PIMA Python wrappers

Date of last modification: 2016.06.25_21:36:59

PIMA was designed to be "scriptable", i.e. called from another programs. The interface: control file that have to define all parameters, task and command line arguments that override the keywords defined in the control file is somewhat heavy weight. PIMA distribution provides 4 scripts and users are encouraged to develop their own. These scripts are also called wrappers, because they wrap calls to various PIMA tasks.

Contents:

The wrappers do not provide new functionality that *PIMA* does not have. They significantly simplify the user interface by expense of reducing functionality and by imposing rules on file names. If you need full functionality, for instance for processing unusual experiment, *PIMA* wrappers will not work for you. But the use of wrappers may be sufficient for processing many experiments.

Wrappers assume the file names obey the following convention:

- *PIMA* control files are located in directory VVVVV/EEE, where VVVVV is the root directory of vlbi experiments specified by --pima-exp-dir during configuration and EEE is the experiment name specified in the keyword **SESS_CODE** of the *PIMA* control file.

The following wrappers are provided:

- **pf.py** — Fringe fitting. Includes tasks data loading, parse log files, coarse fringe fitting,

bandpass computation, fine fringe fitting, data calibration and splitting, generation of the output database in GVF format, auto-imaging, generation of image pictures, etc.

- **pt.py** — Trial fringe fitting. Runs a trial fringe fitting procedure for a given observation.
- **pr.py** — Resolving sub-ambiguities. Parses the listing of the VTD/Post-Solve run, generates control file for re-fringing with a narrow search window, executes that control file, and updates the database.
- **automap.py** — automatic imaging using given averaged and calibrated visibility file.
- **imadir.py** — automatic imaging all the sources with averaged and calibrated visibilities found in a given directory.

pf.py — a general *PIMA* wrapper.

usage: **pf.py**
[-h] [--version] [-v value] [-r] [-s] [-H]
exp band
{***load,logs,gean,coarse,bpas,fine,mkdb,mktxt,splt,gain,autm,pict,map,sres,gaco***}
[opts]

The wrapper has general options followed by three mandatory positional arguments followed by task specific optional arguments. General options that follow immediately after the wrapper name and before positional arguments:

- **-h** — prints a brief information about the wrapper.
- **-H** — print extended manual (this text)
- **--version** — prints the wrapper version.
- **-v** — verbosity level. An integer parameter that describes how detailed informational messages will be printed at the screen. Error message will be printed regardless of verbosity level. **pf.py** will print.
  - **0** — silent mode. No information messages are printed.
  - **1** — moderate verbosity (default)
  - **2** — verbose mode. *PIMA* commands are printed at the screen before execution.
  - **>2** — debugging mode. More verbose than 2.

- **-r** — dry run. **pf.py** prints commands that it is about to execute, but does not execute them.
- **-s** — to use static build of *PIMA*. The directory of the static build is defined in pima_local.py. This option may be useful when you have several versions of *PIMA*.
- **[opts]** — additional options. Some tasks have specific options that starts from hyphen. These options may be followed up by pairs keywords: value, where keywords is a recognized *PIMA* keyword. Wrapper **pf.py** may override values of some keywords defined in the control file. Values of the keywords defined at the end of the wrapper command line have the highest priority and override the values set by the wrapper.

There are three mandatory arguments:

- **exp** — experiment name in low case.
- **band** — band name in the low case.
- **task** — take name. A task may follow by task specific options. The task specific options cannot be be put before the task name.

Supported tasks:

- ***load*** [**-nopcal**] — loading a database into internal *PIMA* data structures. This task searches for SSSSS/EEE_uv.exc file. If such a file exists, it purges its contents. If during executing *PIMA* task ***load*** it finds that there are bad points, and the keyword **UV_EXCLUDE_FILE: AUTO**, it runs the this task the second time, and if necessary, the second time. Option **-nopcal** instructs *PIMA* not to load phase calibration.

  This task creates log file VVVVV/EEE/EEE_load.log.

- ***logs*** — searches for all log files in VVVVV/EEE directory, parses them and writes down in PIMA ANTAB format. This task assumes log names are either in IVS format: VVVVV/EEE/EEElog.SS format, or VLBA format: VVVVV/EEE/EEESS.log, or in KVN format: VVVVV/EEE/EEEKK.log, where SS is a two character long station code, KK is a three character long name of a KVN station: one of KTN, KUS, or KYS. The output file has format VVVVV/EEE/EEE_SS.ant. This task creates log file VVVVV/EEE/EEE_log_antab.log.

- ***gean*** — this task searches for log files in PIMA ANTAB format generated, for example, by wrapper task ***logs*** and/or for legacy VLBI log file, parses them and loads system temperature, cable calibration atmospheric pressure, air temperature, and relative humidity into internal data structure of *PIMA*. It searches for parsed log files in PIMA ANTAB format with file names VVVVV/EEE/EEE_SS.ant and/or VLBA legacy log with

name VVVVVV/EEE/EEEcal.vlba. When the tasks processes legacy VLBA logs, it loads phase calibration phase and amplitude as well as cable calibration. NB: Legacy log files should *not* be used for processing VLBA experiments recorded with a digital backend since 2014. If FITS-IDI files had calibration information, task ***gean*** overwrites it. This task creates log file VVVVV/EEE/EEE_gean.log.

- ***coarse*** — this task performs fringe search in the coarse mode. The goal of performing coarse fringe search is a) to identify failures; b) to find a list of high SNR scans. Coarse fringe search runs in a simplified mode in order to speed up computations. No bandpass calibration, no bandpass mask, no phase calibration mask, no oversampling is applied. Fringe fit uses single polarization data. Fine fringe fits algorithm uses a simplistic parabolic fit. The task generates the output file with fringe fitting results with name VVVVV/EEE/EEE_B_nobps.fri and fringe fitting residuals with name VVVVV/EEE/EEE_B_nobps.frr overriding names specified in the control file by keywords **FRINGE_FILE** and **FRIRES_FILEpf.py** still can override these names. The task creates log file with name VVVVVV/EEE/EEE_B_coarse.log.

- ***bpas*** — this task performs computation of the bandpass and, in a case of dual-polarization data, polarization bandpass. This task has two modes: inspection mode that is invoked with option **-insp** and production mode. The task in the inspection mode computes bandpass in the **init** mode. It generates two plots for observations with the maximum SNR with at all baselines with the reference stations. The first plot shows normalized amplitude (green), a model fit to the normalized amplitude (blue), and amplitude of auto-correlation (red). The second plot shows residual phases (green) and the model fit to residual phases (blue). The task invoked in the inspection node creates a bandpass with name VVVVV/EEE/EEE_B_init.bps and, if dual-band data are processed, the polarization bandpass with name VVVVV/EEE/EEE_B_init_plr.bps as well.

  When task is invoked in the production mode, i.e. without using **-insp** option, it computes the bandpass, and polarization.

- ***fine*** — runs fringe fitting in the fine mode. Unless option *-keep* is specified, this task will purge fringe file and fringe residual file if they exist before processing the first observation. Option *-keep* suppresses deletion of existing fringe file and fringe residual files. The list of observations to be processed is determined by keywords *OBS, INCLUDE_OBS_FILE, EXCLUDE_OBS_FILE*

  The task creates log file with name VVVVVV/EEE/EEE_B_fine.log.

- ***mkdb*** — generates of the output database in GVF format using fringe fitting results.

  The task creates log file with name VVVVVV/EEE/EEE_mkdb.log.

- ***mktxt*** — — generates of the output database in TEXT format using fringe fitting

results.

The task creates log file with name VVVVVV/EEE/EEE_mktxt.log.

- ***gain*** — updates gain table into internal *pima* data structure. The task checks two files vlba_gains.key and ivs_gains.key in the share directory specified by options **--pima-share** during *PIMA* installation. The task does not issue a message if it does not find gain information for one or more stations, since this situation is considered normal. However, if gain is missing for certain stations(s), observations at baselines with such stations will not be used by task ***splt***, and therefore, cannot contribute to imaging.

  This task *updates* the gain only for frequencies and stations specified in the control file (**BEG_FRQ, END_FRQ, FRQ_GRP**) and found in the gain file. Gain for for other stations and/or other frequencies remains unchanged.

  The task creates log file with name VVVVVV/EEE/EEE_B_gain.log. The log file contains information about gain information in *PIMA* data structure after running this task. It is strongly advised to examine this log file to be sure that correct gain is applied.

- ***splt*** — performs calibration of visibilities for system temperature and gain; performs amplitude renormalization, applies results of fringe fitting, averages visibilities over time and frequencies, and writes calibrated and averaged visibilities into output files in FITS format.

  The task creates log file with name VVVVVV/EEE/EEE_B_splt.log.

- ***autm*** — performs automatic imaging. Unless option **-sou** is specified, this task searches for all fits files with calibrated and averaged visibilities with in directory SSSSSS/EEE_uvs with suffix _uva.fits and runs fringe fitting for all these sources. Option **-sou** followed by the value of source names separated by comma instructs ***autm*** to perform automatic imaging for these sources only. Source names can be either in B1950 or J2000. *PIMA* does not report an error if does not find one or more input files with calibrated and averaged visibilities.

  The output files are created for each source: map in FITS image format with suffix _map.fits, self-calibrated visibilities in FITS-format, and ascii log file with suffix dfm.log.

  The task creates log file with name VVVVVV/EEE/EEE_autm.log.

- ***pict*** — generate pictures files in gif format for all source images in the image directory for a given experiment, given band. This task searches for pair of image file in FITS format with suffix _map.fits and self-calibrated visibilities with suffix _uvs.fits in image directory SSSSS/EEE_uvs. It generates two files fir each input: picture of the image in gif format with suffix _map.gif and scan-averaged self-calibrated visibilities versus baseline length in gif format with suffix _uvs.gif.

The task creates log file with name VVVVVV/EEE/EEE_autm.log.

- ***sres*** — runs task gain (see above) and task ***splt*** for reference sources defined in file VVVVV/EEE_B_ref.sou. The file may define sources either with B1950 or J2000 names, one source per line. Lines that start with # are considered as comments and ignored.

  The task creates log file with name VVVVVV/EEE/EEE_sres.log.

- ***gaco*** — computes gain correction for reference source defined in the reference source file with name VVVVV/EEE_B_ref.sou. The file may define sources either with B1950 or J2000 names, one source per line. Lines that start with # are considered as comments and ignored.

- ***map*** — combines tasks ***gain***, ***splt***, ***autm***, ***gain***. It is just consecutively executes these tasks.

pt.py — trial fringe fitting.

This wrapper performs fringe fit for an observation with given index. Fringe results are written in /tmp/1.fri and fringe residuals are written in /tmp/1.frr overriding keywords **FRINGE_FILE** and **FRIRES_FILE**. The task does not purge these files, and therefore, it appends results to their end. Task **pt.py** shows an 1D plot of residual phases and amplitudes versus frequency and a similar plot of residuals versus time. Examining fringe plots is the main function of this task. Task *DEBUG_LEVEL:* **6**, and therefore, it prints verbose report about fringe fitting.

usage: **pt.py**
[-h] [--version] [-v value] [-r] [-s] [-H]
exp band obs
[opts]

Wrapper has general options followed by three mandatory positional arguments followed by *PIMA* options that are pairs keyword: value. There are three mandatory arguments:

- **exp** — experiment name in low case.
- **band** — band name in the low case.
- **obs** — observations index. Should be a positive number not exceeding the total number of observations in the experiment.
- **-h** — prints a brief information about the wrapper.

- **-H** — print extended manual (this text)
- **--version** — prints the wrapper version.
- **-v** — verbosity level. An integer parameter that describes how detailed informational messages will be printed at the screen. Error message will be printed regardless of verbosity level. **pt.py** will print.
  - **0** — silent mode. No information messages are printed.
  - **1** — moderate verbosity (default)
  - **2** — verbose mode. *PIMA* commands are printed at the screen before execution.
  - **>2** — debugging mode. More verbose than 2.
- **-r** — dry run. **pt.py** prints commands that it is about to execute, but does not execute them.
- **-s** — to use static build of *PIMA*. The directory of the static build is defined in pima_local.py. This option may be useful when you have several versions of *PIMA*.
- **[opts]** — additional options, pairs keywords: value, where keywords is a recognized *PIMA* keyword. Wrapper **pt.py** may override values of some keywords defined in the control file. Values of the keywords defined at the end of the wrapper command line have the highest priority and override the values set by the wrapper.

pr.py — re-fringe VLBI experiment.

Task **pr.py** implements interface Solve &arrow; *PIMA*. It 1) analyzes Solve residual file, 2) finds observations that have been suppressed, computes predicted path delay on the basis of a priori path delay and adjustments from the Solve solution and computes correction to the a priori path delay with respect to the model used by the correlator; 3) generates a command file that calls *PIMA* with parameters of the search window centered with respect to the updated a priori path delay and with the specified window semi-width; 4) executes this command file; 5) creates databases in GVF format; 6) updates automatic suppression status.

There are several situations when *PIMA* re-fringe procedure helps to improve results:

1. There was a strong RFI and fringe fitting procedure picked up fringes from the FRI;
2. A priori delay rate was low and fringe fitting procedure picked up fringes from the phase calibration signal.

3. There was a significant phase distortion in IFs after applying measured phase calibration and bandpass calibration. As a result, the amplitude of the secondary maximum of 2D Fourier transform that in the absence of phase distortion would be below the amplitude of the main maximum became higher than the level of the main maximum.

4. A priori source position (and in a case of RadioAstron, a priori Space Radio Telescope position) was significantly (more than 1 mas) improved. Significant errors in a priori source position may result in quadratic term of fringe phase versus time.

5. A source has marginal SNR (typically in range 4.5–6.5), and the thermal noise reduced the main maximum and increased a secondary maximum above the amplitude of the main maximum.

VTD/Post-Solve interactive solution should be made first. Option Print residu(A)ls: ON should be turn on, the spool file with solution listing be rewound, and residuals be generated (command Q). The the spool file with residuals should be copied into file VVVVVV/EEE/EEE_B_init.spl. An analyst should be check carefully the residual file. In particular, an analyst should check a) the experiment name and band name :-); b) whether solution is correct (wrms of residuals is close to what it is supposed to be); c) the spool file contains residuals. Residual section starts after line Residuals from Solve Symbols > at the 8-th position marks suppressed observation. All suppressed observations will be re-fringed.

There is a way to change the list of observations that will be re-fringed. If an analyst does not want to re-fringe some observations, the lines that correspond to these observations should be either removed from the listing file or the character at the 8th column of the rows that correspond to those observations should be changed to K. Alternatively, if an analyst would like to re-fringe a given observation even it it is not suppress, the character at 8th column should be changed to R. Re-fringing an observation of a source that had a priori position errors exceeding 1 arcsec may improve the SNR.

Usage: **pr.py**
[-h] [--version] [-v value] [-r] [-s] [-H]
exp band snr
[opts]

Wrapper has general options followed by three mandatory positional arguments followed by *PIMA* options that are pairs keyword: value. There are three mandatory arguments:

- **exp** — experiment name in low case.
- **band** — band name in the low case.
- **snr** — SNR limit. Should be a positive number. Typical value is 4.8.
  NB: SNR detection limit is lower for re-fringing procedure, because the search

window is less.

- **-h** — prints a brief information about the wrapper.
- **-H** — print extended manual (this text)
- **--version** — prints the wrapper version.
- **-v** — verbosity level. An integer parameter that describes how detailed informational messages will be printed at the screen. Error message will be printed regardless of verbosity level. **pr.py** will print.
  - **0** — silent mode. No information messages are printed
  - **1** — moderate verbosity (default). Only messages about finish of the procedure is printed.
  - **>2** — debugging mode. More verbose than 1.
- **-r** — dry run. **pr.py** prints commands that it is about to execute, but does not execute them.
- **-s** — to use static build of *PIMA*. The directory of the static build is defined in pima_local.py. This option may be useful when you have several versions of *PIMA*.
- **[opts]** — additional options:
  - **-delwin** — specifies the semi-width of the search window with respect to group delay. Units: ns. By default, **pr.py** selects the window semi-width itself depending on frequency.
  - **-nodb** — does not create the database upon completion of re-fringing. This option is required for processing the low band of a dual-band experiment.

The wrapper creates log file VVVVVV/EEE/EEE_samb.log.

pu.py — update suppression status after re-fringing.

VTD/Post-Solve supports so-called automatic suppression status. This status depends on a number of factors including detection status and other parameters. When *PIMA* creates a GVF database it sets this status for version 1. But the status does not automatically propagate to version 2 and higher. Wrapper **pu.py** propagates the status from version 1 database to the higher version.

Let us consider the following situation. A given observations had SNR 4.7 and therefore was treated as unconditionally suppressed. VTD/post-Solve does not show such observation and does not allow to restore it. After re-fringing then SNR grew to 6.8, i.e. the observations was detected. Task ***mkdb*** created GVF database version 1. The suppression status is version dependent. The observation is marked as good in version 1, but unconditionally suppressed in version 2.

Wrapper **pu.py** will set status "suppressed, but recovered" in version 2. Then a user can reset status to "good".

usage: **pu.py**
[-h] [--version] [-v value] [-r] [-s] [-H]
exp

automap.py — Automatic imaging of a given visibility file.

This wrapper calls DIFMAP and preforms automatic imaging using script pima_mupet_01.dfm developed by Martin Shepherd and Greg Taylor.

usage: **automap.py**
[--version]
uva_file

The input for the wrapper is the file with averaged calibrated visibilities in FITS-IDI format.

The wrapper assumes the filename with averaged calibrated visibilities has the following form SSSSSS/EEE_uvs/JJJJJJJJJJ_B_uva.fits where SSSSSS is the directory specified in the keyword **SESS_CODE** of *PIMA* control file. The wrapper generates 5 output files: SSSSSS/EEE_uvs/JJJJJJJJJJ_B_map.fits — FITS image, SSSSSS/EEE_uvs/JJJJJJJJJJ_B_uvs.fits — self-calibrated, scan averaged visibilities in FITS format, SSSSSS/EEE_uvs/JJJJJJJJJJ_B.mod — ascii file with Clean components of the image, SSSSSS/EEE_uvs/JJJJJJJJJJ_B.win — coordinates of four corners of CLEAN windows used be the imaging process, SSSSSS/EEE_uvs/JJJJJJJJJJ_B.par — command file created by the DIFMAP.

The quality of automatic image may or may not be satisfactory. Automatic image does not perform flagging. If the visibility data are either too high or too low for some IFs due to errors in amplitude calibration, or a portion of data has garbage visibilities at some station(s) because the antenna(s) were not on source, the quality of automatic image will be disappointing at best, or totally garbage at worst. Running **onof** and **gaco** tasks usually solve these problems and substantially reduces the chances that the automatically generated images will have unsatisfactory quality.

In general, analyst should scrutinize automatically generated images and decide whether to keep them or re-image them manually.

imadir.py — Automatic imaging for all files with averaged visibilities in a given directory.

usage: **imadir.py**
[-h] [--version] [-pict] [-H]
directory

This wrapper scans the specified directory, searches file with ending uva.fits or uvm.fits and executes wrapper **automap.py** for each file, i.e. generates automatically the image and picture files

- **-h** — prints a brief information about the wrapper.
- **-H** — print extended manual (this text)
- **--version** — prints the wrapper version.
- **--pict** — generates only picture files from results of imaging without image re-generation. The following pictures gif-format are generated for each image: files SSSSSS/EEE_uvs/JJJJJJJJJJ_B_map.gif and scan-averaged self-calibrated visibilities as a function of baseline length in files SSSSSS/EEE_uvs/JJJJJJJJJJ_B_uvs.gif.

This web page was prepared by Leonid Petrov ( )
Last update: