Automatic RFI Identification and Flagging

Urvashi R.V. and A.Pramesh Rao
N.C.R.A.,Pune
NCRA Technical Report

October 1,2003

Abstract

Flagging RFI affected data from recorded radio astronomy data sets is an inevitable first step in the process of data analysis. Routine manual examination of increasingly large data sets is no longer feasible. This report describes the design and implementation of an automatic RFI identification and flagging algorithm for radio interferometric data sets. RFI affected antennas,channels and timestamps are identified from the autocorrelation data recorded at each antenna, and all visibilities obtained from corrupted data in either antenna of the corresponding baseline pair are flagged. The current version of the program is in use at the GMRT.

1 Introduction

At low radio frequencies, stray electromagnetic transmissions often interfere with the incoming radiation from a source, thus corrupting the data being recorded. Sources of Radio Frequency Interference (RFI) may be external (satellite and aircraft radar, wireless network and TV/radio transmissions) as well as internal (antenna cross-talk, imperfect Faraday cages). Flagging involves the identification and masking of RFI affected data points, and is an inevitable step in standard data analysis.

RFI identification and removal, is usually done manually, by visual inspection of the data. This is a time-consuming and highly subjective process. Also as data volumes grow, it is not practical for an observer to examine all the data and manually find and mask corrupted data points.

This work involves the design, implementation and testing of an algorithm that partially automates this process of RFI identification and flagging for GMRT data.

2 Framework and Assumptions

The total power $M(\nu, t)$, recorded at each antenna, can be modeled as follows.

$$M(\nu,t) = [P_{sky}(\nu,t)G_{ant}(\nu,t) + RFI(\nu,t)G_{ant}(\nu,t) + P_{rec}(\nu,t)] * G_{rec}(\nu,t) \quad (1)$$

where P_{sky} is the source power, $RFI(\nu, t)$ is the RFI and P_{rec} is a measure of the T_{rec}. G_{ant} and G_{rec} are complex gain factors due to the antenna and receiver electronics.

This gives

$$M(\nu,t) = [P_{sky}(\nu,t)G_{ant}(\nu,t)G_{rec}(\nu,t) + P_{rec}(\nu,t)G_{rec}(\nu,t)] + RFI(\nu,t)G_{ant}(\nu,t)G_{rec}(\nu,t) \quad (2)$$

Assuming broadband noise, the power at the output of the baseband system is

$$M(\nu,t) = P(\nu,t) + (RFI(\nu,t) * g(\nu,t)) \quad (3)$$

where

$$P(\nu,t) = P_{sky}(\nu,t)G_{ant}(\nu,t)G_{rec}(\nu,t) + P_{rec}(\nu,t)G_{rec}(\nu,t) \quad (4)$$

and $g(\nu,t) = G_{ant}(\nu,t)G_{rec}(\nu,t)$

In interferometry, each RFI source can either be local to a particular antenna, or may affect a group of (or all) antennas.

The effects of RFI are most clearly seen in the autocorrelated visibilities of individual antennas. In cross correlated visibilities, the main effect is an increased effective T_{sys}. (If the RFI affects only one of the antennas in the baseline pair, this is the only way the RFI is visible in the cross correlations).

When both antennas in a baseline pair are affected by RFI, the effect of source fringe correction results in RFI fringing at rates proportional to the projected baseline length. In long projected baselines where the integration time length spans several cycles of the RFI fringes, there is no net increase in mean flux and there is only an increased noise. In short projected baselines, where the integration time may not span multiple RFI fringe cycles, there can be a net increase in mean integrated flux.

This algorithm focuses on flagging visibilities for all baselines of an antenna, based on the RFI detected in the autocorrelations of that antenna. A direct flagging scheme will involve detecting antenna based RFI from autocorrelation data, and flagging all cross correlated visibilities formed from antenna pairs where either one or both are RFI affected.

This algorithm has been implemented in Octave(Matlab) and 'C'. It is adapted to the GMRT database formats, and produces a text flag file,in a format readable by AIPS. It has also been partially implemented as a glish client in AIPS++ and thus can be applied to data stored as a measurement set in AIPS++.

3 Algorithm for Automatic RFI Identification and Flagging

The algorithm operates on the self correlations of each antenna over all scans. The analysis works on two dimensional data sets, comprising of self correlation amplitudes as a function of channel number (128 channels) and timestamps (across the time range of a scan).

The RFI is assumed to be narrow-band, and localized in frequency. The source flux per channel is assumed to not vary erratically over the time-range of an observation scan. Any flags already present in the system will be taken into account while examining the data, but not modified.

The following is a description of the various steps and heuristics that have been adopted, to detect and flag RFI affected data points.

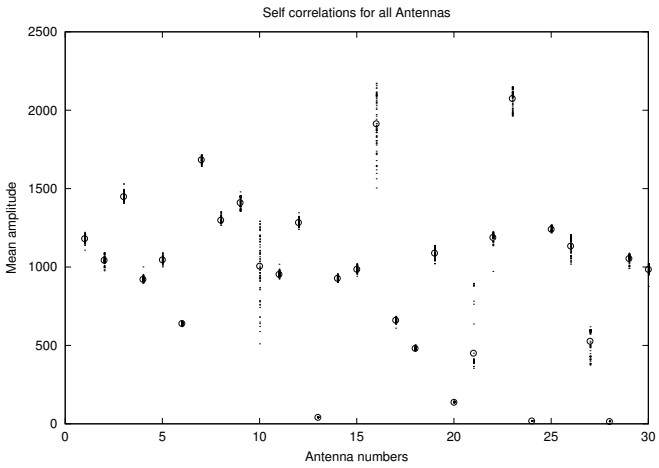

Figure 1: Mean Amplitude of all Antennas

3.1 Identification of live antennas

The mean of the entire two dimensional data set, is first calculated to identify if a particular antenna is 'dead' or not. If the mean value is below a user defined threshold, the antenna is assumed to be dead, and above it, to be good.

A plot of the Mean amplitudes for all antennas, as well as the spread of individual data points, for each antenna across the time-range of an observation scan, is shown in Figure 1.

3.2 RFI identification in time (per channel)

Data for each channel is then examined separately, over the entire time-range of the scan, and sharp peaks corresponding to sudden erratic disturbances are flagged.

This is based on the assumption that the data in a particular channel, does not change significantly during the scan time interval. Any variation that may happen, is due to a change in antenna gains, which is gradual, and will contribute to a general slope in the data values. Hence, a linear fit to the data, will suffice to model this slope.

For a particular channel, a one dimensional data set is obtained with values varying with time. A straight line is fitted to this data set and deviant points are identified with respect to this fitted straight line. The measure of 'goodness' of a data point,is the magnitude of the difference between itself, and the value of the fitted line at that point, in comparison to the standard deviation of the one dimensional data set.

A user specified tolerance level, given as a multiple or fraction of the standard deviation, with respect to the linear fit, is used to decide which data points are to be flagged.

This process of fitting a straight line to the data, and flagging data points with respect to it, is repeated until the standard deviation stops changing significantly. In each iteration, the linear fit is calculated after discarding the data points that were flagged in the previous iteration.

This operation, performed on all channels separately, identifies erratic RFI, present for only short time ranges within the scan.

For RFI that is present over more than half the duration of the scan, the fitted line, is itself biased, and does not provide a good reference to detect any RFI. Hence this first stage will detect only intermittent or erratic RFI, affecting only short time ranges in the scan.

3.3 Bandpass RFI identification (per time-stamp)

In order to identify sustained RFI in any channel, data has to be examined, as a function of frequency. Data remaining after the flagging of the previous step, is averaged over time, for each channel, resulting in a one dimensional data set corresponding to an effective bandpass of that antenna, for that scan. This can be compared to looking at the bandshapes of the data, at the output of the baseband system (as seen by the correlator).

This corresponds to removing the time dependence of the variation of the data left. Channels with sustained RFI across the duration of the scan, will show up as peaks in $M(\nu)$, a one dimensional data set obtained by averaging the data per channel, over the time-range of the scan.

Therefore from Eq. 3

$$M(\nu) = P(\nu) + RFI(\nu) * g(\nu) \tag{5}$$

where $RFI(\nu)$ is sustained RFI as a function of frequency.

This formulation requires the following simplifying assumptions.

The system noise and all other noise, except for RFI, is taken to be broadband, and the signal from the sky, including the RFI is assumed to correspond to far-field radiation. The data is examined at the output of the baseband system, without considering the systematics of the specific parts of the system affecting the data, before and up to the baseband system.

In equation 5 if $P(\nu)$ is known a priori, then from the observed $M(\nu)$ (data averaged over time, for each channel), and $P(\nu)$ one could identify $RFI(\nu)$ and flag it. However, since $P(\nu)$ is not known, it can only be estimated from the observed $M(\nu)$. This can be done by an iterative scheme, that identifies sharp variations in $M(\nu)$ and smooths over the base of the RFI peaks, in the estimation of $P(\nu)$.

The algorithm used to estimate $P(\nu)$ can be outlined as follows..

A piecewise polynomial is fitted to $M(\nu)$, smoothly interpolating across the bases of any sharp peaks. This, if accurate, can be thought to represent $P(\nu)$.

The first step is to identify sharp peaks in $M(\nu)$, assuming that RFI is usually narrow-band, and has a signature localized in frequency. Also since the bandpass is supposed to be inherently smooth, any large sharp deviations, would most probably correspond to RFI.

The polynomial fitting, incorporates the identification of sharp peaks, and it is based on the following heuristics.

1. $P(\nu)$ can be thought to vary smoothly, and hence piecewise polynomials of the order 3 or 4 should suffice. These lower order polynomials, will not fit any sharp high peaks, and this approximation, over several iterations, would discard the high peaks, and smoothly interpolate over their bases.

2. Each 'piece' of the data set, over which each polynomial is fitted, represents approximately 15 to 20 frequency channels. This has been chosen, because any larger data sets per polynomial, would require higher order polynomials for an accurate fit, which might result in sharp peaks being fitted too.

3. The initial piecewise fit is linear, and largely discrepant points are discarded. In the next iteration, third order piecewise polynomials are fitted, and smaller sharp peaks are identified. These iterations proceed a few times, as the high peaks are identified and discarded from the data set.

4. Due to the $sinc^2$ response to RFI, in the frequency domain, it is expected that for channels with very high RFI, adjacent channels may also be affected by the sidelobes of the main high RFI peak. This is accounted for, by replacing values

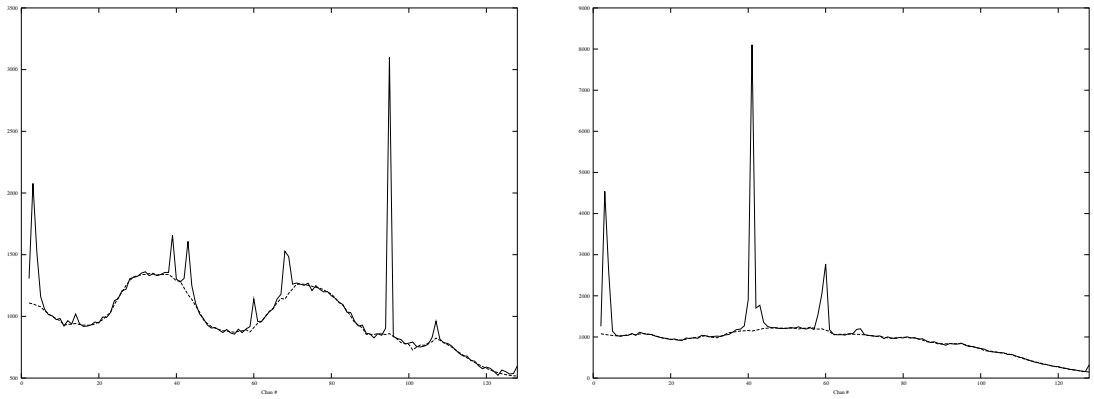

Figure 2: Raw Mean Bandshape and Polynomial fit(`left`)W03 and (`right`)C06

of $P(\nu)$ in channels adjacent to those containing high RFI peaks, by interpolated values. This is done by comparing the height of the main peak, and the expected height of its sidelobes, with the standard deviation. Depending on whether the heights of the expected sidelobes are above or below the standard deviation, adjacent channels are suppressed and interpolated over.

Two examples of the polynomial fits are shown in Figure 2.

The actual values $M(\nu, t)$ for each time-stamp are now compared to this estimate of $P(\nu)$ and sustained RFI signatures are identified and flagged from the entire data set.

This is done, by dividing out $P(\nu)$ from one dimensional data sets obtained as a function of frequency, for each time-stamp t from $M(\nu, t)$. This corresponds to looking almost directly at the RFI itself,above a constant mean, and the RFI can easily be detected.

For each time-stamp,

$$H(\nu, t) = \frac{M(\nu, t)}{P(\nu)} = 1 + \frac{RFI(\nu, t) * g(\nu, t)}{P(\nu)} \tag{6}$$

where $g(\nu, t) = G_{ant}(\nu, t) * G_{rec}(\nu, t)$ and $P(\nu)$ is given by Eq. 4

$$H(\nu, t) \approx 1 + \frac{RFI(\nu, t)}{\left[P_{sky}(\nu, t) + \frac{P_{rec}(\nu,t)}{G_{ant}(\nu,t)}\right]} \tag{7}$$

It can be stated that $H(\nu, t)$ has the characteristics, of the RFI itself, affected mainly by a constant additive offset.This is depicted in Figure 3. The denominator of Eq. 7 only scales the deviant RFI affected points, and will not affect the overall RFI pattern across frequency and time.

$H(\nu)$ is now examined, for each time stamp, and data is flagged according to deviations, from a constant mean, compared to the standard deviation of the values of $H(\nu)$. A user specified tolerance level, given as a multiple or fraction of the standard deviation,

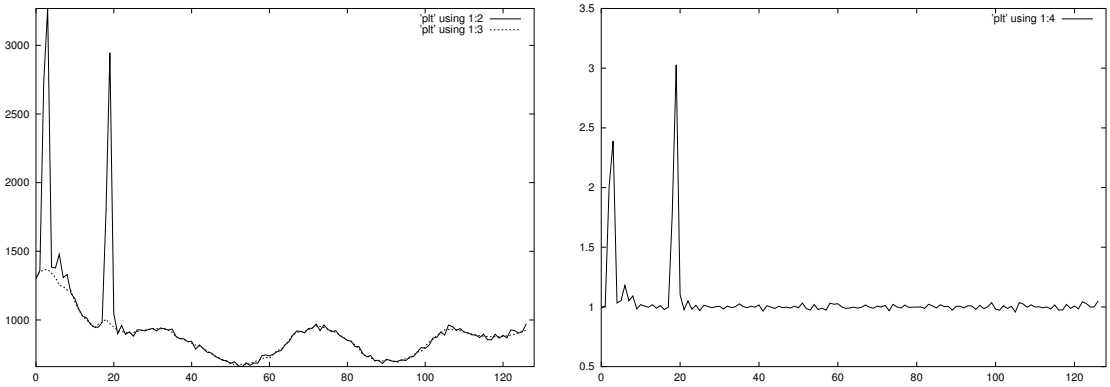

Figure 3: Raw Bandpass with Polynomial Fit divided out - (`left`)$M(\nu)$ and $P(\nu)$ and(`right`) $H(\nu)$

with respect to a now constant mean, is used to decide which data points are to be flagged. This is also done iteratively, until the calculated standard deviation converges. Each time, flagged data points are ignored in the calculation of the mean and standard deviation.

3.4 Flag Heuristics

The result of the above two operations, is a two dimensional flag table (a mask), of the same dimensions as the time-frequency data. Ones correspond to unflagged data points, and zeros correspond to flagged data points.

A few heuristics are now applied to this flag table, and are a part of user defined options. This depends on some known characteristics of the data and comprises of several levels of flagging.

1. The flagged points, are spread out in time by one time-stamp, in an attempt to ensure that any sidelobe of high discrepant points, is also removed.

2. If a channel has more than 60% flagged points, the channel is flagged for the entire time range.

3. The first two and last two channels are flagged, for all antennas, for the entire time-range. This corresponds to making all entries of the first two and last two columns of the flag table, equal to zero. In addition, if any channel is flagged for the entire time-range of the scan, the two adjacent channels are also flagged.

The two dimensional flag table is then scanned and a list of flags is prepared, which can then be converted to any format. Currently, the mechanism has been implemented to create text flag files, with entries corresponding to each stretch of bad data, to be flagged.

3.5 Summary Output

A Summary of the flagging process is generated, and contains details of the mean and rms of the data before and after flagging.In addition, a table is created, with information about the fraction of good data remaining, after flagging, in each channel, per antenna. This can be used to identify inherently good or bad antennas, as well as good or bad channels, which is useful, in later calibration stages, when deciding a reference antenna and channel.

Gray scale plots of the time-frequency data for the E02 antenna, are shown in figures 4 and 5. Figure 4 is a gray scale plot of the raw time-frequency data for E02. Figure 5 is a gray scale plot of the detected flags applied to the raw data.

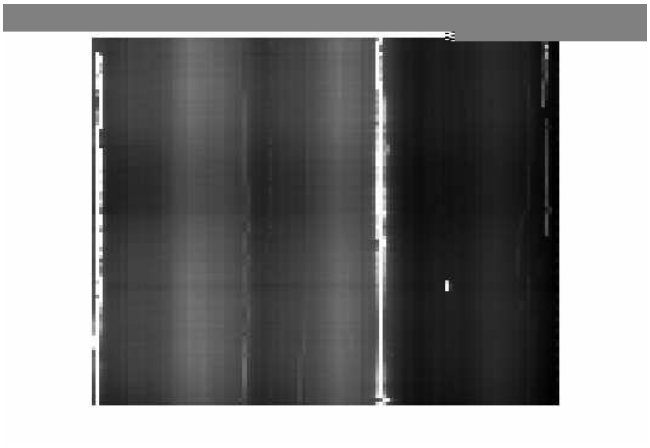

Figure 4: Gray Scale plots for E02 - Raw Data

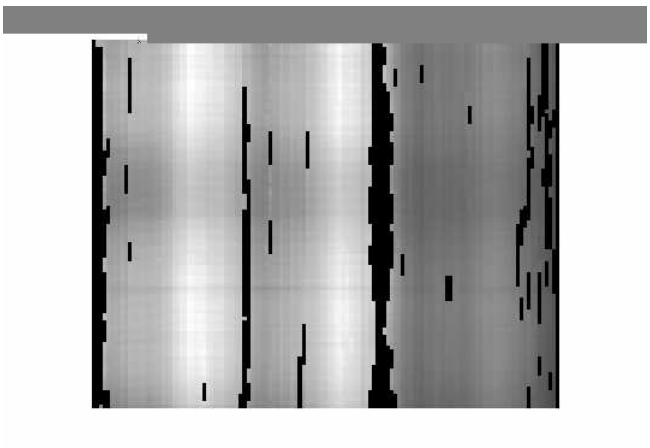

Figure 5: Gray Scale plots for E02 - Flags applied to Raw Data

4 Implementation Details

A test version of the *ltaflag* algorithm was implemented in Octave(MATLAB). The main application has been written in C along with the Numerical Recipes library.

The $C++$ offline software user interface library was used to generate the input user interface. The graphical user interface was provided by Kudale Sanjay, based on the *tax* gui in use at the GMRT. The *shmflag* application uses software contributed by Jayaram Chengalur to read the incoming data from the shared memory.

It has also been packaged as a stand-alone library, with a suitable programmable interface to any application programmer, who would like to use these routines as part of an integrated system. The *genflag* application can be used with time-frequency data in any format, by converting it to the generic input data format.

The main control flow in the algorithm, has been depicted in the following flow-chart (Figures 6 and 7).

A detailed description of the User Interface of the three programs is given in the Appendix.

5 Method of Testing

The performance of the algorithm, has been tested in the following ways.

1. Bandpass plots before and after flagging were examined to make sure that only discrepant channels were detected during the piecewise polynomial fitting.

2. Gray scale plots of the data for each antenna per scan, have been looked at both before and after flagging. Any discrepant data points which would normally be attributed to RFI, were found to have been detected by the algorithm.

3. The text flag file that gets generated, was taken into AIPS and the data was examined using the task 'SPFLAG' , to check the time-stamp numbering conventions and whether the list of flags has been correctly set.

4. The flags applied to the cross correlations (based on the RFI detected in the autocorrelations) were examined from within AIPS and verified visually.

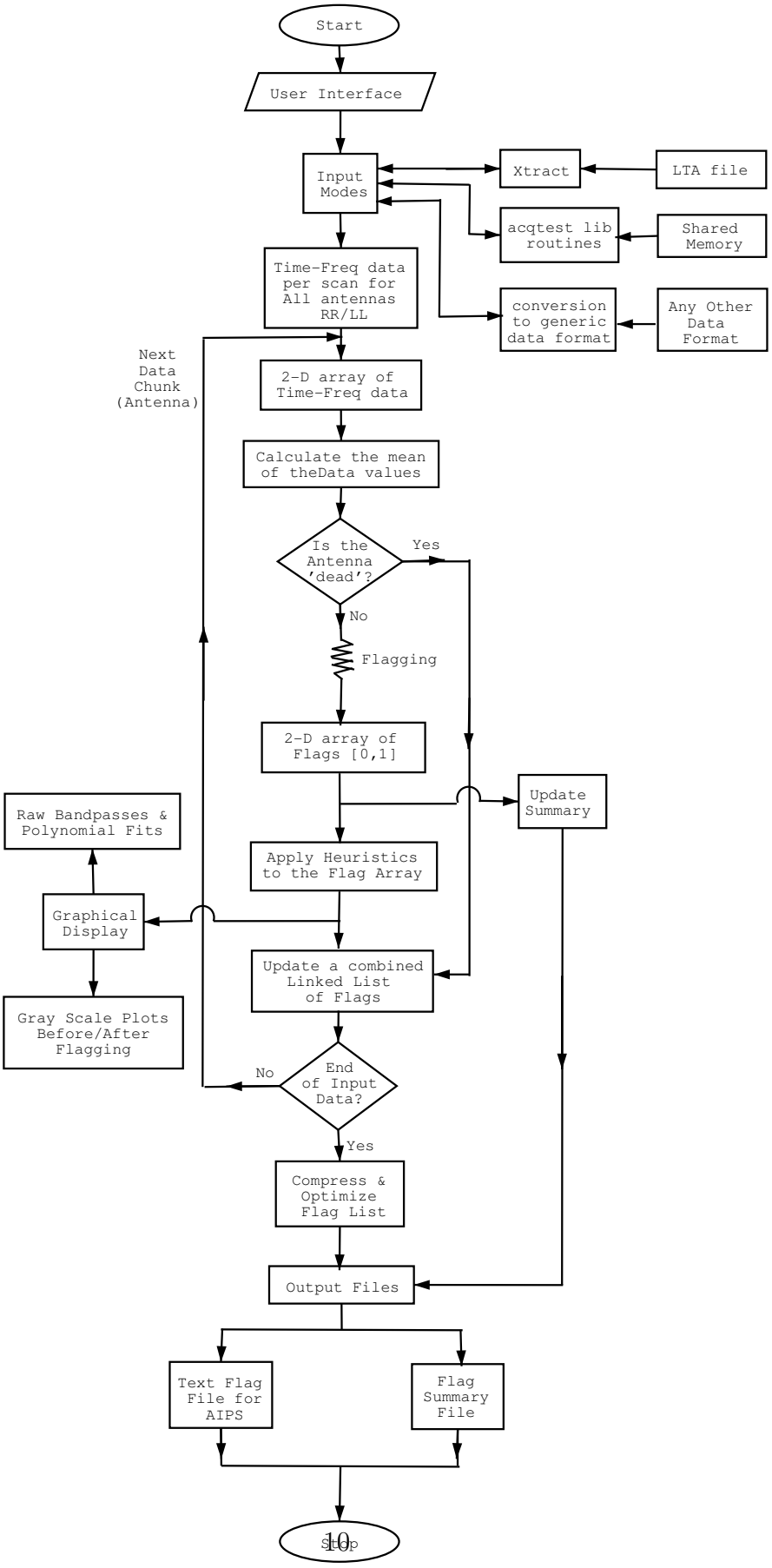

Figure 6: Control Flow for the Flagging Algorithm - 1

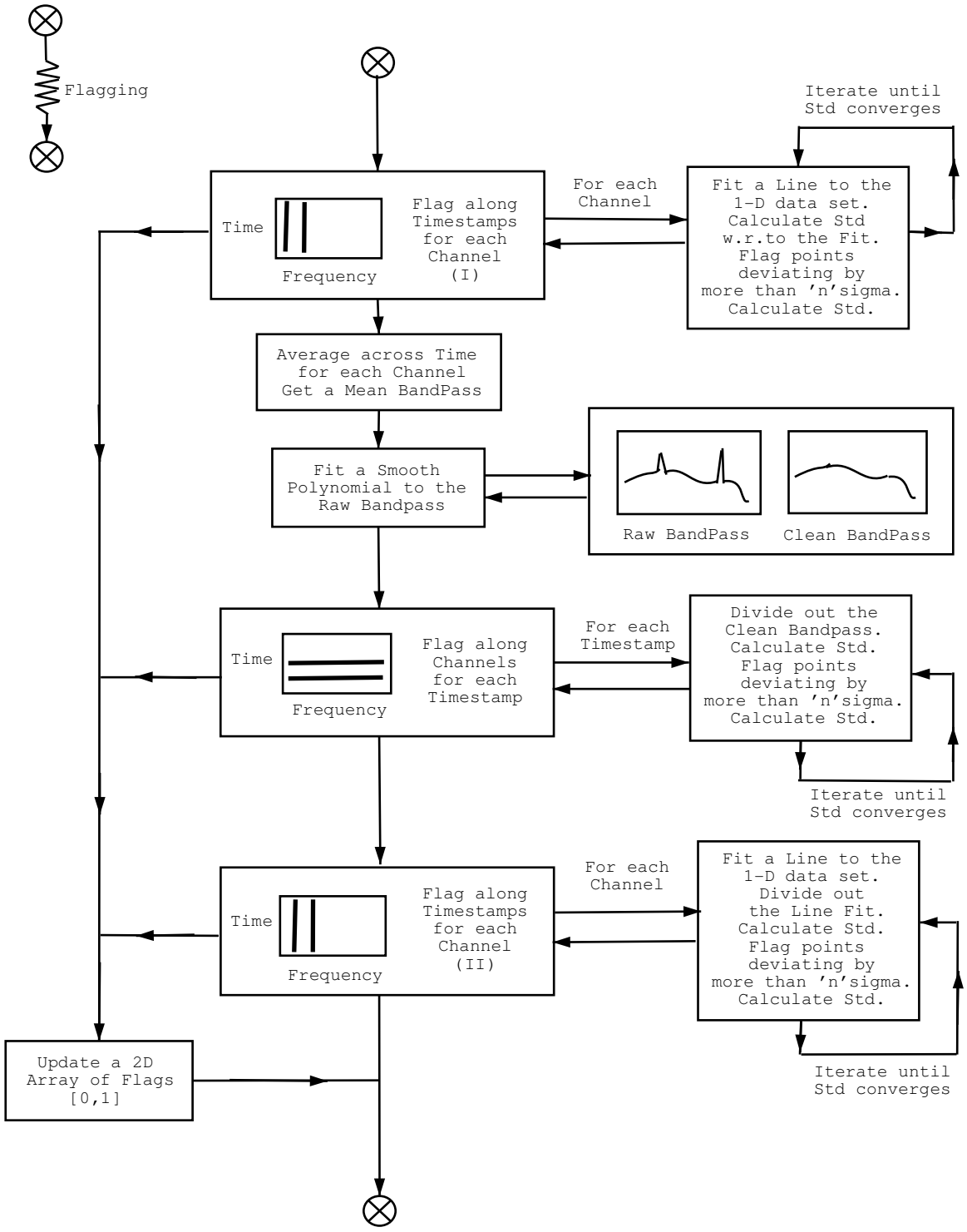

Figure 7: Control Flow for the Flagging Algorithm - 2

6 Current Status and Further directions

6.1 Current Status

1. This algorithm does not work on cross correlation data. It will only flag data points based on what it finds in the self correlation data. At present, this method will be able to detect RFI stronger than a few multiples of T_{sys}. Low level RFI will not be detected by this algorithm. Any spurious lines that appear only in the cross correlations, due to the electronics in the system, are not detected and it is assumed that flags for this corrupted data have already been set.

2. The technique described here is useful mainly for continuum mapping, where any sharp peaks in the spectrum can be deleted with no serious loss of information. For spectral line analysis, if the radio line of interest, lies near some RFI, the astronomer may be reluctant to blindly flag all the RFI, and a different strategy may have to be undertaken. One way to still make use of this program, is to provide lenient flagging cut-offs, so that only large RFI spikes, are detected and flagged. Also to prevent automatic flagging in certain channels, the user can provide a region in the band which should be excluded by the RFI removal algorithm.

3. Broadband RFI can often appear like a feature of the actual bandpass, and hence will not be detected by this scheme. The user can however look at the bandpasses and the corresponding polynomial fits, and judge why a particular part has been flagged or not. The polynomial fitting routines can handle most kinds of bandpasses, but when they are exceptionally bad, the fitted polynomials are not very accurate. But the hope is that if the bandpasses are that bad, then that data may itself not be too reliable.

4. This implementation currently uses the offline software package "xtract" to read specific data from the LTA file. "xtract" is being invoked via a pipe from within the ltaflag application. The entire data that xtract sends to its output is currently being stored in the pipe buffer, and is read only in chunks by ltaflag. Hence if all scans have been chosen, all the data is sent to this pipe buffer until ltaflag reads it chunk by chunk. This causes the 'Loading Data' routines to run sub optimally. This can be rectified, by directly using the lta access routines from one of the available libraries.

5. There is a limit on the number of lines that the AIPS task 'UVFLG' can accept. If the text flag file grows beyond a certain number of lines, 'UVFLG' displays an error message and ignores the rest of the flags. One way to get around this is to split the text flag file into smaller files and load them in sequence into 'UVFLG', or to run "ltaflag", operating on less than 5 scans at a time.

6.2 Work in Progress (as of September 2003)

1. The 'AutoFlag' tool within AIPS++ has been explored. As of July 2003, the status of this tool is as follows. It uses a server-agent framework each flagging algorithm is a separate 'agent'. The user can choose one or more flagging agents to be run simultaneously on the data. The data is then traversed in a strict time-stamp by time-stamp loop, and the chosen algorithms are run, each one filling a different bit in an output flag word. At the end, this flag word is written to the measurement set. The various algorithms available at present are 'TimeMedian','FreqMedian','Sprej' (a Spectral Rejection filter that is similar to the ltaflag algorithm, though it works on single timestamps, on smaller channel ranges, and requires user input on the degree of the polynomial to be used in the fit). Additional flagging options in AIPS++ are available in the display module using gui-based local statistics. This visual module is currently disjoint from the autoflag tool. Currently, no-one maintains the autoflag module, and the lack of technical documentation has so far prevented anyone from completely understanding the implementation design. Some users have tested it, but have reported that they 'are not sure' if it is flagging what they think. The tool has been scheduled for scrutiny.

2. The ltaflag algorithm has been partially implemented into the autoflag framework. It has been demonstrated to work and is usable as a glish client. It has however not been formally integrated completely into the autoflag tool as an agent.

3. The poly-fit routine is being changed to be more robust to varied band-shapes. This routine was re-written while incorporating this algorithm into the AIPS++ autoflag tool.

4. An option of using polynomial fits in time is being added.

5. The 'xtract-bash-C' pipeline currently in place for data input from lta files, is being replaced with routines that directly access the LTA file. This will result in faster 'data loading time' as well as efficient memory management (unlike the current implementation, where the 'xtract-bash-C' pipeline forces all the data to be stored in buffers long before they are actually processed). Larger data sets can then be processed without a visible drop in efficiency.

6. In addition to the output AIPS flag file, an output LTA file with its flag bits set will be an option. This way, programs that create fits files can incorporate these flags into a flag table within AIPS, which can be used or replaced by the user. This will eliminate the need for extremely large output text files which are bulky and sometimes subject to file size limits set in AIPS. Such a filter will also be necessary for it to be used as an element of an automatic data processing pipeline.

7. A few additions to the current options are being implemented. The `flag_more` options are being made list-based and non cumulative. An additional option is being provided to flag all channels for a particular timestamp if RFI is present in any of the channels. This will be useful for spectral line flagging (request by Nissim Kanekar). Also for spectral line users, flagging an RFI affected channel from the entire data set (all baselines) for that particular timestamp will be necessary (to maintain the same UV coverage for all channels). An additional keyword 'sideband' has been added (after a request from Dharam) to distinguish between USB and LSB data.

6.3 Future directions

1. The existing code structure is generic enough to allow other programs like 'gvfits' to add in a flagging module. The advantage of this is is that 'gvfits' can then create a FITS file which includes an optional binary flag table that AIPS can recognize.

2. An alternative to (1) would be to modify the standard program that converts LTA files to the FITS format to recognize the flag bits of the LTA file, creating an AIPS readable optional flag table in the FITS file. `ltaflag` can be then used as a pipeline module which creates an LTA file with the flag bits appropriately set.

3. If (1) or (2) are not done and if the use of an output AIPS text flag file is to continue, the text flag file generation must be optimized, using compression heuristics to the list of flags that the program generates. This can be done either within the ltaflag code structure, by manipulating the linked list of structures that stores the list of flags for all the scans, before writing it out as the AIPS readable flag file, or by running a shell/awk script on the output text flag file, to optimize its list of flags and reduce its size. One heuristic that already exists in the current implementation is to group together flags contiguous in time. An improvement on this would require a heuristic to identify and represent two dimensional clusters in the time-frequency plane.

4. Currently, cross correlation data is directly flagged based on the autocorrelations. An additional feature can incorporate a scheme where 'possibly corrupted' cross correlation data, identified as before from the autocorrelations, is not immediately flagged but subject to further examination. Data points with abnormally high amplitude can directly be flagged as this corresponds to RFI on shorter baselines, where the RFI fringes have not averaged out during the integration time length. Data points with just higher noise levels with respect to nearby points could correspond to RFI effects on long baselines, where the integration time is long enough to average out the RFI fringes and there is no mean increase in amplitude. The weights of these data points can be reduced instead of flagging them. This may help prevent over-flagging (Perley,2002).

5. A pipelined version of ltaflag (Input and Output LTA files) can be used for online flagging of data in real-time. This can be used as the first stage of an online data analysis package.

6. The AIPS++ autoflag tool has some direct algorithms based on time and frequency median filters which could be useful for additional flagging by examining data from all cross correlations. These algorithms can be incorporated into the ltaflag application as additional options.

7. Alternatively, if the ltaflag algorithm is incorporated into the AIPS++ autoflag tool with its data formats, it can be used on generic data sets. It is trivial to make the current ltaflag program a Glish client. This uses only the Glish interface of AIPS++ which has standard routines for consistent and stable data I/O, and standard visualization tools. This will eliminate the use of 'xtract' or any other non-standard routines for data I/O, and the algorithm can be used to process data stored as FITS files and measurement sets.

8. A user-involved systematic testing procedure must be used, during which the usefulness of the program in routine data analysis can be assessed and documented.

Acknowledgements

I thank Kudale Sanjay for his contribution of the complete graphical user interface for the software that was developed, Jayaram Chengalur for his shared memory related software and Sanjay Bhatnagar for his help with the GMRT/NCRA offline software system, the usage of the related libraries, and some design features of AIPS++.

I thank Dharam Vir Lal for his efforts and time spent in running and testing *ltaflag*, providing operational support to external users, giving valuable suggestions regarding improvements to be made to the basic algorithm, and now co-ordinating the formal induction of the application into the offline software system at the GMRT/NCRA. I would also like to thank Nissim Kanekar, Subhashis Roy, Poonam Chandra, Neeraj Gupta and Nimisha G. Kantharia for useful points regarding the algorithms developed, and for testing the programs and reporting bugs. I would also like to thank Rajiv K. Singh and Rajaram Nityananda for their advice on the algorithm and the implementation.

At the NRAO, I thank David King, for his time and ongoing effort in understanding the autoflag framework of AIPS++ and his suggestions about the implementation of the ltaflag algorithm into it. I also thank Athol Kemball for his guidance in implementing the ltaflag algorithm in AIPS++ and George Moellenbrock, Joe McMullin, and Kumar Golap for their advice and pointers on the same.

All of this work was done on computers running the GNU/Linux OS.

References

1. Bhatnagar, Sanjay, Xtract : a flexible data extraction program for GMRT visibility, 1997, Jan., NCRA Technical Report #R00171

2. Bhatnagar, Sanjay, User interface for off-line applications,
1997, March, NCRA Technical Report #R00173

3. Bhatnagar, Sanjay, Tips for programming with GMRT off-line software, 1997,March,
NCRA Technical Report #R00189

4. Autoflag Tool, AIPS++ User Reference Manual,
"http://aips2.nrao.edu/docs/user/Refman/Refman.html"

5. AIPS++ Programming and System Manual,
"http://aips2.nrao.edu/docs/programmer/programmer.html"

6. Perley, Rick.,"Attenuation of Radio Frequency Interference by Interferometric Fringe Rotation",November 2002,EVLA Memo #49.

A Appendix - User/Programmer Interface

A.1 Ltaflag

There are several keywords provided to the user, which when appropriately set, can control the functioning of the application. This application to read data from an lta file and produce an output flag file readable with the AIPS task 'UVFLG' goes by the name of "ltaflag".

The keywords for "ltaflag" and their functions are described below.

1. `'in'` (default=stdin)

   To specify the source of data in the lta format
   in = file.lta

   To read from streamed lta data
   in = < cat file.lta
   in = < nshmdata ...

2. `'mode_out'` (default=0)

   Output modes
   0 : No output files
   1 : Create a text Flag file for the AIPS task 'UVFLG'
   2 : Create a text Summary file
   3 : Create both Flag and Summary files

   For display and visual analysis of the data and RFI, mode '0' would be ideal. The information that is recorded in the Summary file, can appear at run-time, by setting the keyword 'print' to 1.

3. `'out'` (default=myoutfile.flag)

   This is the output text flag file, readable by the AIPS task, UVFLG.
   ***This is a hidden keyword to change the output filename ***

4. `'summary'` (default=mysummary.txt)

   This is a summary file that is created. It contains statistical information about the self correlated data, before and after flagging.

   It contains details of the flagging process in the following form
   (1) A table for all 30 antennas (RR/LL) with fields:
   - Antenna number / name
   - Mean amplitude of the self correlation data for that scan
   - Mean and RMS of the data set after dividing out the estimate of the bandpass, and BEFORE flagging.

- Mean and RMS of the data set after dividing out the estimate of the bandpass, AFTER flagging.
- The fraction of good data left in that antenna
(2) A table of antenna number Vs channel number The values in the table is the Fraction of good data left in each channel/antenna, after flagging.
***This is a hidden keyword to change the filename ***

5. tt 'scans' (default=0)

   Scan numbers of the scans to be read from the lta file. These numbers are the same that appear for each scan in the output of 'ltainfo'.
   scans = 0,1 => scan numbers 0 and 1
   scans = => all scans in the lta file.

6. `'channels'` (default=1,127)

   Channel range and increment of the data to be analysed.
   channels = 1,127,2 means Channel nos 1,3,5,...127
   It is preferable to start the channel range from '1' instead of zero[1] .

7. `'sideband'` (default=USB)

   Sideband of the data to be analysed

   sideband = LSB (or) sideband = USB

   If 'USB', this keyword internally forms the string "A.+-USB-.+" which is passed as the 'baseline' parameter to 'xtract'.

8. `'ant_cutoff'` (default=0)

   Threshold by which to identify a 'dead' antenna
   It is the value to which the mean of the time-channel data points for each antenna for one scan is compared. If the mean is less than this threshold, it indicates a dead antenna which is then completely flagged.

9. `'time_tol'` (default=3)

   Specifies the threshold against which data is flagged while flagging across time for each channel.
   During this first flagging pass, a line is fitted to the one dimensional data set of values for EACH channel across the timerange of the scan. Points deviating from this line by more than 'time_tol' * sigma, are flagged. Sigma is the standard deviation of the data set, calculated with respect to the line fit.
   The level of flagging can be controlled by the user, by changing this parameter.

[1]The channels are numbered from 1 - 127 and this corresponds to channels 2 - 128 inside AIPS.

10. `'freq_tol'` (default=2.5)

    Specifies the threshold against which data is flagged while flagging across channel for each timestamp.
    In this second pass, a piecewise polynomial is fitted to the time average of the raw bandpasses. This polynomial is smooth and interpolates across the base of any sharp peaks.
    Raw bandpasses at each timestamp are now divided by this smooth estimate of the bandpass, and points deviant from mean of the resultant data set are flagged. The threshold for deciding the points to be flagged, is 'freq_tol' * sigma, where sigma is the standard deviation of the data set with respect to the mean.
    The level of flagging can be controlled by the user, by changing this parameter.

11. `'plot'` (default=0)

    Turns the interactive display GUI on and off.
    If plot = 1 , a window similar to 'tax' appears and plots of bandshapes before and after smoothing, can be obtained.
    In addition, gray scale plots of time-frequency data can also be displayed on 'ds9'.

    'ds9' must run in the background, before running 'ltaflag'.
    Failure to do this will signal a broken pipe and the application will exit.

    ```
    $ ds9 &
    $ ltaflag
    ```

    Moving the cursor around on the gray plots in 'ds9' - will cause 'de9' to display the actual data value in the information panel. This is the actual value of the data point being pointed at, by the cursor[2].
    The first channel has not been considered in the analysis and this channel number 1 is by default flagged for all antennas.
    The analysis begins from channel number 2 (AIPS)

12. `'print'` (default=0)

    Turns the interactive TEXT information display on and off.

    If print = 1 , a summary of the data quality is displayed for each antenna (RR/LL) one scan at a time.

    The fields of the tabular display are as follows...

    (1) Antenna number / name
    (2) Mean amplitude of the self correlation data for that scan

[2]The CHANNEL numbers on the ds9 gray scale plots will appear as 1 - 127. This corresponds to the actual channel numbers as in AIPS - to channels 2 - 128.

(3) Mean and RMS of the data set after dividing out the estimate of the bandpass, and BEFORE flagging.
(4) Mean and RMS of the data set after dividing out the estimate of the bandpass, AFTER flagging.
(5) The fraction of good data left in that antenna

A gray-scale PLOT of the FRACTION of GOOD data remaining in the data, after flagging, appears on 'ds9'.

The x-axis is channel number - 2 to 128
The y-axis is antenna number+stokes

Eg. The rows (y) go from bottom to top - 1 - 60 where
1 : C00[RR]
2 : C00[LL]
3 : C01[RR]
4 : C01[LL]
... and so on.

Information about the quality of data present in each antenna across channels, as well as in each channel for different antennas - can be visually obtained from this plot. The dark regions signify flagged data and the gray scales vary from black : completely flagged ant/chan to white : unflagged data.
Moving the cursor around on the gray plot - will cause 'ds9' to display the actual fractional value in the information panel.
In addition to this, the following menu is also displayed.
Flag Query
Set Threshold [1]
Antenna - Bad Channels [2]
Channel - Bad Antennas [3]
Bad Antennas/Channels [4]
Good Antennas/Channels [5]
Quit [0]
Enter your choice :

This is to optionally query the flag summary, to identify 'bad' antennas/channels.

The following numbers correspond to the flag query options listed above.

0 quit from this menu and move onto the next scan.

1 to specify a number between 0 and 1, signifying the fraction of flagged data.

All later displays within this menu, will report antennas and channels with more than THIS fraction of data flagged.

Default = 0.9 i.e. queries [2],[3],[4] will report antennas/channels with more than 90% bad data.

2 Specify an antenna NUMBER (1 to 30) and the display will be a list of CHANNELS in THAT antenna, which have more than 90% data flagged.

3 Specify a channel NUMBER (2 to 128) and the display will be a list of ANTENNAS for which THAT channel has more than 90% bad data.

4 Prints out a list of bad antennas and channels (IF ANY) depending on the specified threshold.
Eg. It prints a list of antennas with more than 90bad data, considering all channels, and a list of all channels that are bad on the whole - for more than 90% of the antennas.

5 Prints out a list of good antennas and channels (IF ANY) depending on the specified threshold.

These results also include the fraction of good data present in the chosen antenna/channel.

13. `'flag_more'` (default=1)

To control the level of flagging. These levels are cumulative.

Eg. flag_more = 3 will include levels 0,1,2 and 3.

0 : Flag ONLY what is detected by the algorithm

1 : Spread all flags in time, by one timestamp on either side.

2 : If a channel has flags for more than 60% of the time range of the scan, then flag that channel for the entire timerange of that scan.

3 : Flag the first two and last two channels being analysed. For channels flagged for the entire timetrange of the scan, flag adjacent channels

4 : For channels that are bad for more than 60% of the antennas, flag that channel for all antennas.

14. `'flag_ants'` (default=¡blank¿)

If any antennas are known to be 'bad' or 'dead', list them here and they will be flagged by default.

Eg. flag_ants=1,2,24

will flag antennas 1, 2 and 24 completely for the scans specified for the program run.

***This is a hidden keyword ***

15. `'flag_chans'` (default=¡blank¿)

    If any channels are known to be bad, they can be specified in a list and will be flagged for the entire timerange of the specified scans.

    Eg. flag_chans-1,2,45,66

    ***This is a hidden keyword ***

16. `'comments'` (default=No Comments...)

    This is to type in text comments that the user may want, to identify the output files. Any text typed in at this keyword will appear in the beginning of the output Flag file and Summary file.

    A single line of text can be typed directly at the keyword
    OR
    The name of a TEXT file can be specified at this keyword. The program will then transfer the textual content of that specified file into the output text files - for later reference.

A.2 Other programs

There are two other modes in which this program works.

1. One mode can directly read from the shared memory, and flag chunks of data in real time. This program uses the acqtest offline libraries, to access the shared memory, and extracts the self correlation data. It works on data chunks of sizes as mentioned with the 'ntimes' keyword in the application "$shmflag$". A description of these keywords and usage, can be obtained from the online documentation.

2. There is a generic data format that one can use to pipe data into the program, in the generic data mode. This will allow one to run the same algorithms on any other kind of data. This mode is incorporated in a symbolic link called "$genflag$".
   This generic data format is as follows...
   ```
   #CHANS <number of channels>
   #TIMES <number of timestamps>
   <timestamp1> <data values for all channels>
   <timestamp2> <data values for all channels>
   ....
   ```
   The algorithm works on chunks of data that comprise of the number of timestamps, as mentioned in the second line of the data.

   A description of these keywords and usage, can be obtained from the online documentation.

3. There is another program that can be run on the data recording machine, to look at gray-scale plots of the data, and average bandpasses, in real time. This goes by the name "$ondisp$" and can be used to monitor RFI in real time. The gray scale display is incremental and scrolls after a maximum limit of 100 timestamps is reached. Integration times can be set, for each timestamp that appears in the gray-scale plots.

4. A generic library of easy to use routines and functions has been provided for anyone to use some of these generic routines in related applications. These routines require the Numerical Recipes library, for the polynomial fitting routines, and are otherwise stand-alone routines.