# FLAGCAL: A flagging and calibration pipeline for GMRT data

*Jayaram N. Chengalur*

## 1 Introduction

FLAGCAL[1] is a flagging and calibration program meant for use with GMRT data. The input to FLAGCAL is a raw GMRT FITS file, and its output is a calibrated and flagged FITS file, which can be directly used for imaging and further processing. The input FITS file is assumed to be a random group (either multi-source or single-source) fits file. The output file has exactly the same structure as the input file (i.e. is multi-source if the input file is multi-source, and single-source otherwise).

The broad underlying idea is to utilize the fact that true visibilities will vary smoothly with time and frequency, while corruptions due to RFI and instrumental errors in general do not. This can be used to identify corrupted data. This approach works best with calibrated data, where the variations due to the different (and time variable) antenna gains has already been corrected for. However, the calibration algorithms cannot work with the raw data, since any corruption in the visibilities input to the antenna gain computing routines will result in incorrect gains. FLAGCAL hence takes an iterative approach to this problem. A first round of data flagging is done, after which the data is calibrated. The calibrated data can then re-examined, re-flagged using more stringent criteria and finally re-calibrated.

Operationally, FLAGCAL allows the user considerably flexibility in choice of algorithms to use for flagging, order in which these algorithms are run, input parameters to these algorithms, the number of flagging-calibration iterative loops to go through etc. These are detailed further below. The data flagging parameters are internally normalized using robust statistics computed from the data itself. In general the parameters do not require much fine tuning. The currently set default parameters have been spot tested on data sets at all of the GMRT frequency bands, and appear to be working satisfactorily. However, more testing is in progress and the program is expected to evolve. The code itself is also currently in pre-release testing - please do report any problems that you encounter to me. Users should also be aware that the currently set default parameter values need not be the best for their particular data set.

The program has been optimized for running on multi-core workstations, and the processing time is in general a few minutes for a 10 hour GMRT observation. Reading and writing the file could take a significant fraction of the total run time, so it helps a lot to be running on a machine with fast local disk.

## 2 Overview

### 2.1 Quick Start

This section is for those who really don't want to know anything about FLAGCAL's innards but want to try it out on some GMRT data that they have just taken. **NOTE:** FLAGCAL has not been tested at all with polar mode FITS files, its quite likely to do something funny with such files. Assuming that your observations are not in the polar mode, follow the steps below.

[1]This writeup documents FLAGCAL version 0.989 (Mar 2014). The first version of FLAGCAL was developed by Jayanti Prasad & Jayaram Chengalur, and is described in Prasad & Chengalur Exp. Astron., **33**, 157, 2012). While this version of FLAGCAL builds on that experience, it uses different algorithms, and also has a completely different architecture and code base. If you have used this version of FLAGCAL for processing data that you are publishing, you could cite this technical document (Chengalur, J. N. NCRA Technical Report,NCRA/COM/001 (April 2013)) and/or Prasad & Chengalur Exp. Astron., **33**, 157, 2012), which describes a broadly similar approach to the problem.

1. Use gvfits to convert your ltafile to fits.

2. Either set the FLAGCAL_DIR environment variable to point to the system wide default, or copy all of the required "rc" files to your current working directory (CWD). At the GMRT the FLAGCAL_DIR environment variable is likely to be already set for you.

3. If your program sources are not listed in the FLAGCAL_DIR "src_info.dat" file (which is likely to be the case) create the file src_info.dat in your CWD which has lines of the sort

   SourceName code

   where SourceName is the name of the source in your fits file, and CODE is F for flux calbirator, B for bandpass calibrator P for phase calibrator and T for target source. A example src_info.dat file would be

   ```
   3C286          FB
   0348+338       P
   NGC1265        T
   ```

   If a source is both the flux and the bandpass calibrator label it FB. If it is the flux, bandpass and phase calibrator, or if it is a bandpass calibrator but not a flux calibrator you will need to edit the default flagcal.rc file. Read on[2]

4. Run flagcal $fits\_in = YourUVdataFile.fits[flagfile = OnlineFlags.FLAG]$ where YourFile.fits is the raw fits file created by gvfits. The optional parameter you can specify with the $flagfile$ option is is the flag file (if any) created by gvfits using the flags generated by ONLINE.

5. Load the output FITS file (called TMP.FITS by default) into AIPS/CASA and proceed to imaging.

Summary log information on what FLAGCAL has done will be in the file fcs.log. A much more detailed log can be found in fc.log. Options for printing out the gain and bandpass solutions, details on what data has been flagged and why, etc. also exist. FLAGCAL by default does a fairly "light" flagging on the target source. You may wish to subtract out the bright continuum sources to create a UVSUB file, and then pass it through FLAGCAL again for a second, more stringent flagging round. Alternatively, you could try fitting a smooth function to the visibilities on the target source and then flagging points which are highly discrepant from this fit (see the function **flag_res()**).

## 2.2 Some more stuff regarding running flagcal

The user has a great deal of control over how exactly how to process data. All actions that one would like FLAGCAL to take (i.e. the actual steps in the pipeline) can be specified either on the command line or in command files, referred to as "rcfiles" in this document. By default, FLAGCAL will look for commands in a file called `flagcal.rc` in the the current working directory or if that is not found in the directory pointed to by the FLAGCAL_DIR environment variable. This can be overridden using the "-r" command line option to specify the name of a different file to use (e.g. flagcal -r test.rc. Once again this file is looked for in the CWD first and the FLAGCAL_DIR directory second). If no rcfile is found, FLAGCAL exits with an error message.

The "-r" option is accepted only on the command line, all other parameters/commands can be passed either from the command line, or from inside rcfiles. Inputs to FLAGCAL are processed in the order in which they are

[2]See the files flagcal2.rc, flagcal3.rc, flagcal4.rc for some commonly used processing pipelines.

encountered. The command line is processed first, and then the rcfiles. All parameters are global, later settings overwrite the current ones. The list of commands and parameters understood by FLAGCAL are listed in Sec. 3 and Tab. 2, 3, 4, 5.

An rcfile can include other files, a specification line "@include.rc" will interpolate the file "include.rc" at the point at which the line is encountered. The rcfile can also contain comments, which are marked by the hash ("#") character. A hash character causes the program to ignore the rest of the input line. Words which end with a parenthesis pair ("()") are interpreted as commands, while words separated by an equal to sign ("=") are interpreted as (keyword, keyvalue) pairs. Parameters for the commands are passed as such (keyword, value) pairs. Since all the parameters are global, once set they apply to all succeeding commands in the file, unless they are reset in between. The rcfile can also contain for loops, which causes all the commands inside the loop to be iterated over scans that match the loop test criteria. At present the loop test can be one of

- scan numbers (*scanno=s1,s2,..*, with -1 being interpreted as all scans).
- source qulaifier code (*calcode=STR*, where STR should consist of one or more letters from the set F,B,P,T. This stands for Flux calibrator, Bandpass calibrator, Phase calibrator and Target source respectively. A "*" will match all qualifier codes).
- source name (*srcname=STR*, which uses minimum match to match *STR* to the name of the source in the current scan. Once again a "*" will match any source name).

Loops can be nested. As the loop is being iterated over, the *scan* parameter is automatically updated to the scan number of the selected scan.

The other general parameters are listed below

- *fits_in* The name of the input FITS file. This can be a single-source or multi-source random group UV data file.
- *fits_out* The name of the output FITS file. This file has exactly the same format as the input fits file.
- *src_info_file* The name of the file in which additional information about the sources in the input FITS file can be found. This additional information tells the program whether the source is a calibrator or a target source, etc. The available classifications are phase calibrator (P), flux calibrator (F), bandpass calibrator (B) or target source (T)). The syntax of the *src_info_file* is

  src_name QUAL

  where QUAL can be one or more of the single letter codes F,B,P,T, discussed above. For example, if the source is both the flux and the bandpass calibrator, the code is FB. The spacing in the file is not critical. By default FLAGCAL looks for the file src_info.dat
- *dolog* Setting *dolog* to an integer greater than 0 causes log information to be written to the specified file. Larger integers produce more verbose information. The statistics of each baseline, antenna, the gain solutions, the flags applied etc. can all be written out to the log file. While this is useful for understanding what the program is doing and for debugging, it can also be quite voluminous. FLAGCAL also can produce a summary log file (fcs.log, see the command **print_summary()**) which has a much more compact log. In addition, the statistics of each scan, as well as the visibilities (and flags) for selected scans and baselines can be written out into FITS data cubes (see the commands **write_base(), write_stats()**).

- *logfile* The name of the file in which to write the log output.
- *dryrun* Setting *dryrun=1* causes the program to parse the rc file, but not actually execute any of the commands. This is useful to check the syntax of the rc file.
- *verbose* If *verbose=1* FLAGCAL reports to stderr the command that is currently being executed. The run time for that command as well as the total cumulative run time so far are also reported. Setting *verbose=0* suppresses this output.
- *num_threads* The number of threads to spawn when parallelizing computations. The appropriate number to use depends on the number of available cores, as well as the number of records/channels in the scan. Since the parallelization is over records and channels, one should not set *num_threads* to a value that is greater than the number of records or channels. FLAGCAL internally does this cross check and temporarily resets *num_threads* as required. A value of *num_threads=16* is commonly used.

## 2.3 A typical FLAGCAL processing run

In general, the sequence of steps required to process a file is as follows.

1. scan through the input fits file and make an index table.
2. read in the visibility data. This can be done scan by scan, allowing the user to read in data for just one scan, process it completely, write it out, and then proceed to the next scan. Scan by scan processing is useful in situations where the file size is large compared to the available RAM.
3. Process the scan data as desired
4. Write out the scan data. Generally a template output FITS file is first generated, after which the scan data is written into it. This allows out of order processing and writing of scan data.

We now look at each of these steps in some more detail. For indexing the data, the program first reads in the file header, source and antenna tables etc. (commands **read_hdu(), read_an()**, etc) and then reads the random parameters associated with each visibility. The information in the random parameters and the associated tables is used to create an index table. The index table consists of a series of scans, with each scan having visibilities corresponding to a single source. These scans in general correspond to the scans made while recording the data. The only exception is when the observer records two scans on the same source back to back. This scan break is lost at the time that the FITS file is made and cannot be recovered. However, the user can also specify a maximum time interval (*scan_maxbreak*) between visibilities in a given scan. This is useful for single-source files, where otherwise all of the data would by default be put into a single long scan. Setting *scan_maxbreak* to a value of about 5m generally causes the visibilities to be split into the same scans as were present during recording. Since the statistics of baselines,antennas, channels etc. are examined for a scan at a time, one should break up single source data into scans. The fits file header, associated tables and as well as the generated index can be written out into the log file by using the *print_hdr()*, *print_an()*, etc. commands.

Note that while making the index table the visibility data itself is not read, only the random parameters associated with each visibility record are. The next stage in the processing is to read in the visibility data (command **read_scan()**). This data once read in can be browed (command **browse_data()**), the user can browse through the data as desired. Data once read in can be processed, and written out. A listing of the available processing steps are given in Sec. 3. To write out the data, one first makes a template output FITS file (command **make_template()**). This template is based on the input FITS file, and hence has exactly the

same structure as the input file. Once the template is created scan data can be written into it (command **write_data()**). Scan data can be written out of order, since for each scan, the location in the file to which the data is to be written is known in advance. Once one has finished with the data for a given scan one can free the memory used to hold the scan data (command **free_scan()**). This is useful when processing files which are large compared to the available computer memory. Note that it is the users responsibility to make sure that the data for all scans is written out into the output file. If not the output file will contain blanks.

The typical processing for a file using FLAGCAL would be as follows. First one process the data for the flux and bandpass calibrators. Any flags that were provided by the ONLINE system can be applied (using the flagfile generated during the observations, and the command **usr_flag()**). The next step is to compute statistics of the calibrator data. The main statistics that FLAGCAL works with are the median, and the median absolute deviation (mad). The mad of a data set is defined as the median of the absolute deviations from the median[3]. These are "robust" statistics, in that they are less affected by outliers than the mean or the standard deviation. Of course if a significant fraction of the data is corrupted, then these statiscs are not very usable. In this case the user will need to provide some prior information (via the command **usr_flag()**) as to what data is good. The statistics are computed and stored separately for the real, imaginary, amplitude and phase for each record, channel, baseline, and antenna in the scan. The phase statistics properly accounts for the fact that the phase is a circular variable (i.e. wraps around after reaching $2\pi$). The over all statistics of all of the data in the scan are also computed. Most of the user settable flag parameters are in terms of these statistics. For example, the user can chose to flag visibilities that deviate from the median visibility of the corresponding baseline by more than $n$ times the mad for that baseline.

It is assumed that calibrators are bright point sources at the phase centre[4] The calibrator visibilities are hence expected to be constant with time, apart from a slow variation due to drift of the amplifier gain/ionospheric phase. One would also expect that for all "working" antennas, the visibility amplitudes would all be within a factor of a few (typically $\sim 2$). Based on these assumptions it is relatively straight forward to flag the pre-calibrated visibilities for calibrator scans. Once the data has been flagged, a "continuum" channel is constructed. The user has some control over the exact channels to be used for constructing the continuum channel, but FLAGCAL also tries to help in this choice. Generally the user specifies a large range of channels from which to make the choice, as well as the number of channels to be averaged to make the continuum channel. FLAGCAL then chooses the first set of contiguous channels within this set for which none of the channels are flagged.

After this the flux for the calibrator source is set to the value appropriate to the frequency of the chosen continuum channel. This is done using the flux scale given in Barrs et al. After this the continuum channel data are used to solve for the antenna gains (command **solve_chan0()**). The solution is a standard steepest gradient minimization, and generally works well since the input data are already flagged. Since the data is already flagged, it is also possible to have FLAGCAL decide which antenna is best suited to be used as a reference[5] This gain is then applied to all the visibilities in the scan (command **calibrate()**). Next the bandpass for each antenna is computed (command **solve_bpass()**). A single bandpass is calculated using the average of the data over the entire scan. This bandpass is then applied to the data. With this the processing of the flux and bandpass calibrators is finished. The data can be written to the output file, and the corresponding memory freed.

The next step is to read in the data for the phase calibrators. These are flagged as before, and the continuum channel (using the same channels as were used for the flux calibrator) constructed. The antenna gains for

[3]That is, if m is the median of a data set x, i.e. $m =< x >_{med}$, then the mad is defined as mad $=< x - m >_{med}$.

[4]In general, it would have been better to use model visibilities for the field instead. But the point source assumption works well for most of the GMRT archival data, and provides a sufficiently good model for calibration and flagging. It is assumed that the user will later improve on this by using self-calibration on the target field.

[5]FLAGCAL will pick the smallest numbered antenna (antenna numbering starts with C00) which is not flagged in either polarization in any calibrator scan. Unless all central square antennas are flagged, this will result in a central square antenna being picked as reference.

the phase calibrator are then computed. The flux of the phase calibrator is then obtained from by comparing the gains on the flux calibrator and the phase calibrator, and the known flux of the flux calibrator (command **getjy()**). This is used to set the scale for the gain tables. For observations in the 21cm band, it is also possible to apply an elevation dependent gain curve while doing the calibration. The bandpass is then transferred from the flux calibrator to the phase calibrator and the phase calibrator data is calibrated for using both the gain and bandpass tables. If there are multiple bandpass calibrator scans, the simple average of the bandpass derived from each scan is used to calibrate the phase calibrator and target visibilities[6]. With this the processing of the phase calibrator data is finished, and the data can be written out to the output file and freed.

The last set of scans to be processed are those for the target source. The data are read in, the gain and band pass tables transferred from the flux and phase calibrator to the target source, and then the target source is data is calibrated. This calibrated data is then flagged, and the output is written to out.

For fields with significant flux and structure, the visibilities will vary with systematically with time. One could try and fit the systematic variation either by polynomial fitting, or low pass filtering, before trying to identify corrupted data. FLAGCAL currently implements robust fitting to the visibilities and flagging based on the residuals from this fit (see **flag_res()** for details). For GMRT archival data, it is reasonably fast to make an image using data that has been lightly flagged by FLAGCAL. The source emission can then be modelled out, and the residuals passed through FLAGCAL for a second, more rigorous round of flagging.

The processing sequence detailed above is summarized in the pseudo rcfile below.

```
for FLUX, PHASE and BANDPASS calibrators
   read data
   compute statistics
   flag data
   compute channel0
endfor

select reference antenna

for FLUX and BANDPASS CALIBRATORS
   set flux scale
   solve for gains and bandpass
   calibrate
   write data
endfor

for PHASE calibrators
   solve for gains
endfor

for PHASE calibrators
   boot strap phase calibrator flux
   transfer bandpass
   calibrate
```

[6]The GMRT bandpass for some antennas sometimes varies with time (typically at the few percent level). This straightforward calibration done by FLAGCAL allows one to easily do a second, finer level correction for the varying part of the bandpass using the scans on the phase calibrator. There are also bandpass calibration modes (see the explanation for **calibrate()** below) where one constant bandpass curve is used to calibrate all the data in the file

```
   write data
endfor

for TARGET sources
   read data
   transfer gain and bandpass
   calibrate
   compute statistics
   flag
   write data
   free data
endfor
```

# 3 List of available commands

In this section has an alphabetical listing of available commands as well as details of what each command does is given. A summary table giving the available commands and their parameters can be found at the end of this section.

1. **bpass_transfer()** *(scan, calsrc)* This command transfers the bandpass table from the calibration source *calsrc* to the current scan. *calsrc* is the name of the calibration source. If there are multiple scans for the source *calsrc* then the bandpass tables for all scans are averaged together and this average bandpass is transferred to the target source. In case *calsrc* is not specified, then the bandpass tables from all the bandpass calibrators in the file are averaged together and transferred to the target scan. Note that this command only transferres the bandpass table, the table itself is applied to the data by the **calibrate()** command.

2. **browse_data()***(scan)* Allows the user to browse through the data for the current scan. The program enters an interactive loop where the user is allowed to specify the exact data (baseline/record/channel, etc) that is to be shown. See also the command **write_base()**.

3. **browse_stats()***(scan)* Allows the user to browse through the data for the current scan. The program enters an interactive loop where the user is allowed to specify the exact data (baseline/record/channel, etc) that is to be shown. See also the command **write_stats()**.

4. **calibrate()***(scan, apply_gain, apply_bpass, bp_mode, amp_phase_int)* Command to apply the gain and band pass tables associated with the selected *scan* to the visibilities in that scan. The gain table is applied if *apply_gain=1*, and the bandpass table if *apply_bass=1*. Both tables can be applied simultaneously. When applying the gain table a linear interpolation in time is done. The linear interpolation is done separately for the (re,im) components of the gain if *amp_phase_int=0*. Otherwise, the interpolation is done separately for the (amp,phs) components of the gain. In this case, it is assumed that the phase change between available solutions is less than $180^o$. The *bf_mode* sets the bandpass interpolation model. Mode 0 applies a simple average bandpass computed from all bandpass calibrator scans to all scans other than bandpass calibrator scans. The bandpass calibrator scans are calibrated using the bandpass computed from the each scan individually. In Mode 1 all scans, including bandpass calibrator scans, are calibrated using the simple average bandpass computed from all bandpass calibrator scans.

5. **compute_chan0()***(scan, chan0_start, chan0_end, chan0_nchan)* Command to construct a continuum channel ("channel0"). This is constructed by averaging together *chan0_nchan* channels and is used for computing the antenna gains. The channels have to lie within the range *chan0_start* and *chan0_end*, which has to be larger than *chan0_nchan*. The program picks the first set of contiguous channels in this range of which none of the channels are flagged. **Note:** If all of these parameters are set to -1, then the program will use some GMRT specific heuristics to set the above parameters.

6. **compute_tsys()***(scan, calsrc)* Command to compute the "system temperature". This is computed only for phase calibrators. The command returns an error if it is run for any other type of source. The "system temperature" is *defined* as the MAD of the antenna amplitude computed after calibration has been done. Strictly speaking this is a measure of Tsys/G, and the computed value can be treated as the system temperature only to the extent that all the antennas have equal gains. This command fills the Tsys table for all phase calibrator scans, and also normalizes the values to have a median of 1.0. Since only the relative values are needed for the weighting, this is fine. The normalization is over all scans, antennas and channels. The Tsys table has an entry per channel, and this is what is used to recompute the antenna weights. See also **tsys_transfer()** and **reweight()**.

7. **fake_scan_data()***(scan,sim_iflux,sim_snr,sim_seed, sim_variable_gain)* Commad to replace the scan data with simulated data. The simulated data is for a point source with flux *sim_ifux* located at the phase center. Noise is added so that the signal to noise ratio per visibility is *sim_snr*. *sim_seed* is the seed to the pseudo-random sequence generator. If *sim_variable_gain=1*, then the antenna gains are allowed to vary with time. Some of the baselines, records and channels and visibilities are filled with "bad" data. This command is largely useful to check the effect of various flagging algorithms.

8. **flag_aggregate()***(scan, ant_min_ok_frac, base_min_ok_frac, chan_min_ok_frac, rec_min_ok_frac)* Command to flag out the data aggregates. The basic idea is that often data corruption affects the entire aggregate (e.g. an antenna, or a channel, or a baseline etc.). The other flagging routines that FLAGCAL uses may result in flagging out most of the data of a channel, or baseline, but there may still be a small fraction that passes the flagging criteria. This command will check how much good data is left in the aggregate (as a fraction of the total data associated with that aggregate) and will flag the entire aggregate if the good data fraction is smaller than specified. So for example if ant_min_ok_frac is set to 0.5, then if more than half of the antenna data has flagged (by some other flagging criteria) then the entire antenna is flagged.

9. **flag_ant()***(scan, ant_min_amp, ant_max_amp, ant_outlier, ant_re_outlier, ant_im_outlier, ant_max_nrms, ant_max_ph_rms, ant_min_ph_rms, ant_max_re_rms, ant_max_im_rms)* Command to flag antennas. If an antenna is flagged, then the data in the *scan* for all visibilities involving this antenna (loosely called the visibilities for the antenna below) are treated as flagged. This routine flags antennas for which *any* of the following criteria are met.

   - If the median amplitude of the visibilities of the antenna is less than *ant_min_amp* times the median amplitude of all the visibilities in the *scan*.
   - If the median amplitude of the visibilities of the antenna is more than *ant_max_amp* times the median amplitude of all the visibilities in the *scan*.
   - If the median amplitude of the visibilities of the antenna differs from the median amplitude of all the visibilities in the scan by more than *ant_outlier* times the mad of the amplitude of all the visibilities in the scan.

- If the median of the real part of the visibilities of the antenna differs from the median of the real part of all the visibilities in the scan by more than *ant_re_outlier* times the mad of the real part of all the visibilities in the scan.
- If the median of the imaginary part of the visibilities of the antenna differs from the median of the imaginary part of all the visibilities in the scan by more than *ant_im_outlier* times the mad of the imaginary part of all the visibilities in the scan.
- If the "normalized rms", i.e. the ratio of mad of the visibility amplitude to the median of the visibility amplitude for the antenna is greater than *ant_nrms* times the value of the same ratio for all the visibilities in the scan.
- If the mad of the phase of the visibilities of the antenna is greater than *ant_max_ph_rms* times the mad of the phase of all the visibilities for the scan.
- If the mad of the phase of the visibilities of the antenna is less than *ant_min_ph_rms* times the mad of the phase of all the visibilities for the scan. This is useful occasionally, when the visibility data is stuck at a constant value because of a correlator fault.
- If the mad of the real part of visibilities of the antenna is greater than *ant_re_rms* times the mad of the real part of all the visibilities in the scan.
- If the mad of the imaginary part of visibilities of the antenna is greater than *ant_im_rms* times the mad of the imaginary part of all the visibilities in the scan.

Note that if any of these thresholds is zero (strictly speaking less than $10^{-8}$) the corresponding test is not done. All these thresholds are also intialized to zero at the start of the program, and can be reinitialized to zero using the command **init_thresh()**.

10. **flag_base()***(scan, base_min_amp, base_max_amp, base_outlier, base_re_outlier, base_im_outlier, base_max_nrms, base_max_ph_rms, base_min_ph_rms, base_max_re_rms, base_max_im_rms)* Command to flag out baselines. If a baseline is flagged, then the data in the *scan* for all visibilities involving this baseline are flagged. This routine flags baseline for which *any* of the following criteria are met.

    - If the median amplitude of the visibilities of the baseline is less than *base_min_amp* times the median amplitude of all the visibilities in the *scan*.
    - If the median amplitude of the visibilities of the baseline is more than *base_max_amp* times the median amplitude of all the visibilities in the *scan*.
    - If the median amplitude of the visibilities of the baseline differs from the median amplitude of all the visibilities in the scan by more than *base_outlier* times the mad of the amplitude of all the visibilities in the scan.
    - If the median of the real part of the visibilities of the baseline differs from the median of the real part of all the visibilities in the scan by more than *base_re_outlier* times the mad of the real part of all the visibilities in the scan.
    - If the median of the imaginary part of the visibilities of the baseline differs from the median of the imaginary part of all the visibilities in the scan by more than *base_im_outlier* times the mad of the imaginary part of all the visibilities in the scan.
    - If the "normalized rms", i.e. the ratio of mad of the visibility amplitude to the median of the visibility amplitude for the baseline is greater than *base_nrms* times the value of the same ratio for all the visibilities in the scan.

- If the mad of the phase of the visibilities of the baseline is greater than *base_max_ph_rms* times the mad of the phase of all the visibilities for the scan.
- If the mad of the phase of the visibilities of the baseline is less than *base_min_ph_rms* times the mad of the phase of all the visibilities for the scan. This is useful occasionally, when the visibility data is stuck at a constant value because of a correlator fault.
- If the mad of the real part of visibilities of the baseline is greater than *base_re_rms* times the mad of the real part of all the visibilities in the scan.
- If the mad of the imaginary part of visibilities of the baseline is greater than *base_im_rms* times the mad of the imaginary part of all the visibilities in the scan.

Note that if any of these thresholds is zero (strictly speaking less than $10^{-8}$) the corresponding test is not done. All these thresholds are also intialized to zero in the program, and can be reinitialized using the command **init_thresh()**.

11. **flag_block()***(scan, smooth_cwidth, smooth_rwidth, smooth_redo_stats, smooth_fudge_fac and all parameters relevant to* **flag_vis()***)*. Command to flag out blocks of data at a time. It is useful to see the description of **flag_vis()** first. **flag_block()** flags out visibilities that are marked out as being discrepant in some way. It is meant to identify and flag persistent low level problem in part of a data set. Since the problem is low level the individual visibilities are close to the expected value and are hence not flagged as discrepant. However because this problem is persistent over time or frequency, it can be identified when one smooths over time/frequency. This is what **flag_block()** does. It smooths over time (i.e. over *smooth_rwidth* records) and/or over frequency (i.e. over *smooth_cwidth* channels) to construct a new visibility set. This visibility set is then flagged using a call to **flag_vis()**, and the flags are then transferred back to the original unsmoothed data set. There are two options for flagging the smoothed data. The first is to recompute the statistics of the smoothed data itself, and then use these for further flagging. This is what is done if *smooth_redo_stats=1*. The other option is to use the statistics of the unsmoothed data *smooth_redo_stats=0*, but scale all the variances by $\sqrt{smooth\_cwidth \times smooth\_rwidth}$. If the user does not want to assume that the noise will decrease exactly as $\sqrt{N}$, then a fudge factor which applies a scaling to the variance over and above the $\sqrt{N}$ scaling can be applied using *smooth_fudge_fac*[7]. See also the command **flag_multi_block()**

12. **flag_chan()***(scan, chan_min_amp, chan_max_amp, chan_outlier, chan_re_outlier, chan_im_outlier, chan_max_nrms, chan_max_re_rms, chan_max_im_rms)*. Command to flag out data for all the channel in a given scan. If the channel is flagged, then the data in the *scan* for all visibilities involving this channel are flagged. The channel is flagged if *any* of the following criteria are met.

    - If the median amplitude of the visibilities of the channel is less than *chan_min_amp* times the median amplitude of all the visibilities in the *scan*.
    - If the median amplitude of the visibilities of the channel is more than *chan_max_amp* times the median amplitude of all the visibilities in the *scan*.
    - If the median amplitude of the visibilities of the channel differs from the median amplitude of all the visibilities in the scan by more than *chan_outlier* times the mad of the amplitude of all the visibilities in the scan.

[7] i.e. the variances are scaled by $smooth\_fudge\_fac/\sqrt{smooth\_cwidth \times smooth\_rwidth}$

- If the median of the real part of the visibilities of the channel differs from the median of the real part of all the visibilities in the scan by more than *chan_re_outlier* times the mad of the real part of all the visibilities in the scan.
- If the median of the imaginary part of the visibilities of the channel differs from the median of the imaginary part of all the visibilities in the scan by more than *chan_im_outlier* times the mad of the imaginary part of all the visibilities in the scan.
- If the "normalized rms", i.e. the ratio of mad of the visibility amplitude to the median of the visibility amplitude for the channel is greater than *chan_nrms* times the value of the same ratio for all the visibilities in the scan.
- If the mad of the phase of the visibilities of the channel is greater than *chan_max_ph_rms* times the mad of the phase of all the visibilities for the scan.
- If the mad of the phase of the visibilities of the channel is less than *chan_min_ph_rms* times the mad of the phase of all the visibilities for the scan. This is useful occasionally, when the visibility data is stuck at a constant value because of a correlator fault.
- If the mad of the real part of visibilities of the channel is greater than *chan_re_rms* times the mad of the real part of all the visibilities in the scan.
- If the mad of the imaginary part of visibilities of the channel is greater than *chan_im_rms* times the mad of the imaginary part of all the visibilities in the scan.

Note that if any of these thresholds is zero (strictly speaking less than $10^{-8}$) the corresponding test is not done. All these thresholds are also intialized to zero in the program, and can be reinitialized using the command **init_thresh()**.

13. **flag_multi_block()***(scan, smooth_width0, smooth_maxwidth, smooth_redo_stats, smooth_fudge_fac, and all parameters relevant for* **flag_vis()***)*. Command to iteratively smooth and flag the data. This command is built on top of **flag_block()**. The user specifies the starting width for the smoothing block (*smooth_width0*), and the maximum width of the smoothing block (*smooth_maxwidth*). The smoothing block is taken to be square in records and channels. The program will loop through calls of **flag_block()**, increasing the smoothing block size by a factor of 2 (in each dimension) between successive calls. Note that the program will do some resizing of the smoothing block, in case the user specified choices are not appropriate for the number of channels or records in the given scan.

14. **flag_rec()***(scan, rec_min_amp, rec_max_amp, rec_outlier, rec_re_outlier, rec_im_outlier, rec_max_nrms, rec_max_re_rms, rec_max_im_rms)*. Command to flag out records. If a record is flagged, then the data in the *scan* for all visibilities involving this record are flagged. The record is flagged if *any* of the following criteria are met

    - If the median amplitude of the visibilities of the record is less than *rec_min_amp* times the median amplitude of all the visibilities in the *scan*.
    - If the median amplitude of the visibilities of the record is more than *rec_max_amp* times the median amplitude of all the visibilities in the *scan*.
    - If the median amplitude of the visibilities of the record differs from the median amplitude of all the visibilities in the scan by more than *rec_outlier* times the mad of the amplitude of all the visibilities in the scan.

- If the median of the real part of the visibilities of the record differs from the median of the real part of all the visibilities in the scan by more than *rec_re_outlier* times the mad of the real part of all the visibilities in the scan.
- If the median of the imaginary part of the visibilities of the record differs from the median of the imaginary part of all the visibilities in the scan by more than *rec_im_outlier* times the mad of the imaginary part of all the visibilities in the scan.
- If the "normalized rms", i.e. the ratio of mad of the visibility amplitude to the median of the visibility amplitude for the record is greater than *rec_nrms* times the value of the same ratio for all the visibilities in the scan.
- If the mad of the phase of the visibilities of the record is greater than *rec_max_ph_rms* times the mad of the phase of all the visibilities for the scan.
- If the mad of the phase of the visibilities of the record is less than *rec_min_ph_rms* times the mad of the phase of all the visibilities for the scan. This is useful occasionally, when the visibility data is stuck at a constant value because of a correlator fault.
- If the mad of the real part of visibilities of the record is greater than *rec_re_rms* times the mad of the real part of all the visibilities in the scan.
- If the mad of the imaginary part of visibilities of the record is greater than *rec_im_rms* times the mad of the imaginary part of all the visibilities in the scan.

Note that if any of these thresholds is zero (strictly speaking less than $10^{-8}$) the corresponding test is not done. All these thresholds are also intialized to zero in the program, and can be reinitialized to zero by the command **init_thresh()**.

15. **flag_res()***(scan, smooth_cwidth,smooth_rwidth,...)* The visibilities for strong extended source could have large (but smooth) variations with time and frequency. This makes it difficult to identify outliers using robust statistics. One could identify outliers by first fitting out the smoothly varying part of visibility, and then looking for outliers in the residuals. **flag_res()** fits for the variation in the visibilities by first robustly smoothing and decimating the data, and then doing a bi-linear fit to the decimated data. Residuals from this bi-linear fit are then used to do the flagging. For each baseline, decimation is done by replacing every smooth_cwidth*smooth_rwidth block of visibilities with the median value inside this block. The residuals from a bi-linear fit to the decimated data are then computed (it is planned in future to make smoother fits to the decimated data) and data with discrepantly large residuals are flagged. Specifically the functions **flag_rec(),flag_chan() and flag_vis()** are called. So all parameters that are relevant to those functions are also relevant here.

16. **flag_transfer()***(scan, calsrc)* Command to transfer antenna, baseline, and channel flags from the calibrator source *calsrc* to the target scan. If *calsrc* is not specified, then the nearest phase calibrator source on either side of the specified *scan* is used. If the antenna, baseline or channel is flagged in either of the nearby scans, then the corresponding flag is applied to the target scan.

17. **flag_vis()***(scan, vis_chan_outlier, vis_chan_re_outlier, vis_chan_im_outlier, vis_rec_outlier, vis_rec_re_outlier, vis_rec_im_outlier)*. Command to flag individual visibilities in a scan. A visibility is flagged if *any* of the following conditions is met.

    - If the amplitude of the visibility differs from the median amplitude of all the visibilities in same channel by more than *vis_chan_outlier* times the mad of the amplitude of all the visibilities in the same channel.

- If the real part of the visibility differs from the median of the real part of all the visibilities in same channel by more than *vis_chan_re_outlier* times the mad of the real part of all the visibilities in the same channel.
- If the imaginary part of the visibility differs from the median of the imaginary part of all the visibilities in same channel by more than *vis_chan_re_outlier* times the mad of the imaginary part of all the visibilities in the same channel.
- If the amplitude of the visibility differs from the median amplitude of all the visibilities in same record by more than *rec_chan_outlier* times the mad of the amplitude of all the visibilities in the same record.
- If the real part of the visibility differs from the median of the real part of all the visibilities in same record by more than *vis_rec_re_outlier* times the mad of the real part of all the visibilities in the same record.
- If the imaginary part of the visibility differs from the median of the imaginary part of all the visibilities in same record by more than *vis_rec_re_outlier* times the mad of the imaginary part of all the visibilities in the same record.

Note that if any of these thresholds is zero (strictly speaking less than $10^{-8}$) the corresponding test is not done. All these thresholds are also intialized to zero in the program, and can be reinitialized to zero by the command **init_thresh()**.

18. **flush_log()** Command to flush the output to the logfile. Inserting **flush_log()** commands into the rcfile alows one to examine the log while while FLAGCAL is running. In the absence of the **flush_log()** command, the log output is buffered, and may be written out only when the program has finished running.

19. **free_gain_table()***(scan)* Command to reinitialize the gain table associated with a scan. This allows one to make a second calibration pass through the data, and not worry about having to apply incremental calibration to all other scans for consistency.

20. **free_bpass_table()***(scan)* Command to reinitialize the bandpass table associated with a scan. This allows one to make a second calibration pass through the data, and not worry about having to apply incremental calibration to all other scans for consistency.

21. **free_scan()***(scan)* Command to free the memory allocated for the scan data. The bandpass and gain tables are not freed, so that they can still be transferred from one scan to another. The flag tables are freed though!

22. **gain_transfer()***(scan, calsrc)* Transfers the gain table from the specified *calsrc* to the target *scan*. If no *calsrc* is specified the nearest phase calibrator on either side of the target source is selected. For phase calibrators, the gain tables have multiple entries, depending on the solution interval chosen at the time of computing the solution. Gain transfer makes a new gain table with only two entries. In the first one it copies over the last entry of the gain table for the phase calibrator observed just before the target scan. In the second it copies over the first entry in the gain table for the phase calibrator observed just after the target scan. This allows linear interpolation at the time of calibrating the data (see **calibrate()**. For antennas that are flagged in one of the two phase calibrator scans, or in case there is only one available phase calibrator scan, the available gain value is copied over to both entries in the target scan gain table.

23. **get_src_info()***(src_info_file)* Command to read in the *src_info_file* in which additional information about the sources in the input FITS file is to be found. By default FLAGCAL looks for the file *"src_info.dat"*.

This additional information tells the program whether the source is a calibrator or a target source, etc. The available classifications are phase calibrator (P), flux calibrator (F), bandpass calibrator (B) or target source (T)). The syntax of the *src_info_file* is

src_name QUAL

where QUAL can be one or more of the single letter codes listed above. The spacing in the file is not critical.

24. **getjy()***(scan, apply_el_corr)*. Command to bootstrap the flux of the phase calibrator. This is done by comparing the gain values for the flux and phase calibrator scans, and using the known flux of the flux calibrator (see the function **setjy()**. This bootstrapping is done separately for every possible combination of phase calibrator scans and flux calibrator scans. This final value that is adopted is the average of all of these different determinations. The value obtained from each combination, as well as the adopted mean value and the standard deviation over all estimates is reported in the logfile. If *apply_el_corr* is set to 1, then an elevation dependent gain correction is applied. At the moment this is non trivial (i.e. not unity at all elevations) only for the 21cm band. Be warned that this is based on a technical report on the elevation dependent gain correction at the GMRT which is currently in draft stage.

25. **init_thresh()** Command to reset all the flagging thresholds to 0.0. A flagging threshold of 0.0 means that no data is flagged. Since the threshold settings are global, this is a useful command to use because it ensures that one is starting with a clean slate.

26. **make_index()***scan_maxbreak*Command to make an index table of the visibilities in the FITS file. For indexing the data, FLAGCAL first reads in the file header, source and antenna tables etc. and then reads the random parameters associated with each visibility. The information in the random parameters and the associated tables is used to create an index table. The index table consists of a series of scans, with each scan having visibilities corresponding to a single source. These scans in general correspond to the scans made while recording the data. The only exception is when the observer records two scans on the same source back to back. This scan break is lost at the time that the FITS file is made and cannot be recovered. However, the user can also specify a maximum time interval in minutes (*scan_maxbreak*) between visibilities in a given scan. A break larger than this value results in the start of a new scan. This is useful for single-source files, where otherwise all of the data would by default be put into a single long scan. Setting *scan_maxbreak* to a value of about 5 minutes generally causes the visibilities to be split into the same scans as were present during recording. Since baselines, antennas, channels etc. are processed for a scan at a time, one should break up single source data into scans.

27. **make_template()**Command to make a template output FITS file. The output file has the same structure and header information as the input FITS file, however the data are blank. The scan data can then be written out into this file in arbitrary order. It is the users responsibility to make sure that all of the scan data is written, otherwise the output file will contain blanks. Files with blanks cannot be processed in AIPS.

28. **print_an()** Command to print out the antenna table to the logfile.

29. **print_an()** Command to print out the scan index table to the logfile.

| Name | Val | Name | Val | Name | Val |
|---|---|---|---|---|---|
| FLG_MIN_AMP | 1 | FLG_MAX_AMP | 2 | FLG_OUTLIER | 4 |
| FLG_NRM_RMS | 8 | FLG_MIN_OK | 16 | FLG_NO_DATA | 32 |
| FLG_RE_OUTLIER | 64 | FLG_IM_OUTLIER | 128 | FLG_RE_RMS | 256 |
| FLG_IM_RMS | 512 | FLG_PH_RMS | 1024 | FLG_INTERPOLATE | 2048 |
| FLG_BAD_ANT | 4096 | FLG_USR | 8192 | | |

Table 1: Flag Values used by FLAGCAL

30. **print_bpass()** Prints out the bandpass table associated with all scans. The bandpass tables are printed out to the file bpasstable.dat.

31. **print_finfo()** Print items from the header of the input fits file. Currently this is a skeleton function, which prints out very little information.

32. **print_fq()** Command to print out the frequency table to the logfile.

33. **print_flag_summary()** Print out summary information about the applied flags to the logfile. The numeric value of the flag is printed. See Table 1 for the numeric values of the different types of flags.

34. **print_gain()** Prints out the gain table associated with all scans. The gain tables are printed out to the file gaintable.dat.

35. **print_hdu()** Command to print out the primary HDU (of the input FITS file) into the logfile.

36. **print_stats()** Command to print out the statistics of the selected data subset into the logfile. This command is meant to be invoked via **browse_stats()** and not directly by the user.

37. **print_su()** Command to print out the source table into the logfile.

38. **print_summary()** Creates a summary log file. This file gives summary information on the input file, the fluxes determined for the calibrators, the processing done, the fraction of data flagged etc. The summary log file is called "fcs.log".

39. **read_an()** Command to read in the antenna table from the input FITS file.

40. **read_fq()** Command to read in the frequency table from the input FITS file.

41. **read_hdu()** Command to read in the primary HDU from the input FITS file.

42. **read_rgmf()** Composite command to read in a random group FITS file. Reads in the tables, makes an index and also reads in all the scan data. This would require enough memory to hold the entire FITS file in RAM. Use of this function is generally discouraged. Users are encouraged instead to use the atomic commands to read in the FITS headers and tables, and to read in the scan data as required.

43. **read_scan()***(scan, fits_in,scan_min_length)*. Command to read in the data of the selected *scan* from the random group fits file *fits_in*. It is assumed that the file already has been opened and indexed by make_index(). If the scan has fewer than scan_min_length records, then the entire scan is flagged. If scan_min_length is set to a value =< 0 this test is ignored.

44. **read_su()** Command to read in the source table from the input FITS file.

45. **restore_data()***(scan)* Command to restore the scan data from the previously made using **save_data()**. See **save_data()** for some examples on when this might be useful.

46. **restore_par()** Command to restore the control parameters back to the values they had at the last call to **save_par()**.

47. **reweight()***(scan)* Command to reweight the data as per the system temperature. The system temperature is available per channel for the phase calibrator scans, and the value that is used is the average value of the nearest phase calibrator scan on either side of the target scan. See also **compute_tsys()** and **tsys_transfer()**.

48. **save_data()***(scan)* Command to save a copy of the scan data. This is useful when one want to do some temporary operation which alters the data (e.g. apply calibration followed by further flagging) but finally want the original data written to file. In the case of calibration followed by flagging, pairs of **save_data()** and **restore_data()** calls allow one to make multiple flagging passes through the data, but finally write out a data set with a consistent (i.e. the final) calibration applied for all scans. The data flags accumulate as one makes multiple passes, but not the calibration tables. For this to work properly one should also call **free_gain_table()** and **free_bpass_table()** before the second calibration.

49. **save_par()** Command to save the current control parameters. This is useful when one temporarily wants to change the value of some parameter, but then later set it back to the old value. Please note that it is not possible to save one parameter at a time. **save_par()** saves all the parameters and **restore_par()** restores all of them to their last saved values.

50. **setjy()***(scan)* Sets the flux of the source in the selected *scan* to the value determined by Baars et al. (1977). If the source is not in the Baars et. al list, then FLAGCAL issues a WARNING. If the source is in the list for which FLAGCAL has some (not very reliable, caveat emptor!) estimates of the flux, then the flux is set to this estimate. If not the flux is set to 1.0.

51. **scan_stats()***(scan)* Compute the statistics for the selected scans. The main statistics that FLAGCAL works with are the median, and the median absolute deviation (mad). The statistics are computed and stored separately for the real, imaginary, amplitude and phase for each record, channel, baseline, and antenna in the scan. The phase statistics properly accounts for the fact that the phase is a circular variable (i.e. wraps around after reaching $2\pi$). The over all statistics of all of the data in the scan are also computed and stored.

52. **solve_bpass()***(scan, sol_alpha, sol_bp_div_by_ch0,sol_epsilon,sol_uvmax,sol_uvmin, sol_ref_ant, sol_min_ant,sol_max_iter, sol_max_retry)* Command to compute the bandpass using the visibility data in the selected *scan*. The parameters are essentially the same as that used in **solve_chan0()** and are described in detail there. The only differences are that **solve_bpass()** does not use the *sol_solint()* parameter. Instead only one solution is obtained by averaging the bandpass solution over the entire scan. Also if the *sol_div_by_ch0* option is selected, then the data is divided by the "channel 0" before computing the bandpass. Otherwise, no division is done. In this case the data should have been calibrated (in time for the variation of the broadband gain) before running **solve_bpass()**.

53. **solve_chan0()***(scan, sol_alpha,sol_epsilon, sol_uvmax,sol_uvmin, sol_solint, sol_ref_ant, sol_min_ant, sol_max_iter, sol_max_retry, sol_solint)* Command to compute the antenna gains using the visibility data in the continuum channel. See **compute_chan0()** for information on how to construct the continuum channel. The

gains are solved for by assuming that the visibilities are for a point source at the phase centre, and using a steepest descent least squares minimization . The parameters used are as follows

- *sol_solint* solution interval (seconds). Data is averaged for solint seconds before computing the solution. For solint¡=0 data is averaged over the entire scan. NB: the median is used as the average value.
- *sol_alpha* Loop gain in the iterative least squares solution.
- *sol_epsilon*When the fractional change in the solution is less than *sol_epsilon* the solution is deemed to have converged.
- *sol_uvmax* UV max (kL) of baselines given as input to the solver.
- *sol_uvmin* UV min (kL) of baselines given as input to the solver.
- *sol ref_ant* Reference antenna number. The phases of the solutions are given with respect to that of the reference antenna. If set to -1, FLAGCAL will pick a reference antenna based on the available flagging. The smallest numbered antenna (antenna numbering starts with C00) which is not flagged in either polarization in any calibrator scan is picked. Hence, unless all central square antennas are flagged, this will result in a central square antenna being picked as reference.
- *sol_min_ant* The minimum number of antennas needed for computing a solution. If less number of antennas are available (or less number of baselines for a given antenna) no solution is attempted. The output solution is flagged.
- *sol_max_iter* The maximum number of iterations to try before giving up in case there is no convergence.
- *sol_max_retry* The maximum number of retries after flagging antennas with bad solutions. After *sol_max_iter* iterations, the antenna gains are examined and all those which have not converged (i.e. for which the fractional change in the solution is greater than *sol_epsilon*) are flagged. A least squares solution is once again attempted, but this time without the flagged antennas. This is done a maximum of *sol_max_retry* times.

54. **tsys_transfer()***(scan, calsrc)* Transfers the Tsys table from the specified *calsrc* to the target *scan*. If no *calsrc* is specified the nearest phase calibrator on either side of the target source is selected. The target source Tsys table has two rows. The first row is a copy of the Tsys table from the nearest phase calibrator observed just before the target source, and the second is a copy of the table from the phase calibrator observed just after the target source. If one of these is missing, then both rows have the same table, viz. that of the nearest phase calibrator on either side. Note that phase calibrator Tsys tables have only one row. The Tsys table can be used to reweight the visibilities (see **reweight()**. See **compute_tsys()** for some information on how the Tsys is computed.

55. **usr_flag()***(scan, flagfile, usr_vis_maxamp, usr_vis_minamp)*. Command to apply user specified flags to the data. These flags are independent of the statistics of the data, and can be regarded as a kind of pre-flagging, or priors that the user can supply to FLAGCAL. The *flagfile* is read, and the flags found there are applied to the data. The *flagfile* is assumed to be in the AIPS flagfile format, however FLAGCAL understands only antenna and channel based flagging. So only lines of the sort

    ANTENNAS=14 TIMERANGE=01,03,47,30,11,00,00,00 REASON="XXX"/

    or

    CHANNELS=0,12

    are understood. The actual details of the line content after the end of TIMERANGE or CHANNELS specification are ignored. The idea is that the flag file generated by ONLINE can be pre applied to the data.

Additionally, the user could specify a range of bad channels to be flagged. This is useful for observations where the IF bandwidth is smaller than the baseband bandwidth (e.g. in some dual freq or 230/150 MHz observations). Further, the user can specify a range of values (*usr_vis_maxamp, usr_vis_minamp*) within which the visibilities for the *scan* have to lie. Visibilities whose amplitudes lie outside this range will be flagged. If the end time is 0, then all data from the start time to the end of the observations is flagged.

56. **write_base()***(scan, print_base_num)*. Command to generate an FITS cube containing data for the selected baselines. *print_base_num* is a comma separated list of baselines, and a separate FITS cube is written for each baseline. The name of the cube is bsNscMdat.fits, where N is the baseline number, and M is the scan number. The standalone functions **lbase, nbase** are useful to associate baseline numbers with antennas and vice versa. For each stokes parameter there are multiple planes, each of which has channel number on the horizontal axis and record number on the vertical axis. The are 5 planes for each stokes parameter, viz. the real, imaginary, amplitude, phase, and flag for each visibility in the baseline.

57. **write_fits()***(fits_out)*. Function that bypasses the **make_template()** step and instead writes out all the data into the output random group FITS file at one go. Use of this function is discouraged. Users are encouraged to use **make_template()** and **write_scan()** instead.

58. **write_chan0()***(scan)*. Write out the "channel 0" (see **compute_chan0()** into a separate FITS image cube. The file has 4 dimensions (baselines,records, stokes,product). The product dimension is itself of length 5, and the order is real, imaginary, amplitude, phase, and weight.

59. **write_scan()***(scan, fits_out)*. Command to write out the data of the selected *scan* to the random group fits file *fits_out*. It is assumed that the file itself has been created using **make_tempate()**.

60. **write_stats()***(scan)*. Command to make a FITS image cube with the statistics of the selected *scan*. Two FITS cubes are created for each selected *scan*. The file scanNchan_stats.fits contains the channel statistics for scan N. The file has 4 dimensions (baselines,channels, stokes, statistics). The statistics dimension is of length 8, and the order is Median of real, imaginary, amplitude and phase, and MAD of the real, imaginary, amplitude and phase. The file scanNchan_stats.fits contains the records statistics for scan N. The file has 4 dimensions (baselines, records, stokes, statistics). The statistics dimension is of length 8, and the order is Median of real, imaginary, amplitude and phase, and MAD of the real, imaginary, amplitude and phase.

61. **write_rgmf()***(fits_out)*. Composite function to write out all the data into the output random group FITS file. Use of this function is discouraged. Users are encouraged to use **make_template()** and **write_scan()** instead.

| Command Name | Command Description | Command Pars |
|---|---|---|
| bpass_transfer | Transfer BP soln from cal to target | scan, calsrc |
| browse_data | Browse through the data | scan |
| browse_stats | Browse throwh the statistics | scan |
| calibrate | Calibrate using the gain table | scan, apply_gain, apply_bpass,bp_mode, amp_phase_int |
| compute_chan0 | Compute the channel0 | scan, chan0_start, chan0_end, chan0_nchan |
| compute_tsys | Compute system temperature | scan, calsrc |
| fake_scan_data | Fill Scan with simulated data | scan, sim_iflux, sim_snr, sim_seed sim_variable_gain |
| flag_aggregates | Aggregate flags | scan, ant_min_ok_frac, base_min_ok_frac, chan_min_ok_frac, rec_min_ok_frac |
| flag_ant | Flag individual antennas | scan, ant_min_amp, ant_max_amp, ant_outlier, ant_re_outlier,ant_im_outlier, ant_max_nrms, ant_max_ph_rms, ant_min_ph_rms, ant_max_re_rms, ant_max_im_rms |
| flag_base | Flag individual baselines | scan, base_min_amp, base_max_amp, base_outlier, base_re_outlier, base_im_outlier,base_max_nrms, base_max_ph_rms, base_min_ph_rms, base_max_re_rms,base_max_im_rms |
| flag_block | Flag blocks of data after smoothing<br>all pars relevant to flag_vis | scan, smooth_cwidth, smooth_rwidth, smooth_redo_stats, smooth_fudge_fac |
| flag_chan | Flag individual channels | scan, chan_min_amp, chan_max_amp, chan_outlier, chan_re_outlier, chan_im_outlier, chan_max_nrms, chan_max_re_rms, chan_max_im_rms |

Table 2: Flag Commands available in flagcal FLAGCAL

| Command Name | Command Description | Command Pars |
|---|---|---|
| flag_multi_block | iterative smoothing & flagging | scan, smooth_width0, smooth_maxwidth, smooth_redo_stats, smooth_fudge_fac |
| flag_rec | Flag individual records | scan,rec_min_amp, rec_max_amp, rec_outlier, rec_re_outlier, rec_im_outlier, rec_max_nrms,rec_max_re_rms, rec_max_im_rms |
| flag_res | Flag based on residuals | scan,smooth_cwidth, smooth_rwidth, and all parms for flag_rec(),flag_chan(),flag_vis(). |
| flag_transfer | Transfer flags from adjacent PhsCal | scan, calsrc |
| flag_vis | Flag individual visibilities | scan, vis_chan_outlier, vis_chan_re_outlier, vis_chan_im_outlier, vis_rec_outlier, vis_rec_re_outlier, vis_rec_im_outlier |
| flush_log | Flush out the log file | |
| free_bpass_table | Free bandpass table associated with scan | scan |
| free_gain_table | Free gain table associated with scan | scan |
| free_scan | Free memory allocated for scan data | scan |
| gain_transfer | Transfer gains from cal to trgt | scan,calsrc |
| get_src_info | Read Src Info from user file | |
| getjy | Set the flux of the phs calibrator | scan,apply_el_corr |
| make_index | Make Index of UVFITS file | fits_in, scan_maxbreak |
| make_template | Make template output UVFITS file | fits_in, fits_out |
| print_an | Print Antenna Table (AN)to LogFile | |
| print_bpass | Print bandpass tables of all scans | |
| print_index | Print index to LogFile | |
| print_finfo | Print fits file header information | |
| print_fq | Print Frequency Table (FQ) to LogFile | |
| print_flag_summary | Print Summary of data flags | scan |
| print_gain | Print gain tables of all scans | |

Table 3: Flag Commands available in flagcal FLAGCAL

| Command Name | Command Description | Command Pars |
|---|---|---|
| print_hdu | Print primary HDU | |
| print_stats | Print scan statistcs to LogFile | scan |
| print_su | Print Source Table (SU) to LogFile | |
| print_summary | Print summary log info to fcs.log | |
| read_an | Read the AN table | |
| read_fq | Read the FQ table | |
| read_hdu | Read the primary HDU | |
| read_rgmf | Read a Random Grp Fitsfile | fits_in |
| read_scan | Read the scan data | scan,scan_min_length |
| read_su | Read the SU table | |
| restore_data | Restore scan data from last | scan |
| restore_par | Restore the control parameters cpar | |
| save_par | Save the control parameters cpar | |
| save_data | Save a copy of the scan data | scan |
| setjy | Set the flux of the flux calibrator | scan |
| scan_stats | Compute Robust Scan Statistics | scan |
| solve_bpass | Compute the bandpass solution | scan, sol_alpha,sol_epsilon, sol_uvmax,sol_uvmin, sol_ref_ant, sol_min_ant, sol_max_iter, sol_max_retry |
| solve_chan0 | Compute the channel0 gains | scan, sol_alpha, sol_epsilon, sol_uvmax,sol_uvmin, sol_ref_ant, sol_min_ant, sol_max_iter, sol_max_retry, sol_solint |
| reweight | reweight visibilities by Tsys | scan |
| tsys_transfer | transfer tsys from PhsCal to trgt | scan,calsrc |

Table 4: Flag Commands available in flagcal FLAGCAL

| Command Name | Command Description | Command Pars |
|---|---|---|
| usr_flag | Apply user specified flags | scan, flagfile usr_vis_maxamp, usr_vis_minamp |
| write_base | Write baseline data to fits file | scan, print_base_num |
| write_fits | Write out all data into a FITS file | fits_out |
| write_chan0 | Write channel 0 data to FITS file | scan |
| write_scan | Write scan data to output UVFITS | scan, fits_out |
| write_stats | Write scan stats to FITS file | scan |
| write_rgmf | Write entire Random Grp FITS file | fits_out |

Table 5: Flag Commands available in flagcal FLAGCAL