1 Introduction

The Pipeline module is used to execute the different modules in a specific order. There are currently three different options for running the pipeline.

These options include: * read multiple rows, `read_n_rows` * read single rows, `read_rows` * read all rows, `read_static`

The constructor of the pipeline module will recognize which method is fit for running which method, by looking at the given arguments to the constructor.

Parameter	Description
filename	The path to the filterbank file.
as_stream	This parameter decides whether the filterbank should be read as stream.
DM	The dispersion measure (DM) is used for performing dedispersion.
scale	The scale is used for performing downsampling the time series.
n	The `n` is the rowsize of chunks for reading the filterbank as stream.
size	The size parameter is used for deciding the size of the filterbank.

After deciding which method to run for running the filterbank in a pipeline, it will measure the time it takes to run each method using `measure_method`. After running all the different methods, the constructor will append the results (a dictionary) to a txt file.

8.2 Read rows

The `read_rows` method reads the Filterbank data row per row. Because it only reads the filterbank per row, it is unable to execute most methods. The alternative for this method is the `read_n_rows` method, which is able to run all methods.

```
pipeline.Pipeline(<filterbank_file>, as_stream=True)
```

8.3 Read n rows

The `read_n_rows` method first splits all the filterbank data into chunks of n samples. After splitting the filterbank data in chunks, it will run the different modules of the pipeline for each chunk. The remaining data, that which does not fit into the sample size, is currently ignored.

The `n` or sample size should be a power of 2 multiplied with the given scale for the downsampling.

```
pipeline.Pipeline(<filterbank_file>, n=<size> , as_stream=True)
```

8.4 Read static

The `read_static` method reads the entire filterbank at once, and applies each method to the entire dataset. If the filterbank file is too large for running it in-memory, the alternative is using `read_n_rows`.

```
pipeline.Pipeline(<filterbank_file>)
```

8.5 Measure methods

The `measure_methods` is ran for each of the above methods, and calculates the time it takes to run each of the different methods. For each method it will create a key using the name of the method, and save the time it took to run the method as a value. At the end, it will returns a dictionary with all the keys and values.

7.6 Overview of pipeline

Apart from the different modules described in the previous paragraphs, additional modules are required for this library to make detecting pulsar signals possible. However, these additional modules have not been developed yet, and are required to be developed in the future. In this paragraph the additional modules are listed and described.

Modules that are missing in the pipeline are highlighted using a `*`.

```
1. Read Filterbank as stream
2. Reduce RFI using clipping
3. Dedisperse radio signal
4. Transform dedispersed signal to TimeSeries
5. Run fast Fourier transformation on TimeSeries
6. * Identify and save birdies in file
7. * Perform Harmonic Summing
8. * Search and identify single and periodic signals
```

9. * Phase-fold remaining signals
10.* Do Transient searches

9. Generating mock data

9.1 Creating a filterbank file

To generate a filterbank file you may use the following example code:

```
import filterbank.header as header

header = {
    b'source_name': b'P: 80.0000 ms, DM: 200.000',
    b'machine_id': 10,
    b'telescope_id': 4,
    b'data_type': 1,
    b'fch1': 400,
    b'foff': -0.062,
    b'nchans': 128,
    b'tstart': 6000.0,
    b'tsamp': 8e-05,
    b'nifs': 1,
    b'nbits': 8
}

header.generate_file("file_path/file_name", header)
```

The header data is passed in a header dict. For a more detailed explanation on the file header, please consult chapter 2.2.

9.2 Generate signal

The generate_signal method generates a mock signal based on the following parameters:

Variable	Description
noise_level	the max amplitude of the generated noise
period	period of the signal
t_obs	observation time in s
n_pts	intervals between samples

9.3 Generate header

The generate_header method generates a header string based on the header dict provided in the example in chapter 8.1. The dict provides keys encoded in bytes and the method converts each keyword to a string using the keyword_to_string method (see below).

9.4 Keyword to string

The keyword_to_string method converts a keyword from the header dict to a serialized string.

9.5 Write data

Once all the required data is generated to a filterbank file. The data is written to the file as bytes.