PIMA User Guide

Date of last modification: `2024.05.13_15:09:46`

*This document describes how to use software **PIMA** for processing VLBI visibility data. PIMA performs data calibration, fringe fitting, and exporting results of fringe fitting in the form that can be digested by VTD/Post-Solve and Difmap software for astrometry/geodesy analysis and for imaging.*

Contents:

Introduction

PIMA is software for processing the visibilities data from VLBI experiments. It performs data inspection, data calibration, and fringe fitting. ***PIMA*** is designed to process multi-source experiments that are common for astronomical surveys and geodesy observations. ***PIMA*** has output interface with AIPS, DIFMAP, and VTD/Post-Solve software.

Principles of *PIMA*

PIMA processes visibility data in FITS-IDI format. ***PIMA*** does not transform and does modify original data. At the first step ***PIMA*** "loads" the data, i.e. examines the specified set of visibility data in FITS-IDI format and creates numerous internal indexing tables that are written in disk. For performing all other operations ***PIMA*** uses these tables for getting access to specific fields of input FITS-IDI files.

PIMA has a flexible command-line interface and it is designed for a non-interactive use. ***PIMA*** is ideal for being incorporated into scripts for shell, python or similar interpreters.

All control parameters that are needed for processing a given experiment are gathered in a control file. ***PIMA*** does not support any defaults: all parameters, even those that are not used for a specific operation, are to be explicitly defined in that file.

PIMA supports the following general syntax:

```
pima control_file task [qualifier value...] [keyword: value...]
```

where

- `control_file` is the name of the control file. The control file contains a list of pairs keyword: value. Keywords are case insensitive, values are case sensitive. The order of keywords is irrelevant. If a keyword is defined more than once, the last definition overrides all previous definitions. The keyword defined in the command line override the keywords defined in the control file.
- `task` is the name of the task performed from the following list:
  - ***acpl*** — autocorrelation plotting
  - ***acta*** — compute average autocorrelation spectrum
  - ***bmge*** — generate bandpass mask for visibility data
  - ***bpas*** — compute a complex bandpass
  - ***bplt*** — plot bandpass
  - ***frib*** — baseline fringe fitting
  - ***frip*** — fringe-fitting with phase referencing
  - ***gean*** — load antenna calibration tables
  - ***gepm*** — automatically generate phase calibration mask
  - ***load*** — load the FITS-IDI files and compute indexing tables
  - ***mkdb*** — make output database
  - ***moim*** — import interferometric model
  - ***mppl*** — plots of multiple tones of phase-cal phases and amplitudes
  - ***onof*** — determine on/off time range automatically by investing visibility data

- ***opag*** — fetch slant path delay, atmospheric opacity and atmosphere brightness temperature on an az/el grid
- ***opal*** — load slant path delay, atmospheric opacity and atmosphere brightness temperature
- ***tsmo*** — to compute the model for Tsys using decomposition of Tsys in the product of time-dependent Tsys in the zenith direction and time-independent Tsys as a function of elevation, remove outliers with respect to that model, remove outliers in the ratios of Tsys with respect to different IFs and created arrays of so-called modeled and cleaned Tsys for all epochs, including the epochs with missing or flagged out measured Tsys. atmospheric opacity and atmosphere brightness temperature
- ***pcpl*** — make a plot of phase calibration signal
- ***pdpl*** — make a plot of the phase calibration signal at the LL polarization with respect to the phase calibration signal at the RR polarization.
- ***pmge*** — generate bandpass mask for phase-cal data
- ***pplt*** — generate polarization bandpass plot
- ***prga*** — print gain information
- ***splt*** — split the data into sources and write output FITS-files
- ***tspl*** — plotting the system temperature
- ***tst1*** — reserved for tests
- ***upgr*** — upgrade control file

- `qualifier value` — additional parameters that are supplied to task. They are specified as a pair qualifier and its value. Some tasks require more than one qualifier. Many tasks do not require qualifiers. The order of qualifiers is irrelevant.
- `keyword value` — additional pairs keyword: value. These pairs override definitions from the control file. These pairs can be viewed as amendments of the control file applied on the fly. If the command line has more than one definition of the same keyword, the last definition takes precedence. If the value of any keyword is omitted, ***PIMA*** will print error message.

PIMA supports a number of optional kludge parameters that alter normal processing in a form of keyword: value. The have a prefix `PIMAVAR_`. They can be either defined in the control file or put in the command line, or defined as environment variables.

Frontend wrappers

PIMA provides several frontend wrappers. They accept arguments, perform some operations and finally call ***PIMA***. A user does not have to use wrappers. They are provided just for convenience. Though, using wrappers sets some restrictions on how to name ***PIMA*** related files. If you are going to use wrappers than the file name are supposed to obey the following convention:

1. Files related to a certain experiment reside in subdirectory `VVVVV`, where `VVVVV` is the root directory of vlbi experiments specified by --pima-exp-dir during configuration.
2. Control file has name `VVVVV/EEE/EEE_B_pima.cnt` where `B` is band name in lower case. For instance, the control file for band `C` (4.3 GHz) for experiment bp192c0 is sought in `/vlbi/bp192c0/bp192c0_c_pima.cnt`, provided ***PIMA*** was configured with `--pima-exp-dir=/vlbi`.
3. Wrappers will create the following files
   - `VVVVV/EEE/EEE_load.log` – log of task ***load*** the database.
   - `VVVVV/EEE/EEE_B_nobps.fri` – results of fringe fitting in the coarse mode.
   - `VVVVV/EEE/EEE_B_coarse.log` – log of fringe fitting in the coarse mode.
   - `VVVVV/EEE/EEE_B_fine.log` – log of fringe fitting in the fine mode.
   - `VVVVV/EEE/EEE_mkdb.log` – log of generation a database in GVF format.
   - `VVVVV/EEE/EEE_splt.log` – log of task ***splt***.
   - `VVVVV/EEE/EEE_B_gain.log` – listing with used antenna gains.
   - `VVVVV/EEE/EEE_B_map.log` – log of automatic data imaging.

Several wrappers are provided. Among them are

- **pu.py** – Fringe fitting. Includes tasks data loading, parse log files, coarse fringe fitting, bandpass computation, fine fringe fitting, data calibration and slitting, generation of the output database in GVF format.
- **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.

Graphic interface

PIMA uses graphics interface DiaGI based on PGPLOT library. DiaGI displays plot into X-window. It allows to resize plot, change plot appearance, inquire a point, make a hardcopy, etc. Refer to DiaGI documentation for details. It is assumed in this manual a reader is already familiar with DiaGI interface. DiaGI documentation can be found here: DiaGI doc-1, Diagi doc-2, and Diagi doc-3.

Minimalistic workflow

The workflow of the minimalistic, simplified analysis:

- ***load*** — parses and loads the data. This is always the first operation. A user is supposed to prepare control file that defines the name of visibility file(s) in FITS-IDI, station catalogue, source catalogue, control file for VTD, experiment description file, and parameters that control further analysis. Results of parsing the visibility data are written in a binary file. For all tasks, except ***load***, ***PIMA*** reads that file immediately after start.
- ***frib*** — performs coarse fringe search without bandpass calibration and masking bad data. Some results of this fringe search will be used for computation of bandpass calibration and automatic flagging. Usually, no oversampling is performed during coarse fringe search.
- ***bpas*** — computes the complex bandpass calibration and complex polarization band-pass (for dual-band data) using results of the coarse fringe search.
- ***frib*** — performs fine fringe search with bandpass calibration applied. Usually, the data are over-sampled with a factor of 4.
- ***mkdb*** or ***splt*** — generate the final product of ***PIMA*** using results of fringe fitting:
  - the database with total group delays, phase delay rates and related parameters for consecutive astrometry/geodesy data analysis;
  - visibilities coherently averaged over frequency and specified time intervals, with averaged visibilities split into files, one file per source, in a form ready for imaging analysis with DIFMAP.

Creation of a configuration file

Supported keywords are described in pima_keywords.html document. Control file contains lines with pairs **keyword: value**. Lines that start with # are considered as comments. The first and the last line of a control file is its label

```
# PIMA_CONTROL file.  Format Version of 2020.04.19
```

Keyword names are in upper case and are terminated by column. Values are case sensitive. If the same keyword is defined more than once, the last definition takes preference. The pair keyword: value defined in the control file are processed as if they appended to the end of the control file. There are two exceptions: **UV_FITS** and **INTMOD_FILE**. More than these keywords are allowed for a case when several input files should be defined.

Some values in ***PIMA*** control file are file names. Although you can define relative file names, defining absolute file names is encouraged.

A user rarely creates a configuration file from scratches. Usually, a control file from a similar experiment is copied to a new name and a user edits it. A user should check carefully every keyword. The following keywords are the most commonly need to change:

- **SESS_CODE** — session code. Usually, this is the experiment code defined in the FITS-file. All names of output files that ***PIMA*** generates will contain this code inside. Therefore, this code should be unique for a given experiment. You can have several trial control files for the same experiment. If you want to distinguish output files generated by this trial control files, you may use different **SESS_CODE**. **NB:** if you change **SESS_CODE**, you must re-run task ***load***.
- **UV_FITS** — name of FITS-files. Obviously, you need to define only these FITS-files that you want to process. NB: if you change the number of FITS-files or rename them, you must re-run task ***load***. If you moved FITS-files to another directory without change of the base name and extension, you do not have re-run task ***load***.

  There may be more than one FITS-IDI file in the processed dataset. In that case, more than one lines with keyword **UV_FITS** should appear in the control file. FITS-IDI files should appear in the chronological order.

  PIMA requires that the number of spectral channels within each intermediate frequency (IF) be the same for each FITS-IDI file. If your experiment has different spectral resolution, ***PIMA*** cannot process it as one experiment. In that case you need to write more than one control file with different **SESS_CODE** keyword and process then separately.
- **SOU_NAMES** — filename with source catalogue. This source catalogue is used for several purposes:
  - defining B1950 and J2000 source names for observed sources. FITS-IDI contains only one name for observed sources.
  - renaming a source. A source catalogue has four columns: IVS name, B1950 IAU name, J2000 IAU name and alternative name. The alternative name column allows to match a non-standard name.
  - defining new coordinates of a source that are different than those used for correlation. Strictly speaking, position of a "source" used for correlation is a position of the center of the field. A source may be off the center of the field of view. If the source is far away from the center of the field, the non-liner term of phase as a function of time may appear significant. ***PIMA*** has an option that allows to compensate this term, but it needs to know the position of the source, not the center of the field.
  - there may be more than one source the field of view. ***PIMA*** allows to "split the source": to define more than one source that corresponds to the same field of view. In order to split the source, put @ in column 32 of *adjacent* rows of the catalogues. Adjacent rows should have the same alternative name defined in column 33:42. You may split the field of view in more than one source. NB: if you split several different fields of view, they should not occupy adjacent rows. Put comment line between different fields of view that are spit into several sources. Internally, ***PIMA*** associate with each source a set of entries of the FITS-IDI file with visibilities.
  - Sometimes it is desirable to swap source names of two sources. Character ^ at the 32th column tells ***PIMA*** to change name of the source specified in columns 33:42 to the name specified in columns 1:8. It will use the a priori positions specified for that sources in columns 46:73. Character ^ at the 32th column is used when we want just to change source name. Using character ^ at the 32th column we can even swap source names. If the field in the column 32 is blank and the field in columns 33:42 contains a source name, ***PIMA*** will search a record with the IVS name equal to the string specified in columns 33:40 and it will use the a priori for that record. Blank at the 32th column is used when we want associated source observed under name A to a known source with name B with its a priori positions.

    The differences between these to cases:

    - when character ^ is used, the a priori specified at the same line are used, while if the 32th columns is blank, a priori for another matching record is used. Since ***PIMA*** applies phase correction due to difference between a priori position specified in the source catalogue and the a priori used by the correlator, the result will be different.
    - when character ^ is used, source name swapping is allowed, but name swapping is impossible when blank is used in field 32.

  Keep in mind that each source should be defined in two catalogues: one catalogue used for association with the center of fields defined in FITS-IDI files. Another catalogue is used for computation of theoretical path delay. That catalogue is defined on VTD control file used by ***PIMA***. The primary source name is the "IVS name" which is B1950 name with same exception. Any observed source (NB: a *source*, not the field!) should have a record in the source catalogue defined in the VTD control file that is associated with the ***PIMA*** source catalogue via the field IVS source name. Internally, ***PIMA*** will use IVS source names, but it also keeps the original name of the center of the field.

  If during task ***load*** ***PIMA*** cannot find a source name(s) in the input catalogue, it issues an error message that contains names and coordinates of all missing sources at the beginning of the task. If ***PIMA*** cannot find a source name in VTD source catalogues, it issues an error message that contains names and coordinates of all missing sources at the end of the task.
- **STA_NAMES** — filename with station catalogue. Each station that participated in the processed VLBI experiment should have a record in the station catalogue. The station catalogue has several columns. The first column is the name of the station used by the correlator. It may use up to 8 characters. The second column contains a standardized IVS 8-character long station name. The main purpose of this station name is to match the station name used by the correlator and the IVS name. Internally, ***PIMA*** will use the IVS station name, but also keeps the original name.
- **EXPER_DIR** — name of the scratch directory where ***PIMA*** will write some results, including intermediate files.
- **MIN_SCAN_LEN**, **MAX_SCAN_LEN**, **MAX_SCAN_GAP** — these parameters control the algorithm for splitting the dataset into scan. Depending the goals of your experiment, you may adjust the strategy for splitting the data into scans.
- **BANDPASS_FILE**, **BANDPASS_MASK_FILE**, **PCAL_MASK_FILE**, **POLARCAL_FILE**, **FRINGE_FILE**, **FRIRES_FILE** — obviously, these files are specific for a given experiment and should have unique names. It is a good practice to keep all these files in the same experiment specific directory.
- **MKDB.OUTPUT_NAME** — defines either suffix if **MKDB.OUTPUT_TYPE**: **GVF** or the file name if **MKDB.OUTPUT_TYPE**: **TEXT**. The name should be experiment specific. If more than one experiment has the nominal start time at the same day, the suffix should be unique. Please check the suffix carefully. Otherwise, ***PIMA*** may override existing database for different experiment!
- **EPHEMERIDES_FILE** — if you process VLBI experiment with RadioAstron, you need to specify the relevant ephemeride file that covers the time interval of the experiment.
- **MKDB.DESC_FILE** — this file defines auxiliary information specific for this experiment. You need to copy the description file for similar experiment to another name and edit accordingly.

Loading the data

Task data loading is the first task. ***PIMA*** parses the control file, finds the data with visibilities in FITS-IDI format, and reads them. It gathers information about station names, sources names, frequencies, a priori models, system temperature, gain, weather information, phase calibration, cable calibration, etc. It reads all cross-correlations and auto-correlations, associates them, and checks for their consistencies. Then ***PIMA*** splits the data into scans and observations. It creates scan tables, observation tables and associates observations with indices of visibilities. All tables are written into a binary file `SSSSS/EEE.pim`, where `SSSSS` is the scratch directory specified in the keyword **EXPER_DIR** and `EEE` is the experiment name specified in the keyword **SESS_CODE** of the ***PIMA*** control file. Data loading takes from 20 seconds for small experiments to 2–4 hours for experiments with visibility files of terabyte size. All other ***PIMA*** operations will read file `SSSSS/EEE.pim` and use indexing tables. Unlike to AIPS or CASA, ***PIMA*** does not rewrite input visibility data in its own format. Instead of it, it creates indexing tables and uses this tables when it needs to collect visibilities for processing a given observation. The advantage of this approach is that no intermediate files is created. The disadvantage is that reading of visibility data may become inefficient when if the input data file that reside on magnetic hard-drive are larger than than the amount of available operative memory due to limitations related to a design of FITS-IDI data. ***PIMA*** does not need input information about learning how the data are split into scans: it does it itself. The advantage of this approach is that ***PIMA*** will process the data even if any auxiliary information is lost. The disadvantage of this approach is that ***PIMA*** can split the data into scans not the same way as an observer designed the experiment.

The first operation of task ***load*** is parsing control file. VTD control file specified in ***PIMA*** control file is also parsed. Finally, the experiment description file specified in the keyword **MKDB.DESC_FILE** is parsed. Any errors, such as syntax errors or files that do not exist are reported. ***PIMA*** will stop and issue an error message in a case of errors.

In the next step ***PIMA*** will check every source name first in the file specified in keyword **SOU_NAMES**, then in catalogue files specified in VTD control file. If it finds at least one source not in the catalogue, ***PIMA*** will issue the error message and print the list of missing source names and their coordinates extracted from the FITS file.

Then ***PIMA*** will check every station name first in the file specified in keyword **STA_NAMES**, then in catalogue files specified in VTD control file. If it finds at least one station not in the catalogue, ***PIMA*** will issue the error message and print the list of missing station names and their coordinates extracted from the FITS file.

Then ***PIMA*** check frequencies in each files and creates the global frequency table for the entire experiment. It converts low side band intermediate frequencies tables (LSB IF) into upper side band IFs by re-ordering frequencies of the channels within each IF for them to following in the ascending order. It merges or combines frequency groups if requested. Finally, it tables of cross indices from the original frequency tables frequency groups to the global frequency table, frequency groups and vice versus.

Next step is to read all visibility data. Visibility data are sorted, cross-correlation data are linked to autocorrelation data, and tables of time indices, cross-correlation indices and auto-correlation indices are created. ***PIMA*** checks for organ visibilities: cross-correlation visibilities without autocorrelation and auto-correlation data within matching cross-correlation data. These visibilities are added to the list of "bad data".

Then ***PIMA*** splits the data into scans. By that time the data are chronologically sorted. There are three parameters in the ***PIMA*** control file that controls the process of data splitting: **MIN_SCAN_LEN**, **MAX_SCAN_LEN**, and **MAX_SCAN_GAP**. ***PIMA*** sets a preliminary scan boundary when a source is changed. If it does not find valid visibilities for **MAX_SCAN_GAP** seconds after the last valid visibility of the previous source, it sets the end of scan of the previous source. If duration of the time from the first valid visibility of a given scan is longer than **MAX_SCAN_LEN**, a border of a scan is set, and a new scan starts. That means that a scan cannot be longer than **MAX_SCAN_LEN** seconds and it cannot have a gap longer than **MAX_SCAN_GAP**. At the same time scans of different sources may overlap, i.e as scan B may have start and stop time within the interval of start and stop time of the scan A. Scans shorter than **MIN_SCAN_LEN** seconds are eliminated and the visibilities within such short scans are marked as bad.

The choice of **MIN_SCAN_LEN**, **MAX_SCAN_LEN**, and **MAX_SCAN_GAP** is determined by scheduling goals and the correlator setup. Usually **MIN_SCAN_LEN** is set to have at least three accumulation periods, otherwise fringe fitting process may fail. For non-phase referencing experiment **MAX_SCAN_LEN** can be set to the scan length set by the schedule. Experiments at 22 GHz and higher **MAX_SCAN_LEN** can be set shorter to be close to the coherence time. **MAX_SCAN_GAP** can be set to 1/2 of the scan length to prevent scan split in a case of data loss within a scan. For scan-referencing observations **MAX_SCAN_LEN** is set to the cycle duration and **MAX_SCAN_GAP** is set to 90% of **MAX_SCAN_LEN**. It should be noted that ***PIMA*** allows to use a portion of a scan in data analysis, but it cannot unite two scans. Parameter **SCAN_LEN_USED** and **SCAN_LEN_SKIP** allows to set up continuous portion of a scan for fringe fitting and split after ***load*** task. But ***PIMA*** cannot increase scan length after task ***load*** is done. If a user needs to change scan allocation or increase scan length, task ***load*** should be re-run. NB: if a new run of task ***load*** changes the total number of observations, fringe fitting should be re-run, since the stale fringe results have different scan and observation indices.

After ***PIMA*** split the data into scans, it checks all cross- and auto- visibility data whether they are claimed by scans. All visibilities not claimed by scans are marked as bad.

If ***PIMA*** finds at least one bad visibility, ***PIMA*** stops withe an error message. Since getting bad visibilities is rather a common situation, ***PIMA*** has a mechanism to accommodate them. ***PIMA*** control file supports keyword **UV_EXCLUDE_FILE** that defines a file with indices of visibilities, either cross or auto, that are to be excluded at the very beginning. These visibilities are excluded from analysis, and ***PIMA*** cannot mark them bad because it does not see them. ***PIMA*** supports a special value of parameter **UV_EXCLUDE_FILE: AUTO**. If value **AUTO** is specified, than when ***PIMA*** finds bad points, it writes visibility indices in the so-called bad visibility file at `SSSSS/EEE_uv.exc` file, where `SSSSSS` is EXPER_DIR and `EEE` is **SESS_CODE**. If that file already exists, ***PIMA*** appends new visibilities to that file. When **UV_EXCLUDE_FILE: AUTO**, tasks ***load*** is executed several times. The first time ***PIMA*** finds bad points, puts them in the `SSSSS/EEE_uv.exc` file and stops with the exit code 23. The second time the bad points in `SSSSS/EEE_uv.exc` are read and excluded from the subsequent analysis. Usually two runs are sufficient. Sometimes the 3rd and 4th is required. Wrapper **pf.py** executes the 2nd, 3rd and 4th run automatically. NB: **pf.py**purges `SSSSS/EEE_uv.exc` file if it exists.

After splitting the data into scans, ***PIMA*** reads and parses phase calibration, system temperature, weather information, interferometric model and interferometric model components. Any these parameters may be missing in the FITS-IDI file. In such cases ***PIMA*** issues a warning, but proceeds.

If **PCAL**: **NO** is specified in the control file, ***PIMA*** will skip phase calibration information present in the FITS-IDI file(s). Keep in mind, if **PCAL**: **NO** was specified during loading, ***PIMA*** cannot re-enable phase calibration later within running task ***load*** again. If phase calibration was loaded and cane be disabled for entire experiment or for the specified station(s) and re-enabled again. If phase calibration is not available for some scans at some stations, such observations are flagged as bad and are skipped for fringe fitting and other operations, unless phase ca libation is disabled for the entire experiment by specifying **PCAL**: **NO** or by disabling pcal at both stations of the baseline of that observation. It should be noted that bandpass and fringe results will be different whether phase calibration was used or not. Therefore, if phase calibration status was changed, bandpass should be re-generated and fringe fitting re-done.

Correlator organzies data by spectral channels, intermediate frequencies (IFs) and frequency groups. Strictly speaking, fringe fitting can be done only within one frequency group ***PIMA*** has two ways to circumvent this resutriction.

If UV data from several frequency groups have the same time tag, such freuqnecy groups are called overlapping. Overlapping frequency groups can be merged to a new virtual group. When task ***load*** is executed with **FRQ_GRP: m:n**, where `m` and `n` is a range of the frequency groups, a new virtual groups created that merges IFs of frequency grops from `m` to `n`. The number of IFs of the new virtual group is `(m-n+1)*Num_IF`. The new virtual group has index 1 and has (m-n+1)*Num_IF IFs, where Num_IF is the number of IFs in orginial groups. After loading either **FRQ_GRP: m:n** or **FRQ_GRP: 1** forms can be specified to us the merged group. Other forms of **FRQ_GRP**, such as **FRQ_GRP: 2** are not accepted.

If UV data from several frequency groups have the different time tag, such groups are not overlapping. A common usual case: change of the reciever or the backend setup while the antennas are on source. The non-overlapping groups can be combined into a new virtial group by involiing task ***load*** with **FRQ_GRP: m-n**, where `m` and `n` is a range of the frequency groups. The number of IFs of the new virtual group is (m-n+1)*Num_IF. The new virtual group has index g+1, where g and has `(m-n+1)*Num_IF` IFs, where Num_IF is the number of IFs in orginial groups. After loading data, the virtial group can be accessed as **FRQ_GRP: g+1**. Other forms of **FRQ_GRP**, such as **FRQ_GRP: 1** are not accepted.

It is important to note that virtial frequency group are created when running task ***load***. Just calling ***PIMA*** with **FRQ_GRP: m:n** or **FRQ_GRP: m-n** after loading will cause an error message, unless the virtial frequency group has been created by task ***load***.

By 2016.01.01 only VLBA put model and all calibration information into the FITS-IDI data. Lack of calibration information does not prevent ***PIMA*** to run fringe fitting by may prevent further tasks. For instance, task ***splt*** cannot run if no Tsys and/or antenna gains is available. Task ***mkdb*** cannot run if the interferometric model is not available. VLBA hardware correlator and DiFX version 2.0 and newer puts phase calibration information into FITS-IDI. Other correlators do not to do it. Missing weather information, Tsys can be loaded by ***PIMA*** task ***gean*** using results of parsing log-files. Missing antenna gains can be loaded by ***PIMA*** task ***gean*** from external gain files. Missing interferometric for VERA, SFXC, and KJCC correlators can be loaded by task ***moim*** from external model files in native format that were used by the correlator.

Example: processing an experiment correlated with SFXC (JIVE correlator). A user need collect all delay files and one clock files. The delay files and the clock file need be put either in a separate directory that keeps the interferometric data for this specific experiment or a tree of sub-directories. The delay files are in a binary format SFXC supports two formats pre-2020 and post-2020. ***PIMA*** recognises them automatically. **INTMOD_FILE:** keyword specifies a file or a directory with delays. More than one keyword **INTMOD_FILE:** can be specified. ***PIMA*** will find all files with extension ".del", extract the apriori interferometric model, sort it and puts in the internal data structure. It will also search for a file with extension ".clk" with the clock model.

Then the ***PIMA*** control file should have these definitinos: **INTMOD_FILE: directory_name_with_del_files**
INTMOD_TYPE: SFXC

NB: If the data are re-loaded, task ***splt*** should be repeated.

SFXC Format description of 2024.05.13:

```
SFXC post-2020 inteferometric model format supports adding extra
rows of delay model before and after each scans. This makes some
things easier in the correlator.

How many extra rows there are is controlled by a variable called
n_padding, the value of which is stored in the delay file header.
So there are n_padding extra rows before and after each scan.

Also new in the header is a version number.  The currently values are
version_number = 1, and n_padding =1.

The scan name is repeated for every source, even if there are multiple
sources in a scan.

Below is the delay file format in pseudo code. Between the square
brackets is the data type and the number of elements.

    [int32_t] # header size excluding this variable
 [int32_t] # version number
      [int32_t] # number of extra rows before and after each scan
   [(header_size - 8) * char]
* For each scan in the experiment
    * For each source in the scan
```

```
      [char * 81]       # Null terminated scan name
      [char * 81]     # Null terminated source name
      [int32_t]               # The modified julian day at the
                                    start of the scan
     * The model values sampled once per second
     time  U  V   W   delay   phase   amplitude [double]
     0 0 0 0 0 0 0 0 [double]   # The end of a scan (for each source)
                                  is marked with a row of zeros

Here time is the number of seconds since midnight on the day the scan
starts. U, V, W are in meters, and delay is in seconds.
```

Parsing log files

This is the most frustrating part of data analysis. If you have data from VLBA, you do not need to run parsing log files. Parsing log files from the KVN and VERA is very straightforward. Unfortunately, parsing log files generated by the Field System developed in the Goddard Space Flight Center often fails, because the format of field system log file is changed without notice, and the developer who maintains the field system refuses to cooperate.

PIMA can directly import log file in VLBA format or in the ***PIMA*** ANTAB format. Non-VLBA logs are parsed by program log_antab and transformed to ***PIMA*** ANTAB format. Program log_antab extracts system temperature, if present, nominal on-off time tags, cable calibration and meteorological information. Modern VLBI analysis does not use in-situ meteorological information, and uses instead of that the output of numerical weather model. The current version of ***PIMA*** does not use nominal on-off time tags from log files, since this information is already used by the correlator. ***PIMA*** can compute on-off from data actual time tags when the antenna was on source. The use of cable calibration in analysis is discretion and rarely improves the fit, and sometimes significantly degrade it. But the use of system temperature is critical for imaging. When ***PIMA*** produces the calibrated averaged visibilities, it discards observations without system temperature.

Syntax of the program for parsing log-files:

```
Usage: log_to_antab {mode} {log_file} {antab_file} [year]
```

where

- MODE = 1 — for IVS log-files after 2008.
- MODE = 2 — for IVS log-files in approximately 1999–2002.
- MODE = 3 — for IVS log-files in approximately 1996–1996.
- MODE = 4 — for IVS log-files in approximately 1996–1999.
- MODE = 5 — DBBC log file with USB/LSB pairs of BBCs.
- MODE = 11 — for KVN log-files

The main difficulty is in extraction system temperature from field system logs. The parsing software needs to identify Tsys record, extract the array of Tsys and match that array with sky frequencies. It needs to determine intermediate frequency with respect to the frequency of the local oscillator, to determine the frequency of the local oscillator, match them, find tsys record, determine to which BBC a field in tsys record belong and match the field.

NB: An analyst should always examine the output of log_antab program. Typical failure: log_to_antab fails to determine sky frequencies. Possible reasons: a log file may have a portion at the beginning or at the end that is related to another experiment, a new change of log format. In the first case, editing a log file solves the problem. In the latter case you need to patch log_to_antab. Please try not to break its ability to parse other log files. If everything else fails, you can either develop your own parser or to parse a log file by hand. Keep in mind that some station do not record Tsys at all.

Wrapper **pf.py** supports log parsing. The following command does this:

`usage:` **`pf.py EEE B`** ***`logs`***

where `EEE` is the experiment name and `B` is the band. It creates output file `EEE_AA.ant`, where `AA` is a two character long low case antenna name.

Calibrating the data

FITS-IDI visibility file is supposed to have all calibration information inside. However, only Socorro correlator inserts all calibration information into FITS-IDI. Visibility files from all other correlator missing some or all calibration information.

Although the visibility data from the old hardware VLBA correlator has all calibration information, it is recommended to re-load it since in some cases the calibration information is not correct and re-loading fixes the problem. There is no need to reload calibration information to FITS-IDI files generated in Socorro by DiFX 2.0 and newer.

Field System log files contain a) on-off start/stop scan time; ' b) meteorological information; c) cable calibration; d) frequency table; e) system temperature. VLBA log files contain phase calibration phases and amplitudes in addition to that.

PIMA task `gean` inserted the calibration tables into ***PIMA*** internal data structures. Task ***gean*** requires a qualifier that is followed by the value. The following qualifiers are supported:

- `pima_antab_file` — loading Tsys, cable calibration, and meteorological information in ***PIMA*** ANTAB format. The value of this qualifier is the file name. One file contains calibration information for one station.
- `vlba_log_file` — loading Tsys, cable calibration, phase calibration and meteorological information in VLBA calibration format. The value of this qualifier is the file name. One file contains calibration information for all stations that have VLBA data acquisition terminal, i.e ten VLBA stations and EFLSBERG. The NRAO pipeline names this file as `EEEcal.vlba`, where `EEE` is the name of the experiment.

  Comment 1: After adopting DBBC in 2013, the NRAO has changed the data processing chain. It still provides legacy calibration file, but that legacy calibration is inadequate for processing DBBC data. **You should not load legacy calibration into *PIMA* when processing DBBC NRAO data.**

  Comment 2: Although FITS-IDI from analogue NRAO observations prior 2013 contains phase-calibration, system temperature, phase calibration, and phase calibration, it is desirable to re-load calibration into ***PIMA*** using task ***gean***. The instances when calibration information into FITS-IDI supplied by the NRAO was incorrect were found.

  Comment 3: ***PIMA*** issues warnings about missing phase-cal and Tsys. If ***PIMA*** uses phase calibration and for a given observation phase calibration is missed, ***PIMA*** will declare that observation as "bad" and will not perform fringe fitting. Though ***PIMA*** will process such an observation if **PCAL**: **NO** is specified in the control file. As of 2016.01.17 ***PIMA*** task ***splt*** will bypass observations with missing system temperature.
- `vlba_gain` — inserts antenna gain information stored in VLBA gain file into ***PIMA*** data structures. The value of the qualifier is the file name. The antenna gain file is supposed to the VLBA gain format. This file can be found at http://www.vlba.nrao.edu/astro/VOBS/astronomy/vlba_gains.key. It is updated several time a year. ***PIMA*** will update gains only for stations that are defined in this file and do not change gain of other stations.

  Comment: Although FITS-IDI generated by NRAO at Socorro contains antenna gains, it is desirable to re-load calibration into ***PIMA*** using task ***gean***. The instances when old calibration information into FITS-IDI supplied by the NRAO was incorrect were found.
- `evn_gain` — inserts antenna gain information stored in the EVN antab format file into ***PIMA*** data structures. The value of the qualifier is the file name. ***PIMA*** will update gains only for stations that are defined in this file and do not change gain of other stations.
- `pcal_off` — turns the phase calibration off for a given station. The value of this qualifier is the station name. ***PIMA*** turns off the flag of phase calibration availability. As a results ***PIMA*** considers that phase-calibration is unavailable for that station. The flag can be turned on back. If pcal was turned off for a given station, it cannot be turned by a fine-grained **PCAL** option.
- `pcal_on` — turns the phase calibration on for a given station. The value of this qualifier is the station name. This operation undoes operation pcal_off and sets flag of phase calibration availability. Of course, this operation will have effect only if there are phase calibration data loaded in ***PIMA*** internal data structure. If pcal was turned on for a given station, it can be turned off by a fine-grained **PCAL** option.

- `tsys_off` — turns off Tsys for a given station. The value of this qualifier is the station name. This task turns Tsys availability. As a results ***PIMA*** skips that station for imaging. The flag can be turned on back. Turning Tsys off may be necessary if Tsys is missing or corrupted, or a user wants to avoid using that station for imaging for any reason.
- `tsys_on` — turns Tsys on for a given station. The value of this qualifier is the station name. This task undoes `tsys_off` and makes Tsys availability if Tsys data are loaded in ***PIMA***.
- `wvr` — processes water vapor radiometer data into ***PIMA*** internal data structures. This qualifier has value either **load**, **plot1**, and **plot2**. All these operations load all WVR data specified in the keyword(s) **WVR_FILE** in the control file. If the value of `wvr` qualifier is **plot1**, then in addition to loading, ***PIMA*** will generate a plot of WVR path delay versus time. If the value of `wvr` qualifier is **plot1**, then in addition to loading, ***PIMA*** will generate a plot of WVR path delay versus elevation.

NB: Antenna gain, system temperature, meteorological information and cable calibration does not change result of fringe fitting. Therefore, these calibration can be applied after fringe fitting. Phase calibration affects result of fringe fitting. Therefore, it is supposed to perform this kind of calibration before fringe fitting. If you changed phase calibration, phase calibration status (pcal_on, pcal_off), you have to redo bandpass calibration and fringe fitting. Otherwise, you will get **wrong results**.

Examine raw data and calibration information

An analyst must always examine the data and calibration information before running fringe fitting as carefully as possible. If an error will not be noticed at the initial examination, then the analysis will be have be redone. Attentiveness during early examination saves time and reduces the probability that the error will not be noticed and will lead to an erroneous result.

- Examining gean log files. This file has information about possible errors in FITS-IDI. Keyword **CHECK_SEVERITY:** 2 in the control file will cause ***PIMA*** stop at many errors. Keyword **CHECK_SEVERITY:** 1 will allow to continue loading with a damaged visibility file. However, ***PIMA*** recovery algorithm may be too permissive. Sometimes, data may require flagging after loading corrupted FITS-IDI.
- When ***PIMA*** successfully loads the data, it creates a number of dumps files. They are created to facilitate inspection of the data.
  - Examining statistics file. Upon successful loading, ***PIMA*** creates statistics file and `SSSSS/EEE.stt`, where is `SSSSS` the ***PIMA*** scratch directory specified in **EXPER_DIR** keyword of the control file, and `EEE` is the lower case experiment name. An analyst should check among other things a) whether all FITS-IDI were read; b) whether the data have all polarizations, c) whether the data for all time range were read (nominal start and stop date); d) whether data from all stations were read.
  - Examining the source list. Upon successful loading, ***PIMA*** writes the source file dump in `SSSSS/EEE.sou`. This file is just the ascii dump of the internal source table. You should examine source names and a priori source coordinates. ***PIMA*** gets a priori source coordinates from the catalogue file. In general, the a priori source coordinates are not the same as the coordinates used by the correlator. However, if the differences are large (say, more than 10 mas), an analyst should be aware of that. Large discrepancies between the a priori coordinates used by the correlator and those specified in the catalogue may trigger a parabolic phase correction. This corrections needed if the coordinates used by the correlator were not precise. But this correction will degrade the results and even cause a non-detection if the a priori coordinates in the catalogue are wrong, f.e. a wrong source name was associated.
  - Examining the frequency list. ***PIMA*** writes the frequency file dump in `SSSSS/EEE.frq`. ***PIMA*** sorts the frequencies and transforms low side band data with frequencies running in the decreasing order into upper side band data with frequencies running in the increasing order. ***PIMA*** control file supported keywords **BEG_FRQ**, **END_FRQ**, and **FRQ_GRP** that have intermediate frequency or frequency group indices as their values. An analyst should be aware to which sky frequencies these indices correspond.
  - Examining the station list. ***PIMA*** writes the frequency file dump in `SSSSS/EEE.sta`. The file has information about station names, station coordinates, and number of Stokes parameters.
  - There are other dump files, but normally they should be inspected in a case of problems. File with extension `.obs` lists the observations, observing stations, observed sources, start and stop time. File with extension .sca lists the scans, observed sources, stop and start dates. File with extension `.tim`lists time tags of valid visibilities. File with extension .mod lists start and stop time for parameters of the interferometric model. File with extension .mdu shows association of observations with intervals of interferometric model. File with extension .mdc lists parameters of clocks used in the correlator model. File with extension .mda lists parameters of the atmosphere path delay used by the correlator. Very long file with extension .uv lists all cross-correlation and autocorrelation visibilities. File with extension .dup shows duplicate visibilities. File with suffix _uv.exc lists indices of excluded visibilities if **UV_EXCLUDE_FILE: AUTO** was specified in the ***PIMA*** control file.
- Examining phase calibration. ***PIMA*** has a special task ***pcpl***. It is useful for examining 1 to 8 phase calibration tones per IF. If there are more phase cal tones, the plot become crowded what makes it difficult to read. An analyst should make a decision whether phase calibration signal is useful or not. Keep in mind, DiFX correlator will extract signal at frequencies where it supposed to be regardless whether phase calibration unit was on or off. If the unit was off, phase calibration phases will be a noise, and applying such phases will ruin observations completely. ***PIMA*** task ***gepm*** will catch this case and mask out such tones.

  What to look? First to look at phase cal phases. Phase cal scatter with respect to a smoothed curved should not be excessive (more than 0.3 rad). Sometimes phase calibration may vary significant with time. Plot of "phase cal relative f0" (R), i.e. differences of phase calibration phases with respect to the phase at the lowest frequency is helpful in this situation. Another useful statistics is "phase cal amplitude" (M). There are several factors that causes variation of phase-cal amplitudes. Phase calibration amplitude is proportional to T_sys, which depends on elevation and may depend on time. The second factor is presence of spurious narrow-band signal(s) generated by the hardware. This signal distorts phase and amplitude of the phase-calibration signal. If front-filters are not tuned well, the phase calibration signal at the image sub-band may distort phase and amplitude of the phase cal signal at the primary sub-band. Plot of "phase amp versus phase" (V) help to reveal the presence of spurious signals. There is no dependence of phase calibration amplitude on phase if the hardware is perfect. Sinusoidal pattern indicates the presence of spurious signals. Spurious signal with amplitude less than 10% of the average amplitude are usually harmless, while the use of phase calibration with spurious signals with the amplitude 50% may significantly degrade results. ***PIMA*** task ***gepm*** implements all of these heuristics in a repeatable and automatic way. After running ***gepm***, it is a good idea to look at phase calibration once more to determine whether any spurious signals remain.

  An analyst should make a decision whether to keep phase calibration for a given station or not. ***PIMA*** allows to disable phase calibration for any given station or for all stations. In order to disable phase calibration for all the stations, **PCAL**: **NO** should be specified in the control file. Task ***gean*** allows to disable phase calibration for a given station. It requires qualifier pcal_off that needs a value: station name. If to run task ***gean*** with qualifier **pcal_on**, the phase calibration for a given station will be enabled. **NB:** if you loaded the experiment with **PCAL: NO**, ***PIMA*** does not read phase calibration, and therefore task ***gean*** cannot be enabled it. You need to load the experiment again in order to enable phase calibration.

  Task ***pcpl*** supports an optional qualifier **pcal_type** that can take value **raw** and **average**. By default, ***PIMA*** averages phase calibration within a scan, and task ***pcpl*** shows averaged phase calibration phases and amplitudes. If **raw** is selected, no averaging is performed.

  In addition, ***PIMA*** provides a fine-grained mechanism for toggling status to use or not to use pcal for given stations. Keyword **PCAL** supports a qualifier that provides a station list.

  Syntax:

  PCAL: **value[:action:[station[:station]...]**

  A separator : (column) or , (comma) between stations is allowed.

  Action is either TO_USE or NOT_TO_USE. The action is case insensitive. If action is TO_USE, then pcal only from the stations form the list will be used. If action is NOT_TO_USE, then pcal from the stations on the list will not be used. Example:

  ```
  PCAL:   USE_ALL:NOT_TO_USE:MEDICINA:NYALE13S:RAEGSMAR:YARRA12M
  ```

  Here phase calibration from the following stations, MEDICINA, NYALE13S, RAEGSMAR, YARRA12M will not be used.

  ```
  PCAL:   USE_ALL:to_use:HART15M,KOKEE,WETTZELL
  ```

  Here phase calibration only from the following stations, HART15M,KOKEE,WETTZELL will be used provided the phase is available and was not turned off with task gean.

  Fine-grained pcal station selection can change phase calibration use status **only if pcal is available and was not turned off using task *gean***.
- Examining system temperature. ***PIMA*** task ***tspl*** displays system temperature for a given IF. By default, ***tspl*** shows Tsys for the first IF. The IF index can be changed by hitting box "Frequency selection" (V). Tsys can be displayed versus time (T) and versus elevation (E). Tsys can be decomposed in the product of Tsys in the zenith direction as a function of time and the Tsys elevation dependence. The first part, Tsys in the zenith direction is show by button (Z). The second, Tsys as a function of elevation angle is shown by button (D).

  What to look? If ***PIMA*** shows no Tsys, that means it was not loaded. If need to check log file and if possible, to fix. Then you need to re-run task ***gean***. Sometimes Tsys is so noisy or wrong that keeping such a station will degrade reconstructed source images. In that case bad Tsys in certain IFs can be disabled by editing so-called gain correction file specified by the **GAIN_CORRECTION_FILE** control file. ***PIMA*** task ***load*** creates and initializes it if the file specified by that control file does not exist. The gain corrections file specifies for each station, each IF a factor that ***splt*** will multiplies Tsys. ***PIMA*** does not modify the file if it exist. An alternative way to initialize gain correction file is to run task ***gaco*** with qualifier **init**. That qualifier requires a value either fill or overwrite. Value **fill** instructs ***PIMA*** to add missing records: if for some IFs, some stations the gain correction was not defined, ***PIMA*** will add record with correction equal to 1 (i.e. no

correction). If the qualifier init has value overwrite, *PIMA* will overwrite previous definitions with 1. If for a given IF, given station the gain correction is 0.0, then *PIMA* task ***splt*** will set amplitude zero and such an IF will not be used for imaging.

- Running a trial fringe fit for a given observation. It has sense to look at fringe plot of several scans of a strong source. A number of strong sources are usually observed in a well designed experiment. Examining dump with extension `SSSSS/EEE.obs` helps to find indices of observations of strong sources. Wrapper **pt.py** is useful for running a trial fringe fit. It has the following syntax:

  Usage: **pt.py [-pt optios] EEE B obs [pima_opts]**

  where is the low case experiment name, band is the low case band name, and obs is the observation index. These mandatory arguments may be followed by additional arguments of the command line for *PIMA* that are in the usual format keyword: value. The wrapper itself supports options --dry-run (-r) and --verbosity (-v). Option --dry-run just shows the *PIMA* command line without execution. Option --verbosity requires a value. Value **0** means no informational messages, value **1** (default) moderate verbosity and values **2** and **3** more and more verbose output.

  If the *PIMA* control file defines **BANDPASS_FILE**, and/or **BANDPASS_MASK_FILE**, and/or **PCAL_MASK_FILE**, and/or **POLARCAL_FILE** that do not exist, wrapper **pt.py** replaces them with **NO** and issues a warning.

  pt.py displays two fringe plots: versus frequency (and averaged over time) and versus time (and averaged over frequency). If a source is weak, the plot may look too noisy. Keyword **FRIB.1D_FRQ_MSEG** averages the data over frequency after performing fringe fit and before plot preparation. The value of the keyword specifies how many spectral channels are coherently averaged out. This parameter should not exceed the total number of spectral channels in an IF. Analogously, **FRIB.1D_TIM_MSEG** averages the data over time after performing fringe fit and before plot preparation. The value of the keyword specifies how many accumulation periods are coherently averaged out.

  What to look? First, whether the source is detected. As a rule of thumb, SNR > 7.0 and higher indicates a reliable detection, SNR in a range of [6.0, 7.0] is a marginal detection, SNR in a range [5.1, 6.0] is unlikely a detection, and SNR < 5.1 usually is a non-detection. If an observation you picked is a non-detection, try another. If *all* observations are non-detections — bad luck, you can stop analysis on this point. Nothing can be done.

  Since no bandpass calibration is applied at this point, the phases are not aligned. However, the residual fringe phases should follow a more or less a smooth line for a high SNR observation. Jumps, or low amplitudes at some IFs raises a concern. Phase behavior at individual IFs can be examined by running **pt.py** with specifying the IF under consideration with keywords **BEG_FRQ** and **END_FRQ**.

Running coarse fringe fitting

The goal for coarse fringe fitting is preparation for bandpass computation and for initial data examination. Coarse fringe fitting uses single polarization data (RR for dual-polarization data), does not use bandpass, because usually bandpass is not known at that time, and uses no oversampling in order to speed up computation, and performs the parabolic fine fringe search. Therefore, the following parameters are always set:

```
BANDPASS_USE:           NO
BANDPASS_FILE:          NO
POLARCAL_FILE:          NO
FRIB.OVERSAMPLE_MD:     1
FRIB.OVERSAMPLE_RT:     1
FRIB.FINE_SEARCH:       PAR
MKDB.FRINGE_ALGORITHM: DRF
```

If the bandpass mask file is not available, then **BANDPASS_MASK_FILE:** NO is set. **POLAR:** RR is set for dual-polarization or RR data and **POLAR:** LL is set to LL-polarization data. Usually *PIMA* runs in both coarse and fine fringe fitting mode. It is desirable to store results of coarse and fine fringe fitting in separate files. Therefore, when we run coarse fringe fitting, we set **FRINGE_FILE** and **FRIRES_FILE** into different files than those specified in the *PIMA* control file.

To run coarse fringe fitting, task ***frib*** is used. *PIMA* wrapper **pf.py** simplifies running coarse fringe fitting. Syntax:

```
Usage: pf.py EEE B coarse
```

where EEE is the experiment name and B is band in lower case. Wrapper will write fringe result in `VVVVV/EEE/EEE_B_nobps.fri` and fringe fitting residuals into `VVVVV/EEE/EEE_B_nobps.frr` files.

Computation of a complex bandpass

Computation of the complex bandpass is the second major task that requires human intervention. The bandpass of the ideal system is rectangular shape for the amplitude and zero for phase. That means that cross-correlation spectrum of a signal from a radio sources with continuum flat spectrum is also flat with some constant phase offset and the multiplicative factor that is proportional to the square root of the products of Tsys at both stations. Unfortunately, up to date perfect VLBI hardware is not yet developed. The cross-correlation spectrum diverts from the ideal (flat phases and flat amplitudes). The use of phase-calibration may alleviate the deviation from the ideal spectrum, may have no visible effect or may even degrade it, but never fixes phases. Therefore, normally a complex function of frequency is computed that being multiplied by the cross-spectrum makes it flat. This function can be computed reliably using sources with SNR > 200, but preferably with SNR > 1000. A good principle investigator does not hesitate to spend a sizable amount of allotted time for observing bright sources that are used as calibrators. Bandpass calibration can still be performed using source with SNR 40–200, but less reliable. Quality of bandpass derived using observations with SNR in a range 10–40 is questionable. Bandpass calibrator sources are supposed to be continuum with the spectrum flat within an IF. Any active galaxy nuclea satisfies this condition.

In a case when phase calibration is applied, the bandpass is computed with respect to the phase calibration. If you use all tones of phase calibration, you should first clean them and mask out the tones affected by internal radio interference (spurious signals).

It is assumed the residual bandpass is stable with time. An experiment may have jumps in the bandpass due to power-off power-on of the VLBI hardware at one or more stations. Unfortunately, as of 2016.02.01 *PIMA* does not provide a convenient way for processing such data. The workaround is to effectively split the dataset into two subsets before and after the jump and compute two bandpasses. Splitting the dataset can use made using keywords **OBS**, **INCLUDE_OBS_FILE**, and **EXCLUDE_OBS_FILE**. Fortunately, jumps in bandpass occur in less than 5% VLBI experiments.

An analyst usually sets the mask for cross-spectrum data during this step. The mask is an array of 1 and 0 that depends on spectral channel index. *PIMA* multiplies the mask by visibilities when it processes the data. Zeroes in the mask effectively replaces the spectral channels with zeroes. Usually unwanted potion of the spectrum is masked out: either affected by RFI or affected by hardware bandpasses. If a signal is narrow-band, for instance from an stellar maser, then masking allows to discard spectral channels that have no signal, but only noise.

Cleaning phase calibration

If you use four or more tones of phase calibration per IF, you should first clean them and mask out the tones affected by internal radio interference. Old VLBA hardware extracted two phase calibration tones per IF. *PIMA* treats this case as a single tone per IF. A user can select which tone to use. If less four tones per IF was extracted in your experiment or you do not apply multiples phase cal tones per IF, just skip this section.

Usually, phase calibration signal is a rail of very narrow-band signals with frequency separation 1 MHz which less than the spectral resolution of visibility data. *PIMA* interpolates spectrum of phase calibration within each IF. In a case if only one tone per IF is used *PIMA* considers phase spectrum of the calibration signal is flat, i.e. it assigns the phase calibration phase to all spectral channels. In a case if all phase calibration tones are used, *PIMA* unwraps phases and performs linear interpolation or extrapolation. The presence off spurious signals distorts calibration phases. If they affect a small fraction of all tones, the tones affected by spurious signals can be masked out. Then *PIMA* will automatically interpolated between tones that are not affected.

Task **mppl** shows phase and amplitudes of phase calibration signal with multiple tones per IF. It may be useful to use keyword **OBS** to control which observations to use for generating plots.

Task **mppl** shows several types of plots. Raw phase usually is not informative, since the calibration signal may have many phase turns per IF. Let *PIMA* to unwrap phase for you. Plot of unwrapped phases (U) shows the spectrum: unwrapped phases (green) and modeled phases (blue). Bandpass is supposed to be smooth. Jumps in phases is due to spurious signals. The sum of the phase calibration tone and the spurious signal depends on the phase of the phase calibration tone itself. Therefore, if at a given plot of a given observation you see phase that does not strongly deviate from the smoothed curve that does not necessarily means that other observation will not be affected even if the source of spurious signals does not depend on time. Mode (C) displays both unwrapped phase and amplitude at the same plot. Since spurious signal affect both phase and amplitude of the calibration signal, this plot helps to identify frequencies affected by spurious signals.

OK, you found a peak at the plot that you believe is due to the spurious signal. What further? *PIMA* supports so-called phase calibration mask file specified by the keyword **PCAL_MASK_FILE**. This file defines the value, 0 or 1, for **each** phase calibration tone. If the value is zero, then that calibration tone is masked out and not used for computation of the smoothed curve that interpolates the phase calibration signal across an IF. One may edit this file manually, but in general, it is too boring. *PIMA* supports so-called mask definition files that allows to write which calibration tones to suppress in a concise way. It allows to define ranges of the tones that are to be mask out. An analyst edits the calibration mask definition file, converts it to the phase calibration mask, visualizes the phase calibration phases and/or amplitudes and repeat this procedure till a satisfactory result is produced.

Phase calibration mask definition file consists of records of variable length. The first record identifies the format. It should always be be `# PIMA PCAL_MASK_GEN v 1.00 2015.05.10` Lines that start

with '#', except the first one, are considered comments, and the parser ignores them. Mask definition records consists of 8 words separated by one or more blanks `PCAL STA: ssssssss IND_FRQ: aa-bb IND_TONE: xx-yy OFF`

where `ssssss` is the station name, `aa` is the index of the first IF of the range, `bb` is the index of the last IF of the range, `xx` is the index of the first tone in a given IF range and `yy` is the index of the last tone in a given IF range. Here is an example:

```
# PIMA PCAL_MASK_GEN  v  1.00 2015.05.10
#
#  Phase calibration mask definition file for VLBI experiment VEPS02
#
#  Last updated on  2015.05.13_12:35:17
#
PCAL  STA:   KUNMING     IND_FRQ:   1-1   IND_TONE:   30-31  OFF
PCAL  STA:   SESHAN25    IND_FRQ:   8-8   IND_TONE:    4-4   OFF
PCAL  STA:   URUMQI      IND_FRQ:   1-16  IND_TONE:    1-1   OFF
```

The first line identifies the format. The file defines the mask that deselects tones with indices 30–31 of the first IF for station KUNMING, the tone with index 4 for the 8th IF for station SESHAN25, and the first tone in all IFs from the 1st to the 16th for station URUMQI.

PIMA does not accept the mask definition file directly. Task **pmge** transforms the phase calibration mask definition file into phase calibration mask file. That task requires qualifier **mask_gen** with value mask definition file. The name of the output file is defined by keyword **PCAL_MASK_FILE** in the PIMA control file. Example:

```
pima ru0186_x_pima.cnt pmge mask_gen ru0186_pcal_mask.gen
```

Comments:

- Masking a phase calibration tone also excludes from unwrapping the phase calibration phase. The phase unwrapping algorithm subtracts group delay in the calibration signal. Therefore, removal of one or more points, especially of they were outliers, changes the unwrapped phases.
- Usually a mask is applied to all observations. Some spurious signals may go on and off. Therefore it is a good idea to look at phase calibration at different parts of the experiment.

Automatic masking is done through the PIMA task **gepm**. This task has three required parameters and three optional parameters. The first is `sta`, which selects the station for which automatic phase calibration masking is to be done. Passing the `sta` parameter the value `all` will automatically mask all phase calibration tones at once and is the normal mode of operation. The second required parameter is `tim_mseg`, which controls the averaging interval of phase calibration data. **gepm** will average phase calibration data such that the interval between phase calibration samples is as close to this parameter as possible. If phase calibration data has a relatively high sample rate, increasing this parameter may improve the results of automatic phase calibration masking. Passing a value of tim_mseg below the sampling interval of the phase calibration data leaves the data unchanged, thus passing `tim_mseg 0` will always result in no averaging of data. The third required parameter is `overwrite`, which should always be given as `overwrite yes`. This indicates that it is acceptable to overwrite the existing phase calibration mask file.

The three optional parameters pertain to the methods used in automatic phase calibration masking. The first is **tim_thresh**, which controls the threshold fraction of flagged epochs at which a phase calibration tone is masked out. The default value is 0.1, or 10% flagged epochs for masking out. An epoch is flagged when the modeled phase of the phase calibration tone differs from the measured phase by more than **diff_thresh**, indicating a discontinuity typical of a spurious signal. **diff_thresh**, the second optional parameter, thus determines the level of discontinuity in the frequency domain in an IF indicative of a spurious signal. The default value of this parameter is 0.15 radians. This can be increased to reduce the number of masked out tones or lowered to increase the number of masked out tones. Be careful of lowering it too much, as healthy tones may be flagged due to normal phase jitter. The third optional parameter is **max_count**. This controls the maximum allowed number of statistically significant jumps in the phase time series a tone may have. The default value is 50.

Upon completion of **gepm**, several files are produced. The first is an indication of the health of the phase calibration tones in the form of the root-mean-square phase jitter in the time domain and the frequency domain. This file is saved in the form `exp_band_pcal_rms.txt`. **gepm** also produces a report file of the form `exp_band_pcal_report.gen`. This report file shows the specific tones that have been masked out as well as the reason for their masking out. In addition, this report file can be hand-edited to turn off additional tones and turned into a phase calibration mask using the task **pmge** with the usual syntax, calling the report file as the generator file. An excerpt of a typical rms file and report file are shown below:

```
# PIMA PCAL_RMS   Format of 2022.06.30
#
# PCAL_RMS file for experiment r41056
#
# created by # PIMA PCAL_RMS_GEN  v  1.00 2022.06.30 on 2023.01.02-20:26:23
# using control file /vlbi/r41056/r41056_s_pima.cnt
#
AVERAGE TIME SPACING:      8.606 SEC
TIME DIRECTION RMS PHASE CALIBRATION JITTER (RAD) BY CHANNEL:
BADARY   IND_FRQ:     1 IND_TONE:      1 IND_ABS_CHN:       9 RMS:      0.009
BADARY   IND_FRQ:     1 IND_TONE:      2 IND_ABS_CHN:      10 RMS:      0.009
FREQ DIRECTION RMS PHASE CALIBRATION JITTER (RAD) BY IF:
BADARY   IND_FRQ:     1           0.017
BADARY   IND_FRQ:     2           0.008

# PIMA PCAL_RPT   Format of 2022.07.07
#
# PCAL_RPT file for experiment r41056
#
# created by # PIMA PCAL_RPT_GEN  v  1.00 2022.07.07 on 2023.01.02-20:26:23
# using control file /vlbi/r41056/r41056_s_pima.cnt
#
PCAL  STA:  BADARY   IND_FRQ:     3-3   IND_TONE:     8-8    OFF !    Problem:  SPURIOUS SIGNAL
PCAL  STA:  BADARY   IND_FRQ:     6-6   IND_TONE:     3-3    OFF !    Problem:  SPURIOUS SIGNAL
PCAL  STA:  BADARY   IND_FRQ:     6-6   IND_TONE:     5-5    OFF !    Problem:  SPURIOUS SIGNAL
#
PCAL  STA:  MEDICINA IND_FRQ:     3-3   IND_TONE:     1-1    OFF !    Problem:  PHASE JUMPS
PCAL  STA:  MEDICINA IND_FRQ:     3-3   IND_TONE:     2-2    OFF !    Problem:  PHASE JUMPS
PCAL  STA:  MEDICINA IND_FRQ:     3-3   IND_TONE:     3-3    OFF !    Problem:  PHASE JUMPS
```

In addition, if all phase calibration tones are noise for a given station, this will be reported to the user as a text output in the terminal. A clock break detection scheme is also implemented, and any detected clock breaks will be reported along with the time in the time series at which the break occurred. A typical execution and terminal output is shown below:

```
pima r11039_s_pima.cnt gepm sta all tim_mseg 10 overwrite yes tim_thresh 0.1 diff_thresh 0.15 max_count 50
 All pcal tones of station SEJONG   are noise. Recommend adding to  PCAL: not_to_use:  in control file
 All pcal tones of station SVETLOE  are noise. Recommend adding to  PCAL: not_to_use:  in control file
 All pcal tones of station YARRA12M are noise. Recommend adding to  PCAL: not_to_use:  in control file
 For station AGGO     short-term spurious signal near t =    68973.830720024562       sec
 For station HART15M  short-term spurious signal near t =    28750.155200028232       sec
 For station NYALES20 short-term spurious signal near t =    11917.828159952907       sec
 For station NYALES20 likely clock break near t =    57313.701440090219       sec
 For station ONSALA60 short-term spurious signal near t =    55581.286400066594       sec
 For station SESHAN25 likely clock break near t =    13436.420480009539       sec
 For station SESHAN25 likely clock break near t =    70135.392319724342       sec
WRI_PCAL_RMS: pcal rms file is written in /vlbi/r11039/r11039_s_pcal_rms.txt
WRI_PCAL_MASK: pcal mask is written in /vlbi/r11039/r11039_pcal.mask
WRI_PCAL_RPT: pcal rpt file is written in /vlbi/r11039/r11039_s_pcal_report.gen
```

gepm has also been added to the wrapper **pf.py** to make it easier to use in the fringe fitting process. To call **gepm** from **pf.py**, the user needs only to specify **overwrite**. All other parameters have listed

defaults. Thus, a minimal function call might look like,

```
Usage: pf.py exp band  gepm -overwrite [-sta,-tim_mseg,-tim_thresh,-diff_thresh,-max_count]
```

When using **pf.py**, the output from the terminal including warnings about clock breaks is written to a log file. A condensed phase calibration generator file is also formed from the report file, which increases readability but does not list reasons for tone being masked out. It is saved according to the name of the phase calibration mask file by replacing `.mask` with `_mask.gen`. i.e. for **PCAL_MASK_FILE:** `r41056_s_pcal.mask`, the mask generator is saved to `r41056_s_pcal_mask.gen`.

Masking auto- and cross-correlation spectral channels

PIMA allows to mask out specified channels of either auto or cross-spectrum or both. The auto-correlation spectrum is corrupted by the presence of internal RFI generated in the vicinity of the data acquisition system. Usually the internal RFI causes appearance of peaks in the auto-correlation spectrum. As a rule of thumb peaks with the amplitude less than 1.5 of the average amplitude can be safely ignored and peaks with the amplitude greater than the average by a factor of 2 should must be masked out since they noticeably affect the fringe fitting procedure and distort the estimate of the average phase and the amplitude. Peaks with the amplitude in a range [1.2, 2] of the average autocorrelation are in the border line. It is not recommended to mask out auto-correlation at the edge of the IFs, since during data processing ***PIMA*** interpolates auto-correlation.

Cross-correlation spectrum can be distorted by external RFI, such as satellite radio. Hardware problem or errors in the hardware setup may cause cause a significant drop in the amplitude or a total loss of signal either at the entire IF, or a range of IFs, or a portion of IFs. If there is no signal in given spectral channels, the SNR will be reduced and weak sources may not be detected. Masking out unwanted noise improves the SNR. Cross-correlation spectrum from narrow-band targets, such as masers or satellites may not have signal beyond edges of the bandwidth even in the absence of hardware failures.

PIMA supports mask file specified by the keyword **BANDPASS_MASK_FILE**. This file defines four sets of values, 0 or 1, for **each** phase spectral channels. ***PIMA*** multiplies visibilities by the mask, which effectively disables spectral channels that corresponds to mask with value 0. The first mask affects autocorrelations, and three remaining masks affect cross-correlations. The second mask used used only for computation of bandpass, the third mask used for fringe fitting, and the fourth mask is used by task ***splt*** for computing visibilities averaged over frequency and time. Usually the second, the third, and the fourth masks are the same, i.e. a common mask for cross-correlation is used. Autocorrelation and cross-correlation masks are usually different since different factors lead to necessity to mask auto-correlation and cross-correlations.

If the mask for a given cross-correlation is zero, the corresponding visibility is replaced with zero. If the mask of a given auto-correlation is zero, the auto-correlation at a given spectral channel is computed by linear interpolation between adjacent channels, or linear extrapolation, if the masked channel is at the edge of the IF. The corresponding cross-correlation is not affected. Very often strong unmasked spurious signals that results in autocorrelation greater than 2–5 of the average level usually affect the cross-correlation as well. Therefore, it is prudent first to mask out strong spurious signals at autocorrelation and then check cross-correlation spectrum.

An analyst may create the mask file by hand, but this is a tedious work. ***PIMA*** supports so-called bandpass mask definition files that allows to specify the spectral channels that are to be suppressed in a concise way. The bandpass mask definition file allows to define ranges of the spectral channels that are to be mask out. An analyst edits the bandpass mask definition file, converts it to the bandpass mask, visualizes the cross- and auto- phase and amplitude spectra, and repeat this procedure till a satisfactory result is produced.

A bandpass mask definition file consists of records of variable length. The first record identifies the format. It should always be be `# PIMA BPASS_MASK_GEN v 0.90 2009.02.05` Lines that start with '#', except the first one, are considered to be comments, and the parser ignores them. Mask definition records consists of 8 words separated by one or more blanks `mmmm STA: ssssssss IND_FRQ: aa-bb IND_CHN: xx-yy ddd`

where `mmmm` is the mask type: one of AUTC (autocorrelation mask), BPAS (bandpass mask), FRNG (fringe fitting mask), SPLT (split mask), CROS (bandpass+fringe_fitting+split masks), ALL (all masks: autocorrelation+bandpass+fringe_fitting+split) `ssssssss` is the station name, aa is the index of the first IF of the range, bb is the index of the last IF of the range, xx is the index of the first spectral channel in a given IF range and yy is the index of the last spectral channel in a given IF range; ddd is disposition: ON or OFF. Station name may be substituted by ALL, what means the definition affects all stations.

The first definition sets the default disposition: ON or OFF. Unless you have really pathological experiment and you have to disable the majority of spectral channels, the first definition is `ALL STA ON`, what means to enable all spectral channels The definitions are processed consecutively. Each new definition alters the mask defined by priori definitions. Here is an example:

```
# PIMA BPASS_MASK_GEN  v 0.90 2009.02.05
#
#  Control for bandpass mask generation for VLBI experiment BP192B3
#
#  Created on 2015.11.14_12:33:11
#
ALL    STA:   ALL         ON
#
ALL    STA:   ALL         IND_FRQ:   3-3    IND_CHN: 121-129   OFF
CROS   STA:   KP-VLBA     IND_FRQ:   1-2    IND_CHN:   1-512   OFF
AUTC   STA:   FD-VLBA     IND_FRQ:   4-4    IND_CHN: 431-436   OFF
```

The first line identifies the format. The first non-comment line sets the initial mask 1. The second non-comment lines disables both autcorrelations and cross-correlations for the IF #3, spectral channels 121 through 129. The third line disables cross-correlation in IFs #1 and #2 (that experiment has 512 spectral channels per IF) for station KP-VLBA. The fourth line disables autocorrelation for FD-VLBA in spectral channels from 431 through 436 in IF #4 keeping autocorrelation.

Masking auto- and cross-correlation spectral channels

PIMA allows to mask out specified channels of either auto or cross-spectrum or both. The auto-correlation spectrum is corrupted by the presence of internal RFI generated in the vicinity of the data acquisition system. Usually the internal RFI causes appearance of peaks in the auto-correlation spectrum. As a rule of thumb peaks with the amplitude less than 1.5 of the average amplitude can be safely ignored and peaks with the amplitude greater than the average by a factor of 2 should must be masked out since they noticeably affect the fringe fitting procedure and distort the estimate of the average phase and the amplitude. Peaks with the amplitude in a range [1.2, 2] of the average autocorrelation are in the border line. It is not recommended to mask out auto-correlation at the edge of the IFs, since during data processing ***PIMA*** interpolates auto-correlation.

Cross-correlation spectrum can be distorted by external RFI, such as satellite radio. Hardware problem or errors in the hardware setup may cause cause a significant drop in the amplitude or a total loss of signal either at the entire IF, or a range of IFs, or a portion of IFs. If there is no signal in given spectral channels, the SNR will be reduced and weak sources may not be detected. Masking out unwanted noise improves the SNR. Cross-correlation spectrum from narrow-band targets, such as masers or satellites may not have signal beyond edges of the bandwidth even in the absence of hardware failures.

PIMA supports mask file specified by the keyword **BANDPASS_MASK_FILE**. This file defines four sets of values, 0 or 1, for **each** phase spectral channels. ***PIMA*** multiplies visibilities by the mask, which effectively disables spectral channels that corresponds to mask with value 0. The first mask affects autocorrelations, and three remaining masks affect cross-correlations. The second mask used used only for computation of bandpass, the third mask used for fringe fitting, and the fourth mask is used by task ***splt*** for computing visibilities averaged over frequency and time. Usually the second, the third, and the fourth masks are the same, i.e. a common mask for cross-correlation is used. Autocorrelation and cross-correlation masks are usually different since different factors lead to necessity to mask auto-correlation and cross-correlations.

If the mask for a given cross-correlation is zero, the corresponding visibility is replaced with zero. If the mask of a given auto-correlation is zero, the auto-correlation at a given spectral channel is computed by linear interpolation between adjacent channels, or linear extrapolation, if the masked channel is at the edge of the IF. The corresponding cross-correlation is not affected. Very often strong unmasked spurious signals that results in autocorrelation greater than 2–5 of the average level usually affect the cross-correlation as well. Therefore, it is prudent first to mask out strong spurious signals at autocorrelation and then check cross-correlation spectrum.

An analyst may create the mask file by hand, but this is a tedious work. ***PIMA*** supports so-called bandpass mask definition files that allows to specify the spectral channels that are to be suppressed in a concise way. The bandpass mask definition file allows to define ranges of the spectral channels that are to be mask out. An analyst edits the bandpass mask definition file, converts it to the bandpass mask, visualizes the cross- and auto- phase and amplitude spectra, and repeat this procedure till a satisfactory result is produced.

A bandpass mask definition file consists of records of variable length. The first record identifies the format. It should always be be `# PIMA BPASS_MASK_GEN v 0.90 2009.02.05` Lines that start with '#', except the first one, are considered to be comments, and the parser ignores them. Mask definition records consists of 8 words separated by one or more blanks `mmmm STA: ssssssss IND_FRQ: aa-bb IND_CHN: xx-yy ddd`

where `mmmm` is the mask type: one of AUTC (autocorrelation mask), BPAS (bandpass mask), FRNG (fringe fitting mask), SPLT (split mask), CROS (bandpass+fringe_fitting+split masks), ALL (all masks: autocorrelation+bandpass+fringe_fitting+split) `ssssssss` is the station name, aa is the index of the first IF of the range, bb is the index of the last IF of the range, xx is the index of the first spectral channel in a given IF range and yy is the index of the last spectral channel in a given IF range; ddd is disposition: ON or OFF. Station name may be substituted by ALL, what means the definition affects all stations.

The first definition sets the default disposition: ON or OFF. Unless you have really pathological experiment and you have to disable the majority of spectral channels, the first definition is `ALL STA ON`, what means to enable all spectral channels The definitions are processed consecutively. Each new definition alters the mask defined by priori definitions. Here is an example:

```
# PIMA BPASS_MASK_GEN  v 0.90 2009.02.05
#
#  Control for bandpass mask generation for VLBI experiment BP192B3
#
#  Created on 2015.11.14_12:33:11
#
ALL    STA:   ALL         ON
#
ALL    STA:   ALL         IND_FRQ:   3-3    IND_CHN: 121-129   OFF
CROS   STA:   KP-VLBA     IND_FRQ:   1-2    IND_CHN:   1-512   OFF
AUTC   STA:   FD-VLBA     IND_FRQ:   4-4    IND_CHN: 431-436   OFF
```

The first line identifies the format. The first non-comment line sets the initial mask 1. The second non-comment lines disables both autcorrelations and cross-correlations for the IF #3, spectral channels 121 through 129. The third line disables cross-correlation in IFs #1 and #2 (that experiment has 512 spectral channels per IF) for station KP-VLBA. The fourth line disables autocorrelation for FD-VLBA in spectral channels from 431 through 436 in IF #4 keeping autocorrelation.

Creation complex bandpass in the inspection mode

Task ***bpas*** compute complex band-pass. This task supports 4 modes: **INSP** (inspection), **INIT** (initial), **ACCUM** (accumulation), and **FINE**. Modes **INIT** and **INSP** differs only by the generated output: ***bpas*** in *INSP* mode generates plots, while ***bpas*** in INIT mode does not.

Task ***bpas*** takes as input result of fringe fitting. In general, it is not required to have fringe fitting results for all observations, although it is desirable. ***PIMA*** will find n observations with the highest SNR at each baseline with the reference station and will compute the complex bandpass using these observations. ***PIMA*** uses one observation with the highest SNR per baseline in **INIT** or **INSP** mode. It may happen that just the observation with the highest SNR is affected by RFI or has another problems. In such a case affected observation can be added into the exclude list.

In a case of dual-polarization observations two bandpasses are computed: for RR polarization and for difference LL minus RR polarization. The second bandpass is called polarization bandpass. The main bandpass is RR for single-polarization RR data, LL for single-polarization LL data, and RR for dual-polarization data. In order to compute both, main RR bandpass and polarization bandpasses, task ***bpas*** should be executed with **POLAR: I**.

Wrapper **pf.py** simplifies generation of the bandpass. It checks exclusion file `VVVVV/EEE/EEE_B_bpas_obs.exc`. If it does not find such a file, the wrapper creates an empty file with a comment line. It supports option **-insp** that overrides value of **BPS.MODE**.

```
Usage: pf.py exp band obs [-insp]
```

When ***PIMA*** task **bpas** is invoked in the **INSP** mode, ***PIMA*** runs a cycle over baselines with the reference station and computes amplitude and phase bandpasses for an observation with the highest SNR among selected observations of each baseline with the referenced station. It displays two plots per baseline: amplitude plot and phase plot. An amplitude plots shows three function: autocorrelation (red), cross-correlation normalized to unity (blue) and the bandpass (green). The phase plot shows residual phase (blue) and phase bandpass (green).

When computing bandpass, ***PIMA*** re-runs fringe fitting for the selected observation. Blue lines in the amplitude and phase plots show just residual amplitudes and phases from that fringe fitting. Then a smooth curve is fitted to the residuals. ***PIMA*** supports two algorithms: fitting with a smoothing spline of the 3rd degree over n equi-distant knots or with a Legendre polynomial of degree b. The smoothing methods should be the same for phase and amplitude, but the degree of the Legendre polynomial or the number of knots for the spine can be different for amplitude and phase.

Smoothing mode is defined by keyword **BPS.INTRP_METHOD**. It can take values **SPLINE**, **LEGENDRE**, or **LINEAR**. Phase initial bandpass is the smoothed model bandpass with opposite sign. The phase bandpass is normalized to have the mean phase over the *band* zero. ***PIMA*** will unwrap phase if phase bandpass exceeds ±π/2. Amplitude bandpass is normalized to unity. If **BPS.NORML:** IF, then the amplitude bandpass is analyzed over each IF separately. If **BPS.NORML**: BAND, then the amplitude bandpass is normalized over the band. Which mode to use? That depends on hardware. If system temperature is measured for entire band, then normalization should be made over the band. If system temperature is measured at each IF (or pairs of IFs) individually, than normalization over IF should be used. Autocorrelation is always normalized over IF.

PIMA task ***bpas*** in the inspection mode displays two plots per baseline: amplitude bandpass and phase bandpass. Reading these plots is an important skill. An analyst should identify spikes in autocorrelation amplitudes and mask them out. To identify a spike in autocorrelation, first set color index 3 by hitting **C** and **3**, then move the cursor close to the spike and hit **LeftMouse** button. Dump file `SSSSS/pima/EEE.frq` helps to identify an IF/channel indices. Using this information, an analyst adds a record in the mask definition file. There is no firm rule when a spike should be mask out. Usually spikes with amplitudes greater than 2 should be mask out, and those with amplitudes in a range [0.8, 1.2] are kept. After all spikes in autocorrelation are mask out, a mask should be generated from the mask definition file using task ***bmge***. Task ***bmge*** requires qualifier mask_gen with the value mask definition file. After than inspection of autocorrelation amplitudes should be repeated. **NB:** Amplitude of masked autocorrelation is set to zero. If necessary, mask definition file should be edited, mask re-generated, and the procedure repeated.

After cleaning autocorrelation spectrum, cross correlation should be cleaned. Several situations are rather common:

1. There is no signal in one or more IFs mainly due to hardware error or errors in the setup. These IFs should be masked out entirely.
2. Cross-correlation amplitude in one or more IF is lower than in others. These IFs *may be* masked out depending how strongly the amplitude is down. If the amplitude is lower by a factor of 4–6, such an IF brings more noise than signal and probably should be mask out. Masking out an IF with the amplitude less than 1.5 times will degrade the SNR and deteriorate the result. Amplitude loss in a range 1.5–5 is the border line.
3. A significant portion of an IF has low amplitude. This may be caused by RFI or the receiver bandpass shape. It is recommended to masked out that portion of the bandpass.
4. Amplitude at the edge of each IF is low. This is the most common situation. This is caused by the hardware. A general recommendation to mask out a portion of the band that is below some threshold in a range of 0.05%ndash;0.2.

A hint: task ***bpas*** in the inspection mode processes baselines in the alphabetic order. If a certain station requires heavy editing the bandpass generation file, you can run the task with **OBS**: num_obs qualifier, there num_obs is the index of the observation with the highest SNR at the baseline of interest.

Task ***bpas*** supports keyword **BPS.AMP_MIN** that defines the threshold on the fringe amplitude as the share of mean amplitude over the IF. Spectral channels with the amplitude threshold below **BPS.AMP_MIN** are ignored by task ***bpas***. The bandpass to these channels is obtained by extrapolation from those channels that are in use. This may be useful if the bandpass has strong changes at channels with low amplitude, since in that case smoothing with Legendre polynomial and spline may not be robust.

Phase may become too noisy in experiment with high spectral resolution. In that case the residuals should be coherently averaged. The number of spectral channels averaged within an individual segment is defined by keyword. Value **1** stands for no averaging. A balance between averaging and spectral resolution should be maintained. From one hand strong averaging (high value of **BPS.MSEG_ACCUM**) results in less random noise in phase. From the other hand, averaging reduces spectral resolution and our ability to model the system response as a function of frequency. Scatter greater than 1–2 rad may result in a failure to resolve phase ambiguity across an IF, which will lead to a wrong result. A general guideline is to select such averaging that the scatter of residual phases with respect to a smooth line be in a range of 0.05–0.2 rad for the sources used for bandpass computation.

Interpolation of the bandpass is important. In the simplest form the bandpass is just reciprocal to the normalized complex cross-correlation function of the observation with the highest SNR at a given baseline. If the SNR of that observation were infinitely high, this would have been the optimal approach. However, when we *apply* the bandpass computed by inversion of the normalize dd complex cross-correlation function of an observation with finite SNR, we propagate the noise of that observation to other observations, which results in an increase of the total noise. To alleviate undesirable noise propagation, we 1) use more than one observation for computing band pass, 2) coherently average n adjacent visibilities; 3) smooth the bandpass with polynomial or splines.

The choice of the magnitude of coherent averaging and the degree of the polynomial or the number of spline knots depends both on the SNR and the number of spectral channels in the IF. The higher SNR of the observations used for bandpass computation, the better, although the quantitative measure of whether a given SNR is high enough is rather subjective. Observations with SNR 200–2000 allows to compute a rather reliable bandpass with the number of spline knots 7–20 (In general Legendre polynomials of degree higher than 5–7 are undesirable since they are prone to end up with bandpass of wiggling shape like a dinosaur's spine).

A wise principal investigator will insert enough strong calibrators to make computation of the bandpass robust, but sometimes either the PI did not think well, or observations of strong calibrators failed, or their were not possible, for instance for space VLBI. In that case we have to compute bandpass using weak calibrators. When observations with SNR in a range 30–100 are used care must be taken to generate a robust bandpass. First, it should be checked that the scatter of the residual phases of the observations used for bandpass compilation is smaller π/3–π/6^ndash and phase ambiguities can be reliable resolved. If an error in ambiguity resolution will happen, it will strongly poison a bandpass: such a bandpass, when applied, will degrade the SNR, not improve it. To mitigate phase ambiguity resolution, the residual cross-correlation is coherently averaged (keywords **BPS.MSEG_ACCUM** and **BPS.MSEG_FINE**). Second, the degree of the smoothing polynomial or the number of spline knots is reduced.

Finally, when the bandpass has to be computed using the very weak observations with the SNR 7–20, **BPS.INTRP_METHOD: LINEAR** is used as the last resource. This mode requires **BPS.MSEG_ACCUM** and **BPS.MSEG_FINE** be equal to half of the number of spectral channels and **BPS.DEG_AMP: 0** and **BPS.DEG_PHS: 1**. In this mode the cross-spectrum is coherently averaged to two points per IF. The

amplitude bandpass is computed as the IF-averaged level. The phase bandpass is computed as a linear function over two points. This mode is the most robust, but it does not model the more grained shape of the bandpass. This omission is not essential for low-SNR observations. According to the reciprocity principle omission in the data reduction of a quantity that cannot be reliably determined from observations cannot significantly degrade goodness of the fit. From the other hand, using **BPS.INTRP_METHOD: LINEAR** for experiments with higher SNR observations may results in worse result with respect to **BPS.INTRP_METHOD: LEGENDRE** or **SPLINE**. The natural low limit of observations used for bandpass generation in **BPS.INTRP_METHOD: LINEAR** mode is the detection limit. Inadvertent inclusion of a non-detection to the list of observations used for bandpass generation will significantly degrade the bandpass. Better use no bandpass than a wrong bandpass.

When the analyst is satisfied with plots that task ***bpas*** generates in the inspection mode, an analyst runs ***bpas*** task in the non-interactive mode to generate the final bandpass. There are three modes for computation of the bandpass: **INIT**, **ACCUM**, and **FINE**. ***PIMA*** computes the residual visibilities averaged over time with parameters of fringe fitting applies. In the **INIT** mode ***PIMA*** picks up for each baseline the observation with the highest SNR among the observations with the reference stations that are subject of filter **OBS**, **INCLUDE_OBS**, and **EXCLUDE_OBS** keywords. It normalizes the amplitude to have the mean value to unity either over the IF or over the band depending on **BPS.NORML** keyword. The residual phase is normalized to have mean value and mean rate to zero over the bandwidth regardless of the value of **BPS.NORML**. ***PIMA*** smooths the residual phases and residual amplitudes and the inverts them: flips the sign of phase band pass, replaces the residual amplitude with the quantity reciprocal to that for each spectral channel, and combines them to form array of complex numbers. These quantities are called initial complex bandpass.

Initial bandpass is computed using only one observation per baseline. If an analyst selected *BPS.MODE:* **INIT**, the task ***bpas*** stops here. If a user selected **ACCUM** or **FINE** mode ***PIMA*** selects N observations per baseline with the highest SNR beyond that that was used in the **INIT** mode. It applies the initial bandpass determine residual phases and amplitudes and averages them out. It reverses sign of residual phases, replaces normalized residual amplitudes, combines them in the array of complex numbers and multiplies it by the initial bandpass. The result is called accumulated bandpass. The number of observations per baseline used for computation of the accumulation bandpass is controlled by two parameters: **BPS.NOBS_ACCUM** and **BPS.SNR_MIN_ACCUM**. ***PIMA*** will select up to **BPS.NOBS_ACCUM** observations for each baseline with the highest SNR, not counting the observation used for computation of the initial bandpass, provided they SNR is **BPS.SNR_MIN_ACCUM** or above. The advantage of the accumulation bandpass is that it is unweighted average over N observations, and therefore, it accounts to some degree bandpass variation. If a user selected **ACCUM** mode ***PIMA*** task ***bpas*** stops here.

If a user chooses **FINE** bandpass computation mode, ***PIMA*** selects K observations per baseline with the highest SNR beyond that that was used in the **INIT** mode. It applies accumulation bandpass and forms the system of linear equations for adjustment to parameters of the model for the phase bandpass and logarithm of the amplitude bandpass, i.e. either coefficients of Legendre polynomial or B-spline. It solves the system using weighted least squares with weights proportional to fringe amplitude. Then it computes residual phases and amplitude corrections, computes their statistics, and if the statistics exceed the specified threshold, it discards the observations with the greatest residual and repeats computation until either the statistics become lower than the threshold or the number of rejected observations per baseline reaches the specified limit.

Keyword **BPS.SNR_MIN_FINE** specifies the minimum SNR. Observations with SNR below that limit are not used by ***PIMA*** for bandpass computation in the **FINE** mode. Keyword **BPS.NOBS_FINE** specifies the number of observations with the highest SNR per baseline that ***PIMA*** selects for bandpass computation in the **FINE** mode, provided their SNR is no less than **BPS.SNR_MIN_FINE**. If a given observation has residual statistics above the threshold, ***PIMA*** will discard it provided the number of remaining used observations is no less than **BPS.MINOBS_FINE**. This mechanism prevents rejection of too many observations.

After performing the first iteration of LSQ adjustment, ***PIMA*** computes for each IF weighted rms of residual phases and normalized residual amplitudes. Then ***PIMA*** finds an observation with maximum phase and maximum amplitude residual. If the rms of phase residual exceeds **BPS.PHAS_REJECT** radians, that observation is marked for rejection. If the rms of normalized amplitude residuals exceeds **BPS.AMPL_REJECT**, that observation is marked for rejection. If the number of used observations still exceeds **BPS.MINOBS_FINE**, the observation is rejected, and the next iteration runs. ***PIMA*** maintains two counters of used observations: for the phase bandpass and for the amplitude bandpass. An observation may be rejected for amplitude bandpass but kept for phase bandpass, or vice versus, or rejected for both amplitude and phase bandpasses.

PIMA task ***bpas*** prints valuable statistics when **DEBUG_LEVEL: 3** or higher. An analyst should always examine it. Lines that starts with "BPASS Removed" are especially important. If there are many rejected observations at a given baseline, especially if their phase or amplitude rms of residuals is large, an analyst should examine these observations: to run a trial fringe fit and look at residuals. Bad observations may skew bandpass evaluation, so it may be necessary to examine residuals with and without applying bandpass (BANDPASS_USE: NO). One of the reasons of computing bandpass in FINE mode is to find observations with large residual phases or residual amplitudes.

It may happen that a fraction of high SNR observations are affected by hardware failure or RFI. If ***bpas*** task does not reject them, they skew the estimate of bandpass. There is another mechanism to get rid of such observations for bandpass computation: to put them in the exclude list. ***PIMA*** wrapper **pf.py** automatically checks file `VVVVV/EEE/EEE_B_bpas_exc.obs`. If it exists, it adds option EXCLUDE_OBS: `VVVVV/EEE/EEE_B_bpas_exc.obs`, i.e. excludes them from participation in bandpass computation.

PIMA task ***bpas*** assumes bandpass is stable. It may happen one or more jumps, f.e. due to power failure. In that case ***PIMA*** will reject many observations. At the moment, ***PIMA*** does not have a capability to accommodate a jump. A workaround is to process two portions of the experiment separately by specifying observation lists using **INCLUDE_OBS_FILE**, **EXCLUDE_OBS_FILE** or **OBS** keywords. If a given station has more than 2–3 jumps in bandpass, it should be discarded and station staff should be alerted.

If there are many rejected observations an analyst may a) mask out bad channels b) add offending observations to the exclude list; c) exclude a list of observations; d) discard a station; e) ignore it.

In a case of dual-polarization data when **POLAR: I** is specified, the procedure is repeated twice: first for the RR polarization bandpass second for the LL polarization with respect to the RR polarization data. Therefore, a trial fringe fit with bandpass applied should run three times: with RR polarization, with LL polarization, and with I polarization. If the polarization bandpass was computed perfectly, then SNR at I polarization should be $\sqrt{\overline{\mathrm{SNR}^2}_{\mathrm{RR}} + \overline{\mathrm{SNR}^2}_{\mathrm{LL}}}$ SNR reduction 2–5% with respect to the expression above is rather common, though if the SNR at I polarization is more than 10% worse, this indicates a problem that should be investigated.

A general recommendation is to run trial fringe fitting for 5–7 observations that are marked as "removed" in the log file of ***bpas*** task with an without bandpass applied in order to familiarize with the data. If fringe plots look satisfactory, the next step: fringe fitting in the fine mode should be done. Otherwise, masking, deselection of bad observations, phase calibration disabling/enabling should be repeated. NB: computation of bandpass should be repeated if a) bandpass or phase cal bandpass mask was changed or b) treatment of phase calibration was changed.

Running fine fringe fitting

Fine fringe fitting is the main task. During coarse fringe fitting, the fine fringe search procedure is disabled in order to speed up the process, and the bandpass was not applied. During fine fringe search this simplification is lifted.

PIMA task ***frib*** performs fringe fitting. Wrapper **pf.py** is called as

```
Usage: pf.py exp band fine
```

PIMA task ***frib*** creates two ascii output files defined in keywords **FRINGE_FILE** and **FRIRES_FILE**. The first file keeps results of fringe fitting and it is used by other tasks. The latter file with fringe fitting residuals is for informational purposes only.

NB: wrapper **pf.py** by default overwrites the files with fringe results if it exists. Wrapper option -keep prevents overwriting the file with fringe results and fringe residuals specified in the control file. ***PIMA*** tasks that reads results of fringe fitting, f.e. ***mkdb*** or ***splt***, processes the fringe file sequentially. If there is more than one record for a given observation, the latest record takes the precedence.

There is a number of parameters that controls fringe fitting.

- Keyword **FRIB.SEARCH_TYPE** is always 2FFT that means a two-dimensional Fast Fourier Transform runs at the coarse fringe fitting step.
- The visibility data are sampled at a uniform grid with a step over time and frequency. The steep over frequency is equal to the spectral resolution times **FRIB.OVERSAMPLE_MD**. The step over time is equal to the accumulation period length (time resolution) times **FRIB.OVERSAMPLE_RT**. When oversampling factor 1 is used the amplitude estimated during the coarse fringe fitting step may be underestimated by a factor of 2.4 and a weak source may be missed. When the oversampling factors are 4, the maximum amplitude underestimation during coarse fringe fitting step is only 5%. Therefore, it is recommended to use **FRIB.OVERSAMPLE_MD**: 4 and **FRIB.OVERSAMPLE_RT:** 4. ***PIMA*** pads grid elements that do not have visibility data with zeroes.
- ***PIMA*** uses library FFTW for performing two-dimensional multi-threaded FFT. Library FFTW requires some customization in order to reach maximum performance. It uses configuration file called "wisdom file" in FFTW documentation. This configuration file should be created before running ***PIMA***. Package fourpack that is required for building ***PIMA*** contains program create_fftw_plan.

  ```
  Usage: create_fftw_plan
  ```

  where method is one of `MEASURE` of `PATIENT`, `num_threads` is the number of threads, request_file is the file with dimension definitions and plan_file is the output configuration file. ***PIMA*** supplies two configuration files pima_wis_big.inp and pima_wis_small.inp . They can be found in $PIMA_DIR/share/pima/ directory, where PIMA_DIR is the environment variable of the directory where ***PIMA*** has been installed. It is suggested to use pima_wise_big.inp unless you have less than 12 Gb memory. In that case you should use pima_wise_small.inp, but you will not be able to process efficiently wide field VLBI experiments with high spectral and temporal resolution. FFTW configuration file depends on the number of threads. If you generated the FFTW configuration file for N threads, but run ***PIMA*** with K threads, the configuration file will not be used. ***PIMA*** will run, but much slower (a factor of 2–5). Therefore you have to create several plans files for different number of threads. Usually, you use the same number of threads as the number of cores, but you may want to reduce the number of threads if you run ***PIMA*** on a busy server). FFTW supports several methods for computing the best configuration file. Method `MEASURE` is recommended. Method `PATIENT` is supposed to improve performance, but it may take several days to compute it. Examples:

  ```
  create_fftw_plan MEASURE 1 $PIMA_DIR/share/pima/pima_wis_big.inp
  ```

```
                        $PIMA_DIR/share/pima/pima_big_measure_1thr.wis

        create_fftw_plan MEASURE 12 $PIMA_DIR)/share/pima/pima_wis_big.inp
                        $PIMA_DIR/share/pima/pima_big_measure_12thr.wis
```

PIMA keyword **FFT_CONFIG_FILE** defines the FFTW configuration file. Keyword **FFT_METHOD** defines the method *that was used for generation of that file* (`MEASURE` or `PATIENT`). Keyword **NUM_THREADS** sets the number of threads that *PIMA* uses for FFTW and some other parallel operations. The FFTW configuration file should be generated with the same number of threads and the same FFTW method, **otherwise *PIMA* performance will be seriously degraded**.

- After computing two-dimension Fourier transform of visibilities, *PIMA* searches for a maximum in the result of the transform. There are for keywords that define the rectangular search window. Keywords **FRIB.DELAY_WINDOW_CENTER** and **FRIB.RATE_WINDOW_CENTER** define the center of the window. Units are seconds for delay window and dimensionless for delay rate. Keywords **FRIB.DELAY_WINDOW_WIDTH** and **FRIB.RATE_WINDOW_WIDTH** defines the **semi-width** of the search window with respect to the center. The total width is twice wider over each dimension. A negative number sets the search window to the total length of the transform over that dimension. For example, **FRIB.DELAY_WINDOW_WIDTH**: **-1** and **FRIB.RATE_WINDOW_WIDTH**: **-1** means to search for fringes in the entire space that is defined as 1/spectral_resolution and F_{ref}/observation_duration where F_{ref} is the reference frequency.

  If you do not know group delay and delay rate of your observation, you need to search for fringes in the entire area of the Fourier transform. This is a usual situation at the first iteration of data processing. After completion of the first iteration, you may be able to predict group delay and phase delay rate. In that case you may want to restrict the search window and re-run fringe fitting with a narrow window. This procedure is called re-fringing. Re-fringing with a narrow window is usually done for two reasons: a) to guide *PIMA* to pick up the main peak of the averaged visibilities that may appear to have a lower amplitude than the secondary peaks due to phase noise; b) to detect weaker sources. The probability of a falls detection at a given SNR is less when the search is done in the narrow window. The gain in the detection limit can reach 30%.

- There are several keywords that affect amplitude but do not affect estimation of group delay an phase delay rate. Keyword **FRIB.AUTOCORR_CALIB** defines the algorithm for autocorrelation normalization. A recommended choice is **SQRT_MEA**, which means to divided the cross correlation by the square root of the products of mean autocorrelation across each IF after applying a digitization correction. If a user want to maintain compatibility with AIPS, value **SQRT_KOG** can be used. It differs from **SQRT_MEA** by the algorithm for digitization correction. From the point of view of *PIMA* developer, **SQRT_KOG** algorithm is incorrect, but the differences are usually in a range of 0.5–3%, which is insignificant.

  Keyword **FRIB.AMPL_FUDGE_TYPE** controls fudge factor correction. Supported values are **VLBA**, **KOGAN**, **DIFX** and **NO**. If your experiment has been correlated by the ***hardware*** NRAO correlator, you should specify either **VLBA** or **KOGAN**. Then a specific fudge factor to take into account register saturation in the hardware correlator is applied. Basically you have to tell *PIMA* your data have been processed with the hardware correlator since there is no reliable way to learn it from the data themselves. The difference between is **VLBA** and **KOGAN** is that in the latter case all weights are considered to be 1 when correction is applied. Values **DIFX** or **NO** mean no fudge factor should be applied.

  Fringe amplitude decays linearly with an increase residual group delay. The attenuation factor reaches 0.5 when the residual group delay reaches a quantity reciprocal to the spectral resolution. **FRIB.AMPL_EDGE_WINDOW_COR** keyword instructs *PIMA* to compensate this attenuation. You should specify its value USE, unless you have a strong argumentation against it.

  Keyword **FRIB.AMPL_EDGE_BEAM_COR** instructs *PIMA* to apply (**YES**) or not to apply (**NO**) a correction for beam attenuation. It is assumed you know source positions with accuracy 1" or better **and** used this a priori coordinates in the catalogue when you loaded *PIMA*. *PIMA* has a table with measured beam factors for ATCA and VLA antennas. For all other antennas it scales the VLA beam pattern using antenna diameter. It is recommended to use value **YES**.

  PIMA discards visibilities with low weights. Two keywords **FRIB.AUTOCORR_THRESHOLD** and **FRIB.AUTOCORR_THRESHOLD** specify the low threshold for autocorrelation and cross-correlation weights. Typically, normal weights are 1 and the thresholds 0.2 are recommended. Weights below that threshold usually means failures. However, sometimes weights for normal visibilities are low. In that case thresholds 0.2 may force *PIMA* to discard all the data. In that case an analyst may try to reduce values of keywords **FRIB.AUTOCORR_THRESHOLD** and **FRIB.AUTOCORR_THRESHOLD**.

  Keyword **FRIB.NOISE_NSIGMA** controls computation of the noise. After fringe fitting *PIMA* selects randomly 32768 samples of the Fourier transform, orders them in decreasing their amplitudes, and computes the rms. Then it runs iterative procedure of excluding samples with amplitudes greater than **FRIB.NOISE_NSIGMA**. After excluding each sample, the rms is updated. Usually, value **3.5** is optimal.

- *PIMA* checks the value of the keyword **FRIB.SNR_DETECTION** and sets and advisory flag whether the was a detection. Depending on a size of the fringe search window, the detection limit is in a range of **4.8**–**5.8** (less for a smaller window). NB: *PIMA* does not check, whether the source has been actually detected, it only checks whether the SNR was less of greater than the specified threshold.

  PIMA supports several algorithms of fine fringe search specified by keyword **FRIB.FINE_SEARCH**. The most commonly used algorithm is **LSQ**. *PIMA* adjusts phase delay rate, group delay, and group delay rate using least squares with both additive and multiplicative reweighting. Method **ACC** performs a similar procedure, but it adjusts phase delay acceleration instead of group delay rate. This mode is used for a case when a priori station position has a very large uncertainty that causes a significant quadratic term in phase. This happens mainly when one of the elements of the interferometer is in space, f.e. RadioAstron. Method **PAR** that adjusts group delay and phase delay rate using parabolic fitting is used mainly for a coarse fringe search.

- *PIMA* task ***frib*** can generate four types of plots. There are four keywords that controls the way how *PIMA* generates these plots: **FRIB.1D_RESFRQ_PLOT**, **FRIB.1D_RESTIM_PLOT**, **FRIB.1D_DRF_PLOT**, and **FRIB.2D_FRINGE_PLOT**. These keywords can take one of six values:
  1. **NO** — do not generate a plot;
  2. **XW** — to display the plot in the current X11 window;
  3. **PS** — to generate a plot in Postscript format;
  4. **GIF** — to generate a plot in GIF format;
  5. **SAV** — to generate a plot in DiaGI format. The plot can be re-displayed with command diagi_rst that is a part of petools package.
  6. **TXT** — to generate a table with columns argument a value in a plain ascii format.

  A plot is written in directory `SSSSS/EEE_fpl`.

  A 1D-plot of amplitudes and residual phases versus frequency is controlled by the keyword **FRIB.1D_RESFRQ_PLOT**. The residuals are coherently averaged over **FRIB.1D_FRQ_MSEG** spectral channels. Averaging is not needed for high SNR observations (i.e. **FRIB.1D_FRQ_MSEG** should be set to **1**), but may be needed for lower SNR observations.

  A 1D-plot of amplitudes and residual phases versus time is controlled by the keyword **FRIB.1D_RESTIM_PLOT**. The residuals are coherently averaged over **FRIB.1D_TIM_MSEG** accumulation periods. Averaging is not needed for high SNR observations (i.e. **FRIB.1D_FRQ_MSEG** should be set to **1**), but may be needed for lower SNR observations.

  A 1D-plot of amplitude of the delay resolution function is controlled by the keyword **FRIB.1D_DRF_PLOT**. The delay resolution is the 1D slice of the the Fourier transform of visibilities along the phase delay found by fringe fitting. Keyword **FRIB.1D_DRF_SPAN** defines the span of the plot in units of the group delay ambiguity spacings. Value **1.2** is usually enough to visualize the DRF. In a case of lack of phase offsets in IFs, the shape of the DRF is regular with low sidelobes. The presence of phase offsets distorts the shape of the DRF and raises the sidelobe which may become stronger than the main maximum.

  A 2D-plot of fringe amplitude versus group delay and phase delay rate is controlled by the keyword **FRIB.2D_FRINGE_PLOT**. A portion of the 2D Fourier transform of visibilities is displayed. Two keywords **FRIB.PLOT_DELAY_WINDOW_WIDTH**, and **FRIB.PLOT_RATE_WINDOW_WIDTH** control the size of the window. Units are seconds for delay and dimensionless for delay rate. The size of a usable window is strongly dependent on the frequency sequence and duration of accumulation period. **1.D-7** for delay and **5.D-12** delay rate may be a reasonable initial choice. The best values are selected by trials.

Export data for astrometry/geodesy solution

Fringe results can be transformed to the form that astrometry/geodesy software Post-Solve can ingest. Task ***mkdb*** reads fringe files, fringe residual file, contents of internal *PIMA* tables that are kept in `SSSSS/EEE.pim` file, and contents of visibility files, computes scan reference time (SRT), computes a priori path delays on the SRT, computes total group delays and phase delay rates on SRT, sorts them, and writes them into database files in either binary Geo VLBI Format (GVF) or plain ascii (TEXT) format. In a case of dual-base observations it reads two fringe results and fringe residual files for both bands and matches them.

Basic operations of this task are a) splitting the data into output scans with their reference time; b)computation of path delay; c) formatting the output.

PIMA performs baseline-dependent fringe fitting, and it processes observations independently. *PIMA* finds the fringe reference time (FRT) automatically as a mean weighted epoch among *used* accumulation periods when **FRT_OFFSET: AUTO**. In general, the FRT is different even if all stations had the same nominal start and stop time. Often VLBI experiments are scheduled in such a way that all stations have the same stop time but different start time. Geodetic schedules tends to have chaotic start and stop time when different antennas of the network have different start and stop epochs.

PIMA has several way to set the scan reference time. The algorithm is controlled by **keyword MKDB.SRT**. If it has value **SRT_FRT**, then *PIMA* sets the **SRT** the same as **FRT**. As a result the number of output scans, i.e. observations with the same epoch, tends to be the same as the number of observations, i.e. each output scan has only one observation. This is usually undesirable.

When **MKDB.SRT**: **MID_SCAN**, *PIMA* consolidates time epochs in order to have as many as possible observations of the same scan to have the same epoch. But when the reference epoch is moved away from the weighted mean epoch, the uncertainty of group delay increases. *PIMA* has two keywords that controls the interval of time the scan reference time can deviate from the reference time for group delay uncertainty not to grow too much: **MKDB.GD_MAX_ADD_ERROR** and **MKDB.GD_MAX_SCL_ERROR**. Keyword **MKDB.GD_MAX_ADD_ERROR** specifies the tolerance of the absolute increase of the

uncertainty in seconds. Keyword **MKDB.GD_MAX_SCL_ERROR** specifies the tolerance of the increase of the uncertainty as a fraction of the original group delay uncertainty derived by *PIMA* task ***frib***. *PIMA* uses the smallest of the these two tolerances. Typical value of **MKDB.GD_MAX_ADD_ERROR** is **5.D-12**, typical value of **MKDB.GD_MAX_SCL_ERROR** is **0.2**. That means that if, for example, the original path delay uncertainty is 40 ps, the tolerance is the smallest of 5 and 0.2*40= 8 ps, i.e. 5 ps. *PIMA* determines the SRT in such a way that the increase of the uncertainty within the tolerance limit be minimal. If there are observations that cannot be combined to the same SRT, *PIMA* splits input scans set by task ***load*** into several output scans.

PIMA allows a user to compute scan reference time, write them in file and supply the file name as the value of **MKDB.SRT**. This may be useful for comparison the results with other fringe fitting software.

PIMA can write the output in three different formats. The format is controlled by the keyword **MKDB.OUTPUT_TYPE**. Value **TEXT** instructs *PIMA* to generate a plain ascii output file with total group delays, total phase delay rates and many other quantities, one line per observation. Value **AMPL** instructs *PIMA* to generate a plain ascii output file fringe amplitude, fringe phase, Tsys, gain, *uv* baseline projections and other parameters are written in a plain ascii table, one line per used observation. Results in **AMPL** format are mainly for non-imaging flux density analysis. Value **GVF** instructs *PIMA* to generates a database in binary **GVF** format. VLBI analysis program Post-Solve for geodesy and absolute astrometry accepts GVF as input. Therefore, task ***mkdb*** provides an interface between *PIMA* and Post-Solve.

The database in plain ascii is not equivalent to the database in GVF format: the GVF database contains more parameters than the ascii database. Therefore, the GVF database can be converted to ascii, but reverse transformation is not feasible.

The ascii database contains parameters for two bands. If the observations were performed only for one band or task ***mkdb*** was called with keyword **MKDB.2ND_BAND: NO**, the values for the second band will be zero. When the output for both bands is available, the band with higher reference frequency, thereafter called *higher* precedes. Path delay is defined as the difference of two intervals of proper time: 1) the interval of proper time measured by the clock of the first (reference) station between event of coming the wavefront to the reference point of the first antenna and clock synchronization and 2) the interval of proper time measured by the clock of the second (remote) station between event of coming the wavefront to the reference point of the second antenna and clock synchronization. The antenna reference point is the point of injection of phase calibration tone if the phase calibration was used in data analysis or the phase center of the antenna. The following parameters are written to the output ascii database:

- Observation index in the database. Starts from 1.
- Observation index in PIMA. This index does not necessarily equal to the observation index in the database. This index is used to associate a given observation in the database with an observation index in *PIMA* internal data structures.
- Long scan name generated by *PIMA* in the form doy_HHMMSS_iiii, where doy is the day of the year, HHMMSS is the scan reference time and iiii is the scan index assigned by *PIMA*.
- Short scan name embedded in the FITS-IDI file.
- Source name, B1950 notation. A non-standard name is allowed.
- 8-character long name of the reference (first) station.
- 8-character long name of the remote (second) station.
- SNR at the higher band
- SNR at the lower band
- Scan reference time in format YYYY.MM.DD-hh:mm:ss.ffff in TAI. This time is computed by ***mkdb*** and is rounded to an integer second.
- Scan reference time in TAI with respect to the nominal start of the first observation of the experiment in seconds. NB: The first observation of the experiment is not necessarily the first experiment in the database.
- Offset of the scan reference time with respect to the nominal observation start time (sec)
- Offset of the fringe reference time with respect to the nominal observation start time at the higher band (sec)
- Offset of the fringe reference time with respect to the nominal observation start time at the lower band (sec)
- Theoretical group delay on scan reference time computed by VTD during fringe fitting analysis (sec)
- Theoretical phase delay rate on scan reference time computed by VTD during fringe fitting analysis (dimensionless)
- A priori geocentric group delay on scan reference time for the higher band computed for the correlator (sec)
- A priori geocentric group delay on scan reference time for the lower band computed for the correlator (sec)
- A priori geocentric phase delay on scan reference time rate for the higher band computed for the correlator (dimensionless)
- A priori geocentric phase delay on scan reference time rate for the lower band computed for the correlator (dimensionless)
- Total group delay on scan reference time at the higher band (sec)
- Total group delay on scan reference time at the lower band (sec)
- Total single-band delay on scan reference time at the higher band (sec)
- Total single-band delay on scan reference time at the lower band (sec)
- Total phase delay rate on scan reference time at the higher band (sec)
- Total phase delay rate on scan reference time at the lower band (sec)
- Reference frequency of the higher band. Usually this is the lowest frequency of the band. (Hz)
- Reference frequency of the lower band. Usually this is the lowest frequency of the band. (Hz)
- Total fringe phase at the higher band (rad)
- Total fringe phase at the lower band (rad)
- Total geocentric phase on scan reference time at the higher band (rad)
- Total geocentric phase on scan reference time at the lower band (rad)
- Residual geocentric phase at the higher band on fringe reference time. (rad)
- Residual geocentric phase at the lower band on fringe reference time. (rad)
- Residual geocentric group delay at the higher band (sec)
- Residual geocentric group delay at the lower band (sec)
- Residual geocentric phase delay rate at the higher band (dimensionless)
- Residual geocentric phase delay rate at the lower band (dimensionless)
- Residual geocentric group delay rate on fringe reference time at the higher band (sec)
- Residual geocentric group delay rate on fringe reference time at the lower band (sec)
- Ratio of the offset of the scan reference time with respect to nominal observation start to the nominal observation duration

- Station order in the baseline: 1 -- direct order; -1 reversed order
- Effective scan duration at the higher band in sec
- Effective scan duration at the lower band in sec
- Elevation angle at the first station at the scan reference time (degrees)
- Elevation angle at the second station at the scan reference time (degrees)
- Azimuth angle at the first station at the scan reference time (degrees)
- Azimuth angle at the second station at the scan reference time (degrees)
- Uncertainty of the group delay estimate on scan reference time at the higher band (sec)
- Uncertainty of the group delay estimate on scan reference time at the lower band (sec)
- Uncertainty of the phase delay rate estimate on scan reference time at the higher band (dimensionless)
- Uncertainty of the phase delay rate estimate on scan reference time at the lower band (dimensionless)
- Effective ionosphere frequency for group delay for the higher band (Hz)
- Effective ionosphere frequency for group delay for the lower band (Hz)
- Effective ionosphere frequency for phase delay for the higher band (Hz)
- Effective ionosphere frequency for phase delay for the lower band (Hz)

The offset and format of each parameter is specified in the header of the ascii database.

Keyword **MKDB.OUTPUT_NAME** controls the name of the output file. Its meaning depends on **MKDB.OUTPUT_TYPE**. If the output type is *TEXT* or **AMPL**, then the value of **MKDB.OUTPUT_NAME** is the file name. If the output type is **GVF**, then the value of **MKDB.OUTPUT_NAME** is the database suffix — a character in lower case. The database name is `yyyymmdd_s` where `yyyy` is the year, mm is the month number with heading zero, dd is the day of the month with heading zero and is the suffix specified by **MKDB.OUTPUT_NAME**. Suffixes `a-e` are reserved to imported databases converted from MARK3-DBH or vgosdb formats to gvf. Suffixes `x-z` are resolved for tests.

In order to generate the output in GVF format, ***PIMA*** requires some additional information that it cannot find in FITS-IDI files with visibilities. This information is supplied in an the experiment description file. The name of that file is the value of keyword **MKDB.DESC_FILE**. If the value of **MKDB.DESC_FILE** is **NO**, then no information that is supposed to be defined in the experiment description is exported to the output database.

PIMA can put in the output database two frequency bands in the dual-band experiment. The fringe fitting procedure runs twice for a dual-band experiment. The second band can either occupy a range of IFs or a frequency group. ***PIMA*** requires to have two control files for the lower and upper bands. Task ***mkdb*** should use the control file with the upper band. The file for the upper band of dual-band data should have a reference to the control file file for the lower band. The name of the control file for the lower band is specified in the keyword **MKDB.2ND_BAND**.

Post-Solve has two slots for group delays, phase delay rates and other quantities: the 1st and the 2nd. Post-Solve assumes the first slot is the the upper frequency and the second one is for the lower frequency but it does not check.

Keyword **MKDB.2ND_BAND** should have value **NO** for single band experiment and for the control file for fringe fitting the lower band. It should have the name of the control file for the lower band inside the control file for the upper band of a dual-band experiment.

PIMA needs to know where to write the output database in GVF format. Keyword **MKDB.VCAT_CONFIG** specifies the configuration file for VLBI database catalogue. That configuration file defines two directories: directory for ascii envelope wrappers (keyword **GVF_ENV_DIR**) and directory for binary files (keyword **GVF_DB_DIR**). The same configuration files is supposed to be used by Post-Solve. Post-Solve assumes that configuration file has name $SAVE_DIR/vcat.conf . Therefore, in order Post-Solve to find the database crested by ***PIMA***, the control file should specify $SAVE_DIR/vcat.conf with environment variable SAVE_DIR expanded.

Database file in GVF format can be read with GVH library. The GVH package has routine gvf_transform that can transform from binary representation of the database to the ascii and back to binary form. The ascii representation of a database is human readable. Moreover, Post-Solve can work with both ascii and binary representation, binary representation being more than one order of magnitude faster. Transformation to an ascii representation can be useful for simple editing such as replacement of source name or station name.

Post-Solve classifies observations as good, bad, but recoverable, and bad unrecoverable. The latter category is not visible and Post-Solve does not allow to recover them. The common reasons for that: 1) SNR less than the limit specified in the keyword **FRIB.SNR_DETECTION** of the ***PIMA*** control file; 2) lack of phase calibration for a given observation when **pcal** is anything else than **NO**; failure in fringe fitting, f.e. because there are less than 3 valid accumulation periods. Since ***PIMA*** does not determine whether an observation is really detected or not, if **FRIB.SNR_DETECTION** is too high, there is a chance that Post-Solve will miss good observations. For this reason, it is recommended to set a rather low **FRIB.SNR_DETECTION** parameters, f.e. **5.0**, and then set the SNR limit in Post-Solve but hitting **J**. Post-Solve sets the SNR limit temporarily. The observations below the limit are considered unrecoverable until the limit is changed. If the SNR limit is lowered, but not below the limit used by ***PIMA***, the observation from bad and unrecoverable becomes bad and recoverable. Reducing the SNR limit in Post-Solve below the limit used by ***PIMA*** does not have effect.

A suggested strategy is to set a **low FRIB.SNR_DETECTION**, **5.0**, and after loading the database into Post-Solve set higher SNR limit in Post-Solve, **5.8**–**6.0**. After cleaning the database for outliers with SNR limit **5.8**–**6.0** the limit is lowered to the value used by ***PIMA***. Such a strategy avoids the problem of contamination the dataset with too many outliers which may cause difficulties in initial analysis, since too many outliers, say more than 10% may significantly skew residuals.

Split and export data for imaging

PIMA has a capability to format its results in the form that is suitable for both absolute astrometry/geodesy or imaging. In the latter case ***PIMA*** coherently averages the visibilities over time and frequency and applies all necessary calibrations and re-normalizations.

If a given observation, given IF does not system temperature or antenna gain, or have Tsys out of range [10, 10000]K or have zero gain, the visibility for such an observations, such an IF is discarded. Therefore, calibration for system temperature and antenna gain must be performed before running task ***splt***. In contrast, the data without amplitude calibration are still usable for absolute astrometry/geodesy. Flagging data for the time intervals when the antennas were off-sources is essential for deriving a good-quality image. Flagging can also be performed before ***mkdb***, although usually it has only a marginal effect.

Import gain curves

As of 2016, only the NRAO generates visibility data that are fully compliant with FITS-IDI specifications and contain gain curves. Data generated by other correlators do not have this information, and therefore, the gain curves should be imported. Import gain curves is beneficial for processing VLBA data, since the gain curves embedded in the database may not be the best one.

PIMA task ***prga*** (PRint GAin) prints gain for each station, each IF. It is recommended to inspect these values.

PIMA supports two formats of gain information: VLBA gain and EVN gain files. The VLBA gain format allows to specify different gain for time ranges, and therefore, this format is preferable.

Gain file specifies gain at the reference elevation, the so-called Degrees Per Flux Unit (DPFU) factor in Jy/K for R and L polarizations for a certain frequency range and a set of coefficients that specify the polynomial that describes the dependence of gain with elevation — that is why it is called "gain curve". If the elevation dependence is not known, than the polynomial has only one coefficient for degree 0: 1.0

PIMA task ***gean*** allows to import gain curves. It requires either qualifier vlba_gain or evn_gain. The value of the qualifier is the file name with gains. When the firsts qualifier is **vlba_gain** ***PIMA*** requires the second qualifier **gain_band** that specifies the band. Supported bands are class="val">90cm, class="val">50cm, class="val">21cm, class="val">18cm, class="val">13cm, class="val">13cmsx, class="val">6cm, class="val">7ghz, class="val">4cm, class="val">4cmsx, class="val">2cm, class="val">1cm, class="val">24ghz, class="val">7mm, class="val">3mm. NB: ***PIMA*** does not check whether the band is supported.

PIMA wrapper **pf.py** does this task as well. It assumes to find files vlba.gains and ivs.gains in directory specified by configuration parameter --stable-share. **pf.py** wrapper tries both gain files.

Task ***gean*** does not report an error, if it does not find a gain for the specific station, specific frequency range, specific time interval. If it does not find gain, the gain is set to zero. If the gain is zero for a given station, given IF, ***PIMA*** task ***splt*** will not export visibilities for a given station. It is recommended to inspect gain values by running ***PIMA*** task ***prga*** after importing gains in order to be sure the are

correct. Wrapper **pf.py** runs ***prga*** automatically at the end. The results an be found in the log file of task ***gain***.

Flagging visibilities with low amplitude at the beginning or end of a scan.

Data acquisition system often record before and/or after the actual scan time. The field system is supposed to record time stamp of nominal start and nominal stop time and the correlator is supposed to flag accumulation periods that were recorded when the antennas were off-source. However, it may happen that visibility data for a given time have intervals when the antenna were off-source. Usually, this happens at the beginning of the scan. This may happen because the fields system software incorrectly determine on/off time, or did not propagate it to the correlator, or the antenna was off while the fields system software reported the antenna was on. Propagation such "data" poses a serious problem for imaging. Such visibilities should be flagged during imaging stage. If left unflagged they distort an image. ***PIMA*** has task ***onof*** that analyzes the data, determines accumulation periods with the fringe amplitude at the beginning and/or the end of a scan with fringe amplitude below the threshold and flag them out. This task should run before ***splt***.

PIMA supports two mechanisms for flagging visibility. When ***PIMA*** loads the data, it checks all visibilities for inconsistencies, such as lack of autocorrelation, wrong source indices, duplicates, etc. It puts indices of damaged visibilities in a separate file and bars them from loading. These visibilities are considered unrecoverable. ***PIMA*** supports keyword **TIME_FLAG_FILE** that defines a so-called time epoch flag file. The file flag file consists of records in plain ascii that defines the visibility to be flagged. A record consists of four words separated by one or more blanks. The first word is the observation index, the second word is the index of the accumulation period within that observation, and the third word is a flag. The flag is multiplied by the visibility. Flag 0 means the visibility will not be used for further processing. Lines that start with # are considered as comments and discarded by ***PIMA***.

Task ***onof*** uses this mechanism to flag out bad accumulation periods. It determines accumulation periods with low amplitude and write their indices and observation indices into the time epoch file. Other tasks, such as ***frib***, ***splt*** read this file and flag out visibilities that have corresponding indices.

Task ***onof*** does not require qualifiers. Its behavior is determined by a number of keywords of the control file. If Keyword **ONOF.GEN_FLAGS_MODE** is **CREATE**, ***PIMA*** will ignore the previous contents of the flag file and overwrite it. If **ONOF.GEN_FLAGS_MODE** is **UPDATE**, then ***PIMA*** will honor input of the flag file specified by the keyword **TIME_FLAG_FILE** and update it. In this mode ***PIMA*** will never reduce the number of flagged visibilities, but it can only increase them.

In order to determine accumulation periods at the beginning or the end of the scan that have to flags, ***PIMA*** needs to get a hint which interval to consider as "good". Two keywords, **ONOF.KERNEL_START_SHARE** and **ONOF.KERNEL_END_SHARE** determine the so-called kernel interval. The value of these keywords are the offsets or the kernel start and stop time as a share of the total nominal scan length. The share runs from 0 to 1. instance, **ONOF.KERNEL_START_SHARE**: **0.25**, **ONOF.KERNEL_END_SHARE**: **0.80** specifies the kernel interval that starts at 0.25*scan_length and ends at 0.80*scan_length. However, the length of the kernel interval is limited by the value of **ONOF.COHERENT_INTERVAL** (in seconds). If parameters **ONOF.KERNEL_START_SHARE** and **ONOF.KERNEL_END_SHARE** specify the interval longer than **ONOF.COHERENT_INTERVAL**, then **ONOF.KERNEL_END_SHARE** is reduced in such a way that the kernel interval will be close, but not exceeding **ONOF.COHERENT_INTERVAL**.

PIMA first computes coherently averaged complex visibilities over the kernel interval and then tries visibilities coherently averaged over frequency over and accumulation periods backwards from the start of the kernel interval and forward from the end of the kernel interval. ***PIMA*** computes the frequency averaged visibility amplitudes, computes the ratio of the visibility amplitude over the trial accumulation to the amplitude over the kernel interval and tries two criteria: a) if the ratio is less than (1-k*σ_a), where k is the value of the keyword **ONOF.NSIG_THRESHOLD** and σ_a, then the accumulation periods is marked as a candidate for exclusion; b) if the ratio is less than **ONOF.AMPL_THRESHOLD**, then the accumulation periods is marked as a candidate for exclusion. If **ONOF.AMPL_THRESHOLD** is zero then the first criterion is disabled. If **ONOF.AMPL_THRESHOLD** is zero, then the second criterion is disabled. If **ONOF.NSIG_THRESHOLD** > 0.0, the second criteria is used only if the ratio of the amplitude in the kernel interval to the uncertainty amplitude at a given accumulation period is less than **ONOF.NSIG_THRESHOLD**.

If ***PIMA*** finds k consecutive candidates, where k is the value of keyword **ONOF.MIN_LOW_AP**, then it flags them and all consecutive visibilities at the beginning or the end of the scan.

Criterion **ONOF.AMPL_THRESHOLD** is suitable for observations with high SNR and long accumulation periods. Visibility amplitude computed over one accumulation period has a large scatter and the fluctuations caused by noise may be mistakenly considered as the source being off source.

Criterion **ONOF.NSIG_THRESHOLD** is suitable for both low SNR and high SNR observations. Value **3** was found satisfactory for most of cases. Task ***onof*** is not able to find time interval when antennas were off-source for observations with low SNR, say less than 10, because amplitude fluctuations due to random noise become too large to be distinguished from antennas being off-source.

Running task splt for splitting and exporting data for imaging

Using results of fringe fitting, ***PIMA*** performs coherent averaging over time and frequency after rotation phases according to group delays and phase delay rates, applies calibration for system temperature, gain curves, bandpass re-normalization, combines all visibilities of a given source and writes averaged visibilities and their weights into output binary files in FITS format that are suitable for imaging with AIPS or DIFMAP.

Task ***splt*** processes the data on source basis. A user can specify the source name that will be processed or to request to process all the sources in a cycle. Keyword **SPLT.SOU_NAME** controls this behavior. Its value can be either B-name, or J-name or ALL, which means to process all the sources.

Keyword **SPLT.FRQ_MSEG** specifies the number of spectral channels to be coherently averaged out. A usual choice for processing legacy data is to specify the number of spectral channels in an individual IF. In that cases all spectral channels will be averaged out. ***PIMA*** does not average spectral channels across IF boundaries. That means that the maximum value of **SPLT.FRQ_MSEG** is limited to the number of channels in an IF. Starting from 2014, observations at 512 MHz band became more and more common. Averaging across 512 MHz band may result to image smearing. Therefore, it may appear beneficial to decrease the number of spectral channels that will be averaged. Task ***splt*** averages **SPLT.FRQ_MSEG** spectral channels in one output IF. If **SPLT.FRQ_MSEG** is less than the number spectral channels in one IF, then the output dataset will have more output IFs than the input dataset.

Keyword **SPLT.TIM_MSEG** specifies the number of accumulation periods to be coherently averaged out. The interval of time that with **SPLT.TIM_MSEG** accumulation periods is called segment. A usual choice is to select segment duration 10–20 seconds. DIFMAP allows to averaged data further increasing segment length (DIFMAP task uvaver), but after the data have been averaged, there is no way to undo averaging. In a case when a very large map will be made, **SPLT.TIM_MSEG** may be reduced in order to avoid image smearing.

Keyword **SPLT.SNR_MIN** specifies the SNR threshold. Observations with the SNR over all frerquencies and over total scan duration less than that threshold are excluded for processing and are not writtebn in the output file.

One output visibility is a result of coherent averaging over **SPLT.FRQ_MSEG***SPLT.TIM_MSEG** input visibilities. It should be noted that if **SPLT.FRQ_MSEG** is not an integer divisor of the number of spectral channels and **SPLT.TIM_MSEG** is not an integer divisor of the total number of accumulation periods of an observation, the number of used input visibilities can be less at the end of the observation or at the upper part of the spectrum can be less.

Phases of input visibilities are rotated before averaging according to phase delay rate and group delays that are found during fringe fitting. ***PIMA*** reads results of fringe fitting from the file specified by the keyword **FRINGE_FILE**.

Since ***PIMA*** performs fringe fitting for each observation independently, in general, fringe reference time is different. As a result, the misclosure of the *raw* phases from the contribution of group delay and phase delay rates is not zero. If **SPLT.STA_BASED**: **YES** or **ALL**, then ***PIMA*** performs a procedure that converts baseline-dependent group delays and phase delay rates to station-based that automatically have zero misclosure and therefore, applying phase rotation from the results of fringe fitting does not change misclosure in original visibilities. ***PIMA*** performs this conversion at each scan. It may happen that observations of a given scan have to be split into several subarrays. A subarray is a set of observations that has common baselines with *every* station within a subarray. For example a station array `ABCDRFG` may have to be split into two subarrays `ABCD` and EFG if there are no *usable* observations at baselines `AE,AF,AG,BE,BF,BG,CE,CF,CG,DE,DF,DG`. If in this example, there are no usable data at baseline `FG`, there will be three subarrays: `ABCF`, `EF`, `EG`. ***PIMA*** splits the data into subarrays for processing each scan. If a subarray for a given station was already used in the previous scans, ***PIMA*** assigns the observations to that subarray. In a case if a new subarray has all the stations that were in one of the previous subarrays, ***PIMA*** assigns observations to that subarray. In a case is a new subarray has all the stations that were in one of the previous arrays, plus one or more new station, ***PIMA*** extends that subarray, and assigns the observations to that subarray.

In a case if all scans were scheduled at all antennas and fringes were detected at all observations and no observations were excluded, there will be only one subarray. If one of these conditions is violated, ***PIMA*** may end up with many subarrays. Since splitting data in many subarrays reduces the number of phase and amplitude misclosures, in general it is undesirable to have many subarrays. ***PIMA*** supports a procedure subarray consolidation controlled by the keyword **SPLT.SUBARRY_CONSOLIDATION**. Value **NO** means to disable subarray consolidation. Value **MIN** instructs ***PIMA*** to preform minimal subarray consolidation: if all stations of subarray `A` are present in the subarray `B`, then the subarray `B` is consolidated with subarray `A`. Value **MAX** instructs to preform maximum subarray consolidation: if a subarray has at least one common station with subarray `B`, both subarrays are consolidated.

In order to compute station-dependent group delay, phase delay, and group delay correctly, baseline-dependent group delay, phase delay, and group delay should be correct. Fringe fitting provides a wrong result for a non-detection, or an observation affected by RFI. Group delays of non-detections have a uniform distribution over the fringe search window, which is several orders of magnitude larger than the scatter of group delays for normal observations. Therefore, a care should taken in order to block using bad group delays by task ***splt***. It is recommended to process the data with VTD/Post-Solve in order to identify outliers and exclude them as input to the procedure for computing station-based group delay, phase delay and group delay rate using keyword **EXCLUDE_OBS_FILE**. If **SPLT.STA_BASED**: **YES** is used, the observations excluded for computing station-based quantities will remain excluded from being further processed and written in the the output file. However, in general, this approach is too restrictive. When **SPLT.STA_BASED**: **ALL** is specified, the filters specified by kewords **FRIB.SNR_DETECTION**, **EXCLUDE_OBS_FILE**, **INCLUDE_OBS_FILE**, and **OBS** is applied only to the input data of the procedure for computing station-based quantities. All observations between the stations for which station-dependent group delays, phase delay, and group delays are computed are used for further processing, regardless whether they passed the input filter or not. Though if there were no observations at baselines at certain stations, there will be no station-based

quatnties for these stations, and therfore, no for the observations with these stations will be used for generating the output. Value **ALL** is recommended.

Alternatively, when **SPLT.STA_BASED**: **NO**, *PIMA* does not convert baseline-dependent quantities to station-based. Important: phase misclosure is distorted when this option is used. This option is useful when the data are to be processed on a baseline basis.

Keyword **SPLT.SNR_MIN** specifies the SNR threshold for the output. Observations with the SNR over all frequencies and over total scan duration less than that threshold are excluded for processing and are not written in the output file. This criteria is used after the averaged visibilities are computed. If **SPLT.STA_BASED**: **YES** or **SPLT.STA_BASED**: **NO** was used, observations with SNR less than **FRIB.SNR_DETECTION** will remained excluded even if their SNR is equal or greater than **SPLT.SNR_MIN**. However, when **SPLT.STA_BASED**: **ALL** is specified, observations with SNR less than **FRIB.SNR_DETECTION** may become valid. For instance, if stations B and C have low sensitivity, but station A has high sensitivity, visibilities at baseline BC can be determined using phase delay rate and group delay at baselines AB and AC. The SNR of visibilities at baseline BC may be very low. Keyword **SPLT.SNR_MIN** allows to filter out such visibilities with low SNR computed on the basis fringe fitting results from visibility analysis at other baselines.

There are several options to generate the output in a case of dual-polarization data. Keyword **SPLT.POLAR** specifies which polarizations to put in the output: **ALL** for all polarizations that present in the data, **PAR** only for **RR** or **LL** polarizations. Other supported values: **I**, **RR**, **RL**, **LR**, and **LL**.

PIMA computes averaged visibilities and their weights. The algorithm for weights computation is controlled by keyword **SPLT.WEIGHT_TYPE**. According to FITS specifications, weight is defined as reciprocal to the fringe amplitude variance. Value **ONE** forces *PIMA* to set all weights to 1. Value **OBS_SNR** instructs *PIMA* to compute weights on the basis of signal to noise ratio SNR. The segment weight is Ampl/SNR**2, where SNR is the signal to noise ratio over visibilities of a given segment. The segment SNR computed from the SNR over all visibilities used in fringe fitting and scaled by square root of the the ratio of visibilities in the segment to the total number is used visibilities in the observation. When **SPLT.WEIGHT_TYPE** is **OBS_RMS**, *PIMA* computes variance of the fringe amplitude over the observation and assigns weights for all segments reciprocal to this estimate of variance. When **SPLT.WEIGHT_TYPE** is **SEG_RMS**, *PIMA* computes variance of fringe amplitude over visibilities of a given segment and assigns weights reciprocal to this variance.

Method **SEG_RMS** is the preferable, since it accounts for temporal variation of the variance. However, it requires a sufficient number of segments for computing meaningful variance. When **SPLT.WEIGHT_TYPE** is **AUTO**, *PIMA* uses different ways to use the weight depending on the number of accumulation periods in a given segments. If the number of accumulation periods per segment specified in the keyword **SPLT.TIM_MSEG** is equal or greater than the threshold (currently 8), the variance is computed over visibilities of a given segment. Otherwise, the variance will be computed over all visibilities of the observation (equivalent to **OBS_RMS**). **SPLT.WEIGHT_TYPE**: **AUTO** is recommended for a general case.

When computing calibrated amplitude *PIMA* applies two renormalizations unless a user disables it. When **SPLT.AUTOCORR_NRML_METHOD**: **AVERAGED**, *PIMA* normalizes system temperature for masking autocorrelation. The system temperature is measured by integrating the total power over entire IF. When a portion of the bandwidth where Tsys was computed is masked out, the total power is changed. In general, the spectrum of noise is not constant over the band, it is proportional to the autocorrelation. *PIMA* normalizes autocorrelation to have the average equal to 1 over the nominal IF width. When **SPLT.AUTOCORR_NRML_METHOD**: **AVERAGED**, *PIMA* computes the mean autocorrelation over the used portion of the bandwidth within each IF, which in general is not 1. Then fringe amplitude is divided by the mean autocorrelation. **SPLT.AUTOCORR_NRML_METHOD: NO** disables applying this renormalization. It is recommended to use **SPLT.AUTOCORR_NRML_METHOD: AVERAGED**.

When **SPLT.BPASS_NRML_METHOD**: **WEIGHTED**, *PIMA* divides the fringe amplitude by the square root of the product of the square root of bandpass renormalization factor. The representative bandwidth of the intermediate frequency used for re-normalization is specified by the keyword **SPLT.BPASS_NRML_RANGE**. The value of this keyword is two numbers from 0 to 1 separated by the colon. These number specify the lower and the high part of the representative bandwidth as a share of the total bandwidth. Example: **SPLT.BPASS_NRML_RANGE: 0.25:0.80**. They define the representative bandwidth as `[F_low + Bl*Fw, F_low*Bh*fw]`. *PIMA* computes renormalization factor `R = (sum` B_r/N_r `) / Sum` B_t/N_t, where B_r — bandpass in the bandwidth is `[F_low + Bl*Fw, F_low*Bh*fw]`, N_r — the number of points in that bandwidth; B_t bandpass in the total bandwidth, N_t the total number of points in the entire IF. Factor R is multiplied by every point of the bandpass and makes its normalized over the representative bandwidth `[F_low + Bl*Fw, F_low*Bh*fw]`. Usually `R > 1.0` For example, if the IF bandwidth is 16 MHz and **SPLT.BPASS_NRML_RANGE: 0.25:0.80**, then the representative portion of the bandwidth used for renormalization starts at 0.25*16=4.0 MHz and ends at 0.8p*16=12.8 MHz. Thus, the portion `[4.0, 12.8] MHz` of the total bandwidth `[0, 16] MHz` is considered representative and the bandpass is normalized to be 1 over the representative portion of the bandwidth. The share of the representative bandwidth depend on quality of hardware. **0.25:0.80** is a good choice for most of the cases.

In addition to generating the output averaged visibilities in FITS-IDI format, *PIMA* task ***splt*** will generate total visibilities averaged over entire can when **SPLT.TOTAL_UV: YES** is specified. **NB:** unlike to averaged visibilities written in the FITS-IDI format, the total visibilities are refereed to the band reference frequency. Name of a file with total visibilities obeys the following convention: `JJJJJJJJJJ_B_uvt.txt`, where `JJJJJJJJJJ` is the 10-character long J2000 source name and B is the band defined in the keyword **BAND**. Total visibilities are written in the plain ascii format. See document Total_visibilities_format.txt for format description.

Compute gain correction

It is rather common from some stations to have some IFs with gain to be wrong by a certain factor. This may be due to unaccounted change in gain curve, or due to systematic error in Tsys. During imaging process gain can be adjusted using amplitude closure. This procedure is called amplitude self-calibration. However, a source should be relatively bright and UV coverage should be rather dense for amplitude self-calibration to produce good results. If the gain is off by a factor that is constant over entire experiment, the gain correction determined from imaging one source can be used as a priori for imaging other sources. This is just that *PIMA* task ***gaco*** (GAin COrrection) does.

PIMA supports keyword **SPLT.GAIN_CORR_FILE** that specifies so-called gain correction file. This file in plain ascii format defines factors for each station and each IF by which fringe is multiplied when task ***splt*** runs.

These factors can be assigned manually or automatically. Task ***gaco*** can run in two modes: manual and automatic. In manual mode *PIMA* expects a qualifier init with the value of the a priori factor. Value **1.0** or **0.0** are usual choices. Task ***gaco*** with qualifier **init** sets all gains to the initial value. Gain correction 1.0 means no correction. Gain correction 0.0 means all IFs all stations should be deselected.

Task ***gaco*** writes the gain correction file. If task ***gaco*** was invoked in **init** mode, the gain correction should be edited in order to be useful. Example: station KP-VLBA had fringe amplitude a factor of 8–10 lower in IFs 3 and 4, and a user would like to get rid of them for imaging purposes. Than *PIMA* task ***gaco*** with qualifier **init** and value **1.0** is called. After that the user edits the gain correction file that the task created ad changes gain correction for KP-VLBA IFs 3 and 4 from 1.0 to 0.0.

In order to to compute gain corrections in the automatic mode, several images should be made first and self-calibrated visibilities be saved. Then *PIMA* analyzes the ratio of original amplitudes before amplitude self-calibration and after amplitude self-calibration and determines their ratios for each station and each IF using least squares. These gain corrections are equivalent to the factors should by task CORPLT of DIFMAP package. In order to do it, *PIMA* should find visibilities before and after imaging. *PIMA* supports convention that a file with visibilities before imaging have name `JJJJJJJJJJ_B_uva.fits`, where `JJJJJJJJJJ` is the 10-character long J2000 source name and B is the band defined in the keyword **BAND**. *PIMA* task ***splt*** created files with averaged visibilities in this format. *PIMA* expects files with self-calibrated visibilities after imaging to have names in the form of `JJJJJJJJJJ_B_uvs.fits`.

When used in the automatic mode, *PIMA* task ***gaco*** expects two qualifiers, **sou** and **dir**, the first is mandatory and the second is optional. The value of the first qualifier is a comma-separated source list. The value of the second optional qualifier specifies the directory where files with original and self-calibrated visibilities can be found (they should be in the same directory). If the second qualifier is omitted, *PIMA* will search for visibilities in the same directory where task ***splt*** put them: `SSSSS/EEE_uvs`, where `SSSSS` is the *PIMA* scratch directory specified by the keyword **EXPER_DIR** and `EEE` is the experiment name specified by the keyword **SESS_CODE**.

PIMA task ***gaco*** used in the automatic mode computes the gain correction file. If a given station observed no sources from the list, the gain correction for that station is set to 1.0. A user may edit the gain correction file, for instance setting zeros for IFs for certain station(s). Setting the gain correction to zero will effectively flag out these IFs for *imaging purposes*, while these IFs are still available for other tasks, for example, fringe fitting.

PIMA task ***splt*** uses gain file, unless **SPLT.GAIN_CORR_FILE**: **NO**. It multiplies the calibrated visibility by the product of gain corrections of both stations of a baseline. It writes the used gain corrections into output FITS file in two places: 1) as an ascii table in the HISTORY records of the main table, 2) as a new table in GACO. It is possible to run task ***taco*** the second time. *PIMA* searches for gain correction in the FITS file with calibrated visibilities, applies these corrections as a priori and writes updated **total** gain correction with respect to a case when no gain correction is applied. Thus, if to run ***gaco*** task more than once, the result will be approximately the same.

Use case of preparing the data suitable for imaging

The imaging analysis and absolute astrometry/geodesy pipeline has a common beginning: loading the data; parsing log files, checking logs, checking phase calibration, cleaning phase calibration for spurious tones, checking autocorrelation, checking Tsys; running coarse fringe fitting, computation of the bandpass, running fine fringe fitting. After that point the pipelines diverge. Next task will be selecting good reference sources and running task ***splt*** for them. A good reference source is strong, observed at all baselines and has relatively simple structure.

It is first recommended to run ***splt*** task *for several reference sources* with **DEBUG_LEVEL**: **2** or **3** and with **FRIB.SNR_DETECTION**: **6.0**. It is recommended to investigate the ***splt*** log file. Search there for lines with `PIMA_SPLT_FITSTA SOU:`, for instance with using grep. This line provides the statistics for a subarray. (Remember: a scan may have one or more subarrays that can be later consolidated into one subarray). An analyst should examine the column followed by `MaxDev_Gr_Del`. This column provides the maximum residual of transforming baseline-dependent group delay to the station-based. Typical value of this residual for a **detected source** is 50–300 ps. A source with complicated structure may have residual 1 – 2 ns. But a residual, say 1000 ns, indicates that at least one observation in the subarray was a **non-detection**. Even one non-detection can spoil entire dataset for a given source to a level of uselessness. If you see large maximum residual in the subarray statistics, just look in the preceding lines that start with `PIMA_SPLT_FITSTA Sou:`. Identify sources with large residuals (say more than 12 &ndash 3 ns), identify their observation indices that can be found in the column followed

by `OBS:`, and add these observation indices to the exclude file. Then run *PIMA* task ***splt*** with specifying this file in keyword **EXCLUDE_OBS_FILE** once again. Then inspect the log file again.

After inspection shows no subarray with large residuals, the selected reference sources are imaged using phase and amplitude self-calibration. If for some reason an image of one of the reference sources is not satisfactory, another that bad source should be replaced with another source.

When good images were produced for all reference sources, *PIMA* task ***gaco*** is invoked with qualifier **sou** with the comma-separated list of reference sources. After that the gain correction file is inspected. If some IFs at some stations are to be masked out, corresponding values of the gain correction file are replaced with zeroes.

After that task ***splt*** runs over entire dataset by specifying **SPLT.SOU_NAME**: **ALL**. A care must be taken to use only detected observations. There are two approaches for cleaning the dataset for non-detections. The first approach is to raise the SNR detection limit defined by keyword **FRIB.SNR_DETECTION**. Depending on the search window, the detection limit is 5.3–6.0, the wider the window the higher the limit. Another approach is to run full absolute astrometry/geodesy pipeline and exclude those observations that are flagged out by Post-Solve. Solve will flag non-detections and other "bad" observations, such as those affected by RFI, low fringe rate problem, etc. Extraction of the list of observations that is performed by program gvf_db that is a part of Solve package. Usage:

```
gvh_db database_name mode
```

where mode is either 10 for a processing a single-band experiment or the upper band of a dual-band experiment and 20 is for processing lower band of a dual-band experiment. The advantage of the second approach is that detections as weak as 4.8 can be used since Post-Solve will eliminated non-detections and other bad detections. Re-fringing allows to recover weak detections. The disadvantage is that Post-Solve should be installed, and running astrometry analysis requires extra efforts.

After running ***splt*** over the entire dataset, the ***splt*** log should be examined the same way as we did when we processed reference sources.

Task ***splt*** creates calibrated visibilities of all the sources, except those that have too few detections and put then in directory `SSSSS/EEE_uvs`. Calibrated visibilities are used for imaging with DIFMAP, AIPS or another software. Result of imaging are two files per source: a file with self-calibrated visibilities in FITS format and image in FITS image format. The image contains two tables: a set of CLEAN components and the binary image that was generated from the table of CLEAN components.

OPAcity Generation

Task ***opag*** provides and interface to the SPD (Slant Path Delay) package for computation of opacity and atmosphere brightness temperature using the output of numerical weather models. This task accesses a remote Web server, downloads data files with slant path delays, opacitites, and atmospheric brightness temperatures, and stores then in directory `SSSSSS/EEE_sob`

Keyword **spd_url** is required. It specifies the URL where the datafiles with atmosperic opacity and brightness temperature are stored. Recommended web-site: http://atmospheric-propagation.smce.nasa.gov/spd/asc/geosit.

OPAcity Loading

Task ***opal*** parses the files with opacity and brightness temperature on the 3D elevation-azimuth-time grid created by package SPD that was invoked by task ***opag***, interpolates them for the scan start and scan end and writes down into *PIMA* internal data structure. It also computes receiver temperature Trec by subtracting atmosphere temperature from measured system temperature. Task ***opal*** also initializes arrays with so-called modeled and cleaned system temperature, i.e. removes if they existed before, fills them zero and sets flags "not available".

Compute TSys MOdel

Task ***tsmo*** computes the so-called modeled Tsys. This task works in two modes, "`elevation`" mode and "`if`" mode.

In the "if" mode the task flags Tsys values that violate the assumption that the ratio of Tsys between IFs of the specified band is constant in time for a given experiment. First, *PIMA* finds the reference IF within the band specified by keywords **BEG_FRQ** and **END_FRQ**. It tries the IFs one by one. It computes the logarithms of the the ratio of Tsys of a given IF to the Tsys of the reference IF, finds the median ratio, computes the rms of the deviation and removes the outliers that are **FRIB.NOISE_NSIGMA** times greater than the rms. The trial reference IF that has the minimum number of outliers becomes the reference IF. After that *PIMA* computes the average Tsys ratios and stores flags. In a case if for a given observation Tsys at the reference IF has to be flagged, a temporary reference IF is sought.

In the "elevation" mode the task decomposes Tsys into the product `T_sys = T_o * a(t) * b(e)`

where OL>

- a(t) is a function of time represented by a linear spline;
- b(e) is a function of elevation represented by linear spline.
- T_o is the minimal system temperature. Functions a(t) and b(e) are normalized to have minimal value 1.0

Parameter T_o and coefficients of the spline a(t) and b(e) are found by iterative non-linear LSQ. Outliers are detected and flagged out during this procedure. If ***tsmo*** task in "`if`" mode ran before, the input for this procedure is the geometric average of Tsys, except those that were previously flagged out, i.e. `T = (Π Tsys(i))^(1/n)`. Otherwise, Tsys for **BEG_FRQ** is taken. It is strongly recommended first to run ***tsmo*** task in "`if`" mode and then in "`elevation`" mode since the latter mode is less stable to outliers. Potentially, a large outlier(s) can distorts significantly the solution.

After the task ***tsmo*** computes Tsys decomposition, it computes the so-called modeled Tsys using parameters of the decomposition for start and stop date of every scan and stores it in the appropriate slot. In addition, *PIMA* computes so-called cleaned array of Tsys. Cleaned Tsys coincides with modeled Tsys for the points with missing or flagged measured Tsys and coincides with measured Tsys for all other points.

Task ***tsmo*** requires keyword **mode** that accepts comma separated values **if** and **elev**. Unless a user has reasons to do otherwise, it is recommended to use both keywords: ***tsmo*** **mode if,elev**. When two values are specified, *PIMA* will first execute Tsys model computation in the "`if`" mode and then in "`elevation`" mode.

For experiments with two band, for instance S/X or C/X, task ***tsmo*** should run for two bands separately since the ration of Tsys between IFs of different receivers may change. Elevation dependence is frequency dependent, and therefore, should be modeled separately.

Results of task ***tsmo*** can be examined with task ***tspl***. When task ***tspl*** is invoked with keyword **TSYS: MEASURED**, it will show measured Tsys. When it is invoked with **TSYS: MODELED** it will show modeled Tsys. When it is invoked with **TSYS: CLEANED**, it will show modeled Tsys. In general, **TSYS: CLEANED** is recommended: from one hand, measured Tsys is used. From the other hand, obvious Tsys outliers are eliminated.

It should be remembered that ***opal*** purges results of task ***tsmo*** and task ***load*** purges results of both tasks ***opal*** and ***tsmo***.

Automatic imaging

In principle, totally automatic imaging is feasible, but development such a system would require an order of magnitude more efforts than it was invested in *PIMA*. Therefore, *PIMA* provides partially automatic imaging capability. In the framework of the approach implemented in *PIMA*, a user runs fringe fitting, runs astrometry/geodesy solution with Post-Solve, runs task ***onof***, runs ***splt*** for 2–4 reference sources, produces their images manually, runs task ***gaco***, and then runs wrapper script **pf.py** task ***map***. Task ***map*** of **pf.py** wrapper invokes *PIMA* task ***gain***, ***splt***, calls DIFMAP in a batch mode, generates images of all sources, and creates pictures of all imaged sources in gif format. **pf.py** scripts puts calibrated visibilities, images, self-calibrated visibilities, images, pictures of source images, and pictures of scan-averaged flux densities as a function of baseline lengths into directory `SSSSS/EEE_uvs`, were `SSSSS` is the *PIMA* scratch directory specified in the keyword **EXPER_DIR** and `EEE` is the experiment name specified in the keyword **SESS_CODE**. *PIMA* generates the for each source with enough usable data the following files:

- calibrated visibilities with name `JJJJJJJJJJ_B_uva.fits`
- slef-calibrated visibilities with name `JJJJJJJJJJ_B_uvs.fits`
- brightness distributions with name `JJJJJJJJJJ_B_map.fits`
- picture of image `JJJJJJJJJJ_B_map.gif`
- picture of scan-averaged self-calibrated visibility as a function of baseline length `JJJJJJJJJJ_B_rad.gif`

where `B` is the band specified in keyword BAND,

Quality of automatically generated images not always satisfactory. Most common reasons: a) not all visibilities when the antennas were off are filtered out; b) visibilities at some IFs require significant corrections, say more than 50%, c) non-detection visibilities used by ***splt***; d) the source is larger than the default field of view. Therefore, automatically generated images require scrutinizing. Recommended approach: an analyst using utility gqvew screens pictures of images and self-calibrated visibilities and selects those sources which images look suspicious. These selected sources are

imaged manually. It is recommended to adhere the same name conventions of output files that ***PIMA*** uses.

Re-fringe the data using results of astrometry/geodesy solution

In general, an iteration ***PIMA*** → Post-Solve → ***PIMA*** is required. Interactive Post-Solve is used for a) setting up parameterization; b) outliers elimination; c) re-weighting; d) setting up constraints. Results of Post-Solve are written in the database during operation "database update" (Cntrl/U). Observation suppression status is kept in the database. This information can be extracted and used for excluding suppressed observations by task ***splt***. Sole suppresses observations with residuals greater than some limit, typically 3–4 σ. There are several reasons why an observation may have a large residual: a) deficiency in the theoretical model; b) low fringe rate that results in ***PIMA*** selecting correlation between phase-calibration signal; c) ***PIMA*** picking up a local maximum in the Fourier transform of visibilities. If the residual is caused by deficiency in the ionosphere or atmosphere model, such an observation is "bad" for astrometry/geodesy, but good for imaging. If the a priori source position used by ***PIMA*** had a large error, say more than 0.5"–1", a quadratic term in residual fringe phase appears, group delay is biased, and the SNR is reduced. This problem can be alleviated if fringe fitting is repeated with corrected a priori source position. ***PIMA*** checks the difference between the source position used by the correlator and the a priori used by ***PIMA*** that it takes from the supplied catalogue. If the differences exceeds a certain threshold ***PIMA*** computes phase correction that compensate the quadratic term. This usually fixes the problem. NB: the dataset should be reloaded in order to change in the a priori source catalogue to take effects. Similarly, if a source position used by the correlator was correct, but the source position in the ***PIMA*** catalogue is wrong, f.e. due to a typo or wrong source association, ***PIMA*** will apply wrong correction and may spoil data that are good otherwise.

If an observation is suppressed because ***PIMA*** picked wrong maximum in the Fourier transform of visibilities, it is possible to correct it. We can predict group delay rather precisely after Post-Solve solution: several nanoseconds at 2 GHz and several hundreds picoseconds at 8 GHz and higher. Therefore, we can guide ***PIMA*** where to search for the maximum and fringe affected data once more. This procedure is called re-fringing.

Solve has a special mode that prints residuals and a priori delay computed by VTD. To turn this mode, hit key **A** at the "Last page" menu and rewind spool file (Find "menu 1" set (C)hange Spooling current: on and hit key **;** to rewind the spool file with solution listing, i.e. to purge its previous contents. After that just run LSQ solution by hitting key **Q**, scroll the listing by hitting blank key two times and leave Post-Solve by hitting key **T**. Then copy the spool file into the experiment directory under name `EEE_B_init.spl`, where `EEE` is the experiment name and `B` is band. Using information in the residual file, one can construct command line for ***PIMA*** with modified keywords **FRIB.DELAY_WINDOW_CENTER**, **FRIB.RATE_WINDOW_CENTER**, **FRIB.DELAY_WINDOW_WIDTH**, **FRIB.RATE_WINDOW_WIDTH** in such a way that ***PIMA*** will search for the maximum within 1–3 ns of the group delay predicted on the basis of Post-Solve solution. Program samb that is a part of Post-Solve package does this for you.

Usage:

```
samb -p {pima_control_file} -w {window_semi_width_in_nsec} -s {snr_min)
     -r {residual_file} -o {output_file}
```

Parameter `pima_control_file` is the name of ***PIMA*** control file. Parameter `window_semi_width_in_nsec` is new window for group delay search. Recommended value is 5 times the wrms of residuals. Parameter `snr_min` is the new SNR limit. Recommended value 4.8. Parameters `residual_file` is the fike with residuals generated by Post-Solve. Finally, parameter `output_file` specifies the name of the output command file.

Program samb analyzes the file with Post-Solve residuals, finds outliers marked by character > or `R` in the 8th column, computes the expected residual group delay delay with respect to the a priori model used by the correlator, which in general is different than a priori models used by VTD plus adjustments found by Post-Solve, and uses this value for **FRIB.DELAY_WINDOW_CENTER** argument for ***PIMA*** command line for re-fringing.

We need to save Post-Solve solution before running ***PIMA*** by hitting `CNTRL/U` key. Then we execute the command line generated by ***PIMA***. Re-fringing may or may not find correct maximum in the Fourier transform of visibility data. The control file generated by samb writes the results in file `VVVVV/EEE/EEE_B_refri.fri` and residuals in `VVVVV/EEE/EEE_B_refri.frr`, where `VVVVV` is the ***PIMA*** scratch directory, `EEE` is the experiment name specified in the keyword **SESS_CODE** of the ***PIMA*** control file, and `B` is the band name.

Next step is to extract records in the output fringe output and fringe residual files generated by the command file created by samb, and to add those that have SNR greater than the limit specified by samb command *to the end* the main fringe results and fringe residual files. Remember, process fringe results file consecutively. If there is more than one record corresponding to the same observation, the latest record overrides the previous record(s).

Next step is to create a GVF database using updated files with fringe results and fringe residuals with ***PIMA*** task ***mkdb***. There is a caveat. When we updated the database, it stores auto suppression and user suppression flags. These flags store the status of observations before re-fringing. Re-fringing may change observation status: an observation that was considered non-detection in the first fringing, may become detected in the re-fringing. Post-Solve does not allow to change status of an observation marked as non-detection. Program `gvf_supr_promote` solves this problem. It updates flags "not detected". If an observation was not detected in the first fringing, but detected during re-fringing, the flag "not detected" is cleared and flag "suppressed" is set. Program `gvf_supr_promote` is a part of Post-Solve. It accepts the full database name, including and extension .env, as an argument. Alternatively, the same operation can be performed with wrapper **pu.py**. Wrapper **pu.py** has two arguments: experiment name and band.

Operations samb, ***PIMA***, and update of suppression flags can be performed with wrapper **pr.py**. Wrapper **pr.py** requires as the first argument the experiment name, as the second argument low case band, as the third argument the SNR limit. Depending on band, **pr.py** will select delay window semi-width. The delay semi-width can be overridden with optional argument **-delwin**. Optional flag **-nodb** causes the wrapper to update only fringe results without creation of a database. This option is necessary when a low band of a dual-band experiment is re-fringed.

After the database is updated, it should be processed with Post-Solve once more. Some observations that were previously suppressed as outliers (ideally all) can be restored. Observations that were considered as non-detections and therefore were considered as unrecoverable if detected during re-fringing appears as "bad", but recoverable. Post-Solve program ELIM in restoration mode should be executed and the database be updated. Usually, there is no need to make a next iteration, unless an error has been made that should be corrected.

Data analysis pipeline

The recommended pipeline consists of three steps: 1) fringe fitting; 2) astrometry/geodesy; 3) imaging. Step astrometry/geodesy can be used for imaging analysis or can be skipped.

Fringe fitting pipeline

1. Create ***PIMA*** configuration file for the experiment.
2. Load FITS-IDI data. Command: **pf.py EEE B load**.
3. Parse log files. Not needed for processing VLBA data after 2014. Command: **pf.py EEE B logs**.
4. Load calibration information into ***PIMA*** internal data structure. Not needed for processing VLBA data after 2014. Command: **pf.py EEE B gean**.
5. If all phase calibration tones are extracted, run automatic phase calibration masking with ***PIMA*** task ***gepm***, then examine phase-cal with ***PIMA*** task ***mppl***. Check phase cal tones for all the stations. If additional spurious signals are found, add the infected tones to the mask definition file and translate the mask definition file to a phase cal mask using task ***pmge***. Iterate the process till plots of phase calibration show satisfactory results. If 1 or 2 tones are IF are extracted, check phase calibration with task ***pcpl***. If phase calibration for some stations is too noisy, disable phase cal for these stations with task ***gean***.
6. Rug coarse fringe fitting. Command: **pf.py EEE B coarse**.
7. Run bandpass generation in the inspection mode. Command **pf.py exp band bpas -insp** To examine cross and auto spectrum. To create bandpass mask definition file. Transform mask definition file into the bandpass mask file with command ***bmge***. Run command **pf.py EEE B bpas -insp**. again and check that masking bad auto- and cross- correlation channels fixed the problem. Update the band definition file if needed and iterate.
8. Run bandpass generation in the non-interactive mode. Examine the log. Check observations that are marked as outliers. If needed, update the mask definition file, disable phase calibration, and repeat.
9. Run fine fringe fitting in fine mode. Command: **pf.py EEE B bpas fine**.

Pipeline for astrometry/geodesy data analysis

This pipeline runs *after* the fringe fitting pipeline.

1. Export the results of fringe fitting into GVF database.
2. Run astrometry/geodesy solution using VTD/Post-Solve.
3. Suppress outliers that include but not limited to non-detections using Post-Solve.

4. Store the GVF database using Post-Solve.
5. Store the full residual file
6. Run Post-Solve program samb that generates the control file that calls ***PIMA*** for re-fringing suppressed observations; run re-fringe with ***PIMA***; select the observations with SNR above the detection threshold and append fringe results for these observations to the end of fringe file; create a new GVF database version 1 using ***PIMA*** task ***mkdb***; propagate auto suppression status to the database. All these steps are executed by wrapper **pr.py**.
7. Run VTD/Post-Solve solution once again. Suppress outliers and restore good observations marked as outliers.

Imaging pipeline

It is recommended to run imaging pipeline after fringe fitting pipeline and astrometry/geodesy pipeline. The latter pipeline allows you to effectively filter out non-detections and corrupted observations, f.e. observations where the fringe fitting algorithm found the maximum that corresponds to correlation of phase calibration signal. Though it is possible to skip astrometry/geodesy pipeline, but in that case you need to screen observations for non-detections or observations' where fringe fitting failed. Setting a higher SNR limit in a range 6.0–6.5 seems prudent. NB: even one non-detection that slipped into the dataset may severely distort an image.

1. Import gain table.
2. Run ***PIMA*** task ***onof***.
3. Extract indices of suppressed observations in VTD/Post-Solve solution in an ascii to use this file as value of the keyword **EXCLUDE_OBS_FILE**.
4. Import antenna gain.
5. Select 2–5 strong sources that were observed at all baselines. Run task ***splt*** for these sources. Check ***splt*** logs. If there are observations with large residual group delays, add their indices to the file pointed by keyword **EXCLUDE_OBS_FILE**, and repeat task ***splt*** one more time.
6. Image selected sources.
7. Run task ***gaco*** for imaged sources. Examine gain correction file and edit it, if necessary.
8. Run task ***splt*** for all the sources. Check ***splt*** logs. If there are observations with large residual group delays, add their indices to the file pointed by keyword **EXCLUDE_OBS_FILE**, and repeat task ***splt*** one more time.
9. Image all the sources. You may want to copy to another directory results of imaging strong calibrator that you run before, since ***PIMA*** will overwrite them otherwise.
10. Create pictures of the brightness distribution of the imaged sources.

The three last three steps are performed by wrapper **pf.py EEE B map**.

Running the analysis pipeline with pir.py

Program **pir.py** is provided for facilitating running the VLBI analysis pipeline in the semi-automatic fashion. As of 2021, the fully automated mode is not yet implemented. However, **pir.py** substantially reduces the amount of manual work. It executes elements of the VLBI data analysis pipeline. In total, there are 16 elements. Elements can be executed separately, or in the group, or all together.

```
usage: pir.py [-h] [--version] [-v verbosity] [-b band]
              [-r run-level] [-s] experiment
```

where experiment is the experiment code following either NRAO, or KVN, or IVS, or KaVA, or EAVN notation.

parameter **--verbosity** controls verbosity of the output.

- 0 -- silent
- 1 -- normal verbosity (defaults)
- 2 -- debugging mode.

parameter **--band** specifies the 1-character long band name. If the experiment has two bands, the code for the upper band should be used.

parameter **--run-level** controls which elements or a group of elements of the VLBI data analysis pipeline should be executed. The run-level is either a positive number when an elementry run level is specified or a low case letter if a compound run level is selected. See the next subsection.

If parameter **-s** was specified, statically linked ***PIMA*** will be used.

If the experiment has data from two bands, both bands will be processed to enable the use of ionosphere-free combinations of group delay observables. The upper band should be specified when running **pir.py**.

Limitations:

1. The full end-to-end pipeline without manual intervention is not yet feasible. More work needs be done to implement it. As of version 1, there are several breaking points that assume manual work.
2. As of version 1, lin-pol 4 band observations are not yet supported.

pir.py run levels

Program **pir.py** splits the VLBI data analysis pipeline into a number of run levels. The run levels are supposed to be executed in the defined order because results from the previous run levels are used for the next run level. The run levels can be elementary or compound. Compound run levels combine several elementary run levels. The granulation of the pipeline into elementary and compound run levels provides flexibility. For some data analysis scenarios compound run levels can be used, for other scenarios additional programs need run between elementary run levels.

The following run levels are supported:

1. Load the experiment to ***PIMA***.
2. Generate bandpass and phase calibration masks from mask definitions files.
3. Parse logs and load information extracted from logs to ***PIMA***.
4. Run coarse fringe fitting.
5. Run bandpass generation.
6. Run fine fringe fitting.
7. Run creation of an output GVF database.
8. Run pSolve using version 2 of the GVF database and generate listing of the solution. **NB:** It is assumed interactive data preprocessing with pSolve has been done prior that step.
9. Run re-fringing of the observations marked as outliers during preprocessing.
10. Run creation of an output GVF database with re-fringed data.
11. Run task onof to determine segments of data when antenna was not on source.
12. Run computation of atmosphere brightness temperature and opacity using the output of numerical weather model. **NB:** You need to have the preprocessed output of the output of numerical weather

model at your local computer to run this task. Automatic downloading these data will be implemented in the future.

13. Run loading atmospheric opacity and brightness temperature followed by automatic editing of raw Tsys values.
14. Generate images of the sources selected as reference.
15. Run gain correction computation using images of reference sources.
16. Run automatic imaging.

The following compound run levels are supported:

- **l** — extended loading. It combines 1,2,3 and performs loading the experiment, generation of the bandpass and phase calibration masks from mask definitions files, parsing Fields System log files, and loading extracted calibration information into ***PIMA***.
- **c** — coarse fringe fitting. The same as 4.
- **b** — bandpass computation. The same as 5.
- **f** — fine fringe fitting. Includes fine fringe fitting and generation of the GVF database at the end. Combines 6 and 7.
- **r** — re-fringing. It runs pSolve solution, uses pSolve residuals to generate control files for re-processing the outliers of the pSolve solution with a narrow window, runs re-fringing, and generates the re-fringed database. Combines 8, 9, and 10.
- **p** — pre-imaging. Includes determination of data segments when antennas were not on source; computation of the atmospheric opacity and brightness temperature; automatic editing Tsys that includes outlier elimination and interpolation of bad or missing Tsys; loading modeled Tsys into ***PIMA***; getting gain and loading into ***PIMA***; and generation of files with time and frequency averaged data in FITS image format for reference sources. Combines 10, 11, 12, 13, and 14.
- **i** — imaging. Includes computation of gain correction, calibration of the data, time and frequency averaging, splitting the data into calibrated FITS-file, one file per source and band, automatic image generation with Difmap, and generation of rad plot and PostScript pictures from images. Combines 15 and 16.

Hints for pir.py use

The pipeline execution still requires manual steps. **pir.py** reduces the number of manual operations to the minimum and automatically runs other steps. The following sequence is recommended:

- (***manual***) Download the data and create ***PIMA*** control file. This step is done manually for processing a new style of an experiment. For processing an experiment from a campaign that contains many segments, this step can be easily automated. That includes downloading the data and generation of ***PIMA*** control files, bandpass and phase calibration masks from templates. The template defines most of the parameters using the control files used for processing a specific experiment. The rest of the parameters, such as experiment name and observing dates, can by automatically updated by a simple program.
- (***automatic***) After the visibility file in FITS-IDI format and VLBI Field System logs are downloaded and named according to the convention ***PIMA*** and Psolve understand, task **pir.py** with run level **l** is executed.
- (***manual***) Phase cal examination. It is recommended to run manually tasks ***plcl*** and ***mppl***. The goals is a) to update pcal mask file; b) to find stations for which phase calibration is bad and needs be disabled. The stations with disabled phase calibration are included in as a comma-separated qualifier of **PCAL:** keyword value. **NB:** If you want to disable phase calibration for a 2-band experiment, do not forget to update ***PIMA*** control files for each band. It is advised to run **pt.py** for several scans of strong sources to catch other problems than may require editing of control files.
- (***automatic***) Run **pir.py** with run level **c** to perform coarse fringe fitting. **NB:** if you have a dual-band experiment, the upper band should be specified. This will cause **pir.py** to process both bands in parallel.
- (***manual***) Phase cal examination. It is recommended to run manually tasks ***bpas*** via wrapper **pt.py** with qualifier **-insp**. You need decide whether you need adjust bandpass mask, for instance to mask out affected IFs either partially or entirely.
- (***automatic***) Run **pir.py** with run level **b**. This will generate bandpass file, and for dual-band experiment, the polarization bandpass as well.
- (***manual***) Examine bandpass logs. If bandpass logs show problems, you may need adjust control files and/or bandpass or phase calibration masks.
- (***automatic***) Run **pir.py** with run level **f**. This will perform fringe fitting and generate version 1 of the experiment geodetic database.
- (***manual***) Run pSolve in the mode of interactive preprocessing. Refer pSolve documentation. You need edit both bands of the dual-band experiment, At the end you need update the database.
- (***automatic***) Run **pir.py** with run level **r**. This will cause re-fringing of all observations. The geodetic database will be automatically updated, but the suppression status will not.
- (***manual***) Load version 2 of the experiment and restore good observations that appear after re-fringing using pSolve. Save updated database. Geodetic part of the processing is completed.

  If you will image observed sources, you need prepare files with 2–4 reference sources at this point. Remember, these files have suffices _{band}_ref.sou, where {band} is a band.
- (***automatic***) Run **pir.py** with run level **p**. This will execute task ***onof***, update Tsys, and generate files with time and frequency averaged calibrated visibility in FITS image format for reference sources.
- (***manual***) Image reference sources using Difmap.
- (***automatic***) Run **pir.py** with run level **i**. This will automatically generate images of all the sources and generates source picture files in GIF format.
- (***manual***) Manually examine the images. Re-image manually poor images.

Processing dual-band observations

Dual-band observations are processed separately. Though in some cases it is possible to run fringe fitting over a very wide bandwidth (several GHz), in that case this would be called wide-band fringe fitting. Depending on the correlator setup dual-band data can be be put in one frequency group, f.e. VLBA S/X observations or observations at remote wings of VLBA C-band receivers or be put into two different groups. If the frequency layout is not known, the following parameters should be set **FRQ_GRP**: **1**, **BEG_FRQ**: **1**, **END_FRQ**: **1** before loading the experiment in ***PIMA***. After that, a user should examine the frequency file `SSSSS/EEE.frq` created by ***PIMA*** task ***load*** and create *two* ***PIMA*** *control files* for the upper and lower bands. The upper band is considered primary band and the low band is considered secondary band. Keywords **BAND**, **FRQ_GRP**, **BEG_FRQ**, **END_FRQ** should define frequency names frequency indices within the frequency band. The control file of the primary (upper) frequency band should define the name of the ***PIMA*** control file for the secondary (lower) frequency band in the keyword **MKDB.2ND_BAND**. The value of this keyword should be **NO** in the control file for the lower band.

PIMA task ***load***, ***gean***, ***pmge***, ***bmge*** are band-independent; other tasks depends on the band an should be executed with the appropriate control file. All operations in ***PIMA*** pipeline, except tasks ***load***, ***gean***, ***pmge***, ***bmge***, and ***mkdb*** are performed two times: first for lower band and for upper band. They can run concurrently. Bandpass and phase calibration mask files are common for both bands. Task ***mkdb*** should be run for the upper band only. When ***PIMA*** finds value of **MKDB.2ND_BAND** that is the ***PIMA*** control file for the lower band, it computes the total observables of two control files and puts them in appropriate slots of GVF database. NB: ***PIMA*** does not check which band is upper and which band is lower frequency — an analyst should define it. If to run ***PIMA*** task ***mkdb*** with the control file for the lower band, ***PIMA*** will create the GVF with total observables only for that band. For historical reasons Post-Solve always marks the upper band as "X" and lower band as "S" regardless the frequency range.

Tasks coarse fringe fitting, bandpass generation, and fine fringe fitting is executed two times, for lower and upper band. Task ***mkdb*** is executed once for the upper band control file only. The GVF database created in the dual-band mode contains the data for both bands. Using VTD/Post-Solve two bands are processed consecutively, first the lower band marked as "S" (Data type "GS"), then the upper band marked as "X" (Data type "GX"). Two files with residuals are created: the upper band and for the lower band. Then wrapper **pr.py** is executed for both bands: first for the lower band and then for the upper band. Option **-nodb** should be used with the wrapper for the lower band. This option prevents creation of a database for the lower band, since such a database would not have the data for the upper band. After wrapper **pr.py** for the lower band is completed, wrapper **pr.py** for the upper band is executed. During next VTD/Post-Solve iteration lower band and upper band data are re-analyzed. After that a liner combination of the upper and lower band data (Data type: "G_GXS") are analyzed.

Imaging the upper and lower bands is done separately. Gain control files should be separate for upper and lower bands.

Although GVF format allows to support up to 8 bands, as of 2016.05.05, VTD/Post-Solve supports only two bands. An experiment with more than two frequency bands is processed similarly as a dual-band

experiment except **MKDB.2ND_BAND** keyword that should be NO. In such case the keyword **MKDB.OUTPUT_NAME** that defines the database suffix should be different for each band. Otherwise, ***PIMA*** task ***mkdb*** will overwrite a database for a different band.

Auxiliary tools

PIMA provides a number of tools for examining the data.

Antenna log processing tool

Program **log_to_antab** processes input log files generated by software Field System and writes results in PIMA Antab format.

```
Usage: log_to_antab mode log_file antab_file  [year]
```

There are three mandatory arguments:

- **mode** — a flavour of log file. Supported modes:
  - mode = 1 — for IVS log-files after 2008.
  - mode = 2 — for IVS log-files in approximately 1999–2002.
  - mode = 3 — for IVS log-files in approximately 1996–1996.
  - mode = 4 — for IVS log-files in approximately 1996–1999.
  - mode = 5 — DBBC log file with USB/LSB pairs of BBCs.
  - mode = 11 — for KVN log-files
- **log_file** — file with the log.
- **antab_file** — output file with results of log file processing.
- **antab_year** — year of observations. Old log files did not provide year in the time tag. This optional argument provides missing information.

Tools for examining data in FITS-IDI format

When you receive the data from the experiment that you intend to analyze, you first need to examine the data. **NB:** ***PIMA*** processes data only in FITS-IDI format. ***PIMA*** provides several utilities that are useful for an initial data check.

- **fitsh** — a tool that prints the contents of FITS headers. It scans all tables and prints the list of keywords, their format and values of keywords (but not contents of the tables).

  ```
  Usage: fitsh fits_file
  ```

- **fitsd** — a tool that examines a directory with files in FITS-IDI format and prints start and stop epochs of visibilities that each file contains.

  ```
  Usage: fitsd directory file
  ```

- **get_source_table_from_fits** — a tool that prints the names and coordinates of the sources that are in the specified FITS-IDI file.

  ```
  Usage: get_source_table_from_fits fits_file
  ```

Tools for manipulation with data in FITS image format

There is a number of tools for processing image data in FITS image format. **NB:** these tools will work with the data generated by ***PIMA***, AIPS, and DIFMAP. They may or may not work with data generated by other programs. There are two level specifications: FITS and contents definition. FITS format defines only the data structure at the lower level. This information is not sufficient to parse *arbitrary* FITS-file without knowledge of contents definition specifications.

There are four contents definitions formats that ***PIMA*** deals with:

1. FITS-IDI — format for correlator output. ***PIMA*** reads this data when executes task ***load***. Data in FITS-IDI format contain visibilities and a lot of information that describes the experiment.
2. FITS-UVA — (UV Averaged) format for calibrated visibilities averaged over time and frequency. ***PIMA*** task ***splt*** writes the data in this format. FITS-UVA data contains visibilities, frequency table and information about experiment (name, date, etc). It may or may not contain gain a correction table. ***PIMA*** adheres file naming convention `JJJJJJJJJ_B_uva.fits` for FITS averaged visibility data, where `JJJJJJJJJ` is a 10 character long J2000 source name and `B` is the upper case band name.
3. FITS-UVS — (UV Self-calibrated) format is the same as FITS-UVA, but it contains self-calibrated visibilities after imaging. Visibility amplitudes and phases are corrected by the imaging process. ***PIMA*** adheres file naming convention `JJJJJJJJJ_B_uvs.fits` for self-calibrated visibility data.
4. FITS-MAP — format is for storing images. It contains several tables: Clean component table, frequency table and the gridded image. It also contains some auxiliary information: source name, source position, experiment date, beam size, etc. **NB:** gridded image is derived from Clean components. It does not bring additional information and is included to facilitate visualization. General purpose FITS viewers will show image in FITS-MAP format.

The following tools are provided:

- **`uva_merge`** — utility for merging several FITS-UVA visbility data files into one, provided the data have the same frequencies. Merged data may have better uv-coverage and this operation usually improves image quality. Since sources are often variable, merging the data for sources with images that changed is not recommended. For majority of AGNs merging the data with epochs within several months is usually safe.

  Usage:

  ```
  uva_merge uva_output input1_uva [input2_uva ...]
  ```

  Up to 30 FITS files can be merged. The order of input files does not matter. Do not forget that the output file comes first!

- **`fits_tim_avr`** — utility for time averaging FITS-UVA calibrated visibility data. The output is written in the file in fits format. Time averaging in seconds is specified. **fits_tim_avr** starts averaging from first the visibility till it fills a chunks of tim_av_sec long. Then it coherently averages, updates weights, writes down starts again. Usage: fits_tim_avr input_uva tim_av_sec output_uva where the first argument is the input file, the second argument is the averaging interval in seconds, and the third argument is the name of the output file in fits format.

  Usage:

  ```
  Usage:  fits_tim_avr input_uva tim_av_sec output_uva
  ```

  where the first argument is the input file, the second argument is the averaging interval in seconds, and the third argument is the name of the output file in fits format.

- **`fits_to_map`** — utility for generation of a picture of the brightness distribution using the Clean component table. The picture can be displayed on the display or written in the output file.

  Usage:

  ```
  fits_to_map [-o output_file] [-box value] [-size code]
              [-color code] [-lev value] [-beam code] fits_map_file
  ```

  A mandatory argument is map file in FITS-MAP format. Options:

  - **-o** — the output file. **XW** means display. If omitted, **XW** is assumed. Supported extensions:

    - **.ps** — output file will be in Postscript format.
    - **.gif** — output file will be in gif format.

  Unrecognized extension is treated as Postscript.

- **-box** — the bounding box size, i.e. the image scale. The argument specifies the semi-width of the bounding box in mas. If omitted, fits_to_map will use the bounding box to show the full image. **NB:** setting the bounding box wider than the image size will increase only the amount of wide space, but will provide no new information.
- **-size** — the image size, one of
    - **1** — 45x45 mm
    - **2** — 80x80 mm
    - **3** — 160x160 mm
    - **4** — 270x270 mm

  Fonts and line width are scaled in order to fit the sizes above. By default, image size is **3**, i.e. 160x160 mm.
- **-color** — the color index of the contour map according to DiaGI in a range [-1, 32]. Color index **-1** indicates the color of the plot the plot will be selected in accordance with the observing frequency. Color index 0 means black. Program diagi_dec (part of petools) shows the color table for codes in the range [1, 32]. By default, color index is **-1**, i.e. is selected automatically based on observing frequency.
- **-lev** — the flux density level for the first (lowest) contour expressed in image root mean square noise. Default is **5.0**.
- **-beam** — the beam style. One of:
    - **0** — elliptical beam. The displayed image is stored in the input FITS-MAP file.
    - **1** — circular beam. The size of the beam is determined by the semi-minor axis of the stored elliptical beam. The displayed image is created by convolving Clean components with the circular beam.
    - **2** — circular beam. The size of the beam is determined by the semi-major axis of the stored elliptical beam. The displayed image is created by convolving Clean components with the circular beam.
    - **3** — elliptical circular beam. The displayed image is created by convolving Clean components with the elliptical beam.
    - **4** — delta-function beam. The displayed image is created by convolving Clean components with the circular beam equal to two pixels.

- **`fits_to_radplot`** — utility for generation of a picture of the dependence of scan-averaged calibrated visibility versus the baseline length (so-called "radplot").

  Usage:

```
fits_to_radplot [-o output_file] [-size code] [-color code]
                [-gap time] [-wei T|F] [-cutoff_err value] [-auto] fits_vis_file
```

  A mandatory argument is map file in FITS-UVA format. Options:

  - **-o** — the output file. **XW** means display. If omitted, **XW** is assumed. Supported extensions:
      - **.ps** — output file will be in Postscript format.
      - **.gif** — output file will be in gif format.
      - **.txt** — output file will be in plain ascii format. An ascii table will be generated instead of a figure.

    Unrecognized extension is treated as Postscript.
  - **-color** — the color index of the contour map according to DiaGI in a range [-1, 32]. Color index **-1** indicates the color of the plot the plot will be selected in accordance with the observing frequency. Color index 0 means black. Program diagi_dec (part of petools) shows the color table for codes in the range [1, 32]. By default, color index is **-1**, i.e. is selected automatically based on observing frequency.
  - **-size** — the image size, one of
      - **1** — 45x45 mm
      - **2** — 80x80 mm
      - **3** — 160x160 mm
      - **4** — 270x270 mm

    Fonts and line width are scaled in order to fit the sizes above. By default, image size is **3**, i.e. 160x160 mm.
  - **-color** — the color index of the contour map according to DiaGI in a range [-1, 32]. Color index **-1** indicates the color of the plot the plot will be selected in accordance with the observing frequency. Color index 0 means black. Program diagi_dec (part of petools) shows the color table for codes in the range [1, 32]. By default, color index is **-1**, i.e. is selected automatically based on observing frequency.
  - **-gap** — the maximum gap between observations to consider them to belonging to one scan. Units: sec.
  - **-wei** — to use (**T**) or not to use (**F**) weights when compute scan averaged visibilities.
  - **-cutoff_err** — the cutoff for discarding points with excessive scatter. The points with the normalized statistical uncertainties determined as `Err(Amp)/Amp` exceeding this parameter are flagged out and removed from computations. Here `Err(Amp)` is the statistical error of the coherently averaged amplitude determined on the basis of the scatter with respect to average and `Amp` is the averaged amplitude.
  - **-auto** — auto-detection of the direction of minimal scatter. If this options is not set up (default), plot of calibrated amplitude versus baseline length will be generated. If this option is set, then a plot of the averaged fringe amplitude versus the length of the baseline projection to the direction which makes the scatter of the amplitude with respect to a smoothed curve minimal.

- **`fits_to_uvplot`** — utility for generation a plot of baseline vector projection to the source tangential plain (so-called uvplot). Visibilities are averaged over a scan. Units are wavelengths.

  Usage:

```
fits_to_uvplot [-o output_file] [-size code] [-color code] uva_fits_file
```

  A mandatory argument is map file in FITS-UVA or FITS-UVS format. Options:

- **-o** — the output file. **XW** means display. If omitted, **XW** is assumed. Supported extensions:
  - **.ps** — output file will be in Postscript format.
  - **.gif** — output file will be in gif format.
  - **.txt** — output file will be in plain ascii format. An ascii table will be generated instead of a figure.

  Unrecognized extension is treated as Postscript.
- **-color** — the color index of the contour map according to DiaGI in a range [-1, 32]. Color index **-1** indicates the color of the plot the plot will be selected in accordance with the observing frequency. Color index 0 means black. Program diagi_dec (part of petools) shows the color table for codes in the range [1, 32]. By default, color index is **-1**, i.e. is selected automatically based on observing frequency.
- **-size** — the image size, one of
  - **1** — 45x45 mm
  - **2** — 80x80 mm
  - **3** — 160x160 mm
  - **4** — 270x270 mm

  Fonts and line width are scaled in order to fit the sizes above. By default, image size is **3**, i.e. 160x160 mm.

- **`fits_to_cfd`** — utility for generation a table of median correlated flux densities at three ranges of baseline projection lengths. The table has two rows: a header and the body of the table.

  Usage:

  ```
  fits_to_cfd [-help] [-o output_file] [-wei T|F] [-cutoff_err value] fits_vis_file fits_map_file
  ```

  Mandatory arguments are the file with self-calibrated visibilities in FITS-UVS format and image in FITS-MAP format. Options:

  - **-o** — the output file. If omitted, the table is printed in stdout, i.e. in the screen.
  - **-wei** — to use (**T**) or not to use (**F**) weights when compute scan averaged visibilities.
  - **-cutoff_err** — the cutoff for discarding points with excessive scatter. The points with the normalized statistical uncertainties determined as `Err(Amp)/Amp` exceeding this parameter are flagged out and removed from computations. Here `Err(Amp)` is the statistical error of the coherently averaged amplitude determined on the basis of the scatter with respect to average and `Amp` is the averaged amplitude.

This document was prepared by Leonid Petrov
Last update: 2025.01.07