Chapter 3

Data analysis software

This chapter describes briefly the off-line data analysis software, developed to satisfy the data analysis needs for debugging the GMRT as well as for developing new algorithms for calibration and imaging at low frequencies. A modular, hierarchical software environment was therefore developed over the course of this dissertation. This software system is organized in the form of general purpose libraries at the lowest layer and a suit of general purpose application programs at the top-most layer. This design required much extra work, but it was perceived to be well worth the effort with the hope that ultimately, at the highest software layer, a large number of programmers would be professional astronomers writing complicated (but interesting) numerical analysis code.

The entire software now constitutes in excess of 50,000 lines of `C++` code and in excess of 10,000 lines of `C` and `FORTRAN` code. Some others in the GMRT group have used these general purpose tools to write more software. In a software development cycle, the number of bug reports per unit of time (week or month) is generally considered to be an indicator of the stability of the software. Assuming this to be a good measure, the libraries are now well debugged and stable (however, a low rate of bug reports could also be due to lack of sufficient number of careful users).

3.1 Introduction

The correlator output is used for on-line monitoring of the health of the array (cross and self correlated band-shapes, closure phases, amplitude and phase of various baselines, antenna based amplitude and phase, system temperature (T^s), antenna temperature (T^a) variations as a function of time, etc.) as well as for on-line phasing of the array for the phased array mode. Recorded data is extensively used for the measurements of various telescope parameters (baseline calibration, *fixed delay* calibration, antenna pointing calibration, beam shape measurements at various frequency bands, measurement of T^s, T^a, antenna efficiency (η) and their variation as a function of time/elevation, etc.). All this requires extensive data analysis and data display capabilities to be easily available for on-line and off-line usage. The GMRT correlator produces 435 cross correlations plus 30 self correlations corresponding to 465 complex numbers per integration time per IF per frequency channel. If all the 128 frequency channels are recorded, this corresponds to 465 baselines $\times$ 128 channels $\times$ 2 IFs $\times$ 2 floating point numbers of size 4 bytes each$= 952320$ bytes of data per integration cycle. With a typical observing time of 8 hours with an integration time of ~ 20 seconds, this corresponds to a database of size ~ 1.5 GBytes. Hence the software should also efficiently handle such large multi-dimensional database and allow easy browsing through the database.

The visibility data is an explicit function of the baseline length (projected separation between the antennas). Implicitly however, it is a function of many other parameters like local sidereal time (LST), observing frequency, the antenna co-ordinates, the co-ordinates of the *phase center*, *compensating delay* applied to the various antennas, etc. Most of the processing (on-line as well as off-line) requires efficient access to the visibility data potentially as a function of many of these parameters. During the debugging stage of the telescope, it is also important to have a short turn-around time between observations and results. This in turn demands a fairly sophisticated data analysis package to analyze the data recorded in the native recording format as well as evolve continually with the potentially rapidly changing environment (including the format itself!). Preferably, such a data analysis package should also be usable on-line. The application programs must also provide a user interface for the software to be usable by a larger community.

Section 3.2 describes the design of the software system which was designed to meet most of the above mentioned requirements. Section 3.3[1] briefly describes the design of the low level libraries used for accessing the visibility database. Section 3.4 describes the design of the user interface, while Section 3.5 describes some of the application programs used in the observations and data analysis for this dissertation.

[1]A description of the latest application programs and libraries is available via http://langur.ncra.tifr.res.in/~sanjay/Offline

3.2 The software design

The GMRT off-line data analysis package was designed to (1) provide efficient means for browsing the multi-dimensional dataset represented by $V(b_{ij}(t), \nu, \tau, \partial\tau/\partial t, ...)$ (2) be robust to changes in the database format (3) provide a higher level astronomically useful view of the data, (4) provide a programming environment which can be used by knowledgeable astronomers to write novel instrumental debugging, data calibration and analysis tools, (5) provide easy means of data selection and (6) provide a simple user interface to the data analysis program.

The design for the entire software was therefore dominated by the following basic design goals:

- the application programming interface (API) be such that the application developer need not manage software/book keeping details not relevant for the goal in mind.

  In terms of the design of the libraries, this translates to a modular software with the API of each module carefully designed to mesh well with the other modules

- to keep the user interface simple and easy (but robust) while still providing mechanisms for more general, possibly more complicated usage by more enterprising users.

  In terms of software development, this translates into implementing generalized software solutions with a programmable interface.

- it should be easy to supply the results from one application program to another.

  With this, it is often possible to effectively perform a complicated data processing/filtering operation using a number of stand-alone programs without writing a monolith program (which is difficult both to write as well as maintain).

- it should be easy to display and view the data and the results of data processing.

  Since it is virtually impossible to write a data display software which satisfies all present and future needs, it is best to design the data analysis software such that external stand-alone display software can be used. This way, the task of development of display software (requiring very different software skills and professional motivation) is separated.

  This also allows the freedom to use a display software that one is most familiar with and satisfies the needs. A large number of stand-alone plotting/data display software packages are now freely available. With such a design, one is not restricted to use locally designed specific display software allowing enormous scope for creativity and an unrestricted, independent development path.

The Object Oriented design methodology was used to design a hierarchical software where each layer hides the details of the layers below it via the interface presented by each layer. Data access, manipulation, extraction and selection mechanisms have, wherever possible, been implemented in a general form in the libraries (see Section 3.3) while keeping their usage as simple as possible at higher levels via their API.

The package is organized in the form of three layers of libraries over which a suite of application/data analysis programs are built. The lowest layer has been written in the `C` programming language while the rest of the software have been written in `C++`. With a lightweight wrapper (also written but almost never used), this lowest layer provides a `FORTRAN` interface for basic access to the LTA database. The user interface was designed as an embedded shell and is implemented as a set of four separate libraries including an arbitrary mathematical expression parser. An application using this interface provides an interactive shell which is largely independent of any external files.

Most application programs have been written as filters (which operate on an LTA database and produce an LTA database) . The user interface provides easy mechanisms for piping such filters to effectively perform complicated tasks.

3.2.1 On-line usage of the software

The correlator output is first written in a shared memory resource provided by POSIX compliant operating systems. The program `record`, which is part of the Data Acquisition Software, converts the data in the shared memory format to the *LTA-format*. This program was modified to write the output LTA formatted visibility data to the standard output instead of writing to a disk file. As mentioned above, most off-line programs have been written to accept the input data from a pipe. This feature of the off-line software system can then be used, in combination with the modified `record`, for on-line data analysis by piping the data to the programs from the shared memory. Thus, *all* programs used for off-line data browsing, data validation, automatic detection of bad data, etc. (see Sections 3.5.1, 3.5.2, 3.5.3, 3.5.4) can also be used as on-line programs. The locally developed data display/plotting software also accepts plotting instructions and arbitrary data via `UNIX` sockets. The output of the off-line programs, along with embedded plotting instructions, can be easily supplied to the display software over `UNIX` sockets. Thus, with a combination of the data piping facility, modified `record` and the display software, the off-line software can be effectively used for on-line data display.

3.3 Data manipulation libraries

The GMRT visibility database is recorded in the *LTA format*. The database is always multi-channel and is composed of zero or more *scans*. Each scan represents the data for one pointing direction and is composed of zero or more data records. The database begins with an ASCII header referred to as the *global header* which carries information about the observations common to all the scans. Each scan begins with an ASCII header of its own, referred to as the *scan header*. The *scan header* constitutes information relevant for the scan alone (e.g. source RA, DEC, object name, information about the frequency contents of the data, time reference information, etc.).

The LTA data manipulation libraries are organized in three layers. The lowest layers is written in `C` and is responsible for parsing and reading/writing the LTA database headers. It is also responsible for maintaining the consistency of the output LTA database. The interface for reading from and writing to the *global* and *scan* headers in the memory is provided by this layer. Higher level `C` and `FORTRAN` bindings are provided via light weight wrappers, which hide the details of managing the linked lists etc. However, using these interfaces, only one LTA database can be opened in a single application. Using the `C++` interface, a single application program can handle any number of databases. This interface also provides many more astronomically useful features while manipulating the visibility database, and is the recommended interface to use. These libraries maintain two copies of the *global header* in the memory as a doubly linked list for the entire lifetime of the application (unless explicitly destroyed). Hence all changes, if any, in the output are represented in the output-header in the memory, which becomes the header of the output LTA database. One of them represents the input database while the other applies to the output database. However, only the latest *scan header* is maintained in the memory.

At even higher level, the `C++` classes (also referred to as `C++` *objects*) called `LTAFMT`, `LTAVIEW` and `LTARec` encapsulate the LTA format. Any number of these objects can be instantiated in a single `C++` application allowing the handling of multiple LTA databases. These objects, briefly described below, were used in application programs for data calibration and testing some automatic data filtering algorithms. There are other `C++` objects which were used for maintaining and manipulating antenna gains and other tables (in the memory and on disk), which are not described here. Those are described in a set of NCRA Technical reports.

3.3.1 The `LTAFMT` Object

The `LTAFMT` object is the base class for handling LTA database. This object encapsulates the functionality which is expected to be a common factor between all classes which manipulate LTA databases and perform higher level operations (e.g. see the `LTAVIEW` object below). It provides the I/O interface to the LTA headers, and presents the visibility data as a two dimensional array of complex numbers. The data-records, along with the visibility data, also carry other auxiliary data (e.g., parameters for *fringe stopping*, *fringe rotation*, *FSTC*, etc.). The interface to this auxiliary data is also provided by this object.

All higher level objects derived from this class (also referred to as *derived* classes) would have all the functionality of `LTAFMT` object plus the additional functionality of the inherited class. Almost all the methods of the `LTAFMT` (functionally, the services offered by this object), have been declared as `virtual` which allows the *derived* objects to modify the behavior of these methods. A redefinition of these methods in a derived object would override their functionality. This leaves the scope for development of higher level objects with minimal effort.

3.3.2 The `LTAVIEW` object

This is an object derived from `LTAFMT` and is designed to provide a *view* of the data with selection on the source name and integration in time and/or frequency applied to the data.

With a source selection, this object behaves exactly as `LTAFMT` except that it will give a view of the database with scans only of the selected source. The source selection can be specified via a `POSIX egrep` compliant regular expression (with the exception that '+' regular expression operator has been disabled since its interpretation as a regular expression operator causes conflict with the astronomical source naming convention where '+' appears as a valid character in the name). Data selection based on scan number is also possible.

Vector, scalar and robust averaging algorithms are built into this object and can be used to average the data in time or frequency. The averaging interval is supplied as the number of seconds of effective integration required. `LTAVIEW` will then average the data for the closest possible interval of time. This averaging of the data is, however, restricted to the scan and does not cross the scan boundary. Most of the application programs use this object for data manipulation.

3.3.3 The `LTARec` object

Data recorded in the LTA database is composed of the data timestamp and the auxiliary data, followed by the real and imaginary parts of the visibility and self correlation data for each baseline for all recorded frequency channels. The `LTAVIEW` and `LTAFMT` objects provide access to the *raw* contiguous buffer of floating point numbers in the memory representing the complex visibility data record.

The `LTARec` object provides an astronomically useful view of this *raw* buffer. It is initialized using an LTA database object (`LTAFMT` or `LTAVIEW`). It can then be used to access the data timestamp, auxiliary data indexed by the antenna number, and the visibility data as a two dimensional array of complex numbers. The array indexing operator (the '`operator[]`' in `C++` syntax) is overloaded in this object to allow indexing inside the array of visibility with the baseline and channel number as the index. It also provides services for mapping between the baseline number and the two antenna numbers which constitute the baseline.

3.4 The user interface for off-line programs

The user interface libraries provide an embedded interactive shell in the application programs – the user interface is part of the compiled binary code of the application programs. In an interactive session, the user is presented with a list of keywords corresponding to the parameters which control the behavior of the application. The user can set/reset, save, load and transfer values from one keyword to another, etc. in the interactive shell. Each keyword can potentially take multiple values which can be supplied as a set of comma-separate values. Numerical values can be expressed as arbitrary mathematical expressions which are parsed at run-time.

The keywords presented to the user are detected automatically at runtime. Since the code for the user interface is part of the binary executable file, this ensures that the interface presented to the user is *always* consistent with the functionality of the program. This eliminates the possibility of the application program and the user interface going out of synchronization, which is, in the long term, a very desirable property. The user interface can (and frequently does!) go out of synchronization with the functionality of the programs, when the user interface depends on external files to determine the keywords; these must be independently maintained and evolved as the application code evolves.

The input data file name is usually specified via the 'in' keyword while the output file name by the 'out' keyword. By default, the input is read from the standard input and output written to the standard output. If the input file name begins with the '<' character, the rest of the file name is treated as a command of the underlying operating system. The output of this command becomes the input of the application. Similarly if the output file name begins with the '|' character, the rest of the file name is treated as a command which reads the output of the parent application. These features are used to supply the output of one program to another.

The applications can also be started in the non-interactive mode by supplying "help = *noprompt*" as one of the command-line options. Other keywords of the application can also be set via "key=<val0>,<val1>,<val2>,..." styled command-line options. This is often useful in setting up a UNIX pipe consisting of a number of programs or when using these off-line programs as part of a shell script. On-line help is also available via the "explain" command in the interactive sessions and via the "help=explain" as one of the the command-line options in non-interactive sessions.

The user interface can be customized to a large extent and can also be used to effectively generate specialized versions of existing applications (see Appendix A). These can be generated by writing the .def and .config files for the applications (see the explanation for the environment variables GCONF and GDEFAULTS in Section A.2).

3.5 Data analysis and display programs

This section describes a few of the application program which were extensively used while observing with the GMRT as well as for instrumental calibration and the measurement of various telescope parameters.

3.5.1 The program xtract and its variants

The visibility function, denoted by V, depends on a number of parameters such as the local sidereal time (LST), observing frequency, the antenna co-ordinates, the co-ordinates of the *phase center*, the *compensating delays* applied to the various antenna, antenna fixed delays, antenna positions, etc. (i.e. $V(b_{ij}(t),\nu,\tau,\,\partial\tau/\partial t,...)$). During debugging, it was frequently required to view this data in various representations (e.g., Cartesian versus polar representation of complex numbers). Also, V depends on a multitude of parameters, and different debugging purposes require viewing V with respect to different quantities. It was therefore not useful to implement a program to extract the data as a function of a fixed set of parameters. Since, as a design philosophy, stand-alone data display software was used, the need for the extracted data to be in a variety of formats (binary as well as in plain ASCII) directly readable by the display/plotting programs was also frequently felt. It was therefore necessary to develop a compact macro language parser to extract and display the data in a flexible and programmable manner.

`xtract` (Bhatnagar 1997b) was designed to extract the visibility data and its parameters in a programmable fashion via a compact macro language. It was also designed to be as general as possible and easily interface with the many stand-alone data display programs in use. *In fact, it was designed with the wider goal of making it easier for a large spectrum of astronomers/engineers using the GMRT to be able to access the visibility data and do further processing if required.* This program was extensively used for this dissertation and is now being regularly used for a variety of applications ranging from on-line and off-line data browsing and display, beam shape, pointing offset, antenna sensitivity measurements, band shape monitoring, etc.

The `xtract` macro language (described in Appendix B) provides a mechanism to define the contents and the output data format. The macros are constructed using the three operators of the language, namely `ant`, `base` and `chan` and the various *elements* (Table B.1). The three operators loop over all the selected antennas, baselines and frequency channels respectively. The language syntax allows arbitrary grouping of the operators with a semi-colon separated list of *elements* as the body of these operators. This effectively provides a compact way of defining macros to extract visibilities and/or other parameters as a function of antenna, baseline and/or frequency channel number. These macros are, in fact, nested loops over the lists of antennas, baselines and/or frequency channels. The entire macro is implicitly the body of the loop over time and is executed for every input data record. The list of available *elements* are listed in Table B.1.

For example, the `xtract` macro to produce a table with the first column containing the time stamp (Indian Standard Time (IST)) followed by two columns for the amplitude and phase of the visibility from each selected baseline, will be "`ist;base{chan{a;p}};\\n`". The list of baselines and channels for which data is to be extracted can be specified via the user interface keywords `baselines` and `channels`. Similarly the u, v, w co-ordinates of antennas can be extracted in a table as a function of HA by the macro "`ha;ant{ua;va;wa};\\n`".

The list of antennas can be specified via the `antennas` keyword.

The `xtract` macro language complier is also available as a stand-alone library. The API of this library can be used by other application programs to parse, execute and get the result of the macros. An attempt was also made to write one such application for the graphical display of the LTA database (line and gray scale plots). This application unfortunately did not stabilize and is not in use.

Antennas are identified by the correlator sampler to which they are connected. In the final double sideband GMRT correlator, there will be 4 samplers per antenna, i.e. 2 samplers for the two polarization channels per sideband. The four signals from each antenna (2 polarizations from each sideband) can therefore be treated as four *logical antennas*. All samplers in the correlator are also uniquely numbered. The *logical antennas* can therefore be specified by the sampler number to which they are connected, or by an antenna name consisting of three hyphen separated fields – the antenna, the side-band and the polarization names. A name for a *logical antenna* is said to be "fully qualified" when all the three fields in the name are specified. All these fields can be regular expressions. Similarly, baselines can be specified by a baseline number or by a baseline name composed of colon (':') separated *logical antenna* names. Here again, the *entire* antenna names (before and after the ':') can be regular expressions. The use of regular expressions for the components of *logical antenna* names as well as for specifying the two antennas of a baseline provides a very general, compact and powerful selection mechanism. The antenna and baseline naming conventions are described in detail in Appendix C. This convention is uniformly followed in all off-line programs where data selection based on antenna and/or baselines is required.

`xtract` normally writes the output in ASCII format with a header specifying the names of different columns and other information about the extracted data. Formats required by a number of freely available line and gray scale plotting programs can be generated by writing the required `xtract` macro. The extracted data can be directly read by these programs for display. The data can also be extracted with the header in ASCII followed by the data in binary format by specifying the output file name beginning with the character '`*`' (it is conceivable to write a `xtract` macro to produce a FITS file to be displayed using any of the FITS image display programs, e.g. for the dynamic spectra from selected baselines).

The user interface provides mechanisms to externally set keywords to some fixed default value(s) and to suppress these keywords from the user interface. This is used to effectively generate specialized variants of `xtract`. One such variant named `oddix` was extensively used for on-line display of the amplitudes and phases from various baselines. The data was read from the shared memory using a modified version of the `record` program, and an `xtract` macro defined to produce the output as a binary table with each row corresponding to a single integration time. The output was then piped to a program which further supplied the data to a display program over the network via a `UNIX` socket. The display program, also written as part of the off-line package, displays an arbitrary number of stacked scrolling line plots. The display surface itself is scroll-able, allowing the viewing of a very large number of line plots at a time.

3.5.2 The `rantsol` and `badbase` programs

The observed normalized visibility can be written as

$$\rho_{ij}^{Obs} = g_i g_j^\star \rho_{ij}^\circ + \epsilon_{ij} \tag{3.1}$$

where g_i is the antenna based complex gain, ρ_{ij}° is the ideal point source visibility and ϵ_{ij} is the baseline based noise. For an unresolved source of unit flux density, $\rho_{ij}^\circ = 1$. For an N antenna array, the g_is represent $2N - 1$ unknowns - the amplitudes and phases of all the g_is (the phase of one of the antennas can be treated as the reference and hence set to zero). ρ_{ij}^{Obs} represents $N(N-1)/2$ observed complex quantities. For $N > 4$ this is an over-determined problem and hence solvable. The program `rantsol` implements an algorithm to solve for the antenna-based complex gains for an unresolved source. The algorithm itself is described in Appendix D.

The visibility data includes data from non-working or malfunctioning antennas as well as data affected by closure errors due to any malfunctioning of the correlator. Since the antenna based complex gains are obtained using a global fit involving all the data, the presence of bad data can result into problems ranging from noisy solutions even for the good antennas to no convergence at all. This can happen even in the presence of a few bad antennas and/or presence of as few as $10 - 20\%$ bad baselines. It is therefore important to remove bad data before attempting to solve for the antenna gains.

This identification and flagging of bad data is done automatically in two passes for every solution interval. Antenna based complex gains are first solved for using *all* the data. Solutions from the first pass are then examined and antennas with amplitude gain less than the user defined threshold are flagged for the second pass. Antennas which are found to be bad in this manner are assigned a complex gain of 1. The solution for an antenna can also be interpreted as a weighted average of the complex visibilities from all baselines with the given antenna (see Section D.1). This averaging in each successive iteration can be done robustly by on-the-fly flagging of data points, which deviate by more than a threshold defined in units of the variance of the series of complex numbers being averaged. Data from a baseline with large closure errors will have large deviations from the mean defined by the data from 'good' baselines. Such data will be identified and flagged in robust averaging. Both these techniques are used in the algorithm implemented in the program `rantsol` to make it robust, even in the presence of time variant closure errors (the latter have been noticed on several occasions).

In practice, `rantsol` has been found to be very robust in the presence of non-working or malfunctioning antennas and malfunctioning `MAC`s in the correlator (which produce large closure errors). It can be used almost as a black box for most of the calibrator databases without the need to identify and flag bad data. `rantsol` was regularly used to compute the antenna based complex gains and another program, `badbase` was used to identify bad data from calibrators scans. `badbase` examines the amplitude and phase of the calibrated visibilities (defined as $\rho_{ij}/g_i g_j^\star$) and reports the fraction of time a given baseline was found to be bad (badness defined as $|\rho_{ij}|$ and $arg(\rho_{ij})$ greater than a user defined threshold). Antennas and baselines which are continuously bad for large fractions of the total observing time were easily identified and flagged before mapping. *This was found to be extremely important as the calibration tasks of the* `AIPS` *package (used for calibration and mapping) are very sensitive to bad data and sometimes resulted in no convergence at all due to the presence of about 10% bad baselines!*

The GMRT typically produces a few hundred baselines per snapshot. Monitoring the data quality for phase, amplitude and closure errors corresponds to monitoring data streams from each of these baselines. This is obviously not practical. However there are only N antenna based complex gains corresponding to N antennas. Solution for antenna-based complex gains using `rantsol` effectively enforces closure constraints. On-line monitoring of these antenna based complex gains, derived using `rantsol` therefore gives a good summary of the data quality. `rantsol` output was also therefore routinely used to on-line inspect the quality of data. The gains were supplied to the online display software mentioned above which displayed the solutions as a set of scrolling line plots (the antenna based amplitudes and phases for the calibrator scans). This was of immense use in identifying time variable problems while observing and ensured that the recorded data was of reasonably good quality.

`rantsol` is now regularly used as a black box for a variety of purposes ranging from baseline and fixed delay calibration (Figs. 2.11 and 2.12 are examples of typical `rantsol` output), pointing offset, beam shape, sensitivity, system temperature measurements to the GMRT phase array operations for pulsar observations (Sirothia 2000).

The output of `rantsol` can be formatted to be directly read by the `QDP`[2] plotting package using the `awk` scripts `getamp`, `getphs` and `getres`. These scripts extract the amplitude and phase of the calibrated visibilities ($\rho_{ij}/g_i g_j^\star$) respectively.

3.5.3 The `closure` program

The antenna based complex gain g_i can be written as $g_i = a_i e^{-\iota\phi_i}$ where a_i and ϕ_i are the antenna based amplitude and phase. For the ideal case with no baseline based errors ($\epsilon_{ij} = 0$ for all i and j), the phase of the triple product for an unresolved source, $\rho_{ijk} = arg(\rho_{ij} \cdot \rho_{jk} \cdot \rho_{ki}) = 0$ and the amplitude of $\rho_{ijkl} = |\rho_{ij} \cdot \rho_{kl}/\rho_{kj} \cdot \rho_{il}| = 1$. The phase of ρ_{ijk} is referred to as the *closure phase* and the amplitude of ρ_{ijkl} is referred to as the *closure amplitude*. These closure quantities are a good measure of the baseline based errors in the system. Ideally, the signals from various antennas, flowing through independent paths, are mixed *only* in the *MAC* stage of the correlator. Although there are several sources which can produce small closure errors, catastrophic closure errors, which severely limit the final dynamic range in the maps, can be traced to malfunctioning *MAC*s in the correlator. The closure phases are therefore a very important quantity to monitor during the observations. It is also important to examine these closure quantities while processing so as to identify bad data; this is done using the `closure` program.

The output of this program, which computes all the closure quantities from the data, was also used for on-line display of the closure phases. This output was simultaneously also supplied to another program which raised an alarm in case the closure phases for the calibrator scans deviated from the expected value by more than some threshold amount. This helped in identifying problems with data before spending long hours of observing and recording of otherwise unusable data.

[2]Written by Allyn Tennant, originally at Cambridge, but developed at NASA Marshall Space Flight Center

3.5.4 Conversion to `FITS` format

The final inversion and mapping of the visibility data was done using the `AIPS`[3] package. The visibility data was imported into `AIPS` by first converting the data from the *LTA format* to `FITS` format using the off-line program `gl2fit`[4] (however, see Section 4.4.1 for some details about use of other data filters before `gl2fit`). The `FITS` file was then imported to `AIPS` using the `AIPS` task `FITLD`.

3.5.5 Miscellaneous programs

In addition to the programs mentioned above, various other programs, which were routinely used during the course of this work, were also developed. Although these programs are not used for numerical computation, they are nevertheless useful and often indispensable.

- The program `ltainfo` lists the astronomically useful summary of each scan in the LTA database (total number of scans in the database, string identifier for the astronomical object, RA and Dec of the pointing center (in date epoch), the hour angle corresponding to the first data record, centre frequency of the RF band and the width of each frequency channel).

- The program `ltacat` concatenates multiple LTA databases into a single database. Since the LTA-*global header* applies to the entire database, all constituent databases are checked for consistency with the first LTA database. Those LTA databases which do not conform to the *global header* of the first database are rejected.

  Quite often, due to various interruptions during long observations, the LTA database was split into several files. These individual files were concatenated into one using `ltacat` (this was necessary for data browsing and detection of bad data as discussed in Chapter 4).

- The program `ltacleanup` checks a LTA database for conformity with the LTA format and attempts to fix some of the inconsistencies. It can also be used as a filter to select scans and/or perform integration in time and/or frequency.

  Due to problems related to synchronization between the online array control and correlator control softwares, bad scans may be recorded in the LTA database (e.g.

[3]Developed by the National Radio Astronomy Observatory (NRAO), USA., primarily for calibration and analysis of the data from the Very Large Array (VLA) but now used for other telescopes as well.

[4]Developed by V.K.Kulkarni of NCRA(TIFR), Pune

scans with zero data records, scans with wrong values for the frequency/bandwidth settings, scans with wrong pointing centre, scans with un-usable data due to a variety of reasons, etc.). Such scans need to be removed to produce a database with only usable scans. `ltacleanup` was used to remove such scans as well as to correct some inconsistencies with respect to the LTA format.

- The program `fixit` was used to edit the values of keywords in the *global header* and/or the *scan headers.*

- The program `tmac` checks for time stamp related errors in the LTA database. Due to synchronization problems between the various network programs comprising the Data Acquisition System (DAS), situation sometimes arise where the time stamp of successive data records does not monotonically increase. `tmac` rejects all data records till it finds a valid time stamp. It was often used as a filter before `gl2fit` to eliminate any offending data records.

- The program `offstop` was written for off-line fringe stopping when on-line fringe stopping in the correlator was not implemented. It later evolved into a program to apply residual compensating delay due to frequency offsets or antenna position offsets. The frequency and/or antenna position offsets for which fringe-stopping should be carried out are the input parameters. This program was extensively used to test the derived antenna co-ordinate corrections and fixed delay corrections during baseline and delay calibration efforts.

- The program `calibbp` does primary phase and bandpass calibration. This was written to allow freedom to test various calibration schemes for the GMRT data. The antenna based complex gains are determined using the phase calibrator scans and antenna based band-passes are determined using the bandpass calibrator. Average gains can be found by integrating the calibrator visibility data. As an alternative, antenna based gains can be averaged to determine the average gains. The latter has the advantage that the automatic bad data detection algorithm is effective while determining the antenna gains.

- The program `lin` subtracts linear slopes across the band for each baseline. This was required for tracing the source of spectral features across the band at P-band (and possibly at other bands as well). The data was time and bandpass calibrated using `calibbp` and linear slopes removed from each baseline. This is the equivalent of the `UVLIN` (Cornwell et al. 1992) operation done for spectral line data analysis. The spectral features seen across the band after passing it though the program `lin`, probably limits the detection of weak spectral lines, at least at P-band, and this is still an unsolved problem.