LoFASM Tools

r0.1

Outline

- Introduction
- Downloading and installing the LoFASM Tools
- LoFASM Executables
- LoFASM API
- Examples

Introduction

The purpose of this document is to serve as a manual for the LoFASM Tools.

Downloading and installing the LoFASM Tools

Platforms

The LoFASM Tools have been tested on the following operating systems:

- Mac OS X Yosemite 10.10.2
- Redhat Enterprise 6
- Ubuntu 14.04

I cannot confirm that the LoFASM Tools will work on Windows since it has yet to be fully tested.

LoFASM Tools Requirements

In order to install the *lofasm* Python package, the following conditions must be met:

- Python version is 2.7.X
- numpy version 1.6.2 or newer is installed
- matplotlib version 1.1.1 or newer is installed

- scipy is installed (version to be confirmed)
- astropy is installed (version to be confirmed)

Cloning into the LoFASM repo

The LoFASM Tools are hosted at ARCC's Github Account. The repository can be downloaded by using the Git clone command in the directory you would like the repository to copied in.

```
git clone https://github.com/arcc/lofasm.git
```

> If you want to save the lofasm repo in the directory *~/repos* then navigate there with `cd ~/repos` before running the command above.

If all goes well, a new directory called `lofasm` should appear in your local directory. Use `ls` to check if the repository was created.

Installing the LoFASM Tools

Installing the LoFASM Tools **should** be as easy as running the Python `setup.py` file.

First, you will need to navigate to the new *lofasm* directory.

> If you are still in the directory from which you ran the `git clone` command above, then navigating to the *lofasm* directory is as easy as `cd lofasm`.

The contents of the *lofasm* directory should like this:

The LoFASM tools can be installed by using the *setup.py* script:

```
sudo python setup.py install
```

> If you're using a virtual environment then you most likely do **not** need the `sudo` in front of the command above.

Here is an example of the output:

If all went according to plan, then the LoFASM Tools should now be completely installed. To confirm that the installation succeeded try pulling up the help menu for the LoFASM Plotter.

```
lofasm_plot.py -h
```

> If the file was not found, then try looking at the output from the *setup.py* step to identify where the LoFASM executables have been stored. In the image above, the lines beginning with 'changing mode of' state the location of the executable LoFASM scripts.
>
> 
>
> Once you've identified where the executables have been stored then make sure the directory is in your path. Use `echo $PATH` to view your current path (in BASH).

LoFASM Executables

In this section I will list the LoFASM Executables and how to use them.

Note: some of the executables in the lofasm directory are no longer used. They will soon be completely dropped. That being said, I will not be mentioning them any further in this document.

lofasm_plot.py

The *lofasm_plot.py* animates all channels from LoFASM Data in .lofasm format. This script can also scan .lofasm files and check them for errors and identify corrupt integrations.

> Usage: lofasm_plot.py -f filename [options]
>
> Access the help menu using `lofasm_plot.py -h`:
>
> 
>
> The only flag that *lofasm_plot.py* requires is `-f`, which points to the LoFASM data file to be plotted. *The only exception to this is when the `-h` flag is used; this flag causes the program to print the help menu and exit. All other options are ignored.*
>
> Using *lofasm_plot.py* without any options will simply result in an animated plot of the LoFASM data that the plotter is pointed to.
>
> `lofasm_plot.py -f 20150328_210002.lofasm`
>
> 
>
> When the end of the file is reached the plotter will simply stop and wait until the plot window is manually

closed.

about the plot

lofasm_plot.py produces a figure with 11 different plots. The smaller plots represent the auto and cross correlations of the four LoFASM trunk lines. The four along the diagonal (INS, IEW, ONS, & OEW) are the auto correlations (self-power) in each of the four LoFASM signals. The other plots (above the diagonal) are the cross-powers.

The larger plot labeled 'All Channels' contains the auto-correlations plotted over each other.

The x-axis and y-axis are frequency and power in all of the plots.

axis limits

To change the limits on either axis use any combination of the `--xmin`, `-->xmax`, `--ymin`, and `--ymax` options. xmin and xmax refer to the minimum and >maximum limits to the x-axis, respectively. ymin and ymax refer to the minimum and maximum limits to the y-axis, respectively.

start position of data

To force *lofasm_plot.py* to start reading data from a particular place in the file either `-s` or `--start_position`. The position must be given in bytes.

```
lofasm_plot.py -f 20150328_210002.lofasm -s 96
```

If *lofasm_plot.py* cannot read the first LoFASM integration when initializing then an IntegrationError will be raised and the program will attempt to exit cleanly.

lofasm-chop.py

lofasm-chop.py is designed to scrape a little bit of data off the top of a large file for data health checking. Instead of downloading a 20GB data file just to discover that the data is not healthy, *lofasm-chop.py* can be used to 'sample' a .lofasm file.

All you have to do is tell lofasm-chop.py how many bytes of data (not including the file header, which gets transferred over automatically) you want to 'scrape off the top'. Optionally, an output filepath can be provided.

Usage: lofasm-chop.py [-h] [-b B] [-o O] filename

The help menu can be accessed with `lofasm-chop.py -h`.

To copy the first 10 integrations from file *20150328_210002.lofasm* and save them in a new file called *20150328_210002_chop.lofasm* use

```
lofasm-chop.py -b 1392640 -o 20150328_210002_chop.lofasm 20150328_210002.lofasm
```

The `-b` flag accepts the number of *data* bytes to be copied. Each LoFASM integration is 139264 bytes long. In the example above, I am sampling the first 10 LoFASM integrations in *20150328_210002.lofasm*.

The `-o` flag takes a path to the location of the new file to be created.

simulate_signal_as_AA.py

simulate_signal_as_AA.py uses the simulate subpackage to simulate LoFASM data.

This script is meant to be used as a template for future data simulation. Currently this script will generate a square wave and inject the signal into the AA channel of an other wise 'zeroed out' LoFASM data file.

Usage: simulate_signal_as_AA.py [-h] [-p PERIOD] [-t DURATION] -f FILENAME

There are only three pieces of input needed for this script.

`-f` : path to where the simulated signal should be saved

`-p` : the period of the simulated square wave in seconds.

`-t` : the total duration of the simulated data set in seconds.

LoFASM API

The LoFASM API is a Python package written to read and write LoFASM data.

Package Structure

```
lofasm
|--simulate
|   |--data.py
|   `--signal.py
|--animate_lofasm.py
|--filter.py
|--future.py
|--mkid.py
|--parse_data.py
|--parse_data_H.py
|--roach_comm.py
`--write.py
```

Package Conents Description

lofasm.simulate subpackage

The *simulate* subpackage has two modules: *data* & *signal*.

simulate.data

The class definitions in this module provide the framework to write filterbank data to disk in the .lofasm format.

simulate.signal

This module contains functions that either generate or facilitate the generation of signals in LoFASM filterbank data format.

signal.square_wave

usage: s, t = square_wave(f[, fsamp][, T][, offset]) *f* is the frequency of the signal in Hz.

fsamp is the sampling frequency in Hz.

T is the length of the data in seconds.

offset is the phase offset in radians.

return numpy array with square wave

lofasm modules

animate_lofasm.py

The *animate_lofasm.py* module is meant to provide functions and constants to be used for animation purposes.

module attributes

animate_lofasm.FREQS

> defined using numpy as `FREQS = np.linspace(0, 200, 2048)`. The *FREQS* attribute is used as the x-axis array when plotting LoFASM filterbank data.

animate_lofasm.autos

> definition: `autos=['AA', 'BB', 'CC', 'DD']`
>
> A Python list containing the labels of the auto-correlation plots. Each element is a string.

animate_lofasm.cross

> definition: `cross = ['AB','AC','AD','BC','BD','CD']`
>
> A Python list containing the labels of the cross-correlation plots. Each individual element is a string.

animate_lofasm.beams

> definition: `beams = ['NS', 'EW']`
>
> A Python list containing the labels of the two LoFASM beam polarizations. This is not currently being used since there is a bug in the current version of the tools affecting beam generation.

animate_lofasm.BASELINE_ID

> A Python dictionary containing the labeling of the trunk lines for each LoFASM site. This dictionary is how the LoFASM Tools interpret the LoFASM polarizations.
>
> Each element in this dictionary is itself a dictionary as well. These 'second layer' dictionaries hold the labels themselves.
>
> Legend: INS: Inner North-South ONS: Outer North-South IEW: Inner East-West OEW: Outer East-West
>
> The available LoFASM baseline arrangements are: 'LoFASMI', 'LoFASMII', 'LoFASM3', 'LoFASMIV', & 'simdata'.

animate_lofasm.BASELINES

> A Python list containing both the auto-correlation and cross-correlation labels.
>
> Definition:
> `BASELINES = ['AA', 'BB', 'CC', 'DD', 'AB', 'AC', 'AD', 'BC', 'BD', 'CD']`

module methods

animate_lofasm.setup_all_plots(xmin, xmax, ymin, ymax, station, crawler [, norm_cross=False])

> Docstring: setup all auto and cross-power plots
>
> Create necessary figure plots in proper arrangement and return the corresponding matplotlib line objects
>
> *xmin*, *xmax*, *ymin*, and *ymax* are used for limits on the plot axes. The x-axes is frequency (MHz) and the y-axis is power (dB).
>
> The *station* argument is one of the baseline arrangments (as a string) accepted by *animate_lofasm.BASELINE_ID*.
>
> *crawler* is an instance of *lofasm.parse_data.LoFASMFileCrawler*. Refer to the documentation of the *lofasm.parse_data* module for more information on the LoFASM file crawler class.
>
> The optional argument *norm_cross* is deprecated and no longer used. It is left in this definition for compatiblity purposes but will soon be done away with.

animate_lofasm.update_all_baseline_plots(i, fig, crawler, lines [, norm_cross] [, forward])

> Update all plots created by *animate_lofasm.setup_all_plots* by incrementing the file crawler to the next LoFASM integration and replacing the matplotlib line object data arrays. Iterating this function using the Matplotlib animation library will animate the LoFASM plots as a function of time.
>
> The *i* argument is an integer. It does not matter what integer is placed here. This argument is required by the Matplotlib animation module. *i* will be incremented in between iterations by matplotlib's *FuncAnimation* function.
>
> *fig*: a matplotlib figure object representing the figure that contains the LoFASM plots.
>
> *crawler*: an instance of *lofasm.parse_data.LoFASMFileCrawler*. Refer to the documentation of the *lofasm.parse_data* module for more information on the LoFASM file crawler class.
>
> *lines*: a Python list of matplotlib 2D line objects used in the LoFASM plots.
>
> *norm_cross*: This is a **deprecated** argument. It is being kept in this definition for compatibility purposes. This will soon be removed.

> *forward*: boolean argument. If True then increment the crawler by a single integration before updating plots. If False, then leave crawler where it is but still update all plot data arrays.

filter.py

> A Python library for LoFASM filtering methods.
>
> Available filters: *filter.running_median(y [, N])*
>
> Docstring: Given a list, y, perform a running median filter. Return the resulting list.
>
> N is the total number of points to be considered for the running median. The default is 50, so for any point X(n) the values considered will be [X(n-25),X(n+25)], inclusive.
>
> If N is not an even number then it will be changed to even number N-1.
>
> If N is not an integer it will be truncated.

future.py

> Classes and function definitions that need a home. The methods and classes in this file were written to accomplish a certain task at a certain time but did not truly get integrated into the LoFASM tools.
>
> If needbe these will be fully integrated into dedicated libraries at some point in the future.
>
> class *future.ComparableMixin*:
>
> > This class can be inherited to facilitate comparing class instances to each other. This allows for the use of the <,>,==, <=, and >= operators between the instances of two future.ComparableMixin child classes.
> >
> > This class is derived from *object*.
> >
> > Child classes **must** have a _cmpkey attribute in order to use future.ComparableMixin capabilities.
>
> class *LoFASM_file*:
>
> > function *get_total_file_size(fname)*:
> >
> > > Description: Return total file size in bytes.
> > >
> > > If *fname* points to a regular file then use *future.syscmd* to retrieve file's total size in bytes by parsing the output of `ls -l <fname>`.

> function *syscmd(cmd)*:
>
> > Description: execute Linux *cmd* as a subprocess and catch output.
> >
> > *cmd* is a string containing a Linux command.
>
> function *file_datetime(filename)*:
>
> > Description: return datetime.datetime object from filename
> >
> > *filename* is a string containing the name of a .lofasm file as labeled by the LoFASM Data recorder.
> >
> > Returns a datetime.datetime object representing the timestamp in the filename.

mkid.py

> Library for parsing FPGA snapshots. Written by the CASPER community.

parse_data.py

> Module for parsing LoFASM Data
>
> function *getSampletime (Nacc)*
>
> > Docstring: Return the sample time corresponding to the number of accumulations, Nacc.
> >
> > *Nacc* can be either an integer or a float.
>
> function *freq2bin (freq)*
>
> > Docstring: Return bin number corresponding to frequency freq
> >
> > Returns an integer bin number.
> >
> > *freq2bin* will calculate a bin number by dividing *freq* by the resolution bandwidth, which is defined by `rbw = 200.0/2048`, and casting the result as an *int*.
>
> function *bin2freq (bin)*
>
> > Return frequency (MHz) corresponding to *bin*.
> >
> > This is the opposite of *freq2bin*.
> >
> > *bin* should be an integer.

> The frequency is calculated by multiplying *bin* by *rbw* (see *freq2bin* for information on how *rbw* is defined).

function *parse_filename (filename)*

> return the file's UTC time stamp as a list [YYmmdd, HHMMSS, pol]
>
> *filename* is a string containing the name of a LoFASM data file. *filename* does not need to actually point to a regular file. The UTC time stamp is obtained from the filename string itself.
>
> It is important that the LoFASM filename be in the format that it was originally saved in.

function *fmt_header_entry (entry_str [, fmt_len])*

> Ensure that every header entry is *fmt_len* characters long. If longer, then truncate. If shorter, then pad with white space.
>
> returns formatted string

function *parse_file_header (file_obj [, fileType])*

> Read LoFASM file header using file_obj.read() and return parsed information as a Python dictionary.
>
> This function preserves *file_obj*'s file pointer location.
>
> *file_obj* should be a valid Python File Object
>
> *fileType* is optional and contains a string of recognized file extensions. Currently, the only file extension recognized is *.lofasm*.

function *parse_hdr (hdr [, hdr_size_bytes] [,version])*

> Parse integration header and return corresponding header dictionary
>
> Usage: parse_hdr (<64bit_string> [, version])
>
> Docstring: Parse the first 64 bits of a LoFASM data packet and return a dictionary containing ech header value.
>
> If *hdr* has a length greater than 8 bytes then it will be truncated and only the first 8 bytes will be parsed; everything else will be ignored.

function *print_hdr (hdr_dict)*

> Iterate through a header dictionary and print all fields and their values to the screen.

> *hdr_dict* should be a dictionary as returned by _parse_file_header or *parse_hdr*.

function *check_headers (file_obj [, packet_size_bytes] [, verbose] [, print_headers])*

> Iterate through LoFASM Data file and check that all the header packets are in place.
>
> > Note: The *verbose* keyword argument is no longer used. It is being left in the definition for now for compatibility purposes.
>
> Returns a tuple (*best_loc*, *err_counter*), where *best_loc* and *err_counter* are the best data start position and the number of bad integrations in the file, respectively.
>
> *packet_size_bytes* is the size of a LoFASM Network packet in bytes. The default value is 8192 and the data type should be int.
>
> *print_headers* is a boolean argument. If set to True then print out all integration header information while scanning the file.

function *get_filesize (file_obj)*

> Usage: get_filesize(file_obj)
>
> Returns file size, in bytes, of file pointed to by *file_obj*. *file_obj* must be a valid Python file object.
>
> File size is calculated by using the file object's *seek* function to navigate to the last byte in the file and retrieve the location of the file using the file object's *tell* function.

function *get_number_of_integrations (file_obj)*

> Returns the number of LoFASM integrations in a .lofasm file. The return value is of type *float*.
>
> The number of integrations is calculated using *get_filesize* to obtain the size of the file in bytes and divide it by the integration size.

function *get_next_raw_burst (file_obj [, packet_size_bytes] [,packets_per_burst] [, loop_file])*

> Usage: get_next_raw_burst([, packet*size*bytes] [, packets*per*burst] [, loop_file])
>
> Python generator that yields a string containing data from the next 17 LoFASM packets in *file_obj* that make up a single 'burst'. If file_obj's pointer is not at zero, then assume it is in the desired start position and begin reading from that point in the file.

function *find_first_hdr_packet (file_obj [, packet_size_bytes] [, hdr_size])*

> Return start location of the first valid header packet in file.

> *file_obj* is a Python file object.
>
> The optional parameters, *packet_size_bytes* & *hdr_size*, must both be integers. Their default values are 8192 & 96, respectively.

function *is_next_packet_header (file_obj [, packet_size_bytes] [, hdr_size_bytes])*

> Check if the next LoFASM packet is a header packet.

class *LoFASM_burst*

> Class to represent an entire LoFASM burst sequence. A LoFASM burst is a collection of 17 User Datagram Protocol (UDP) packets in a particular order.
>
> The first of these packets is always the header packet. The following 16 network packets contain raw LoFASM filterbank data.
>
> All network packets have the same dimensions and are the same length.
>
> *self.autos*:
>
> > A dictionary containing the LoFASM auto-correlation channels.
>
> *self.cross*:
>
> > A dictionary containing the LoFASM cross-correlation channels.
>
> *self.beams*:
>
> > A dictionary containing both polarizations of the LoFASM beams. **Note: this dictionary is not created until *self.create_LoFASM_beams* is executed.**
>
> *self.pack_binary (self, spect)*:
>
> > Convert filterbank data into writable binary string format.
> >
> > Usage: pack_binary (spect)
> >
> > Returns: A binary string containing the filterbank data
> >
> > The data type of the information stored in spect must correspond to one of the data types used in LoFASM bursts. (int, np.complex, or np.float64)
> >
> > all elements in spect must be of the same type.

getAutoCorrelationDataType (self):

> Return the data type used for auto correlation data.

getCrossCorrelationDataType (self):

> Return the data type used for cross correlation data.

getBeamDataType (self):

> Return the data type used for LoFASM beam data.

create_LoFASM_beams (self):

> Generate both LoFASM beams and store them as class attributes.
>
> This function will create the dictionary *self.beams*.

class *LoFASMFileCrawler*

> File crawler for LoFASM data files.
>
> Usage: LoFASMFileCrawler(filename [, scan*file] [, start*loc])
>
> Where *scan_file* is a boolean value. If True then scan and print all integration headers in file. This is an optional argument.
>
> *start_loc* is the starting location of the data in the file. If _start_loc is set then *scan_file* will be ignored and the crawler will be initiated at the given *start_loc*.
>
> function *forward (self [, N])*:
>
> > Move forward by N integrations. Default is 1.
>
> function *backward (self [, N])*:
>
> > Move backward by N integrations. Default is 1.
>
> function *reset (self)*:
>
> > Move back to first integration in file.
>
> function *getIntegrationHeader (self)*:
>
> > Return integration header information as a dictionary.

function *getFileHeader (self)*:

> Return LoFASM file header info as a dictionary.

function *getAccNum (self)*:

> Return LoFASM accumulation number of current integration.

function *getAccReference (self)*:

> Return reference accumulation number. This is the the accumulation number corresponding to the first valid integration.

function *getFilePtr (self)*:

> Return file pointer location.

function *getIntegrationSize (self)*:

> Return the size of a LoFASM integration in bytes.

function *getFilename (self)*:

> Return the filename of the current LoFASM file.

function *print_int_headers (self [, state])*:

> If *state* is True then print integration headers after every transition. This will print integration header information after every *forward()*, *backward()*, and *reset()* command.
>
> If *state* is None then simply print current integration header.

parse_data_H.py

Constants and Error Classes

LoFASM_FHEADER_TEMPLATE:

> Dictionary containing the accepted LoFASM header templates.

LoFASM_SPECTRUM_HEADER_TEMPLATE:

> Dictionary containing the accepted LoFASM integration header templates.

class *Header_Error*:

> Error class for bad header issues.

class *IntegrationError*:

> Error class for bad LoFASM integrations.

roach_comm.py

> Library for functions that require talking to the ROACH Board
>
> function *connect_roach()*:
>
> > Connect to roach board and return the fpga handle.
> >
> > The ROACH board's IP address must be stored as environement variable 'ROACH_IP'. If the environment variable is not set then the default, `192.168.4.21`, will be used.
> >
> > the fpga handle is an instance of corr.katcp_wrapper.FpgaClient
> >
> > once the fpga handle is received, the roach connection can be confirmed by looking at the output of `fpga.is_connected()`.
>
> function *getSampletime (Nacc)*:
>
> > Return the ROACH board's sampling time.
>
> function *getRoachAccLen ()*:
>
> > Return the value of the ROACH register 'acc_len'
>
> function *getNumPacketsFromDuration (obs_dur)*:
>
> > Return the integer number of network packets corresponding to an interval of time.

write.py

> Methods for writing LoFASM Data to disk
>
> function *fmt_header_entry (entry_str [, fmt*\len])_:
>
> > Ensure that every header entry is fmt_len characters long. If longer, then truncate. If shorter, then pad with white space. The resulting formatted string is then returned.
>
> function _write_header_to_file (outfile, host [, Nacc] [, fpga_clk_T] [, Nchan] [, fileNotes])

> Prepends data file with LoFASM spectrometer header. *fpga_clk_T* is the period of the FPGA clock in seconds. *Nchan* is the number of FFT bins in the spectrometer data. *Nacc* is the number of accumulations averaged over before dumping each integration.