Volume 3 - Telescope Specific Processing

AIPS++ Group, eds

August 1, 2006

Contents

Current Status

- Early 2002: Initial *Glish*-based document created by George Moellenbrock.
- June 2002: Modified by Debra Shepherd (DSS) for the VLA summer school to include more discussion and some GUI/AIPS++ startup introduction.
- June 2002: Reviewed and modified by George Moellenbrock. Included in the August release of AIPS++
- Sept 2002: DSS added GUI instruction up to page 41. GUI instructions are first, *Glish* instructions come second. Sent document out for review to the NAUG and to Chris Depree (Users committee member).
- Nov 2002: DSS Received comments from Chris Depree and NAUG on restructuring Introduction. GUI approach approved. DSS restructured whole document, it is now a stand-alone document with chapter headings. Organizational changes approved by Joe Mcmullin. Added GUI instruction through chapter 3, imaging.
- Dec 2002: DSS added GUI instruction for Chapter 4: Analyzing Images. Need to do final check of whole thing before it can be installed on the web.
- Jul 2003: DSS deleted most GUI instruction for most of the document in preparation for new AIPS++ framework based on ACS with python command language and JAVA GUIs.

Chapter 1

VLA reduction

Contributors: Athol Kemball, George Moellenbrock, Debra Shepherd, & the NRAO AIPS++ Users Group
Version: July 22, 2003

This document describes how to calibrate and image VLA data using the Astronomical Information Processing System (AIPS++). AIPS++ is a suite of astronomical data reduction tools that can be run via an interactive command language called *Glish*.

The development of mouse-based Graphical User Interfaces (GUIs) have been suspended for the time being as we await the development of a new AIPS++ framework using the ALMA Common Software (ACS). The plan is to convert the command language to Python (although *Glish* will still be available) and GUIs will be developed using JAVA.

Besides this cookbook, you will also need to access AIPS++ documentation on the web at: http://aips2.aoc.nrao.edu/stable/docs/user/documentation.html This will tell you what input parameters are available for each function as well as provide guidelines for procedures not covered in this cookbook. AIPS++ documentation consists of:

- *Getting Started* with AIPS++ – an introduction to the AIPS++ environment.
- *Getting Results* with AIPS++ – three volumes are available on:
  - *Basic Tools* (e.g. plotting, data display, image analysis).
  - *Generic Processing* (e.g. data calibration & imaging)
  - *Telescope Specific Processing* (e.g. The Very Large Array, the Green Bank Telescope, and other facilities).
- *User Reference Manual* – detailed software documentation.

If you have experience with AIPS or MIRIAD software packages to reduce interferometric data, you may be interested in AIPS-aips++ and MIRIAD-aips++ dictionaries which provide a partial mapping of common AIPS and MIRIAD tasks/adverbs to their counterparts in AIPS++. These dictionaries are also given in Appendicies 1 and 2.

1.1 Basic initialization

This section assumes that AIPS++ has been installed on your UNIX or LINUX system. First, create an .aipsrc file in your home directory which will set some defaults and optimize run time. See Getting Started for a description of how to set up your .aipsrc file. An example .aipsrc file is given in Appendix 3.

To start AIPS++, cd to the directory you want to work in and type in an xterm window:

```
source /aips++/stable/aipsinit.csh    # source the stable version of
                                      #   the aipsinit.csh file that
                                      #   specifies setup parameters
aips++                                # start aips++
```

Check with your system administrator if you cannot locate the aipsinit.csh file.

You should get a *Glish* command prompt in the xterm window where you started AIPS++, a `Tool Manager` GUI window, and a `Logger` GUI window (Fig. 1.1). AIPS++ will take about a minute to initialize at startup. It is active when you get a "-" prompt at the command line interface.

Figure 1.1: Tool manager and logger GUI windows – displayed when AIPS++ is initialized.

The `Tool Manager` GUI has several possible views which are accessed by the buttons near the top. When you launch AIPS++, there are two buttons: one called **Manager** which is activated, and a **Tools in use** button. To change between different views, point the mouse cursor to the appropriate button and click on it with the left mouse button.

How do I exit from AIPS++?

There are two ways to exit AIPS++: From the `Tool Manager` GUI, select

`file` → **Exit AIPS++**

Or simply type at the *Glish* command line interface:

```
exit
```

and press return.

1.2 Tools in AIPS++

AIPS++ is a tool-based reduction package where related functions are gathered into 'tools'. The user creates a working version of a particular tool by logically associating it with a dataset and giving the tool (data + functionality) a name. The different capabilities of this tool are then accessed by running one or more functions within that tool. For example, to plot calibration solutions, you would first create a calibration tool and run the associated plotcal function. This involves the following steps:

- Create a `calibrater` tool attached to your dataset. Specify the TOOLNAME to be *cal* & FILENAME to be *mydata.ms*,
- Select the function `plotcal`, choose PLOTTYPE to be *AMP* and TABLENAME to be *cal.solutions* (e.g. the table with calibration solutions). Leave remaining parameters to their defaults to produce a plot of calibration solutions versus time;
- When you are finished with the tool, remove the tool and detach it from the dataset, mydata.ms.

To execute this sequence, type at the *Glish* command line prompt:

```
include 'calibrater.g';                  # Include the calibrater tool
cal:=calibrater(data='mydata.ms');       # Create a calibrater tool
                                         #   for the data 'mydata.ms'
                                         #   and call it 'cal'
cal.plotcal(plottype='amp',              # Execute the function plotcal
            tablename='cal.solutions');  #   specifying parameters
                                         #   plottype & tablename
cal.done();                              # End use of tool
```

All tools have a `Done` function which undoes the explicit association of the tool and data.

To see what parameters are associated with the plotcal function (or any function or tool in AIPS++ go to: http://aips2.aoc.nrao.edu/stable/docs/user/documentation.html The User Reference Manual gives a listing of each parameter with some examples. Detailed help on Synthesis calibration and imaging can be found in the "Generic Processing" chapter. The "Basic Tools" chapter gives details on Plotting, image display and analysis.

When working from the *Glish* command line in AIPS++, basic tools for VLA data reduction and imaging must be initialized first. Type at the *Glish* command prompt:

```
include 'vlafiller.g';
include 'ms.g';
include 'flagger.g';
include 'imager.g';
include 'msplot.g';
include 'calibrater.g';
include 'image.g';
include 'imagepol.g';
include 'viewer.g';
```

A note about *Glish* syntax: Notice the semi-colon at the end of each *Glish* line? This is an explicit end-of-line indicator that tells *Glish* that you are finished with the inputs. Although it is not necessary, it can be helpful. For instance, say you are typing in a long series of inputs for a tool, you have made multiple carriage returns, you think you are done but *Glish* is just sitting there waiting for more inputs. What goes? Well, you probably made a typo somewhere along the way. So, just type a semi-colon at the command line prompt to force an end-of-line. You will then get an error message telling you that the syntax is wrong. Fix the syntax error, re-submit the command and AIPS++ will start execution.

It is also convenient to start up a viewer tool to view and analyze images easily. To start the default viewer type:

```
include 'viewer.g';       # Include the viewer tool
dv.gui();                 # Start up the default viewer (dv) GUI
```

You will get two popup GUIs displayed, the viewer **Display Panel** and the viewer **Data Manager** panel that you can use to select and view any AIPS++ or FITS image (Fig. 1.2). You can put these off to the side until you are ready to use them or left-click on their **Done** buttons to close the tools. Note: once the aipsinit.csh file has been sourced, you can start the viewer directly from the UNIX prompt by typing `viewer`.

1.2.1 How to find a tool

To find the location of a particular tool, go to the `Tool Manager` GUI, type in the name of the desired tool in the **Search string** space and hit return. Left-click with your mouse on the **Show search Results** button. A list of possible locations for the tool will appear in a popup section. The general syntax will be:

package.module.tool *or*
package.module.tool.function *or*
package.module.tool.constructor *or*
package.module.global function

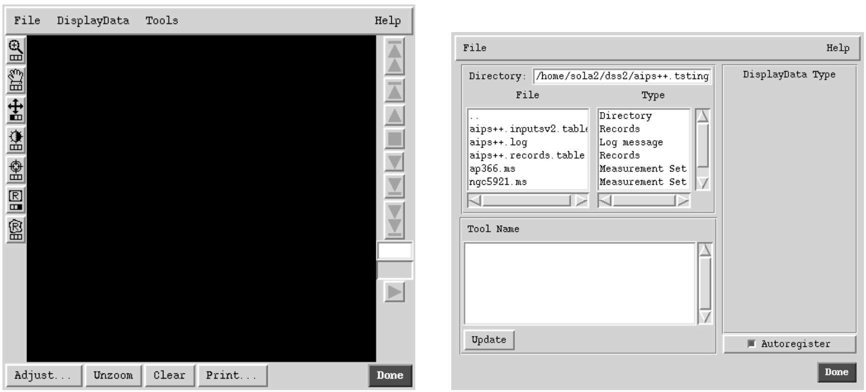

Figure 1.2: AIPS++viewer and data manager GUI for displaying images.

When you are done, dismiss the list. As an example: Type 'advise' in the **Search string** entry. One of the strings you will see when you left-click on the **Show search Results** button is: `synthesis.imager.imager.advise` which implies that '`advise`' is a function of the tool `imager` which is contained in the module `imager` of the package called `synthesis`.

1.2.2 Tools used in this document

The data reduction categories covered in this document are those for continuum data, continuum polarimetry and spectral-line data. Individual steps in each data reduction sequence are illustrated using *Glish* scripts. The reduction sequences given in this document use the following AIPS++ tools:

vlafiller is used to load VLA archive *uv* data into an AIPS++ format called a Measurement Set (MS). The MS is stored on disk as a directory with a "table" name specified by the user, e.g. 3C273.ms.

ms is used to access uv-data in a Measurement Set, and to list and summarize visibility data.

flagger is used to select and flag *uv*-data in a Measurement Set.

msplot is used for plotting and interactively editing visibility data. It is also useful to investigate and edit the data between calibration cycles.

calibrater is used to solve for uv-plane calibration effects, apply calibration corrections to the observed data, and plot calibration solutions.

imager is used to image (e.g. Fourier transform, deconvolve, restore) the data in the Measurement Set.

image is used for image analysis, e.g. fitting 1–D profiles or 2–D models to images, creating moment maps, regridding images.

viewer is used to display images and create publication quality postscript renderings of them. In addition, image tool statistics are available from within the viewer. The viewer is also used to create CLEAN deconvolution regions or masks.

1.3 What happens if something goes wrong?

First, check that your inputs are correct. You can get help from the Tool Manager window by left-clicking on the **Help** button and choosing the **Reference Manual** option. This will drive your browser to the AIPS++ detailed documentation on each tool. From this web page, you can also reach the main AIPS++ documentation site (left-click on Documentation in the upper left of the web page) or you can run a Search on key words you are interested in.

If your inputs are correct and you are still having problems, you may have discovered a software bug. If you find a bug during your data reduction session, type at the *Glish* command line prompt:

```
bug()
```

Or, using the Tool Manager **Help** button, choose **Report a bug** option. This will bring up a bug report GUI (Fig. 1.3) that you can fill in and submit automatically within AIPS++.

If something has gone wrong and you want to force an exit, type a 'Control-C' in the *Glish* window. *Glish* will ask you 'exit glish (y/n)?' Answer yes by typing 'y.' Save this technique for emergencies.

If you manage to find yourself in a hung state (don't worry, this is rare), you can always kill AIPS++ from a LINIX or UNIX command line with:

```
LINUX:   ps -aux | grep aips++
 UNIX:   ps -aef | grep aips++

Note the process ID numbers listed and kill them with the command:

         kill -9 process_id
```

1.4 MeasurementSets

Visibility data in AIPS++ are stored in an AIPS++ table known as a MeasurementSet (MS). An MS consists of a main table containing the visibility data and associated sub-tables containing auxiliary or secondary information. The full MS data format is described in AIPS++ Note 229. Tables 1 & 2 identify the commonly accessed components of the MAIN table and list data selection parameters used during typical data reduction sessions.

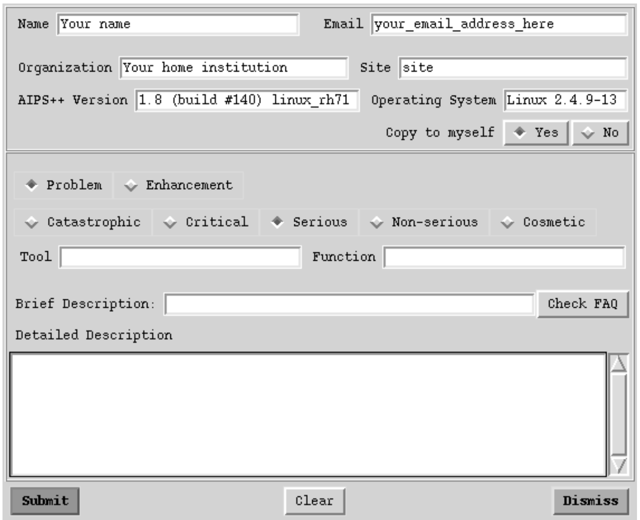

Figure 1.3: AIPS++ bug report GUI. Fill in the blanks and left-click on the **Submit** button. Your report will be sent to the AIPS++ project team and you will be sent e-mail giving you the status of this problem or enhancement request.

All AIPS++ data files, including MeasurementSets, are written into the current working directory by default, with each AIPS++ Table represented as a separate sub-directory. MS names therefore need only comply with UNIX file or directory naming conventions, and can be referred to from within AIPS++ directly, or via full pathnames.

Each row of the DATA column in the MS contains a matrix of observed complex visibilities at a single time stamp, for a single baseline in a single spectral window. The shape of the data matrix is given by the number of channels and the number of correlations (voltage-products) formed by the backend instrument of the telescope (e.g. a correlator for an array, or a spectrometer for a single dish).

1.5 Basic Calibration Fundamentals

Calibration and imaging responsibilities are divided between the calibrater and imager tools. The calibrater tool handles visibility-plane calibration while the imager tool deals with image formation, deconvolution and image-plane calibration.

In a typical synthesis observation, a number of calibrators of known structure (usually point sources) and at least one of known flux density are observed in addition to the scientific target(s) in order to obtain a measure of the systematic

Table 1.1: Commonly accessed MAIN Table components (Data, Flags, & Weights *only*)†

Name	Format††	Comments
DATA	Complex(N_c, N_f)	Complex visibility matrix Observed DATA for synthesis arrays
WEIGHT	Float(N_c)	Weight for whole data matrix
FLAG	Bool(N_c, N_f)	Cumulative data flags
CORRECTED_DATA	Complex(N_c, N_f)	Corrected data created by calibrater or imager tools
MODEL_DATA †††	Complex(N_c, N_f)	Model data created by calibrater or imager tools
IMAGING_WEIGHT	Float(N_c)	Imaging weight, e.g. uniform, natural, robust Created by calibrater or imager tools

† Additional data attributes are described in AIPS++ Note 229
†† N_c= number of correlators, N_f= number of frequency channels.
††† The MODEL_DATA column is created with initial values: amplitude = 1 & phase = 0°.

properties of the instrument and, if relevant, the corrupting media above it. The effects of these systematic properties are then removed from the data for the scientific target. This technique is often referred to as *cross-calibration* and its success depends implicitly on the relationship between: 1) the true brightness distribution of the observed radio sources (what we are after); 2) the formal response of the instrument and its variety of imperfections of varying importance (what we are calibrating); 3) and the data that is actually recorded (what we have measured).

The AIPS++ Measurement Equation, originally defined by Hamaker, Bregman, & Sault (1996a, 1996b), is an explicit description of this relationship. The Measurement Equation (in the visibility plane) for a spectral line polarization calibration is given by

$$\vec{V}_{ij} \; = \; B_{ij}\; G_{ij}\; D_{ij}\; P_{ij}\; T_{ij}\; \vec{V}_{ij}^{\,\mathrm{IDEAL}} \tag{1.1}$$

where:

$\vec{V}_{ij}$ = cross-correlations between two polarizations for each of two feeds (i, j) which characterize an individual baseline (e.g. the measured visibilities).

$\vec{V}_{ij}^{\,\mathrm{IDEAL}}$ = visibilities measured by an ideal interferometer (e.g. no instrumental errors or calibration effects). $\vec{V}_{ij}^{\,\mathrm{IDEAL}}$ = is directly related to the Fourier Transform of the true polarized sky brightness, $\vec{I} = (I, Q, U, V)$.

T_{ij} = Complex gain effects which are polarization-independent, e.g. tropospheric effects, high-frequency opacity corrections, antenna gain as a function of elevation, baseline corrections.

P_{ij} = Parallactic angle.

Table 1.2: Common Data Selection Parameters†

Parameter	Contents
ANTENNA1	First antenna in baseline
ANTENNA2	Second antenna in baseline
FIELD_ID	Field (source no.) identification
SPECTRAL_WINDOW_ID	Spectral window number (IF no.)
ARRAY_ID	Subarray number
OBSERVATION_ID	Observation identification
POLARIZATION_ID	Polarization identification
SCAN_NUMBER	Scan number
TIME	Integration midpoint time
UVW	UVW coordinates

† Selection of data from an MS currently appears in different guises in the various synthesis tools due to independent development of these tools. An effort to reconcile these differences is underway.

D_{ij} = Instrumental polarization response. "D-terms" describe the polarization leakage between feeds (e.g. how much the R-polarized feed picked up L-polarized emission, and vice versa).

G_{ij} = Electronic gain response due to components in the signal path between the feed and the correlator. This complex gain term G_{ij} includes the scale factor for absolute flux density calibration, and may include phase and amplitude corrections due to changes in the atmosphere (in leiu of T_{ij}).

B_{ij} = Bandpass response.

Note that the Measurement Equation is a matrix equation. $\vec{V}_{ij}$ and $\vec{V}_{ij}^{\mathrm{IDEAL}}$ are 4-vectors with elements for each correlator product, and the T_{ij}, etc., are 4×4 matrices that 'corrupt' the correlator products according to the properties of the specific term. The order of the corrupting terms from right-to-left, for the most part, is the order in which the errors affect the incoming wavefront. AIPS++ hardwires the Measurement Equation order and all of the specific algebra associated with the individual terms. As such, it is usually possible to ignore the fact that the Measurement Equation is a matrix equation, and treat the different terms as labeled 'components' (black boxes) along the signal path.

In practical calibration, it is often best to begin the calibration process by determining solutions for those terms which affect the data most. Thus, the user would normally determine gain solutions (G) first (applying pre-computed parallactic angle corrections P if doing polarization calibration), then bandpass solutions (B) and/or polarization solutions (D). Errors which are polarization-independent (T) can be determined at any point in the calibration process. T has nearly identical form to G but it located at a different point in the Measurement Equation. A practical advantage to this factorization is the ability to store polarization-independent effects which are characterized by a certain timescale or a particular parametrization (e.g., tropospheric opacity) as a separate calibration factor, thus isolating them from electronic gain effects which may vary on a different timescale and/or with different systematics. Although most data reduction sequences will probably proceed in this order, the user has complete freedom and flexibility to determine the order in which calibration effects are solved for. Self-calibration, the process by which an improved model of the target source is used to obtain improved solutions for the calibration, is also not limited to just the electronic gain (G) corrections, as has been traditionally the case. Improved solutions for, say, the instrumental polarization (D) may be obtained by self-calibration as well. In general, decision-making along the calibration path is a process of determining which effects are dominating the errors in the data and deciding how best to converge upon the optimal set of calibration solutions (for all relevant components) and source models which best describe the observed data.

Once the calibration solutions have been determined, the AIPS++ software applies the solutions according to the order defined by the Measurement Equation. The individual calibration components represented in the Measurement Equation are described in detail in the Synthesis Calibration chapter of `Getting Results`. The generalization of the Measurement Equation to image plane effects is also described there.

The observed data, $\vec{V}_{ij}$, are stored in the DATA column in the MAIN table of the MS; these are the raw data as loaded by the filler tool or imported from a UVFITS file. Associated with the DATA column are related columns to hold the most recent version of the calibrated data (CORRECTED_DATA), and the latest visibility plane representation of the source or field model, $\vec{V}_{ij}^{IDEAL}$, (MODEL_DATA). The latter two columns are filled in by the calibrater and imager tools, respectively. The actual calibration information is stored in separate calibration tables. The observed DATA column does not change during reduction, but the related columns do. When plotting the data using a tool such as msplot, the data sub-types (`observed, corrected, and model`) can be selected individually.

1.6 Summary of Data Reduction Steps

Listed below are the basic steps for typical continuum and spectral line data reduction sessions which solve for and apply the standard complex gain (G), the polarization leakage (D), and the relative bandpass (B).

Since reduction of continuum, spectral-line, and polarimetry data in AIPS++ differ essentially only in the addition or omission of specific steps, and in the details of a few parameter settings, only one pass through the basic reduction path is described here. In the remainder of this document, annotated sequences of glish for both continuum polarimetry and spectral-line reduction are included in parallel. Some actions apply to only one type of reduction, but all methods are instructive for either type of observation.

The basic reduction sequence is:

1. Import the data. Data may be imported from either a VLA archive tape using the vlafiller tool, or from a UVFITS disk file (e.g., as written by AIPS) using the ms tool's ms.fitstoms tool constructor. The result is a Measurement Set table (the AIPS++ internal data format). The ms tool is used to obtain a summary of the dataset.

2. Examine and edit the data using the flagger tool for basic data editing, the autoflag tool for automatic data editing, and the msplot tool for interactive editing.

3. Set the initial calibration models. By default, all sources have unit flux density point source models. Use the Imager.setjy function to set (point-source) flux densities for the absolute flux calibrators.

4. Obtain the complex gain (G) calibration for calibrators using the calibrater tool. If planning to do instrumental polarization (D) calibration, application of parallactic angle (P) corrections is required at this point.

5. Obtain the bandpass calibration (B) (spectral-line observations only) for bandpass calibrator(s) using the calibrater tool. The G calibration obtained above (and P, if necessary) is applied on-the-fly to ensure that the B calibration is normalized.

6. Transfer the flux density scale derived from the absolute flux calibrator(s) using the calibrater tool. Only the G calibration solutions obtained for the amplitude calibrators are properly scaled in amplitude. The solutions for the other calibrators are rescaled with the calibrater.fluxscale function by demanding that the mean gain amplitude be the same for all calibrators.

7. Apply the derived calibration to the source data.

8. Obtain the instrumental polarization calibration (D) (polarization observations only). Use a full-polarization model for a calibrator (obtained by imaging[1], and set with the Imager.setjy function). All previous calibration factors are applied before the D-terms or "leakage terms" are determined.

9. Apply all calibrations to the target source data. The calibrater.correct function applies calibration to the observed data and writes the result in the CORRECTED_DATA column in the MeasurementSet.

10. Image the target source. The imager tool is used to form images from the calibrated data, deconvolve these images, and restore the residual errors to the model image.

11. As necessary, self-calibrate the target source data and reimage. New and improved calibration may be obtained by repeating the initial calibration step(s), this time using the model image obtained in the first round of imaging. Re-image.

[1] Other means are in the works, stay tuned!

12. Repeat the self-calibration as necessary. Self-calibration need not be limited to only the G calibration factor; any solvable calibration component (G, T, B, and/or D, in the present example) may be self-calibrated.

Keep in mind that these examples are basic ones. More sophisticated sequences are possible. For example, it may be desirable to obtain bandpass calibration from an optimum calibrator, and to apply this on-the-fly during full-out G calibration on a different calibrator. This operation can be performed in the G and B calibrations steps listed above for the bandpass calibrator only, then repeating the G calibration step for the other calibrators with the B calibration applied. In fact, any sequence of operations of this nature is possible; the fundamental requirement is that calibration components that significantly influence the one being solved for should be applied prior to the solution.

1.7 Example data and scripts

The two sample datasets used in the examples below are available from the AIPS++ Data Repository. The Data Repository is usually created in the directory /aips++/data/ when AIPS++ is installed on a UNIX or LINUX machine (check with your AIPS++ system administrator if the data are not here). You can create AIPS++ MeasurementSets from this data directly as described below using the ms tool.

The *Glish* sequences listed below describe calibration and imaging steps and are available as full end-to-end scripts in the AIPS++ Recipe Repository. Scripts in the Repository can be easily modified to reduce and image similar data.

The continuum polarimetry data set is a full-polarization, VLA A-configuration observation at 5 GHz (one spectral window) of the gravitational lens 0957+561. A nearby point-like calibrator (0917+624) has been observed alternately with the target source. 3C 286 is used for both flux density scaling and instrumental polarization calibration. The *Glish* script is available here: continuum polarimetry.

The spectral-line dataset is a short VLA D-configuration observation of HI (redshifted to 1413 MHz) in NGC5921. This dataset was observed in total intensity only (RR & LL), with 63 channels in a single spectral window. A nearby phase calibrator (1445+099) was observed, as well as 3C 286 for flux density calibration. The *Glish* script is available here: spectral line.

For clarity in distinguishing operations for each of the sample datasets, tools associated with the continuum polarimetry reduction have names ending in 'C' (e.g. `imgrC`, `calC`), and those associated with the spectral-line reduction have names ending in 'S' (e.g. `imgrS`, `calS`).

1.8 Filling VLA data into AIPS++

There are two paths for importing VLA data into AIPS++: directly from a VLA archive or distribution tape, or from a UVFITS disk file generated by the

AIPS task FITTP. Both methods are described here. The example data sets from the Data Repository are UVFITS files that have been created using the AIPS task FITTP.

1.8.1 Filling data from UVFITS format

Data which have been exported to a disk file from AIPS using the task FITTP can be read into AIPS++ using the general UVFITS filler ms.fitstoms, which itself creates an ms tool. The example data sets from the AIPS++ Data Repository are in this format:

Continuum polarimetry case

To fill the example continuum data into a MeasurementSet from a UVFITS file, type at the *Glish* prompt (or just cut & paste from a file):

```
msetC:=fitstoms(msfile='ap366.ms',
                fitsfile='/aips++/data/demo/AP366-09C1.fits');
                                            # Construct the ms tool for
                                            #  the Continuum data set located
                                            #  in the AIPS++ Data Repository
msetC.summary(verbose=T);                   # Write a summary to the logger window
msetC.done();                               # Finish ms tool
```

The detailed summary of the example data set is given in Appendix 5.

Spectral-line case

To fill the example spectral line data into a MeasurementSet from a UVFITS file, type at the *Glish* prompt (or just cut & paste from a file):

```
msetS:=fitstoms(msfile='ngc5921.ms',
                fitsfile='/aips++/data/demo/NGC5921.fits');
                                            # Construct the ms tool for
                                            #  the spectral line data set located
                                            #  in the AIPS++ Data Repository
msetS.summary(verbose=T);                   # Obtain summary
msetS.done();                               # Finish ms tool
```

The detailed summary of the example data set is given in Appendix 5.

1.8.2 Filling data from VLA archive format

VLA data in on-line archive format are read into AIPS++ from disk or tape using the vlafiller tool. Tape drives can be located on the local system or on a remote system accessible on the network. Use of a remote tape drive requires knowledge of the network host name (e.g. `gibbon.cv.nrao.edu`, or `gibbon`), and the tape device name (e.g. `/dev/rmt/0ln`). This can often be determined from the physical drive itself, or by logging on to the remote host and using a

UNIX utility such as `sysinfo` or the UNIX command `ls -lut /dev/r[ms]t*` to list the device name.

Remote tapes need not be explicitly mounted from AIPS++ as the vlafiller itself will be run on the remote system directly. If you encounter problems in using the filler on a remote host, make sure that the AIPS++ initialization command `source /aips++/release/aipsinit.csh` is included in your `.cshrc` file (or the `sh` equivalent).

The vlafiller tool allows selection on frequency band, calibrator name, frequency range, project, time, subarray, and source, and supports concatenation to an existing *uv*-data set. To run the vlafiller using a local tape drive, you can use the vlafillerfromtape global function, e.g.:

```
vlafillerfromtape(msname='cals00feb.ms',        # Load project TESTT from tape to
                  device='/dev/rmt/0ln',        # MeasurementSet cals00feb.ms
                  files=[2,3],                  # Read only files 2 and 3 on tape
                  project='TESTT');             # in current directory
```

Note: the first file on a VLA archive tape is always a house keeping file. Thus, if you try to read it, the AIPS++ filler will report that it is not in the appropriate format.

To fill data into AIPS++ using a remote tape drive, set the host name, e.g.:

```
vlafillerfromtape(msname='/home/cals00feb.ms',   # Load project TESTT from remote
                  device='/dev/rmt/0ln',         # tape on host 'gibbon' to MS
                  files=[2,3],                   # Read files 2 and 3 on tape
                  project='TESTT',               # cals00feb.ms in /home directory
                  host='gibbon');                # on remote host
```

Note that it is necessary to specify a full path name (recognizable by the remote host) for the output file when filling from a remote tape drive.

Alternatively, you can use the vlafiller tool (rather than the global function) to set up the filler inputs separately and then run the filler from the *Glish* command line:

```
include 'vlafiller.g';
vfill:=vlafiller();
vfill.tapeinput(device="/dev/rmt/0ln" , files=[2,3] );
vfill.output(msname='cals00feb.ms', overwrite=F);
vfill.selectproject(project='TESTT');
vfill.fill(verbose=F, async=F);
vfill.done();
```

To obtain a summary of the dataset's properties after filling, construct an ms tool, and run the ms.summary function:

```
mset:=ms(filename='cals00feb.ms');       # Construct the ms tool for the data
mset.summary(verbose=T);                 # Write a summary to the logger window
mset.done();                             # Finish ms tool
```

Copy the summary information into a file for later use (source or FIELD_ID numbers and the times different sources are observed are particularly useful).

1.9 Data examination and inspection

Once the data have been filled to a MeasurementSet, they can be examined or displayed using various tools. The Catalog tool provides an overview of AIPS++ data files at the highest level, as individual AIPS++ tables of different types (e.g. images, MS, etc) in a given directory. To bring up the Catalog tool (Fig. 1.4) using Glish type:

```
include 'catalog.g'
dc.gui()                    # dc is the default catalog tool.
```

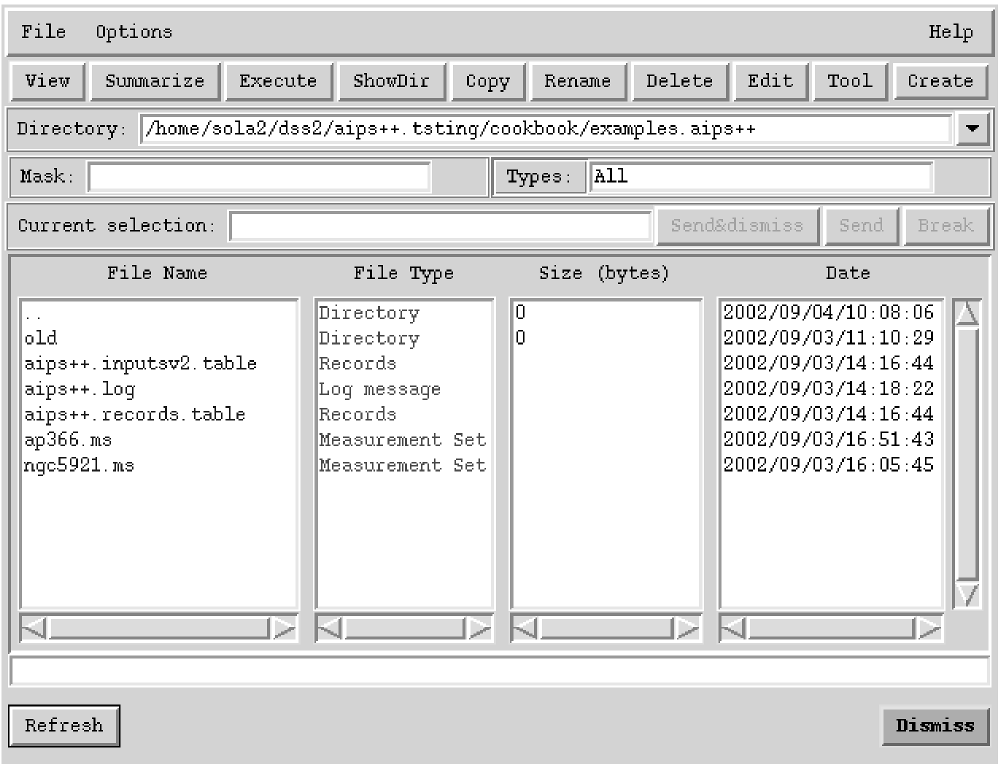

Figure 1.4: The Catalog GUI showing example MeasurementSets that have been created with the vlafiller tool.

When you click on the View button (to view the presently selected catalog item) you will launch the tablebrowser which provides facilities to browse and examine all data fields in the MS main and sub-tables.

Note that indices in the MS, such as FIELD_ID, are stored as integers beginning with zero when viewed at the low level provided by the tablebrowser. All

AIPS++ synthesis tools correct for this artifact of C++ and present selection of indices beginning at one.

The primary graphical visibility data visualization tool in AIPS++ at present is msplot. `msplot` allows line and raster plots of the observed, corrected, and model data against a range of MS indices and parameters, as well as associated plots such as uv-coverage. Facilities are provided to support versatile data selection, range setting and plot iteration. Interactive editing is also supported, as described in the next section.

1.9.1 Continuum polarimetry case

To start msplot from the *Glish* command line interface type:

```
include 'msplot.g'
plot := msplot(msfile='ap366.ms',
           edit=T);                 # Allow editing as well as inspection
```

The msplot control GUI will come up (Fig. 1.5). Left-click on the `Data selection` button to get data selection criteria. Type a field number in the FIELD_ID input to see a particular source. Then, in the top left corner of the msplot control GUI, select "Plot X vs Y" – the default is to plot *uv* distance vs. observed amplitude. To plot something different, click on the `X axis` or `Y axis` buttons to select and scroll your cursor to the option you want. Once you have selected the subset of the data you want to plot and how you want to plot it, left-click on **Go** (upper right on the `msplot` interface) to create the plot. In the plot window (right panel of Fig. 1.5), you can use the cursor to select regions and then left-click on the **Locate** button to identify baselines associated with each *uv* point. Left-lick on the **Flag** or **Unflag** buttons on the plot to flag/unflag selected data.

While you are in the `msplot` function, you can get a plot of antenna locations in the array. To do this, Left-click on the `Data selection` button. To the right of the ANTENNA selection space is a "wrench" icon, left-click on the wrench and choose the FROM MS option. A plot of antenna locations (Fig. 1.6) will pop up in a separate window. You can use this plot to select antennas in the array to plot (e.g. use your left-mouse button to draw one or more boxes around antennas to select them, then click on the **Accept** button to write those antenna names to the `msplot` data selection window for plotting). You can print this plot to hardcopy by left-clicking on the **File** button and choosing the print option. This plot is useful to choose a well-behaved antenna near the center of the array for your reference antenna. For the continuum polarimetry dataset, Antennas 2, 4, 8, & 9 are all good choices. When you are finished with the plot, left-click on the **Dismiss** button in the lower right corner of the antenna plot. *Note: you can also identify antennas near the center of the array using the ms.summary listing of antenna locations (Appendix 4). For example: Antenna 4, located at Station VLA:W8, is only 8 meters from the array center on the West arm.*

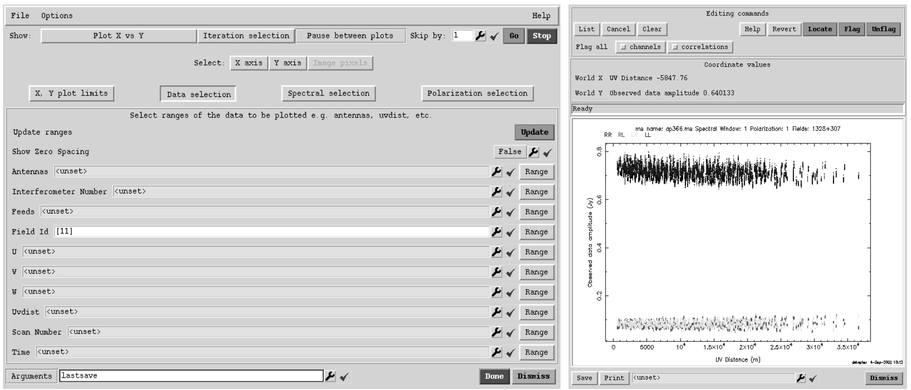

Figure 1.5: Visualizing your data. **Left:** The `msplot` control GUI. Left-click on the **Data selection** button and fill in the FIELD_ID blank (here source number 11, 1328+307, is chosen). **Right:** A plot of *uv* distance vs. observed amplitude created when you left-click on the **Go** button in the `msplot` control GUI. RR and LL polarizations have an amplitude of about 0.75 Jy while RL and LR are near 0.1 Jy.

When you are finished with the `msplot` tool, click on the **Done** button in the lower right corner of the `msplot` GUI.

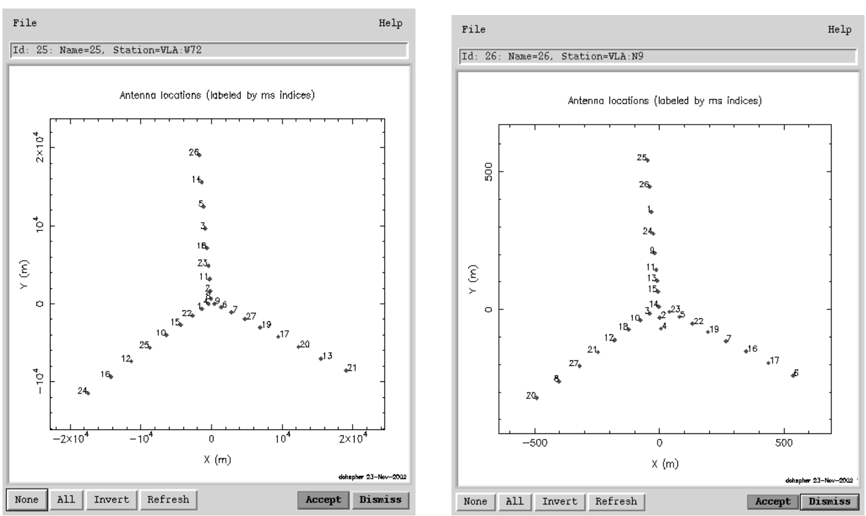

Figure 1.6: Antenna plots as a function of position in the array. **Left:** for the continuum polarimetry data. **Right:** for the spectral line data.

1.9.2 Spectral-line case

To start msplot from the *Glish* command line interface:

```
include 'msplot.g'
plot := msplot(msfile='ngc5921.ms',
            edit=T);                # Allow editing as well as inspection
```

(See Fig. 1.5.) Obtain an antenna plot as described in the previous section (Fig. 1.6; Antennas 2, 3, 14, or 23 are all good choices for references antennas because they are centrally located in the array and are well-behaved. Click on the **Done** button in the `msplot` GUI when you are finished with the tool.

1.10 Data editing

1.10.1 Interactive editing

Interactive editing is possible on almost all graphical views of the data when msplot is created with the option EDIT=*True*. While the data is being displayed and edited, flag commands are saved in a file in your current directory. The flags are not actually applied to the data until you exit msplot. When you click on the **Done** button in msplot, you will be asked if you want to apply the flags or not: click on **Yes** if you want the flags applied.

To flag data, use the `msplot` control GUI to select all or a subset of the data (e.g. select all data for a single source, and plot amplitude versus time). In the plot window, draw boxes with the left mouse button to select regions, e.g.:

1) Move the cursor to a corner where you want to box emission;
2) Click on the left mouse button and hold it down while you
drag the mouse to the opposite side of the region;
3) Release the mouse button when you have the region the size you want
it.
4) Hit the "Escape" key on your keyboard to remove a region.

Click on the **Locate** button to identify the antennas or time range of the selected data (results will be displayed in the `logger` window). Note that the listing may be very long if the region selected covers a large number of visibility points. Click on the **Flag** button to write flags to a file. Multiple flag regions may be selected before executing the **Flag** option.

Continuum polarimetry case

Examine the data in ap366.ms using `msplot`. First, plot calibrator data in FIELD_ID 11 (1328+307 = 3c286 - the flux calibrator). In the `msplot` control panel (see e.g. Fig. 1.5):

Choose the plot type PLOT X VS Y.
Select X AXIS=*uvdist*.
Select Y AXIS=*amplitude*.
Click on the **Go** button to make the plot in Fig. 1.7.

Use the **locate** button to identify low points.
Observed data points between 0.7 and 0.8 Jy are the RR & LL correlations (corresponding to the actual amplitudes of the *uv* data). Plot data in an XY plot with time vs. amplitude next. The lower points near 0.1 Jy are the RL & LR correlations. Repeat this procedure for FIELD_ID 9 (0917+624 - the gain calibrator). The calibrator data look very good - there are no major outliers so you do not have to edit this data before calibration. It is best not to edit or flag

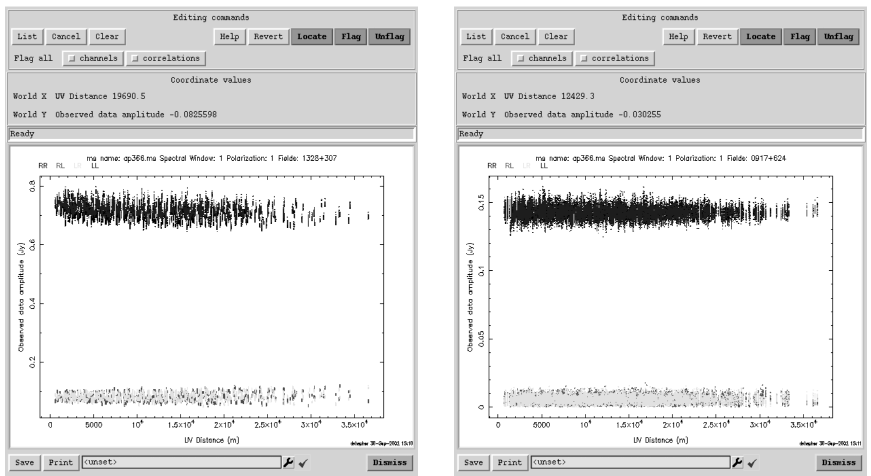

Figure 1.7: `msplot` of the calibrators in the VLA continuum dataset example. **Left:** 3c286, the flux calibrator. **Right:** 0917+624 - the gain calibrator.

---

source data until after it is calibrated.

Spectral-line case

Examine the data in ngc5921.ms using `msplot`. First, plot calibrator data in FIELD_ID 1 (1328+307 = 3c286 - the flux calibrator) In the `msplot` control panel:

Choose the plot type PLOT X VS Y.

Select X AXIS=*uvdist*.

Select Y AXIS=*amplitude*.

Limit the spectral selection to look at a single channel, say one in the middle: channel 30.

(click on the SPECTRAL SELECTION button in the `ms control panel` to choose channels).

Click on the **Go** button to make the plot in Fig. 1.8.

Use the **locate** button to identify low points.

The observed data plotted are the RR & LL correlation. Cross correlation (RL & LR) are not in this dataset. Plot data in an XY plot with time vs. amplitude next. Hey! The data look pretty good (Fig. 1.8)! No obvious flagging needed based on the `msplot` plots. It is noisier than the continuum data set example but that is because the RMS is higher because the band width of a single channel is relatively narrow. Now examine all the channels in a raster plot (Fig. 1.9):

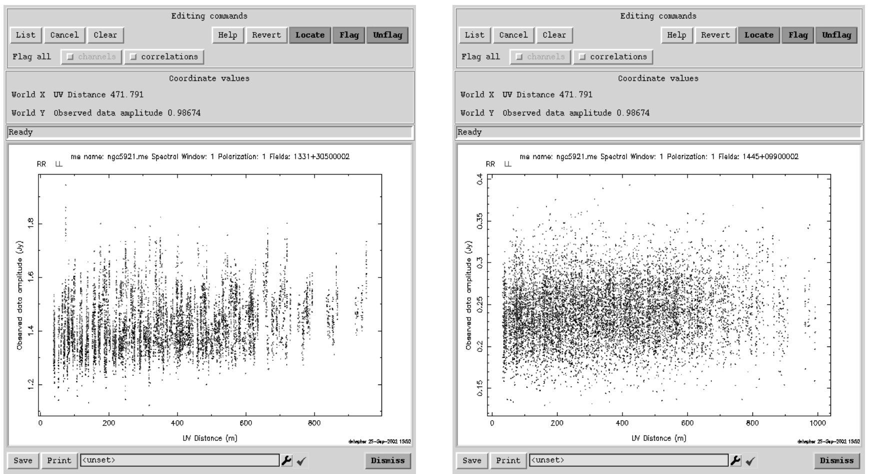

Figure 1.8: `msplot` of the calibrators in the VLA spectral-line dataset example. **Left:** 3c286 (1331+305), the flux calibrator. **Right:** 1445+099 - the gain calibrator.

Choose the plot type Display data as an image.

Choose all 63 channels starting with 1 and incrementing by 1 (click on the

spectral selection button in the `ms control panel` to see this). Use the buttons on the right of the raster display to see a movie of the channel data. The data look great here too but beware - the autocorrelations were filled with this data set. We will use the `flagger` tool to flag autocorrelations in the next section (or they will corrupt the final image). It is best not to edit or flag source data until after it is lcalibrated.

1.10.2 Non-interactive editing: the `flagger` tool

Command-based editing capabilities are supported in the flagger tool. `flagger` is useful because it can select data of a particular type, or with particular attributes and flag these data automatically. In general, this involves selecting the data to be flagged and then running a flagging operation. The flagging commands all have the option to run them on a trial basis.

Continuum polarimetry case

No flagging is necessary for the continuum data in ap366.ms.

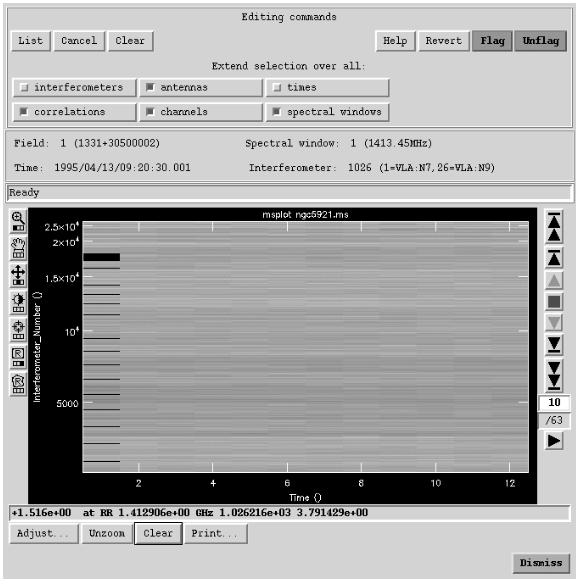

Figure 1.9: `msplot` of the calibrator 3c286 as a raster image. Note that antenna 17 is gone for the first integration.

Spectral-line case

We want to flag the autocorrelations in the MeasurementSet ngc5921.ms, type:

```
fg:=flagger(msfile="ngc5921.ms" );
fg.flagac();
fg.done();
```

The logger window will report:

```
Flagging 1619 rows
```

1.10.3 Useful flagging examples

Example 1:

To clip data in data.ms in fields 9 & 11, spectral window 1, for which the RR & LL data (pols 1 & 4) are outside of the amplitude range 5–10 Jy:

```
fg:= flagger(msfile='data.ms')                # Start flagger tool.
fg.setids(fieldid=[9, 11],                    # Select fields &
          spectralwindowid=1);                #  and spectral window.

fg.setpol(pol=[1, 4] );                       # Select RR & LL for testing.
fg.filter(column="DATA",                      # Select filter,
          operation="range",                  #  test for data outside of
```

```
          comparison="Amplitude",              #  amplitude range, run trial
          range="5.0Jy 10.0Jy",                #  to see how many data points
          fullpol=T,                           #  would be flagged.
          trial=T);
fg.filter(column="DATA",                       # Flag the data now.
          operation="range",
          comparison="Amplitude",
          range="5.0Jy 10.0Jy",
          fullpol=T,
          trial=F);
fg.done();                                     # Close flagger tool.
```

In this example, the `setids` and `setpol` functions select the data that will be tested. The `filter` tests to see whether the selected data (the observed DATA column) is within the specified range, if it is not, then the data is flagged. The range here is applicable to the RR & LL polarizations (pols 1 & 4). But, if the RR & LL polarizations are bad, it is a good idea to flag the cross-polarizations, RL & LR. This is done by setting FULLPOL=T to flag RL, LR data even though it wasn't tested by the `filter` function. The first call to the `filter` function was just a trial. The logger window will report how many data will be flagged, e.g.:

```
Filtering (in range mode) would have flagged 6220 additional data points
```

The second call to the `filter` function actually flags the data.

Example 2:

To flag data from antenna = 5, spectral window = 1, field_id = 2, within a specified timerange using the `Tool Manager` GUI, select:

```
fg.setantennas(ants=5)                       # Select data including antenna 5
fg.setids(spectralwindowid=1,fieldid=2)      # Select spectral id and field id
fg.settimerange(starttime="24-FEB-2000/10:00:05",  # Select timerange
                endtime="24-FEB-2000/10:09:37");
fg.state();                                  # Review data selection
fg.flag(trial=T);                            # Trial run to see how many rows
                                             #   will be flagged
fg.flag();                                   # Flag it
fg.reset();                                  # Reset the data selection
```

In this example, the `state` function reviews the state of the `fg` tool to make sure all the inputs are the way you want them. The state of the tool will be reported in the rolldown `Tool status` bar (see Fig. **??**) Running the `fg.flag` function with TRIAL=T produces a message in the logger window of how many rows of data will be flagged with this operation. Setting TRIAL=F actually flags the selected data. Finally, the `fg.reset` function resets the state of the `flagger` tool and you can start with a clean slate.

Example 3:

To flag data from baseline 3-4, spectral window 2, field_id 1, within ±10s of a timestamp:

```
fg.setbaselines(ants=[3,4])                  # Select data for baseline 3-4
fg.setids(spectralwindowid=2,fieldid=1)      # Select spectral id and field id
fg.settime(centertime="24-FEB-2000/10:00:05",  # Select time and range
           delta="10s");
fg.state();                                  # Review data selection
fg.flag();                                   # Flag it
```

In the above examples, time-related data selection options (set by flagger.settimerange and flagger.settime) and the two antenna-related data selection options (set by flagger.setantennas and flagger.setbaselines are logically OR'd. The net time selections, net antenna selections, and all other selections are then AND'd together. Note that the data selection persists after the flagging operation flagger.flag is executed. To clear the selection, execute the flagger.reset function, or execute individual selection commands without arguments to clear only that particular selection.

Example 4:

To flag all data on antenna 11 for the same spectral window, field, and timestamp as for baseline 3-4 in the previous example:

```
fg.setbaselines();               # Clear the baseline selection
fg.setantennas(ants=11);         # Select antenna 11
fg.state();                      # Review data selection
fg.flag();                       # Flag it
```

Example 5:

To restore or unflag data for baseline 11-15 (that was flagged above):

```
fg.setantennas();                # Clears antenna selection
fg.setbaselines(ants=[11,15]);   # Select baseline 11-15
fg.state();                      # Review data selection (note
                                 #  other selections still in
                                 #  effect)
fg.unflag();                     # undo previous flags
```

Note that use of flagger.unflag may restore data that had been flagged by other means (e.g., by the on-line system, or by other criteria in previous use of flagger or msplot).

Example 6:

One can also use the `flagger` tool to select and flag specific types of data. For example, it is usually desirable to flag the auto-correlation data from the VLA if they were filled with the cross-correlations (otherwise, the imager tool will attempt to use it in image formation as if it were single dish data). To do this:

```
fg.reset();                    # Reset the data selection
fg.flagac();                   # Flag the auto-correlations
```

Example 7:

It is also often the case that the VLA data near the beginning of scans is bad (e.g. after field or frequency band changes). To flag such data, use the flagger.quack function. For example, to remove 20 seconds of data at the beginning of scans (defined by gaps larger than 30 seconds) on field 3:

```
fg.reset();                            # Reset the data selection
fg.setids(fieldid=3);                  # Limit quack to fieldid=3 only
fg.quack(scaninterval='30s',           # Trial run of quack
         delta='20s',
         trial=T);
fg.quack(scaninterval='30s',           # Quack it
         delta='20s');
```

Where the first execution of the `quack` function was a trial run to see exactly how much data would be flagged. The second run with TRIAL=F actually 'quacked' the data.

Example 8:

To clip data in field 2 and spectral windows 1 & 2, outside of a specified amplitude range of 1.12–1.34 Jy you would do the following:

```
fg.setids(fieldid=2, spwid=[1,2]);    # Select field, spw
fg.setpol(pol=[1,4]);                  # Select polarizations 1,4
                                       #  (e.g., RR and LL)
fg.filter(column='DATA',               # Flag data if, DATA column
          operation='range',           #   Amplitudes are outside of
          comparison='Amplitude',      #   the range 1.12-1.34 Jy.
          range=['1.12Jy','1.34Jy'],
          fullpol=T,                   # Extend flags to pols that
          fullchan=T);                 #  weren't tested by the filter
                                       #  and flag all channels in a
                                       #  uv data point if any channels
                                       #  are outside the amp range.
fg.done();                             # Finish the flagger tool.
```

In this example, the `setids` and `setpol` functions select the data that will be tested. The `filter` tests to see whether the selected data (the observed DATA column) is within the specified range, if it is not, then the data is flagged. The range here is applicable to the RR & LL polarizations (pols 1 & 4). But, if the RR & LL polarizations are bad, it is a good idea to flag the cross-polarizations, RL & LR. This is done by setting FULLPOL=T to flag this data even though it wasn't tested by the `filter` function. The choice of FULLCHAN=T specifies that, if this is channel data and one channel fails the test, then all channels in that uv data point will be flagged.

1.11 Basic Gain Calibration

1.11.1 Setting the calibration source model

When solving for visibility-plane calibration components, the calibrater tool compares the observed DATA column with the MODEL_DATA column. The first time that an imager or calibrater tool is constructed for a given MS, the MODEL_DATA column is created and initialized with unit point source flux density visibilities (unpolarized) for all sources (e.g. AMP=1, phase=0°). The Imager.setjy function is then used to set the proper flux density for flux density calibrators. For sources that are recognized flux density calibrators (listed in Table 1.3), Imager.setjy will calculate the flux densities, Fourier transform the data and write the results to the MODEL_DATA column. For the VLA, the default source models are customarily point sources defined by the Baars or Perley-Taylor flux density scales, or point sources of unit flux density if the flux density is unknown. The MODEL_DATA column can also be filled with a model generated from an image of the source (e.g. the Fourier transform of an image generated after initial calibration of the data).

Table 1.3: Recognized Flux Density Calibrators

3C Name	*B1950 Name*	*J2000 Name*
3C 286	1328+307	1331+305
3C 48	0134+329	0137+331
3C 147	0538+498	0542+498
3C 138	0518+165	0521+166
−−	1934−638	−−
3C 295	1409+524	1411+522

Continuum polarimetry case

First, set the flux density for 3C 286 (FIELD_ID 11) according to Perley-Taylor (1999):

```
imgrC:=imager(filename='ap366.ms'); # Start the imager tool
imgrC.setjy(fieldid=11);            # Calculate & set flux density for
                                    #  1328+307 (3C 286) to 7.462Jy
```

After running `setjy` you will get a report in the logger giving the status of the task along with the end results:

```
  Starting imager::setjy
    1328+307  spwid=  1  [I=7.462, Q=0, U=0, V=0] Jy, (Perley-Taylor 99)
  Finished imager::setjy
```

Check that the flux is actually set by using the `imager.plotvis` function to see the value in the MODEL_DATA column. First select only the 3C 286 data using the `imager.setdata` function, then run plotvis to create Fig 1.10:

```
imgrC.setdata(fieldid=11);          # Setdata to 1328+307 (3C 286)
imgrC.plotvis(type='model');        # Plot model amplitudes vs uv distance
imgrC.done();                       # Close out the tool.
```

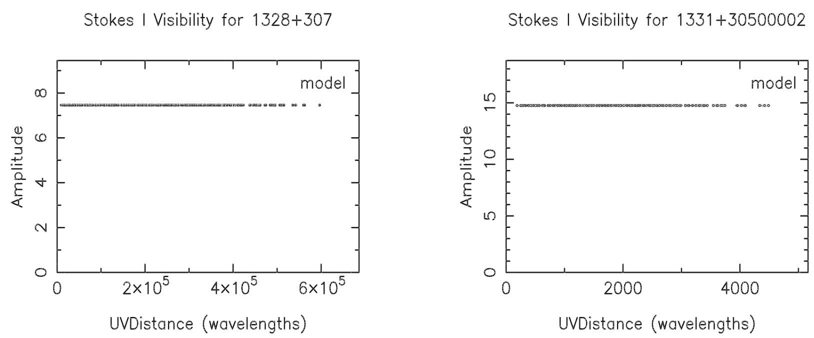

Figure 1.10: **Left:** Use the imager.plotvis function to create a plot like this to check the imager.setjy result. The amplitude of 1328+307 (3C 286) for the continuum polarimetry dataset is now 7.462 Jy at 4885 MHz. **Right:** Plot of 3C 286 amplitude of 14.8 Jy at 1413 MHz for the spectral line dataset.

Spectral-line case

Set the flux density of 3C 286 (field 1):

```
imgrS:=imager(filename='ngc5921.ms'); # Start the imager tool
imgrS.setjy(fieldid=1);               # Calculate & set flux density for
                                      #  1328+307 (3C 286) to 14.76 Jy
```

After `setjy` is run you will get a report in the logger giving the status of the task along with the end results:

```
  Starting imager::setjy
    1331+30500002  spwid=  1  [I=14.76, Q=0, U=0, V=0] Jy, (Perley-Taylor 99)
  Finished imager::setjy
```

Check that the flux is actually set by using `imgrS.plotvis` (Fig. 1.10):

```
imgrS.setdata(fieldid=1);             # Setdata to 1331+305 (3C 286)
imgrS.plotvis(type='model');          # Plot model amplitudes vs uv distance
imgrC.done();                         # Close out the tool.
```

1.11.2 Solving for complex gain, G

At this point, the source model and observed data exist for all calibrators at all uv-points, and it is possible to begin solving for the gains. It is almost invariably the case that the instrumental and atmospheric amplitude and phase gains (hereafter "complex gains") are the dominant calibration error in VLA data. Therefore we solve for this factor first.

For the standard cross-calibration scheme described here, only one term (G) will be used to describe the net complex gain (and so it will include atmospheric effects). One complex gain factor, as a function of time, will be determined for each field, spectral window, polarization, and antenna in the dataset.

Note: Well-behaved antennas that are located near the center of the array are chosen as reference antennas for each dataset. For non-polarization datasets, reference antennas need not be specified although you can if you want to. If no reference antenna is specified, an effective phase reference that is an average over the data will be calculated and used. For data that requires polarization calibration, you must choose a reference antenna that has a constant phase difference between the right and left polarizations (i.e., no phase jumps or drifts). If no reference antenna (or a poor one) is specified, the phase reference may have jumps in the R–L phase, and the resulting polarization position angle response will vary during the observation, thus corrupting the polarization imaging.

To solve for G, first choose a solution interval which samples the time scale over which you think the instrument or atmosphere is changing. Time scales depend on the array configuration and the observed frequency. Here, a time scale of 60 sec is chosen for the 6 cm continuum polarimetry data while scan-based solutions (one solution per integration) are chosen for the 21 cm spectral line data.

Continuum polarimetry case

To obtain gain calibration solutions for the continuum polarimetry data set, create the calibrator tool, select which data you want to find solutions for (e.g. the gain and flux calibrators in FIELD_IDs 9 & 11), set up to apply pre-computed polarization calibration, set up to solve for phase and amplitude solutions, and then solve for the calibration solutions. The actual solutions will be written to a file (here called ap366.gcal) that will be written to disk. The dataset itself will not be written to at this point. Thus:

```
calC:=calibrater(filename='ap366.ms');      # Create Calibrater tool

calC.setdata(msselect='FIELD_ID IN [9,11]');
                                            # Select calibrator sources
                                            #  in MS fields 9 & 11
calC.setapply(type='P',                     # Arrange to apply parallactic
              t=5.0);                       #  angle correction (for polarization
                                            #  only).  Choose a 5 sec timescale.
                                            #  The P-table corrections are
```

```
                                             #  computed in AIPS++.
calC.setsolve(type='G',                      # Arrange to solve for G
              t=60.0,                        #  on a 60.0 sec timescale
              refant=4,                      #  Choose reference antenna 4,
              table='ap366.gcal');           #  a well-behaved antenna
                                             #  near the center of the array.
calC.state();                                # Review setapply/setsolve settings
calC.solve();                                # Solve for the net complex gains
                                             #  and write solutions to the table
                                             #  ap366.gcal located on disk.
calC.plotcal(tablename='ap366.gcal');        # Inspect solutions
```

The state function provides the current state of the calibrator tool in a logger message:

```
  The following calibration components will be applied:
    P table=<pre-computed> t=5 select=[]
  The following calibration components will be solved for:
    G table=ap366.gcal t=60 preavg=0 phaseonly=F refant=4 append=F
```

While the `calibrater.solve` function is running, the logger will report the status of the convergence with messages like:

```
  G Jones Slot=1, 0917+624, spw=1: 22-May-1998/22:40:05 to 22-May-1998/22:41:00
  G Jones    Initial fit per unit weight = 0.989737 Jy, sum of weights = 6210
  G Jones    Final   fit per unit weight = 0.013267 Jy after 7 iterations
  G Jones Slot=10, 1328+307, spw=1: 23-May-1998/00:53:15 to 23-May-1998/00:53:55
  G Jones    Initial fit per unit weight = 7.41213 Jy, sum of weights = 4920
  G Jones    Final   fit per unit weight = 0.0821705 Jy after 7 iterations
```

Good solutions will have a final fit per unit weight that is a few percent of the initial fit per unit weight, as shown above. You can plot the solutions graphically using the `calibrater.plotcal` function, e.g. (Fig. 1.11):
If you want solutions plotted for each antenna separately, choose MULTIPLOT=*True*.

Note that all calibrater.setapply and calibrater.setsolve executions prior to the calibrater.solve execution will be in effect when obtaining the solutions. Thus, the parallactic angle correction, P, is applied before the G solutions are obtained. The parallactic angle correction is required for polarimetry reduction, and may be omitted otherwise. If data imported from AIPS already had the parallactic angle correction applied using the AIPS task CLCOR, then the calibrater.setapply step for P should be omitted (it is the user's responsibility to keep track of this).

If your calibration solutions do not converge well, try longer solution intervals. You may also have bad data remaining in the calibrators: take a closer look at your data and flag anything that looks suspect.

NOTE: To specify scan-based solutions (e.g. determine gain solutions for each scan) set t=0.0 in calC.setsolve.

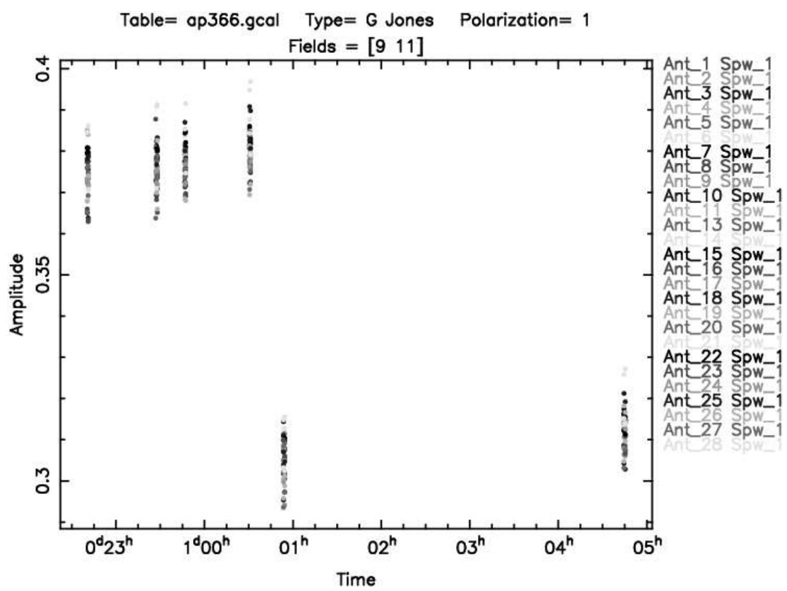

Figure 1.11: A plot of calibration solutions for the continuum data set as a function of time for all antennas (generated with the `calibrater.plotcal` function). Solutions for the gain calibrator, 0917+624, and 3C 286 are plotted.

Spectral-line case

To obtain gain calibration solutions for the spectral-line data set: select the data you want to find solutions for (the inner 55 channels, starting at channel 3, for the gain and flux calibrators in FIELD_IDs 1 & 2), set up to derive scan-based solutions for phase and amplitude, and then solve for the calibration solutions. Write the gain solutions to a file called ngc5921.gcal and plot the solutions:

```
calS:=calibrater(filename='ngc5921.ms'); # Create calibrater tool for
                                         #  spectral line data set
calS.setdata(msselect='FIELD_ID <= 2',   # Select data for calibrators
             mode='channel',             #  (Fields 1 & 2)
             start=3,                    #  and drop the outer channels
             nchan=55);                  #  that may bias the gain solution.
calS.setsolve(type='G',                  # Arrange to solve for G for
              t=0,                       #  each integration.
              refant=14,                 #  Choose reference antenna 14:
              table='ngc5921.gcal');     #  a well-behaved antenna
                                         #  near the center of the array.
calS.state();                            # Review setsolve settings
calS.solve();                            # Solve for the net complex gains
                                         #  and write solutions to the table
                                         #  ngc5921.gcal located on disk.
```

```
calS.plotcal(tablename='ngc5921.gcal');  # Inspect solutions
```

Note that we have omitted application of P since there is no polarization data. The state function provides the current state of the calibrator tool in a logger message:

```
  The following calibration components will be applied:
    None.
  The following calibration components will be solved for:
    G table=ngc5921.gcal t=0 preavg=0 phaseonly=F refant=14 append=F
```

While the `calibrater.solve` function is running, the logger will report the status of the convergence with messages like:

```
  G Jones Slot=1, 1331+30500002, spw=1: 13-Apr-1995/09:19:00 to 13-Apr-1995/09:24:30
  G Jones    Initial fit per unit weight = 1.94099 Jy, sum of weights = 6.32923e+06
  G Jones    Final   fit per unit weight = 0.0274396 Jy after 8 iterations
  G Jones Slot=2, 1445+09900002, spw=1: 13-Apr-1995/09:27:30 to 13-Apr-1995/09:29:30
  G Jones    Initial fit per unit weight = 0.324512 Jy, sum of weights = 2.65356e+06
  G Jones    Final   fit per unit weight = 0.00756852 Jy after 8 iterations
```

Good solutions will have a final fit per unit weight that is a few percent of the initial fit per unit weight, as shown above. You can plot the solutions graphically using the `calibrater.plotcal` function, e.g. (Fig. 1.12). If you want solutions plotted for each antenna separately, choose MULTIPLOT=*True*.

A Note on Calibrater Tool Operation

As illustrated above, the calibrater tool has three 'set' functions (calibrater.setdata, calibrater.setapply, and calibrater.setsolve) and two 'operational' functions (calibrater.solve and calibrater.correct). The calibrater.setdata function selects data and the calibrater.setapply and calibrater.setsolve functions set up the state of the calibrater tool. Both calibrater.setapply and calibrater.setsolve can be executed more than once (for different calibration components) prior to calibrater.solve or calibrater.correct (e.g., calibrater.setapply will typically be executed for both G and P before solving for D). Use the calibrater.state function to review the current state and the calibrater.reset function to clear all or part of the calibrater.setapply/calibrater.setsolve state.

The calibrater.solve function will apply the calibration components (specified with calibrater.setapply) to the *observed* DATA, and solve for the components specified with calibrater.setsolve. Note that it is possible to specify more than one component for solving. In this case, the solutions are done serially (not simultaneously) in the right-to-left order prescribed by the Measurement Equation formalism (e.g., G before D), with each newly-solved component applied before solving for the next (e.g., G solutions will be applied before solving for D).

The calibrater.correct function applies a set of calibrations to the observed data and saves it for imaging in the CORRECTED_DATA column of the MeasurementSet. Anything set by calibrater.setsolve is ignored by calibrater.correct.

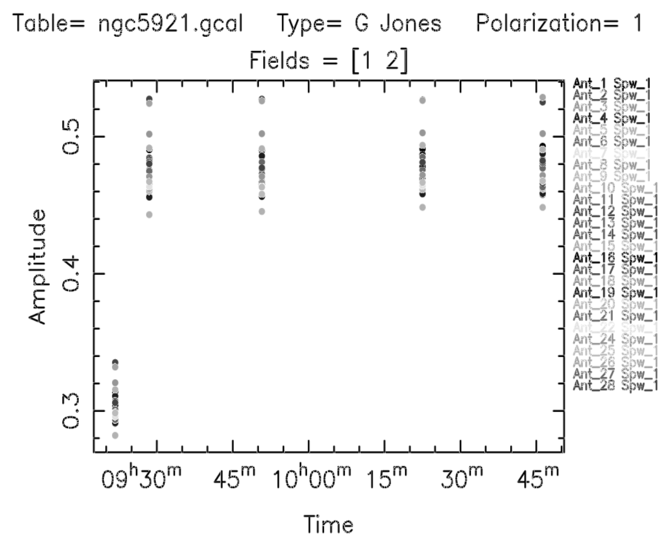

Figure 1.12: A plot of calibration solutions for the spectral-line data set as a function of time for all antennas (generated with the `calibrater.plotcal` function). The first set of solutions are for 3C 286, the last four are for the gain calibrator, 1445+099.

1.11.3 Bandpass calibration

In the case of spectral-line data, it is necessary to remove gain fluctuations that are a function of frequency. This is achieved by using the calibrater tool to solve for B calibration, in the present example, using 1331-305 as the bandpass calibrator. The G calibration obtained above is applied to ensure that the B calibration will be normalized. The solution interval is set very large so only one B solution (in time) is obtained.

Spectral-line case (only)

To obtain bandpass solutions for the spectral-line data set: select the data you want to find solutions for (the bandpass calibrator in FIELD_ID 1), reset the calibrator tool, set up to derive a single solution for phase and amplitude, and then solve. Write the bandpass solutions to a file called ngc5921.bcal and plot the solutions:

```
calS.setdata(msselect='FIELD_ID==1');      # Select bandpass calibrator
                                           #  (1331+305 = 3C 286)
calS.reset();                              # Reset apply/solve state
                                           #  of the calibrater tool
```

```
calS.setapply(type='G',                    # Arrange to apply G solutions
              t=0.0,                       #  from the table ngc5921.gcal
              table='ngc5921.gcal');
calS.setsolve(type='B',                    # Arrange to solve for a single
              t=86400.0,                   #  bandpass solution for the
              refant=14,                   #  entire observation.
              table='ngc5921.bcal');
calS.state();                              # Review setapply/setsolve settings
calS.solve();                              # Solve for the bandpass solutions
                                           #  and write them to the table
                                           #  ngc5921.bcal located on disk.
calS.plotcal(tablename='ngc5921.bcal',     # Inspect the solutions
             plottype='AMP',
             multiplot=F);
```

The state function provides the current state of the calibrator tool in a logger message:

```
  The following calibration components will be applied:
    G table=ngc5921.gcal t=0 select=[]
  The following calibration components will be solved for:
    B table=ngc5921.bcal t=86400 preavg=0 phaseonly=F refant=14 append=F
```

While the `calibrater.solve` function is running, the logger will report the status of the convergence:

```
  Initializing solvable bandpass terms (B-matrix)
  For interval of 86400 seconds, found 1 slots
  Applying G table from ngc5921.gcal
  Solving for B
  B Jones Slot=1, 1331+30500002, spw=1: 13-Apr-1995/09:19:00 to 13-Apr-1995/09:24:30
  B Jones    Initial fit per unit weight = 0.398437 Jy, sum of weights = 3.99722e+08
  B Jones    Final   fit per unit weight = 0.0425429 Jy after 6 iterations
  Storing B matrix in table ngc5921.bcal
```

As with the gain calibration, good solutions have a final fit per unit weight that is low relative to the initial fit per unit weight, as shown above. Plot the solutions graphically using the `calibrater.plotcal` function (Fig. 1.13):

`calS.plotcal`: PLOTTYPE=*AMP* TABLENAME=*ngc5921.bcal*, MULTIPLOT=*False*

→ **Go**

If you want solutions plotted for each antenna separately, choose MULTIPLOT=*True*.

1.11.4 Transfer the flux density scale

For point sources of unknown flux density, but for which valid solutions have been derived against a unit point source model, the absolute flux density can be established by scaling the mean gain amplitudes against those of the known amplitude calibrators. This scaling is achieved using the calibrater.fluxscale function.

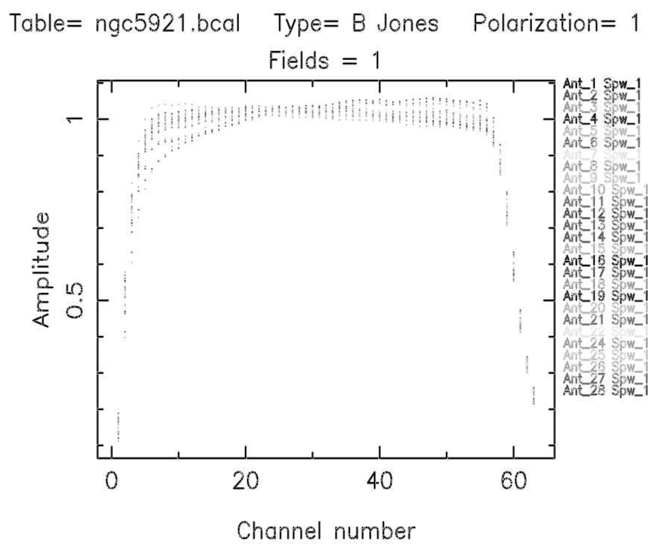

Figure 1.13: A plot of the bandpass solutions for the spectral-line data set for all antennas.

In both examples below, the amplitude scale is set with 3C286 as the reference flux density calibrator. The output calibration table contains the scaled G calibration factors for all of the transfer sources, and will be used in subsequent reduction steps. Proper scaling is indicated in the calibrater.plotcal plots by smoothly varying (ideally constant) amplitude gains for each antenna, *despite* source changes. At high frequencies ($\nu > 15$ GHz), this scaling step may be biased by elevation-dependent variation of the atmospheric contribution to the system noise temperature. Methods for treating this case properly will be available in the near future.

Continuum polarimetry case

In the `calibrater` tool, transfer the flux scale calculated for 3C 286 to the gain calibrator, 0917+624, and write the scaled solutions to a file on disk called ap366.fluxcal. Plot the solutions with the `plotcal` function (Fig. 1.14):

```
calC.fluxscale(tablein='ap366.gcal',      # Transfer the flux density scale
               tableout='ap366.fluxcal',  #  from 1328+307 (B1950: 3C286)
               reference='1328+307',      #  to the gain calibrator 0917+642.
               transfer='0917+624');      #  Write solutions to ap366.fluxcal.
calC.plotcal(tablename='ap366.fluxcal',  # Inspect the solutions
             plottype='AMP',
             multiplot=F);
```

The logger window will report the flux density of the gain calibrator:

```
Setting the flux density scale using reference calibrators
Flux density for 0917+624 (spw=1) is:     1.482 +/-  0.001 Jy
```

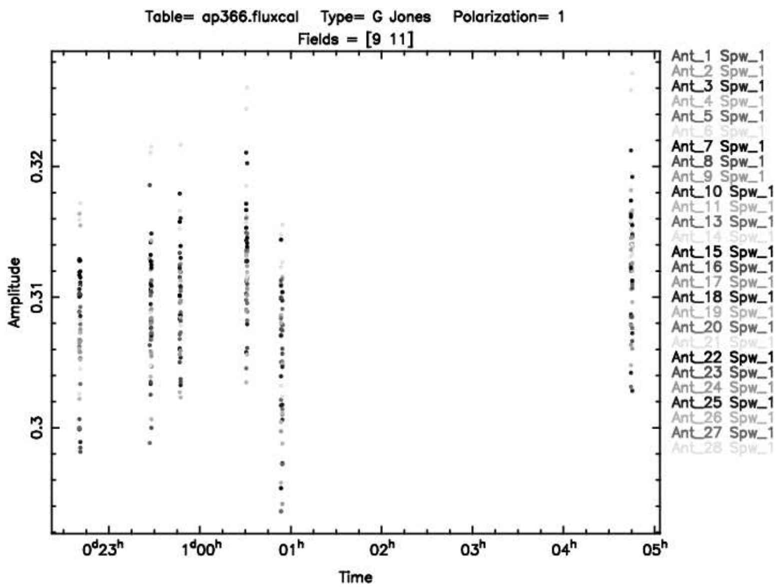

Figure 1.14: A plot of scaled gain solutions for the for the continuum data set as a function of time for all antennas.

Spectral-line case

In the `calibrater` tool, transfer the flux scale calculated for 3C 286 to the gain calibrator, 1445+099, and write the scaled solutions to a file on disk called ngc5921.fluxcal. Plot the solutions with the `plotcal` function (Fig. 1.15):

```
calS.fluxscale(tablein='ngc5921.gcal',          # Transfer the flux density scale
               tableout='ngc5921.fluxcal',      #  from 1328+307 (J2000: 3C286)
               reference='1331+30500002',       #  to the gain calibrator
               transfer=['1445+09900002']);     #  1445+09900002.  Write
                                                #  solutions to the table
                                                #  ngc5921.fluxcal located on disk.
calS.plotcal(tablename='ngc5921.fluxcal',       # Inspect the solutions
             plottype='AMP',
             multiplot=F);
```

The logger window will report the flux density of the gain calibrator:

```
Setting the flux density scale using reference calibrators
Flux density for 1445+09900002 (spw=1) is:     2.486 +/-  0.000 Jy
```

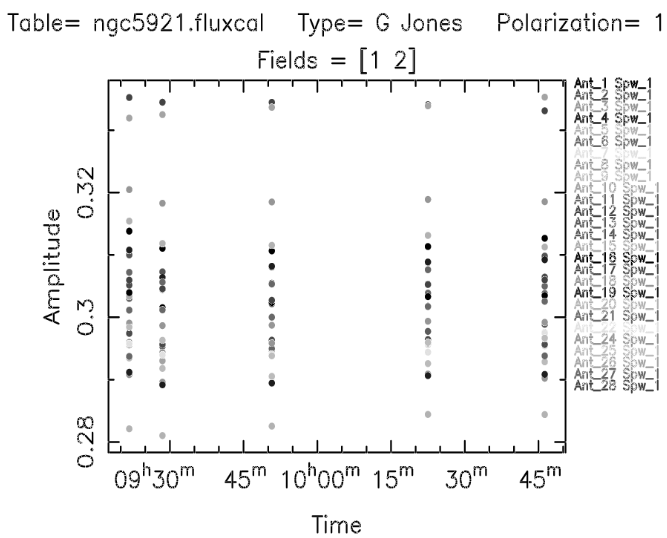

Figure 1.15: A plot of scaled gain solutions for the spectral-line data set as a function of time for all antennas. Amplitudes for all sources are now roughly the same, as expected for a good set of scaled gain solutions.

1.11.5 Correct the data

Before imaging, the data need to be corrected by applying the calibrations determined above. The calibrater.setdata function selects which data are to be corrected, and the calibrater.setapply function selects which solutions are to be applied. The calibrater.setapply function may be run several times (in any order) to specify simultaneous application of different calibration components. The components will be applied by the calibrater.correct function in the appropriate order according to the Measurement Equation. In AIPS++, calibration tables are always cumulative, and so are always applied to the observed data in the DATA column. The corrected data are written to the CORRECTED_DATA column in the MS, which will be used directly in imaging. Note: the observed DATA is *NOT* changed. If you want to change the solutions you have applied, you can create new calibration tables and correct the data again – the CORRECTED_DATA column will be over-written.

Continuum polarimetry case

In the `calibrater` tool, select all data: source, gain and flux calibrators (Fields 9, 10, & 11), reset the tool state, set up to apply parallactic angle and flux-scaled gain solutions, and then correct the observed data, writing the result to the CORRECTED_DATA column in the measurement set:

```
calC.setdata(msselect='FIELD_ID IN [9:11]');  # Select sources in MS fields
                                              #  9, 10 & 11 to which calibration
                                              #  is to be applied
calC.reset();                                 # Reset setapply/setsolve
calC.setapply(type='P',                       # Arrange to apply parallactic angle
              t=5.0);                         #  correction on 5 sec intervals
calC.setapply(type='G',                       # Arrange to apply flux-scaled G
              t=0.0,                          #  solutions from the ap366.fluxcal
              table='ap366.fluxcal');         #  table.
calC.state();                                 # Review setapply settings
calC.correct();                               # Apply solutions and write the
                                              #  CORRECTED_DATA column in the MS.
```

The `calibrator.state` and `calibrator.correct` functions will produce the logger messages:

```
  The following calibration components will be applied:
    G table=ap366.fluxcal t=0 select=[]
    P table=<pre-computed> t=5.0 select=[]
  The following calibration components will be solved for:
    None.
  Initializing non-solvable parallactic angle (P-matrix)
  For interval of 5 seconds, found 213 slots
  Applying G table from ap366.fluxcal
```

At this point, it is a good idea to examine the calibrated source data (CORRECTED_DATA column) with `msplot` and flag any bad data. The continuum data set doesn't have any bad source data so this step will be skipped.

Spectral-line case

In the `calibrater` tool, select source & gain calibrators (Fields 2 & 3), reset the tool state, set up to apply flux-scaled gain solutions derived from the gain calibrator (Field 2) along with the bandpass solutions, and then correct the observed data, writing the result to the CORRECTED_DATA column in the measurement set:

```
calS.setdata(msselect='FIELD_ID IN [2,3]');  # Select sources in MS fields
                                             #  2, & 3 to which calibration
                                             #  is to be applied
calS.reset();                                # Reset setapply/setsolve
calS.setapply(type='G',                      # Arrange to apply flux-scaled G
              t=0.0,                         #  solutions (from field 2) from
              table='ngc5921.fluxcal',       #  the ngc5921.fluxcal table.
              select='FIELD_ID==2');
calS.setapply(type='B',                      # Arrange to apply B solutions
              t=0.0,                         #  from the ngc5921.bcal table.
              table='ngc5921.bcal',
```

```
             select='');
calS.state();                              # Review setapply settings
calS.correct();                            # Apply solutions and write the
                                           #  CORRECTED_DATA column in the MS.
```

The `calibrator.state` and `calibrator.correct` functions will produce the logger messages:

```
  The following calibration components will be applied:
    B table=ngc5921.bcal t=0 select=[]
    G table=ngc5921.fluxcal t=0 select=[(FIELD_ID+1)==2]
  The following calibration components will be solved for:
    None.
  Applying B table from ngc5921.bcal
  Applying G table from ngc5921.fluxcal
```

At this point, it is a good idea to examine the calibrated source data (CORRECTED_DATA column) with `msplot` and flag any bad data. The spectral line data set doesn't have any bad source data so this step will be skipped.

Note: The flux calibrator (3C 286, FIELD_ID 1) could have been calibrated at the same time by selecting it in the initial `calS.setdata` and applying the G and B solutions to all the data (as in the previous example with the continuum data set). To calibrate the 3C 286 data separately, execute `calS.setdata` and `calS.setapply(type=G)` again with FIELD_ID==1 selected in both, followed by `calS.correct`.

1.11.6 Instrumental Polarization Calibration

For the continuum polarimetry data set, it is necessary to calibrate the instrumental polarization (D) before high-quality polarization images can be formed. Accurate D calibration depends on an appropriate polarization model for the calibrator. If the calibrator is unpolarized (Stokes Q = U $\sim$ 0), and point-like[2], the simple total intensity model (non-zero Stokes I only) used in the G calibration can be used as a polarization model. If this is the case, the D calibration can be obtained at the same time as G by issuing an additional `calC.setsolve` execution before the `calC.solve` is executed.

If the instrumental polarization calibrator is significantly polarized (as in the case of 3C 286) a non-zero model for Stokes Q and U must be determined. This model can be obtained by imaging the full polarization image prior to D calibration, as described below. In the near future, a function for simply summing the polarization visibilities will be available to provide an initial polarization model estimate.

For the time being, the polarization model of 3C 286 is obtained by imaging the corrected data after applying the G (and P) calibration. Since 3C 286 is virtually a point source at this frequency, the Stokes I, Q, and U values are obtained using the image.statistics function in the image tool.

[2]Usually, the instrumental polarization calibrator is one of the phase calibrators used for cross-calibration of G, which are ideally point sources for that purpose.

The calibrater tool operations used in solving for D are similar to those used to solve for G. However, note that one additional parameter—`preavg`—is used explicitly in the calibrater.setsolve execution. Since only one D solution is desired for the entire observation, the solution interval is set very long, and the data for each baseline within the solution interval would be averaged before the solution is determined. However, the parallactic angle varies on timescales much shorter than the solution interval[3]. Thus, the `preavg` parameter is set to prevent averaging of the data beyond an acceptable timescale for the parallactic angle.

Continuum polarimetry case (only)

First make IQUV images of the polarization calibrator 3C 286 to determine the polarization model (a more detailed description of imaging is given in the next section). First, start the imager tool and click on the **Function Group** button to the left of the GUI. Choose the option *ALL* function groups so you can see all possible functions available within the `imager` tool. Then select the 3C 286 data, define the image characteristics and weighting scheme. Finally, make a simple, un-CLEANED image called 3c286.cal1.im and write it to disk:

```
imgrC:=imager(filename='ap366.ms');          # Start the imager tool
imgrC.setdata(mode='none',                   # Select continuum data for
              fieldid=11);                   #  for 3C 286.
imgrC.setimage(nx=512,                       # Set up imaging parameters.
               ny=512,
               cellx='0.1arcsec',
               celly='0.1arcsec',
               stokes='IQUV',
               fieldid=11);
imgrC.weight(type='uniform');                # Choose weighting
                                             #  e.g. uniform, natural.
imgrC.makeimage(type='corrected',            # Make an image using the
                image='3c286cal1.im');       #  CORRECTED_DATA column
                                             #  in the MS.  Write the image
                                             #  3c286cal1.im to disk.
```

The resulting image, 3c286cal.im, written to disk, has 4 planes, with I, Q, U, and V images. Use the `viewer` to obtain statistics on each plane of the IQUV image (see the Image Analysis section of this document for details about Viewer operation). Very briefly, bring up the image with the `viewer` by left-clicking on the wrench icon to the right of the image name in the GUI input, and choosing the option **View**. Once the `viewer` window comes up, use the arrow buttons on the right of the `viewer` to display the Q image (2^{nd} image plane). Then, left-click on the **Tools** button at the top of the `viewer` and choose the **ImageStatistics**

[3]The P calibration only rotates the parallactic angle dependence out of Measurement Equation terms containing the nominal interferometer response; the corrupting terms (in particular, those containing D) still have a parallactic angle dependence. It is this dependence which makes the D calibration separable and thus solvable. Refer to the Synthesis Calibration chapter of *G*etting Results for more information.

option to bring up an Image Statistics GUI. Click on the **Statistics** option in this GUI. To obtain statistics in the 2^{nd} image plane, left-click on the **Plane** button. Record the maximum OR minimum value, which ever has the largest absolute value. Use the viewer arrow buttons to display the U image and repeat this procedure. Now, using the Stokes I value reported by the Imager.setjy function run earlier and assuming V=0, the final polarization model is:

```
I = 7.462  Q = 0.4982  U = -0.6988  V = 0
```

If your data set has two spectral windows, create an image with spwid=2 and repeat this procedure to find I, Q, and U.

Now, set the flux density model of 3C 286 with `imager.setjy` and proceed to the `calibrator` tool to solve for the polarization "leakage" terms (aka D-terms):

```
stokes := [7.462, 0.4982, -0.6988, 0.0]      # Define polarization model
                                             #  determined from imaging
imgrC.setjy(fieldid=11,                      # Set model for IQUV
            fluxdensity=stokes);

calC.setdata(msselect='FIELD_ID==11');       # Select data for polarization
                                             #  calibrator, 3C286
calC.reset();                                # Reset setapply/setsolve
calC.setapply(type='P',                      # Arrange to apply parallactic angle
              t=5.0);                        #  correction
calC.setapply(type='G',                      # Arrange to apply flux-scaled
              t=0.0,                         #  G solutions
              table='ap366.fluxcal');
calC.setsolve(type='D',                      # Arrange to solve for D over a long
              t=86400.0,                     #  time scale, average data within
              preavg=600.0,                  #  the solution to no more than 600
              table='ap366.dcal');           #  sec per chunk.
                                             #  Write the solutions to the table
                                             #  ap366.dcal located on disk.
calC.state();                                # Review setapply/setsolve settings
calC.solve();                                # Solve
calC.plotcal(plottype="DRI" ,                # Examine solutions
             tablename="ap366.dcal");
```

The `calibrator.state` and `calibrator.solve` functions will produce the logger messages:

```
  The following calibration components will be applied:
    G table=ap366.fluxcal t=0 select=[]
    P table=<pre-computed> t=5 select=[]
  The following calibration components will be solved for:
    D table=ap366.dcal t=86400 preavg=600 phaseonly=F refant=4 append=F
  Solving for D
  D Jones Slot=1, 1328+307, spw=1: 23-May-1998/00:53:15 to 23-May-1998/04:45:40
  D Jones    Initial fit per unit weight = 0.191679 Jy, sum of weights = 25290
  D Jones    Final   fit per unit weight = 0.13899 Jy after 8 iterations
  Storing D matrix in table ap366.dcal
```

The D-term solutions (Fig. 1.16) are plotted with PLOTTYPE=*DRI*: D-term real versus imaginary components. Most solutions are less than 1% except for a few antennas with about 3% leakage. This is a reasonable solution. See: http://www.aoc.nrao.edu/∼gtaylor/calman/polcal.html for a detailed description on how to plan for good polarization calibration and what to expect when solving for D-terms.

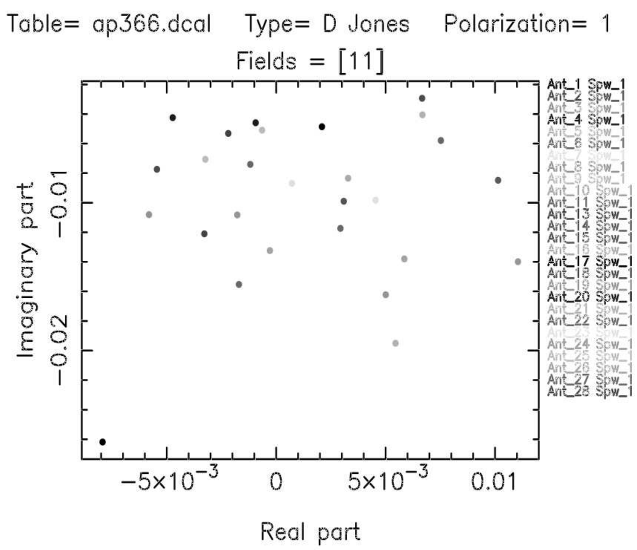

Figure 1.16: Polarization D-term or "leakage" solutions for the for the continuum data set for all antennas. X and Y axes are plotted as percent polarization.

Now, correct the data for P, G and D as follows:

```
calC.setdata(msselect='FIELD_ID IN [9:11]'); # Select fields 9, 10, & 11 to
                                             #  which calibration will be applied:
                                             #  9  = gain calibrator 0917+642
                                             #  10 = target source 0957+561
                                             #  11 = polarization cal 3C 286
calC.reset();                                # Reset setapply/setsolve
calC.setapply(type='P',                      # Arrange to apply parallactic
              t=5.0);                        #  angle correction
calC.setapply(type='G',                      # Arrange to apply flux-scaled
              t=0.0,                         #  G solutions
              table='ap366.fluxcal');
calC.setapply(type='D',                      # Arrange to apply D solutions
              t=0.0,
              table='ap366.dcal');
calC.state();                                # review setapply settings
```

```
calC.correct()                              # Write the corrected the data
                                            #  to the CORRECTED_DATA column
                                            #  in the MS
```

The calC.state and calC.correct functions will report in the logger window:

```
  The following calibration components will be applied:
    D table=ap366.dcal t=0 select=[]
    G table=ap366.fluxcal t=0 select=[]
    P table=<pre-computed> t=5 select=[]
  The following calibration components will be solved for:
    None.
  Initializing non-solvable parallactic angle (P-matrix)
  For interval of 5 seconds, found 213 slots
  Applying G table from ap366.fluxcal
  Applying D table from ap366.dcal
```

Polarization calibration is almost complete at this point – the final step, calibrating the electric vector position angle, will be done during the imaging steps below.

1.12 Making Images

This section describes single-field imaging/deconvolution for the continuum and spectral-line datasets as well as how to interactively define a CLEAN region.

There are almost as many ways to deconvolve an image as there are radio astronomers. The examples given in this section illustrate two relatively simple ways to create images. They have been designed to step the beginning user through the imaging process without getting bogged down with too many details. For an in depth discussion about imaging strategies and the imager arguments see the web documentation at synthesis imaging and imager.

1.12.1 Continuum polarimetry case

For the continuum data set, make a single-field, full-polarization (I, Q, U, & V), continuum image of the target source, 0957+561 (Field 10), using the Clark CLEAN deconvolution algorithm. To do this, first, set which source you want to image, use the imager.advise function to get an estimate of what the image properties should be, then set the image parameters and weight:

```
imgrC:=imager(filename='ap366.ms');  # Create imager tool if not already done.
imgrC.setdata(mode='none',           # Select continuum data for the target
              fieldid=10);           #  source in field 10 (0957+561)
imgrC.advise(takeadvice=F,           # Determine image and cell size using
             fieldofview='1arcmin'); #  advise function.  Do not take advice,
                                     #  note parameters in logger
                                     #  window and use as a basis for setimage.
```

The Advice function will produce logger messages which you can use to guide your choice of image properties:

```
Advising image properties
Maximum uv distance = 585844 wavelengths
Recommended cell size < 0.176041 arcsec
Recommended number of pixels = 360
Dispersion in uv, w distance = 243745, 81981.6 wavelengths
Best fitting plane is w = 0.345636 * u + 0.345275 * v
Dispersion in fitted w = 24694.9 wavelengths
Wide field cleaning is not necessary
```

Set the image parameters and weights:

```
imgrC.setimage(nx=512,               # Set image plane parameters
               ny=512,
               cellx='0.1arcsec',
               celly='0.1arcsec',
               stokes='IQUV',        #   (full polarization)
               fieldid=10);
imgrC.weight(type='uniform');        # Set uniform weighting
```

At this point, you can examine the synthesized beam or Point Spread Function to determine the resolution resulting from the chosen weighting scheme and see what artifacts to expect from beam sidelobes during CLEANing (Fig. 1.17) (Note: this is not a required step to produce an image):

```
imgrC.makeimage(type='psf',          # Form the PSF image if desired
              image='0957+561.psf'); #  use viewer to see sidelobe response

imgrC.fitpsf(psf='0957+561.psf');    # Measure the beam size
```

The `imgrC.fitpsf` function will report the FWHM of the synthesized beam in the logger:

```
Beam fit: 0.423026 by 0.335217 (arcsec) at pa 30.068 (deg)
```

CLEAN the image by first creating a dirty image first, define a mask around obvious source structure, clean for a number of iterations, and redefine the mask until you have all obvious source emission within the masked region. Then continue to clean until you have reached the noise level:

```
imgrC.clean(algorithm='clark',       # Image, and deconvolve using
            niter=5000,              #  the Clark CLEAN
            gain=0.1,                #  Write the cleaned image to the file
            model='0957+561.mod',    #  0957+561.im on disk.
            image='0957+561.im',
            residual='0957+561.resid',
            mask='0957+561.mask',    # Clean mask file name.
```

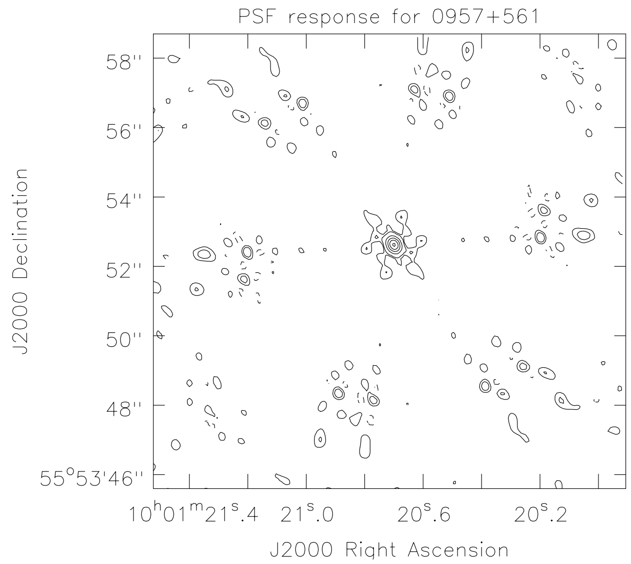

Figure 1.17: Point Spread Function or "Dirty beam" for the continuum data. Contours are plotted at -5, 5, 10, 30, 50, 70, & 90% of the peak.

```
          interactive=T,            # Clean interactively, with the option
          npercycle=500);           #  to choose a new clean region every
                                    #  500 cleaning cycles.  Stop cleaning when
                                    #  you have reached desired level.
dv.gui();                           # Bring up default viewer tool to
                                    #  examine final image.
```

During the CLEAN run, you will be asked to interactively define CLEAN regions (polygons or boxes) in a pop-up viewer. Details about how to do this is described in Section 1.12.3. Interactive masking allows you to add and subtract multiple polygon structures to create a mask around complex source structure. Fig. 1.18 shows an example of the interactive mask GUI interface that pops up during the CLEANing cycle.

Use the viewer to display the image 0957+561.im and obtain statistics (Fig. 1.19). The final image should have an RMS ~ 0.45 mJy beam^{-1} and an image peak ~ 26.3 mJy beam^{-1}.

Note that the CLEAN component model is written as an image file. If necessary or desired, CLEANing may be restarted by specifying an existing (non-zero) model image, which will be subtracted before the deconvolution is resumed. This provides a simple means of monitoring progress of the deconvolution outside of the interactive mode. Note that the imager.setdata and imager.setimage functions need not be run again when restarting a CLEAN with the previously setup imager tool.

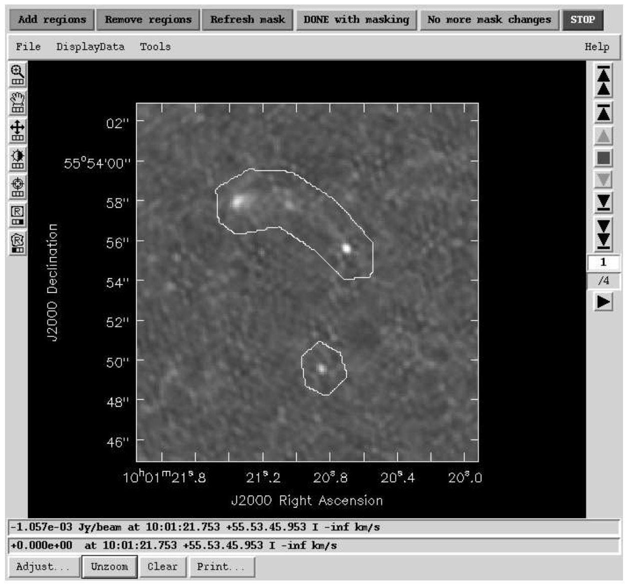

Figure 1.18: The interactive mask GUI showing two CLEANing regions that were defined at the start of the CLEANING process for 0957+561.

1.12.2 Spectral-line case

For the spectral-line data set, total intensity images of the HI in the galaxy NGC 5921 are generated for each channel to form a spectral-line cube. In this example, the CLEANing process is done on the line+continuum data and then the continuum is subtracted in the image plane.

Cleaning a spectral-line cube with the continuum in each channel can be inefficient if most of the clean cycles are spent cleaning the same structure (e.g. continuum emission) in each plane. Thus, it is sometimes better to subtract the continuum emission before the CLEAN process. A *Glish* script along with a description of how to do continuum subtraction in the *uv* plane, before imaging, will be available soon.

CLEANing a cube is similar to CLEANing a single continuum channel except that a mask can be defined for each channel. The mask can be a single shape that is defined for all channels (e.g. as in the continuum polarimetry example) or it can have a variable shape depending upon the location of the emission in each channel. In the simple example below, the image data, parameters, and weighting function are defined first. The inner quarter of the cube (continuum + line emission) is then CLEANed for 3000 iterations:

```
imgrS:=imager(filename='ngc5921.ms');# Create imager tool if not already done.
imgrS.setdata(mode='channel',        # Select channel data for field 3
              nchan=60,
```

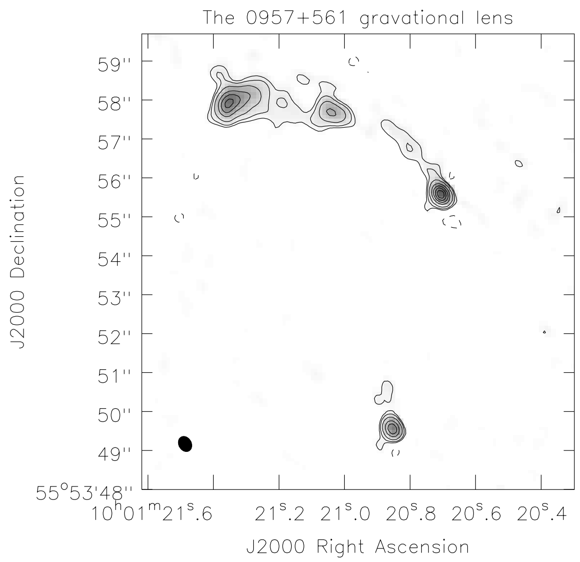

Figure 1.19: Continuum emission at 6 cm in the gravitational lens 0957+561. Grey scale is plotted from 0 to 26.3 mJy beam^{-1}, the RMS is 0.45 mJy beam^{-1}, and contours are plotted at –3, 3, 5, 10, 20, 30, 40, & 50σ. The synthesized beam is shown in the bottom left corner of the image.

```
              start=3,
              step=1,
              fieldid=3);
imgrS.advise(takeadvice=F,           # Determine image and cell size using
            fieldofview='60arcmin'); #  advise function.  Do not take advice,
                                     #  note parameters in logger
                                     #  window and use for setimage.
```

The Advice function will produce logger messages which you can use to guide your choice of image properties:

```
Advising image properties
Maximum uv distance = 4798.32 wavelengths
Recommended cell size < 21.4934 arcsec
Recommended number of pixels = 180
Dispersion in uv, w distance = 1928.57, 788.584 wavelengths
Best fitting plane is w = -0.253496 * u + -0.574818 * v
Dispersion in fitted w = 178.7 wavelengths
Wide field cleaning is not necessary
```

Set the image parameters and weights:

```
imgrS.setimage(nx=256,               # Imaging parameters
```

```
               ny=256,
               cellx='15arcsec',
               celly='15arcsec',
               stokes='I',
               mode='channel',
               nchan=60,
               start=3,
               step=1,
               fieldid=3);
imgrS.weight(type='briggs',           # Robust weighting
             rmode='norm',
             robust=0.5);
```

To examine the synthesized beam or Point Spread Function to determine the resolution resulting from the chosen weighting scheme and see what artifacts to expect from beam sidelobes during CLEANing (Fig. 1.20):

```
imgrS.makeimage(type='psf',           # Form the PSF image if desired
                image='ngc5921.psf');
imgrS.fitpsf(psf='ngc5921.psf');      # Measure the beam size
```

The `imgrS.fitpsf` function will report the FWHM of the synthesized beam in the logger:

```
    Beam fit: 53.2801 by 45.8792 (arcsec) at pa -164.714 (deg)
```

To Hogbom CLEAN the inner quarter of the image down to a threshold level of about the expected RMS level in a single channel (2 mJy):

```
imgrS.clean(algorithm='hogbom',       # Image and deconvolve inner quarter
            niter=6000,               #  with Hogbom CLEAN down to a threshold
            gain=0.1,                 #  of 2 mJy
            threshold='2mJy',
            model='ngc5921.mod',
            image='ngc5921.im',       #  Write the cleaned image to the file
            residual='ngc5921.resid');#  ngc5921.im on disk.
dv.gui();                             # Bring up default viewer tool to
                                      #  examine final image.
```

Use the viewer to display the image ngc5921.im and obtain statistics (Fig. 1.21). The final image should have a peak of $\sim$ 89.2 mJy beam^{-1} and an RMS $\sim$ 1.3 mJy beam^{-1} in each channel. You should see several continuum sources and the HI emission in the galaxy as you move through the cube.

1.12.3 How to interactively define one or more CLEAN regions

1. To zoom on a region, left-click on the Zoom function on the top left of the pop-up Viewer. Use the left mouse button to draw a region around the

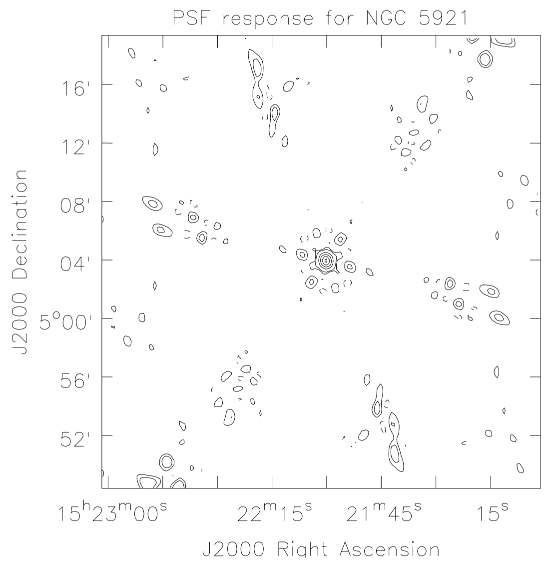

Figure 1.20: Point Spread Function or "Dirty beam" for the spectral-line data. Contours are plotted at -5, 5, 10, 30, 50, 70, & 90% of the peak.

---

source emission. Note: once the box is finished, you can move each point defining the region until it is the size you desire. Double left-click in the region to zoom in.

2. To make a box-mask, left-click on the box region function. Use the mouse button to create a box. To make a polygon-mask, left-click on the polygon region function on the left of the pop-up Viewer. Use the mouse button to create a polygon. Note: once the polygon is finished, you can move each point defining the region until it is the shape you desire.

3. Left-click on 'Add Region'

4. Double left-click in the region you just made (this will add the region to the mask file). The region will be reported on the *Glish* command prompt.

5. Left-click on 'Refresh mask' to see the clean region(s) you have specified displayed on the image.

6. Hit the <ESC> key on the keyboard to remove the polygon or box outline.

7. Define more regions and add if desired.

8. To delete all or part of the mask:

    - Left-click on 'Remove region'

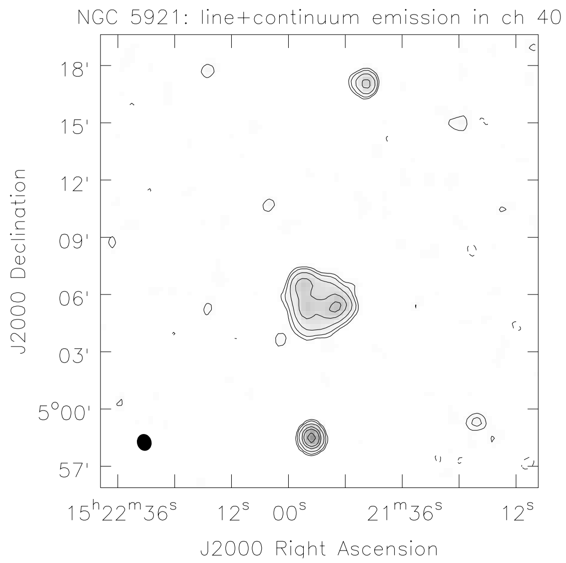

Figure 1.21: Line+continuum emission at 6 cm in channel 40 of the NGC 5921 cube. Grey scale is plotted from 0 to 89.3 mJy beam^{-1}, the RMS is 1.3 mJy beam^{-1}, and contours are plotted at –3, 3, 5, 10, 20, 30, 50, & 70σ. The synthesized beam is shown in the bottom left corner of the image.

- Draw a box or polygon around the mask area you want to remove,
- Double left-click in the box or polygon.
- Left-click on 'Refresh mask' to see your new region.

9. When finished, left-click on 'Done with Masking'

10. The CLEAN algorithm will continue using the region(s) to restrict the location of clean components.

11. You will be given the option to examine the results every npercycle time selected. If you want to stop the cleaning process, left-click on the 'Stop' button.

Note: at this time, if you want to interactively define the CLEAN region or mask and then CLEAN down to a specified threshold level without continuing to set or change the mask, run CLEAN with `niter` set to a small number and `interactive=T` to define the mask. Then re-start CLEAN with `niter` set to a large number, `threshold` set to the level you want, `mask` set to your newly made mask, and `interactive=F`.

1.13 Self-Calibration

If you have sufficient signal-to-noise on your target source or any source in the field, then you may now use the calibrater tool to improve the dynamic range of your images. This improved calibration or "self-calibration" technique assumes that your images have been degraded by complex gain errors which vary too rapidly with time or direction to have been fully calibrated with the calibrator sources. The calibrater tool compares the observed uv DATA with a model of the data that you created during the initial CLEANing and determines amplitude and/or phase corrections which bring the CORRECTED_DATA into better agreement with the model.

Do not self-calibrate unless your target source has enough signal-to-noise to warrant improvement. Ask yourself whether your externally-calibrated CLEAN images contain artifacts well above the noise. If so, self-calibration may improve the image. If your images are limited by system noise, self-calibration may produce erroneous results, including the fabrication of weak sources!

1.13.1 Continuum polarimetry case

First, create the imager tool if it is not already created. Then, fill the MODEL_DATA column in the MS for the target source using the output model from the last execution of `imager.clean`:

```
imgrC.setdata(mode='none',              # Select continuum data for the target
              fieldid=10);              #  source in field 10 (0957+561)
imgrC.ft(model='0957+561.mod');         # Fill MODEL_DATA column in the MS
```

Now it is possible to determine new and improved calibration solutions. We want an incremental solution over the current G, so do an "atmospheric phase" or T calibration (phase only, polarization-independent solutions):

```
calC.setdata(msselect='FIELD_ID==10'); # Set data for target source
calC.reset();                          # Reset apply/solve state
calC.setapply(type='P',                # Arrange to apply P corrections
              t=5.0);
calC.setapply(type="G",                # Arrange to apply flux-scaled
              t=0,                     #  G solutions
              table="ap366.fluxcal");
calC.setapply(type="D",                # Arrange to apply polarization
              t=0,                     #  D-term solutions
              table="ap366.dcal");
calC.setsolve(type="T",                # Arrange to solve for T to
              t=10,                    #  get an incremental solution to
              preavg=0,                #  G with a phase-only correction.
              phaseonly=T,
              table="ap366.tcal");
calC.state();                          # Review the setapply/setsolve settings
```

```
calC.solve();                          # Solve for T
                                       #  Write the output to a table
                                       #  called ap366.tcal1 on disk
calC.plotcal(plottype="PHASE",         # Examine solutions
             tablename="ap366.tcal",
             fields=10);
```

The `calC.state` function reports in the logger window:

```
  The following calibration components will be applied:
    D table=ap366.dcal t=0 select=[]
    G table=ap366.fluxcal t=0 select=[]
    P table=<pre-computed> t=5 select=[]
  The following calibration components will be solved for:
    T table=ap366.tcal t=10 preavg=0 phaseonly=T refant=4 append=F
```

The `calC.solve` function reports messages like:

```
  Initializing solvable atmospheric gain/transmission (T-matrix)
  For interval of 10 seconds, found 96 slots
  Applying G table from ap366.fluxcal
  Applying D table from ap366.dcal
  Solving for T
  T Jones Slot=1, 0957+561, spw=1: 22-May-1998/23:05:30 to 22-May-1998/23:05:35
  T Jones    Initial fit per unit weight = 0.0826564 Jy, sum of weights = 270
  T Jones    Final   fit per unit weight = 0.05584 Jy after 5 iterations
  T Jones Slot=2, 0957+561, spw=1: 22-May-1998/23:05:40 to 22-May-1998/23:05:45
  T Jones    Initial fit per unit weight = 0.0727398 Jy, sum of weights = 1300
  T Jones    Final   fit per unit weight = 0.0501334 Jy after 4 iterations
  ...
  Storing T matrix in table ap366.tcal
```

There is not a whole lot of difference between initial and final fits as expected because this is only an incremental solution.
Now apply the P, G, D, and T solutions and write a new CORRECTED_DATA column for the target source in the MS. Note: Only the `calC.setapply` for T needs to be set now because P, G, & D solutions are already set up to apply:

```
calC.setapply(type="T",                # Arrange to apply T solutions
              t=0,                     #  note: P, G, & D solutions are
              table="ap366.tcal");     #  already set to apply
calC.correct();                        # Correct the data
```

Now, CLEAN the self-calibrated image of 0957+561 to see how this process improved the image:

```
imgrC.setdata(mode='none',             # Select continuum data for the target
              fieldid=10);             #  source in field 10 (0957+561)
imgrC.setimage(nx=512,                 # Set image plane parameters
               ny=512,
```

```
               cellx='0.1arcsec',
               celly='0.1arcsec',
               stokes='IQUV',          # (full polarization)
               fieldid=10);
imgrC.weight(type='uniform');          # Set uniform weighting
imgrC.clean(algorithm='clark',         # Image, and deconvolve using
            niter=5000,                #  the Clark CLEAN
            gain=0.1,                  #  Write the cleaned image to the file
            model='0957+561.mod2',     #  0957+561.im2 on disk.
            image='0957+561.im2',
            residual='0957+561.resid2',
            mask='0957+561.mask2',
            interactive=T,             # Clean interactively, have the option
            npercycle=500);            #  to choose a new clean region every
                                       #  500 cleaning cycles.
dv.gui();                              # Use default viewer to view and analyze images.
```

Figure 1.22 (left) shows the "dirty image" that pops up on the interactive CLEAN `viewer` interface along with an outline of the 2 mask regions that have been specified for the image. After 500 iterations, the residual image will be displayed along with an outline showing your previously defined mask region. You will find that additional, low-level source structure becomes visible. Add additional mask regions around the low-level emission (Fig. 1.22 (right)) and continue cleaning. Use the **ImageStatistics** function in the mask `viewer` to compare the maximum residual reported in the logger window with the residual image RMS. For this example, the final logger report shows:

```
Clean used 500 iterations to get to a max residual of 0.000248331
```

While the RMS in the image is about 0.27 mJy beam^{-1}. It is usually not a good idea to clean below the RMS level in the residual image, therefore, the CLEAN process was stopped at this point.

The final, self-calibrated image of the gravitational lens, 0957+561 shown in Fig. 1.23, shows additional source structure that was only hinted at below the 3σ level in the pre-self-cal image. The image peak is now $\sim$ 29.7 mJy beam^{-1}, and the RMS is 0.29 mJy beam^{-1}, a factor of 1.6 times better than before self-calibration.

If you have a data set in which there is a strong continuum source or strong line emission (e.g. a maser feature), and self-calibration improved the image significantly, then you can repeat the self-calibration as necessary. Note: Self-calibration can be applied to any solvable calibration component (G, and/or D in the present example).

1.13.2 Spectral-line case

In the spectral-line data set, there is an 80 mJy continuum source just south of the main line emission in NGC 5921. We can use this source to self-calibrate the line emission.

Create an image with line-free continuum channels only. First, create an image from line-free channels 5–8: First, create the imager tool if it is not already

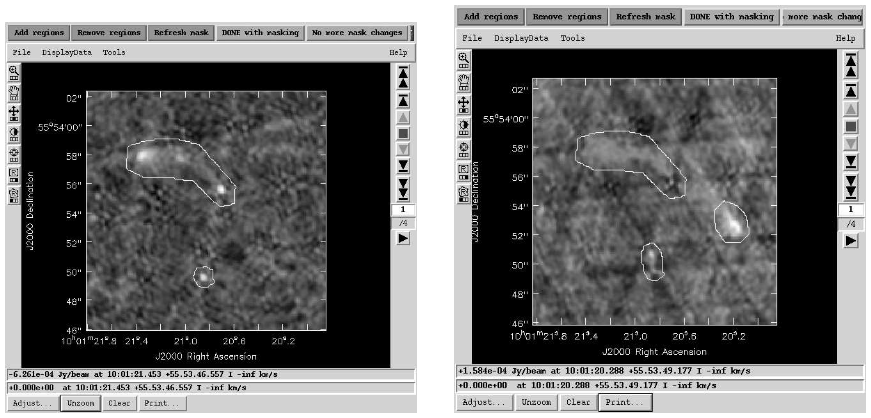

Figure 1.22: Interactive masks for self-calibrated 0957+561 during imaging. **Left:** The first mask regions captures the major source structure. **Right:** During the second interative CLEAN cycle, low-level emission becomes visible and new mask regions are added.

---

created. Then, create an image of line-free continuum channels only and fill the MODEL_DATA column for the specified channels in the MS for the target source using the output model from the last execution of `imager.clean` on the continuum-only data:

```
imgrS:=imager(filename='ngc5921.ms');# Create imager tool if needed.
imgrS.setdata(mode='channel',        # Select channel data for 4
              nchan=4,               #  continuum-only channels in
              start=2,               #  field 3 (target field)
              step=1,
              spwid=1,
              fieldid=3);
imgrS.setimage(nx=256,               # Imaging parameters
               ny=256,
               cellx='15arcsec',
               celly='15arcsec',
               stokes='I',
               mode='channel',       # Choose channel mode
               nchan=1,              # Grid selected channels into one.
               start=2,
               step=4,
               fieldid=3);
imgrS.weight(type='briggs',          # Robust weighting
             rmode='norm',
             robust=0.5);
imgrS.clean(algorithm='clark',       # Image and deconvolve with Clark CLEAN
            niter=6000,              #  to a threshold of 2 mJy
```

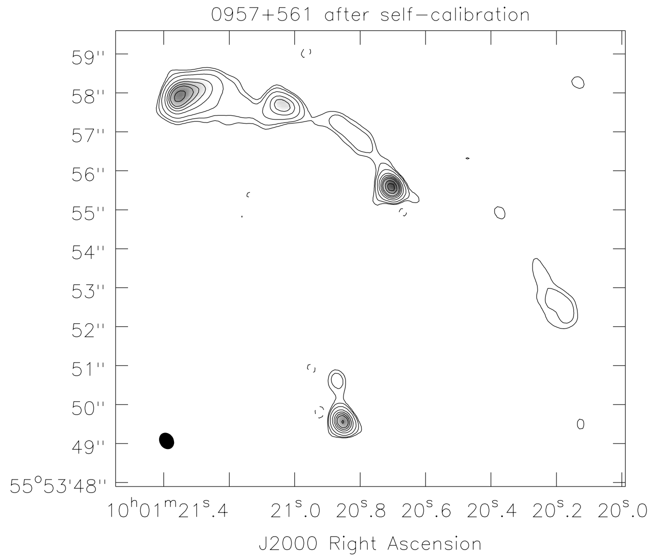

Figure 1.23: Continuum emission at 6 cm in the gravitational lens 0957+561 after self-calibration. Grey scale is plotted from 0 to 29.7 mJy beam^{-1}, the RMS is 0.29 mJy beam^{-1}, and contours are plotted at –3, 3, 5, 10, 20, 30, 40, 50, 70, & 90σ. The synthesized beam is shown in the bottom left corner of the image. This can now be classified as a beautiful image!

```
            gain=0.1,                #  Write the cleaned image,
            threshold='0Jy',        #  ngc5921.c1.im and the model,
            model='ngc5921.c1.mod',  #  image ngc5921.c1.mod to disk
            image='ngc5921.c1.im',
            residual='ngc5921.c1.resid');
imgrS.setdata(mode='channel',        # Select channel data for 4
              nchan=4,               #  continuum-only channels in
              start=2,               #  field 3 (target field)
              step=1,
              spwid=1,
              fieldid=3);

imgrS.ft(model='ngc5921.c1.mod');    # Fill MODEL_DATA column in the MS
```

Now, to derive an incremental solution over the current G, do an "atmospheric phase" T calibration which will be derived from the continuum channels which were used to create the model (phase only, polarization-independent solutions): Note: the MODEL_DATA column is read automatically when solving for T. Set the time interval for the T solutions to be roughly the same as the solution intervals to get scan-based solution intervals.

```
calS.setdata(mode='channel',         # Select channel data for 4
             nchan=4,               #  continuum-only channels in
             start=2,               #  field 3 (target field)
             step=1,
             msselect='FIELD_ID==3');
calS.reset();                        # Reset apply/solve state
calS.setapply(type="G",              # Arrange to apply flux-scaled
             t=0,                   #  G solutions
             table="ngc5921.fluxcal");
calS.setapply(type="B",              # Arrange to apply
             t=0,                   #  Bandpass solutions
             table="ngc5921.bcal");
calS.setsolve(type="T",              # Arrange to solve for T to
             t=100,                 #  get an incremental solution to
             preavg=0,              #  G with a phase-only correction.
             refant=14,
             phaseonly=T,           #  Write the output to a table
      table="ngc5921.tcal"); #  called ngc5921.tcal on disk
calS.state();                        # Review the setapply/setsolve settings
calS.solve();                        # Solve for T
calS.plotcal(plottype="PHASE",       # Examine solutions
            tablename="ngc5921.tcal");
```

The `calS.state` function reports in the logger window:

```
  The following calibration components will be applied:
    B table=ngc5921.bcal t=0 select=[]
    G table=ngc5921.fluxcal t=0 select=[]
  The following calibration components will be solved for:
    T table=ngc5921.tcal t=100 preavg=0 phaseonly=T refant=14 append=F
```

The `calS.solve` function reports messages like:

```
  Initializing solvable atmospheric gain/transmission (T-matrix)
  For interval of 100 seconds, found 21 slots
  Applying B table from ngc5921.bcal
  Applying G table from ngc5921.fluxcal
  Solving for T
  T Jones Slot=1, N5921, spw=1: 13-Apr-1995/09:33:00 to 13-Apr-1995/09:33:00
  T Jones    Initial fit per unit weight = 0.273765 Jy, sum of weights = 1.06142e+06
  T Jones    Final   fit per unit weight = 0.26221 Jy after 9 iterations
  T Jones Slot=2, N5921, spw=1: 13-Apr-1995/09:34:00 to 13-Apr-1995/09:35:00
  T Jones    Initial fit per unit weight = 0.275376 Jy, sum of weights = 2.12285e+06
  T Jones    Final   fit per unit weight = 0.26615 Jy after 6 iterations
```

There is not a whole lot of difference between initial and final fits as expected because this is only an incremental solution. Now apply the G, B, and T solutions to ALL the data and write a new CORRECTED_DATA column for the target source in the MS:

```
calS.setdata(mode='channel',         # Select channel data for all
             nchan=63,               #  channels in field 3 (NGC 5921)
             start=1,
             step=1,
             msselect='FIELD_ID==3');
calS.reset();                        # Reset apply/solve state
calS.setapply(type="G",              # Arrange to apply flux-scaled
             t=0,                    #  G solutions
             table="ngc5921.fluxcal");
calS.setapply(type="B",              # Arrange to apply
             t=0,                    #  Bandpass solutions
             table="ngc5921.bcal");
calS.setapply(type="T",              # Arrange to apply T solutions
             t=0,
             table="ngc5921.tcal");
calS.correct();                      # Correct the data
```

CLEAN the self-calibrated image of NGC 5921 to see if the image is improved:

```
imgrS.setdata(mode='channel',        # Select channel data for field 3
             nchan=60,
             start=3,
             step=1,
             fieldid=3)
imgrS.setimage(nx=256,               # Select imaging parameters
              ny=256,
              cellx='15arcsec',
              celly='15arcsec',
              stokes='I',
              mode='channel',
              nchan=60,
              start=3,
              step=1,
              fieldid=3);
imgrS.weight(type='briggs',          # Robust weighting
            rmode='norm',
            robust=0.5);
imgrS.clean(algorithm='hogbom',      # Image and deconvolve the inner quarter
           niter=6000,               #  of the image with Hogbom CLEAN
           gain=0.1,                 #  to a threshold of 1 mJy.
           threshold='1.0mJy'        #  Write the cleaned image to the file
           model='ngc5921.mod2',     #  ngc5921.im2 on disk.
           image='ngc5921.im2',
           residual='ngc5921.resid2'
           interactive='F'
```

Use the viewer to examine the final source image ngc5921.im2. The RMS is ~ 1.1 mJy beam^{-1} (versus 1.3 mJy beam^{-1} before self-calibration). The peak of the continuum source is now 84.5 mJy beam^{-1}. Self-calibration provided a marginal improvement to the image.

If you have a data set in which there is a strong continuum source or strong line emission (e.g. a maser feature), and self-calibration improved the image significantly, then you can repeat the self-calibration as necessary. Note: Self-calibration can be applied to any solvable calibration component (G, T, and/or B in the present example).

1.14 Analyzing Images

See the Display and Image analysis chapters of *Getting Results* for a detailed discussion on display and image analysis tools and the options available. A brief orientation on the use of the viewer a key element in image analysis, is given below. The `viewer` is described in detail in the chapter on Display of Data while the 'Adjust' options to taylor the image display parameters are described at Adjust.

The Viewer
To start the viewer type:

```
include 'viewer.g'
dv.gui();
```

The viewer Data Manager and Display Panel GUIs will appear. Using the Data Manager, left-click on an image you would like to see in the Display Panel and choose the display type (e.g. Raster Image, Contour Map). The image will be loaded into the Display Panel. You can load as many images and display types as you want.

To zoom in on a region, left-click on the **zoom** button (magnifying glass icon) in the upper left of the Viewer display, use the left mouse button to make a region, double-left-click in the region to zoom. To un-zoom, hit the **Unzoom** button at the bottom of the Display.

Image, axis, and label display options are controlled via the 'Adjustment' panel. Left-click on the **Adjust...** button to bring up this panel. There are a wide range of options to choose from: e.g. plot the beam, change the color map or stretch of an image, change contour lines, change the line width or color, add or delete labels. Once you have the image the way you like it, you can save the state of the image by left-clicking on the 'Save' button in the 'Adjustment' panel. The display parameters can then be restored later by left-clicking on the 'Restore' button.

To obtain statistics on all or part of an image, left-click on the **Tools: ImageAnalysis** button at the top of the Viewer Display. An `Image Analysis` GUI will be created. Left-click on **Statistics** to show the statistics panel. The **Full** button will produce statistics on the full image (all planes). The **Plane** button will produce statistics on only the region being displayed on the Viewer panel. Thus, to estimate the RMS in an image, zoom in on a region that has no emission and left-click on the Statistics panel 'Plane' button.

Use the 'DisplayData: Register' Panel to choose which images you want to register and display together. Use the 'DisplayData: Remove' Panel to delete images from the stack.

To print the image, left-click on the **Print** button at the bottom of the Viewer Panel. You will get a `Canvas Print Manager` GUI to specify the output postscript file name or send the image to a local printer. Postscript files will be written to the directory you started AIPS++ in.

Below are image analysis examples that are useful for analyzing the example data sets.

1.14.1 Continuum polarimetry case

GUI Interface:

We want to form a linear polarization intensity image, e.g. $P = \sqrt{Q^2 + U^2}$ and a vector image of the electric vectors of the polarization, e.g. $(Q + iU)$ as described in the Image analysis chapters of *G*etting Results. First, start an `image polarization` tool which provides access to a slew of tools that are useful for analyzing polarization images:

```
impolC:=imagepol(infile="0957+561.im2");   # Start the image polarization tool
                                           #  for the image 0957+561.im2
```

Create a `linear polarization intensity` tool (leave all inputs to their default values):

```
linpolintC:=impolC.linpolint();            # Form linear polarization intensity
                                           #  image tool, linpolint
linpolintC.subim(outfile='0957+561.pint'); # Form the subim tool and write
                                           #  the image 0957+561.pint
                                           #  to a file on disk
subim.done();                              # Close the tool (no longer needed)
linpolintC.done();                         # Close the tool (no longer needed)
impolC.complexlinpol(outfile="0957+561.cmplx");
                                           # Create a vector image of the
                                           # electric vectors of the polarization
impolC.done();                             # Close the tool (no longer needed)
```

You should now have two image files called *0957+561.pint* & *0957+561.cmplx* on disk as well as the original *0957+561.im2* polarization image. Follow the instructions above to load the images into the `viewer`:

- Load 0957+561.im2 as a raster image
- Load 0957+561.pint as a contour image
- Load 0957+561.cmplx as a vector map

Bring up the `Image Analysis` GUI and determine the RMS of the polarization intensity image using the 'Statistics' panel (for the 0957+561.pint image, the RMS should be $\sim$ 11 mJy beam^{-1}). We will use this value to set a mask expression in the 0957+561.cmplx vector image to get rid of polarization vectors that are below the RMS level.

Left-click on the 'Adjust' button to bring up 3 Adjustment panels, one for each image. Then:

- Adjust the color map for 0957+561.im2 (Color palette of choice: Greyscale 1 for black & white on the Viewer, Greyscale 2 for output to a psfile).
- Adjust the size of the polarization vectors in 0957+561.cmplx using the 'amplitude scale factor', 'x-increment', and 'y-increment'.
- Set the mask expression in the 0957+561.cmplx image to be: '0957+561.pint' $>$ 0.00033 (3σ in the polarization intensity image). Note, the name 0957+561.pint must be in quotes because it contains a + sign which will be interpreted as an expression.

Now, calibrate the electric-vector positions angle (EVPA):

$$\mathrm{EVPA} = 0.5\tan^{-1}(U/Q) = 0.5 * (\mathrm{R} - \mathrm{L}\ \text{phase difference}).$$

First, it is necessary to specify the true ratio U/Q found for the polarization calibrator, 3C 286. During the polarization calibration, we found for 3C 286 ($I = 7.462\ \ Q = +0.4982\ \ U = -0.6988\ \ V = 0$), thus, the measured R–L phase difference is: $\tan^{-1}(-0.6988/0.4982) = -54.5°$. Since U is negative and Q is positive, this is in the 4^{th} quadrant.

From the VLA calibration web page (http://www.aoc.nrao.edu/∼gtaylor/calman/polcal.html) we find the true R-L phase difference for 3C 286 is 66°. Thus, our phases should be rotated by $66 + 54.5 = 120.5°$ or $EVPA = 0.5 * 120.5°$. In the 'Adjust' panel for the 0957+561.cmplx image, enter `Extra rotation = 120.5/2`. To plot the magnetic field vectors, enter `Extra rotation =` $120.5/2 + 90$. The final images are shown in Fig. 1.24.

1.14.2 Spectral-line case

Now, use the image tool to subtract the continuum emission in the image plane using the line-free channels (5-8) and (55-58).

First, the image coordinate system is stored so that regions can be made in world coordinates. Then make two regions with the line-free emission and union them together. Note, a union of regions is a compound region that has to be made from two *world* regions, not pixel regions, so it must be constructed from regions made with the wrange function from the default regionmanager (drm). Once the line-free channels are unioned together, an integrated continuum image is made and subtracted from the line-cube.

```
imS:=image('ngc5921.im2');        # Start Image tool

cs:=imS.coordsys();               # Store image coordinate system
                                  #  in the variable 'cs'

range1:=dq.quantity([5,8],'pix');
```

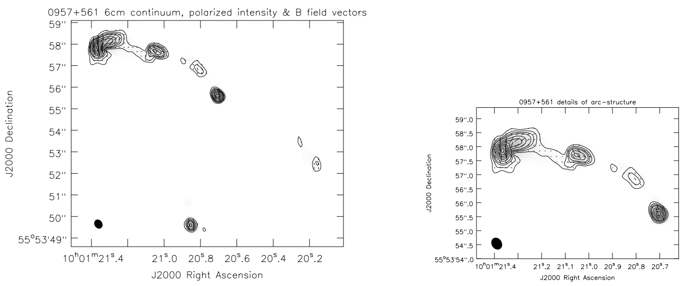

Figure 1.24: Final polarization images of the gravitational lens 0957+561. **Left:** Full image showing all detectable structure. Grey-scale is the 6 cm continuum emission plotted from 58 mJy beam^{-1} (2σ) to 27.7 mJy beam^{-1}. Contours show the polarized intensity beginning at $\pm$0.33 mJy beam^{-1} (3σ) and continuing at levels of 5, 7, 9, 13, 17, 21, 25, 29, 33σ. Magnetic field vectors are plotted as straight lines, the amplitude of the polarization is proportional to the length of the lines. **Right:** Close up of the brightest arc showing position angle rotation across the emission.

```
                                  # Create variable 'range1'

cont1:=drm.wrange(pixelaxis=4,    # Make first region for lower
                 range=range1,   #  continuum channels,
                 csys=cs);       #  Pixelaxis 4 = frequency axis

range2:=dq.quantity([55,58],'pix');
                                  # Create variable 'range2'

cont2:=drm.wrange(pixelaxis=4,    # Make second region for upper
                 range=range2,   #  continuum channels,
                 csys=cs);

cont3:=drm.union(cont1,cont2);    # Union the two chunks of data together

imS.moments(outfile='ngc5921.cont',
           moments=-1,           # Create an integrated
           axis=4,               #  continuum image, ngc5921.cont,
           region=cont3);        #  and write the file to disk.

nocont:=imagecalc(outfile='ngc5921.nocont',
                 pixels='ngc5921.im2 - ngc5921.cont');
                                  # Subtract the continuum from the
                                  #  line cube and write the line-only
                                  #  cube, ngc5921.nocont, to disk.

imS.done();                       # close the tool, no longer needed
```

```
cs.done();                        # close the tool, no longer needed
nocont.done();                    # close the tool, no longer needed
```

View the final image in the Viewer. The continuum sources should now be gone and you can see the rotation curve of the HI in the galaxy as you move through the line cube Fig. 1.25.

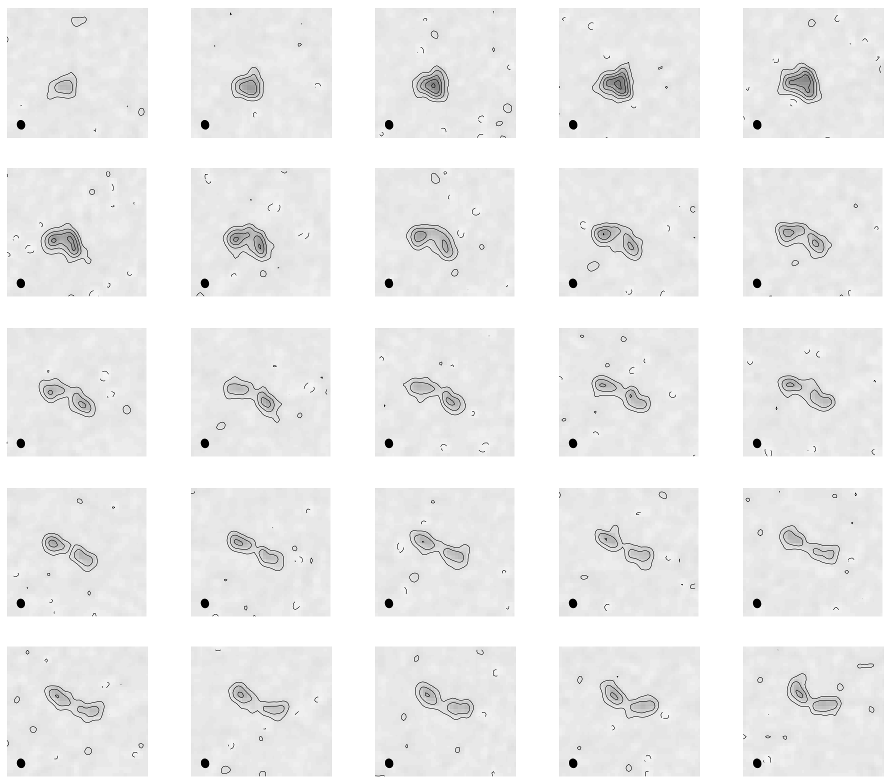

Figure 1.25: Channel map of 5921 (line only, channels 10-34). RMS = 1.1 mJy beam^{-1}. Contours are plotted at –3, 3, 10, 20, 30, & 40 $\times\sigma$.

Now, take a spectrum through the cube. There are two ways to pick positions for taking spectra: 1) view the continuum image and take spectra at the locations of the continuum sources to make sure that continuum subtraction has gone OK; 2) view the 0^{th} moment image (generated by the imageprofilefitter tool automatically) and use this to select spectra in regions with line emission. The latter is the default behavior. However spectral line cubes can have noisy beginning and end channels, so it is often better to view your own 0^{th} moment image when selecting positions for spectra. Both of these ways are shown below:

Use the continuum image to check that the continuum subtraction has gone well.

```
include 'imageprofilefitter.g';   # Create image profile fitter tool

pf:=imageprofilefitter(infile='ngc5921.nocont',
                       infile2='ngc5921.cont');
                                  # Get a spectrum from ngc5921.nocont
                                  #  But use the image of ngc5921.cont
                                  #  To see where to place the cursor
```

You will get a Viewer window labeled 'Image Profile Fitter' which shows the line+continuum image. To get a spectrum:

- Use the 'Adjust' button to adjust the stretch so you can see where the line emission is.
- Left-click on the 'Positioning' button (bulls-eye) to the left of the image.
- Use the left mouse button to left-click on a position. A line-only spectrum will be displayed in a Profile Fitter GUI. In the Image GUI, adjust the 'Width' parameter which specifies the number of pixels to obtain an average spectrum, and the spectrum position until you get the spectrum you want.
- Alternatively, you can generate a region around the source with the Box-region or polygon-region buttons, double left-click in the region to get a spectrum.
- In the Profile Fitter window, left-click on 'Print' to print the spectral line plot to the printer or postscript file.

The spectrum taken at the position of the brightest continuum source, shows that the continuum subtraction appears to have done a good job – the spectrum is flat and amplitude is zero (Fig. 1.26).

Create a line-only 0^{th} moment image and use this to select regions of line emission to take spectra: Moment analysis for line-emission cubes. Viewing the image we find that there is line emission in channels 7–49.

```
imS:=image('ngc5921.nocont');                # Start Image tool

cs:=imS.coordsys();                       # Store image coordinate system

line_range:=dq.quantity([7,49],'pix');    # There is line emission in
                                          #  channels 7-49

line_region:=drm.wrange(pixelaxis=4,      # Define the line region in
                        range=line_range, #  world coordinates
                        csys=cs);

imS.moments(moments=[0, 1, 2] ,
          smoothaxes=[1,2,4] ,            # Pixels to be included in
```

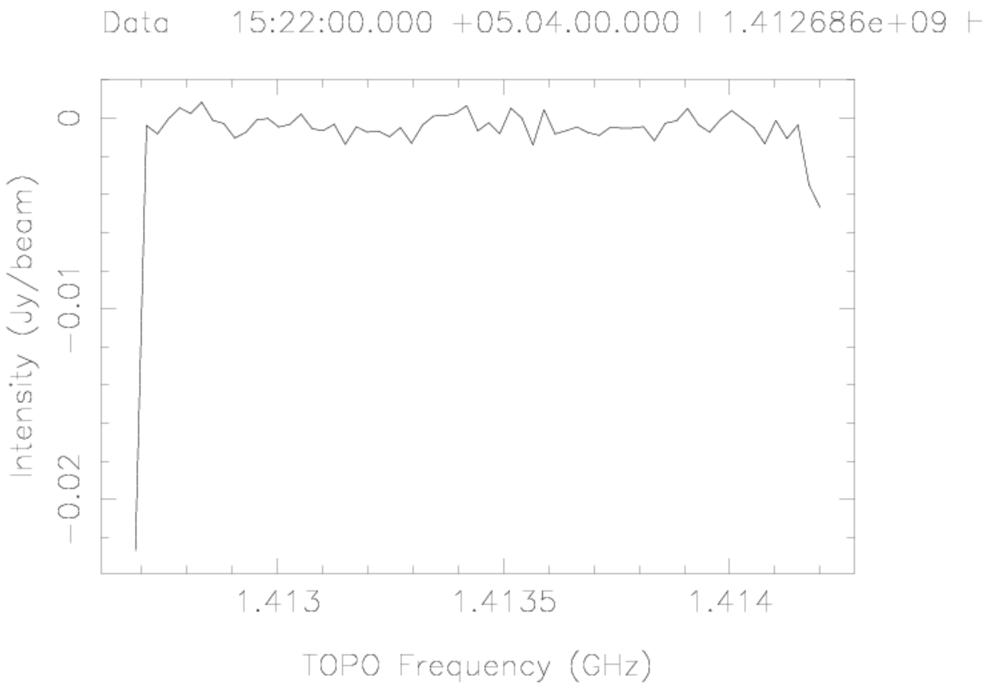

Figure 1.26: Quality check on continuum subtraction. Spectrum taken at the position of the brightest continuum source after the continuum has been subtracted from the image.

```
                                # moment analysis will be
                                # decided by examing the
                                # flux in a smoothed version
                                # of the data.
                                # We will smooth in ra, dec
                                # and velocity (axis=1,2,4)

region = line_region,           # Only use channels with
                                # line emission

mask='$imS>0',                  # Only include pixels with
                                # positive flux in original map

excludepix=[0.003],             # Only sum flux from pixels
                                # in smoothed map with flux
                                # greater than abs(0.003 Jy)

smoothtypes="gauss gauss hann" ,# Gaussian smoothing in ra
                                # and dec; Hanning smoothing
smoothwidths=[5,5,3]  ,         # Width of smoothing kernals
```

```
        doppler="OPTICAL" ,             # optical velocity definition

        outfile='ngc5921.nocont2');
```

The moments command will create 3 images:

```
        ngc5921.nocont2.integrated                # Moment 0 image
        ngc5921.nocont2.weighted_coord            # Moment 1 image
        ngc5921.nocont2.weighted_dispersion_coord # Moment 2 image
```

Now take a spectra using the clipped, integrated image to isolate the area of line-emission:

```
include 'imageprofilefitter.g';
pf2:=imageprofilefitter(infile='ngc5921.nocont',
                        infile2='ngc5921.nocont2.integrated');
```

The spectrum shown in Fig. 1.27, illustrating a classic double-peaked line profile showing the galactic rotation, was made with the 'Polygon-Region' option in the Viewer window to circle the emission in NGC 5921, then double left-clicking on the region to create the spectrum.

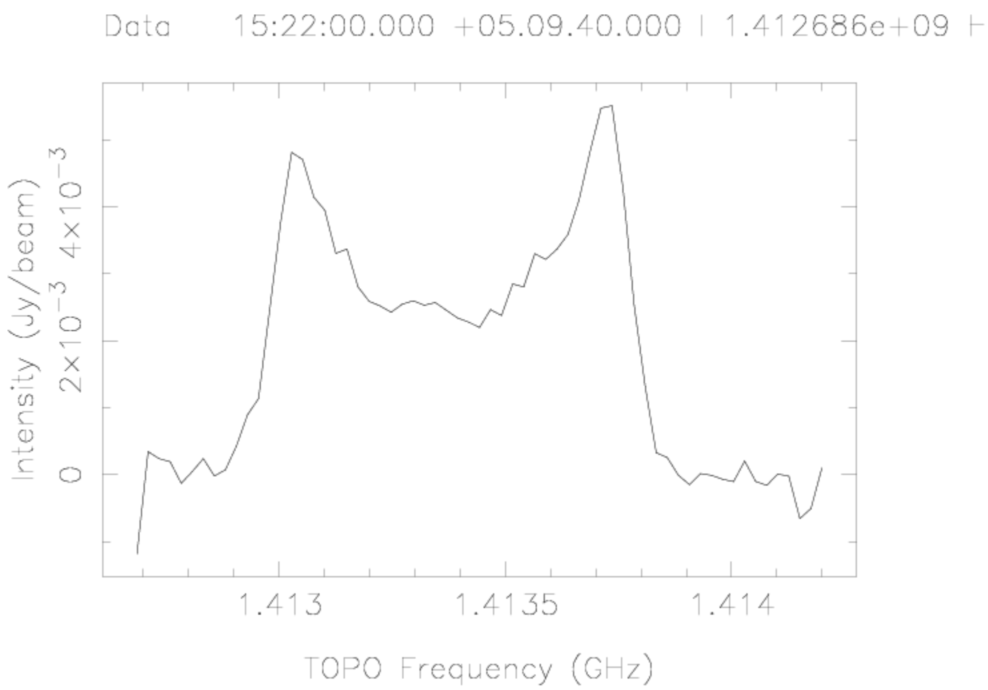

Figure 1.27: Spectrum of NGC 5921.

1.15 Cleanup

After completing work with AIPS++ tools, it is desirable to close them out by issuing the done function for each, as follows:

1.15.1 Continuum polarimetry case

```
calC.done();                     # Finish imager tool
imgrC.done();                    # Finish calibrater tool
imC.done();                      # Finish image tool
```

1.15.2 Spectral-line case

```
calS.done();                     # Finish imager tool
imgrS.done();                    # Finish calibrater tool
imS.done();                      # Finish image tool
```

1.16 References

Hamaker, J.P., Bregman, J.D., and Sault, R.J., 1996a, *A&AS*, **117**, 137.
Sault, R.J., Hamaker, J.P., and Bregman, J.D., 1996b, *A&AS*, **117**, 149.

1.17 Appendicies

1.17.1 Appendix 1: AIPS – AIPS++ dictionary

This appendix provides a list of common AIPS tasks, and their equivalent AIPS++ tool or tool function names. The two packages differ in both their architecture and calibration and imaging models, and there is often not a direct correspondence. However, this index does provide a scientific user of AIPS++ who is familiar with AIPS, with a simple translation table to map their existing data reduction knowledge to the new package.

The index includes common AIPS tasks and verbs; adverbs are not included. Capabilities which exceed those available in AIPS are not listed here.

Table 1.4: AIPS – AIPS++ dictionary

AIPS Task/Verb	Description	AIPS++ tool/function
APCLN	CLEAN deconvolution	imager.clean
BLCAL	Baseline-based (A,ϕ) calibration	calibrater.solve
BLING	Baseline-based fringe-fitter	*not yet available*
BPASS	Bandpass calibration	calibrater.solve
CCEDT	Edit CC tables	componentmodels
COMB	Image combination	image.calc, imagepol
CPASS	Polynomial bandpass calibration	calibrater.solve
CALIB	(A,ϕ) self-calibration	calibrater.solve
DBCON	uv-data concatenation	*not yet available*
DTSIM	Simulator	simulator
DTSUM	uv-data summary	ms.summary
FILLM	VLA filler	vlafiller
FITLD	VLBA filler	*not yet available*
FITTP	UVFITS writer	ms.tofits
FRING	VLBI fringe-fitter	*not yet available*
GETJY	Set flux density scale	calibater.fluxscale
IBLED	Interactive VLBI editor	*not yet available*
IMAGR	Synthesis imaging	imager

1.17.2 Appendix 2: MIRIAD – AIPS++ dictionary

This appendix provides a list of common Miriad tasks, and their equivalent AIPS++ tool or tool function names. The two packages differ in both their architecture and calibration and imaging models, and there is often not a direct correspondence. However, this index does provide a scientific user of AIPS++ who is familiar with MIRIAD, with a simple translation table to map their existing data reduction knowledge to the new package.

1.17.3 Appendix 3: Example .aipsrc file

```
#
# Work in the current directory:
#
user.aipsdir:   .
#
# Put scratch files here: can be a list of directories
#
user.directories.work:  /export/home/your_computer/aips++/scratch
#
# Put the aips++ cache files in one place: it shouldn't be
# over an NFS mount or it will slow aips++ down to a crawl!
#
user.cache:     /export/home/your_computer/aips++/cache
#
# Set these so that bug(), ask(), etc. know who you are
#
userinfo.name:  Your_name
userinfo.email: Your_email@institution.edu
userinfo.org:   Your Organization
#
# Switch on the display of memory used by each server
#
user.display.memory:    True
#
# Uncomment this to send log messages to the deep blue yonder
#
#logger.file:           none
#
# plot axis labels as the default in the viewer
# and set initial character size (normal default size = 1.2):
#
display.axislabels: on
display.axislabels.charsize: 1.2
#
#
system.aipscenter:       NorthAmerica
logger.file:             aips++.log
logger.height:           15
catalog.gui.auto:        T
catalog.confirm:         T
catalog.view.PostScript: ghostview
toolmanager.gui.auto:    T
toolmanager.refresh:     10s
```

1.17.4 Appendix 4: Continuum polarimetry dataset summary

Summary created with the command `msetC.summary(verbose=T)`:

```
           MeasurementSet Name:  ap366.ms      MS Version 2

   Observer: AP366     Project:
Observation: VLA
Data records: 93239       Total integration time = 21935 seconds
   Observed from   22-May-1998/22:40:05   to   23-May-1998/04:45:40

   ObservationID = 1         ArrayID = 1
  Date        Timerange                Scan  Field          DataDescIds
  22-May-1998/22:40:05.0 - 22:41:10.0     1  0917+624       [1]
              23:05:30.0 - 23:14:00.0     2  0957+561       [1]
              23:26:55.0 - 23:28:00.0     3  0917+624       [1]
              23:46:25.0 - 23:47:30.0     4  0917+624       [1]
  23-May-1998/00:10:50.0 - 00:18:20.0     5  0957+561       [1]
              00:30:15.0 - 00:31:20.0     6  0917+624       [1]
              00:53:15.0 - 00:54:50.0     7  1328+307       [1]
              04:44:05.0 - 04:45:40.0     8  1328+307       [1]
Fields: 3
  ID   Name          Right Ascension  Declination   Epoch
  9    0917+624      09:21:36.23      +62.15.52.18  J2000   ** Gain calibrator
  10   0957+561      10:01:20.70      +55.53.52.65  J2000   ** Source
  11   1328+307      13:31:08.28      +30.30.32.94  J2000   ** Flux calibrator, 3C 286
                                                            ** (B1950 name, J2000 coords.)
Data descriptions: 1 (1 spectral windows and 1 polarization setups)
  ID    Ref.Freq    #Chans  Resolution  TotalBW     Correlations
  1     4885.1 MHz  1       50000  kHz  50000  kHz  RR  RL  LR  LL
Feeds: 28: printing first row only
  Antenna   Spectral Window     # Receptors    Polarizations
  1         -1                  2              [         R, L]
Antennas: 27:
  ID   Name  Station   Diam.    Long.         Lat.
  1    1     VLA:W16   25.0 m   -107.37.57.4  +33.53.33.0
  2    2     VLA:N16   25.0 m   -107.37.10.9  +33.54.48.0
  3    3     VLA:N48   25.0 m   -107.37.38.1  +33.59.06.2
  4    4     VLA:W8    25.0 m   -107.37.21.6  +33.53.53.0
  5    5     VLA:N56   25.0 m   -107.37.47.9  +34.00.38.4
  6    6     VLA:E16   25.0 m   -107.36.09.8  +33.53.40.0
  7    7     VLA:E24   25.0 m   -107.35.13.4  +33.53.18.1
  8    8     VLA:N8    25.0 m   -107.37.07.5  +33.54.15.8
  9    9     VLA:E8    25.0 m   -107.36.48.9  +33.53.55.1
  10   10    VLA:W40   25.0 m   -107.41.13.5  +33.51.43.1
  11   11    VLA:N24   25.0 m   -107.37.16.1  +33.55.37.7
  13   13    VLA:W56   25.0 m   -107.44.26.7  +33.49.54.6
  14   14    VLA:E64   25.0 m   -107.27.00.1  +33.50.06.7
  15   15    VLA:N64   25.0 m   -107.37.58.7  +34.02.20.5
```

```
  16   16    VLA:W32   25.0 m   -107.39.54.8  +33.52.27.2
  17   17    VLA:W64   25.0 m   -107.46.20.1  +33.48.50.9
  18   18    VLA:E48   25.0 m   -107.30.56.1  +33.51.38.4
  19   19    VLA:N40   25.0 m   -107.37.29.5  +33.57.44.4
  20   20    VLA:E40   25.0 m   -107.32.35.4  +33.52.16.9
  21   21    VLA:E56   25.0 m   -107.29.04.1  +33.50.54.9
  22   22    VLA:E72   25.0 m   -107.24.42.3  +33.49.18.0
  23   23    VLA:W24   25.0 m   -107.38.49.0  +33.53.04.0
  24   24    VLA:N32   25.0 m   -107.37.22.0  +33.56.33.6
  25   25    VLA:W72   25.0 m   -107.48.24.0  +33.47.41.2
  26   26    VLA:W48   25.0 m   -107.42.44.3  +33.50.52.1
  27   27    VLA:N72   25.0 m   -107.38.10.5  +34.04.12.2
  28   28    VLA:E32   25.0 m   -107.34.01.5  +33.52.50.3


Tables:
   MAIN                   93239 rows
   ANTENNA                   28 rows
   DATA_DESCRIPTION           1 row
   DOPPLER             <absent>
   FEED                      28 rows
   FIELD                     11 rows
   FLAG_CMD             <empty>
   FREQ_OFFSET         <absent>
   HISTORY                  108 rows
   OBSERVATION                1 row
   POINTING                 168 rows
   POLARIZATION               1 row
   PROCESSOR            <empty>
   SOURCE              <absent> (see FIELD)
   SPECTRAL_WINDOW            1 row
   STATE                <empty>
   SYSCAL              <absent>
   WEATHER             <absent>
```

1.17.5 Appendix 5: Spectral-line dataset summary

Summary created with the command `msetS.summary(verbose=T)`:

```
           MeasurementSet Name:  ngc5921.ms      MS Version 2

   Observer: TEST     Project:
Observation: VLA
Data records: 22653       Total integration time = 5280 seconds
   Observed from   13-Apr-1995/09:19:00   to   13-Apr-1995/10:47:00

   ObservationID = 1         ArrayID = 1
  Date        Timerange                Scan  Field          DataDescIds
  13-Apr-1995/09:19:00.0 - 09:24:30.0     1  1331+30500002  [1]
              09:27:30.0 - 09:29:30.0     2  1445+09900002  [1]
              09:33:00.0 - 09:48:00.0     3  N5921          [1]
              09:50:30.0 - 09:51:00.0     4  1445+09900002  [1]
              10:22:00.0 - 10:23:00.0     5  1445+09900002  [1]
              10:26:00.0 - 10:43:00.0     6  N5921          [1]
              10:45:30.0 - 10:47:00.0     7  1445+09900002  [1]
Fields: 3
  ID   Name          Right Ascension  Declination   Epoch
  1    1331+30500002 13:31:08.29      +30.30.32.96  J2000   ** Flux/bandpass cal, 3C 286
                                                               (J2000 name & coords.)
  2    1445+09900002 14:45:16.47      +09.58.36.07  J2000   ** Gain calibrator
  3    N5921         15:22:00.00      +05.04.00.00  J2000   ** Source
Data descriptions: 1 (1 spectral windows and 1 polarization setups)
  ID    Ref.Freq    #Chans  Resolution  TotalBW     Correlations
  1     1413.45MHz  63      24.4141kHz  1550.2 kHz  RR  LL
Feeds: 28: printing first row only
  Antenna   Spectral Window     # Receptors    Polarizations
  1         -1                  2              [         R, L]
Antennas: 27:
  ID   Name  Station   Diam.    Long.         Lat.
  1    1     VLA:N7    25.0 m   -107.37.07.2  +33.54.12.9
  2    2     VLA:W1    25.0 m   -107.37.05.9  +33.54.00.5
  3    3     VLA:W2    25.0 m   -107.37.07.4  +33.54.00.9
  4    4     VLA:E1    25.0 m   -107.37.05.7  +33.53.59.2
  5    5     VLA:E3    25.0 m   -107.37.02.8  +33.54.00.5
  6    6     VLA:E9    25.0 m   -107.36.45.1  +33.53.53.6
  7    7     VLA:E6    25.0 m   -107.36.55.6  +33.53.57.7
  8    8     VLA:W8    25.0 m   -107.37.21.6  +33.53.53.0
  9    9     VLA:N5    25.0 m   -107.37.06.7  +33.54.08.0
  10   10    VLA:W3    25.0 m   -107.37.08.9  +33.54.00.1
  11   11    VLA:N4    25.0 m   -107.37.06.5  +33.54.06.1
  12   12    VLA:W5    25.0 m   -107.37.13.0  +33.53.57.8
  13   13    VLA:N3    25.0 m   -107.37.06.3  +33.54.04.8
  14   14    VLA:N1    25.0 m   -107.37.06.0  +33.54.01.8
  15   15    VLA:N2    25.0 m   -107.37.06.2  +33.54.03.5
  16   16    VLA:E7    25.0 m   -107.36.52.4  +33.53.56.5
```

```
  17   17    VLA:E8    25.0 m   -107.36.48.9  +33.53.55.1
  18   18    VLA:W4    25.0 m   -107.37.10.8  +33.53.59.1
  19   19    VLA:E5    25.0 m   -107.36.58.4  +33.53.58.8
  20   20    VLA:W9    25.0 m   -107.37.25.1  +33.53.51.0
  21   21    VLA:W6    25.0 m   -107.37.15.6  +33.53.56.4
  22   22    VLA:E4    25.0 m   -107.37.00.8  +33.53.59.7
  24   24    VLA:E2    25.0 m   -107.37.04.4  +33.54.01.1
  25   25    VLA:N6    25.0 m   -107.37.06.9  +33.54.10.3
  26   26    VLA:N9    25.0 m   -107.37.07.8  +33.54.19.0
  27   27    VLA:N8    25.0 m   -107.37.07.5  +33.54.15.8
  28   28    VLA:W7    25.0 m   -107.37.18.4  +33.53.54.8


Tables:
   MAIN                   22653 rows
   ANTENNA                   28 rows
   DATA_DESCRIPTION           1 row
   DOPPLER             <absent>
   FEED                      28 rows
   FIELD                      3 rows
   FLAG_CMD             <empty>
   FREQ_OFFSET         <absent>
   HISTORY                  270 rows
   OBSERVATION                1 row
   POINTING                 168 rows
   POLARIZATION               1 row
   PROCESSOR            <empty>
   SOURCE              <absent> (see FIELD)
   SPECTRAL_WINDOW            1 row
   STATE                <empty>
   SYSCAL              <absent>
   WEATHER             <absent>
```

AIPS Task/Verb	Description	AIPS++ tool/function
IMFIT	Image-plane component fitter	imagefitter
IMLOD	FITS image filler	image.imagefromfits
IMH	File header summary	table.summary
IMSTAT	Image statistics	image.statistics
ISPEC	Plot image slice	*not yet available*
JMFIT	Image-plane component fitter	imagefitter
LISTR	List uv and calibration data	tablebrowser
MCAT	List image catalog	catalog
MK3IN	MK3 VLBI filler	*not yet available*
MOVE	Move uv-data files	tablecopy
MSORT	Sort uv-data	tablecommand
MX	Synthesis imaging	imager
PBCOR	Primary beam correction	imager.setvp, vpmanager
PCAL	Polarization calibration	calibrater.solve
PRTAB	List table data	tablebrowser
PRTAN	List AN table data	tablebrowser
PRTUV	List uv-data	tablebrowser
RENAME	Rename file name	tablerename
SETJY	Set source properties	imager.setjy
SNPLT	Plot calibration solutions	calibrater.plotcal
SUBIM	Extract sub-image	image.subimage
SPFLG	Interactive line data editing	*not yet available*
SPLIT	Apply calibration	calibrater.correct, map.applycal
	Split out calibrated data	*not yet available*
TAPLT	General table plotting	pgplotter, table
TBIN	Read table from ASCII format	tablefromascii
TBOUT	Write table to ASCII format	table.toascii
TRANS	Transpose an image	not required in AIPS++
TVBOX	Set regions in an image	regionmanager
TVFID	Adjust TV display	viewer
TVFLG	Interactive TB data editing	msplot
TVLOD	Load image to TV display	viewer
UCAT	List uv-data catalog	catalog
UVCON	Simulator	simulator
UVFIT	uv-plane component fitter	*not yet available*
UVFLG	Command-based flagging	flagger
UVLOD	UV-FITS filler	ms.fitstoms
UVMOD	Simulator	simulator
UVPRT	List uv-data	tablebrowser
UVSRT	Sort uv-data	tablecommand
UVSUB	Source model computation	imager.ft
VBGLU	Concatenate VLBI data	*not yet available*
VBPLT	Baseline-based uv-data plotting	msplot
VMEM	MEM deconvolution	imager.mem
VPLOT	Baseline-based uv-data plotting	msplot
ZAP	Delete a file	tabledelete

Table 1.5: MIRIAD – AIPS++ dictionary

MIRIAD Task	Description	AIPS++ tool/function
atlod	load ATCA data	atcafiller
blflag	Interactive baseline based editor/flagger	msplot raster displays
cgcurs	Interactive image analysis	Viewer
cgdisp	Image display, overlays	Viewer
clean	Clean an image	imager
fits	FITS image filler	image.imagefromfits
gpboot	Set flux density scale	calibrater.fluxscale
gpcal	Polarization leakage and gain calibration	calibrator with 'G' and 'D'
gpcopy	copy calibration tables	*generally not needed*
gpplt	Plot calibration solutions	calibrater.plotcal
imcomb	Image combination	imager
imfit	Image-plane component fitter	imagefitter
impol	Create polarization images	imagepol
imstat	Image statistics	image.statistics
imsub	Extract sub-image	image.subimage
invert	Synthesis imaging	imager
linmos	linear mosaic combination of images	imager
maths	Calculations involving images	images.imagecalc, images.calc
mfcal	Bandpass and gain calibration	calibrator with 'G' and 'B'
prthd	Print header of image or uvdata	image.summary, ms.summary
restor	Restore a clean component model	imager
selfcal	selfcalibration of visibility data	imager, calibrater
tvclip	automated flagging based on clip levels	autoflag
tvdisp	Load image to TV display	viewer
tvflag	Interactive TB data editing	msplot
uvaver	Average/select data, apply calibration	*average not available*
uvfit	uv-plane component fitter	*not yet available*
uvflag	Command-based flagging	flagger
uvgen	Simulator	simulator
uvlist	List uv-data	tablebrowser
uvmodel	Source model computation	imager.ft
uvplt	uv-data plotting	ms
uvsplit	split uv file in sources and spectral windows	*not yet available*

Chapter 2

GBT Standard Data Reduction Guide

GBT Reduction Guide

Chapter 3

ATCA reduction

Mark Wieringa

3.1 Introduction

This chapter describes how to reduce data from the ATNF's Compact Array (ATCA) in AIPS++. It should be read in conjunction with several other chapters in *Getting Results*, including the chapters which cover i) synthesis calibration, ii) synthesis imaging (including general imaging and mosaicing), iii) display of data, and iv) image analysis.

The data reduction categories covered in this chapter include those for continuum data, continuum polarimetry, spectral line data, and spectral line polarimetry. Individual steps in each data reduction sequence are illustrated using Glish script excerpts. These data reduction steps can equivalently be executed from the Toolmanager GUI interface.

The reduction sequences described in this chapter require some familiarity with the `tools` listed here:

atcafiller is used to load ATCA RPFITS file data into an AIPS++ MeasurementSet (MS).

ms is used to access the MS (e.g. list and summarize visibility data).

msplot is used for plotting and editing of visibility data, and is useful for investigation and editing during calibration cycles.

Calibrater is used to apply or solve for a range of antenna-based calibration effects (like gain, bandpass and leakage), and to plot calibration solutions.

Imager is used to image (Fourier Transform, deconvolve etc.) the data held in the MeasurementSet (MS).

Viewer is used to examine and display data and perform related operations.

Image is used to access and analyse images.

The User Reference Manual contains the primary documentation for all these tools and their functions. The aim of this document is to provide pointers to the relevant chapters and examples, and to describe their use in ATCA reduction.

Users are also referred to the Miriad-aips++ dictionary which is a separate chapter in *Getting Results*, which provides a mapping of some common Miriad tasks to their counterparts in AIPS++.

3.2 MeasurementSets

Visibility data in AIPS++ are stored in an AIPS++ table known as a MeasurementSet (MS). An MS consists of a main table containing the visibility data and associated sub-tables containing auxiliary or secondary information. The full MS data format is described in AIPS++ Note 229. Tables 1 & 2 identify the commonly accessed components of the MAIN table and list data selection parameters used during typical data reduction sessions.

All AIPS++ data files, including MeasurementSets, are written into the current working directory by default, with each AIPS++ Table represented as a separate sub-directory. MS names therefore need only comply with UNIX file or directory naming conventions, and can be referred to from within AIPS++ directly, or via full pathnames.

Each row of the DATA column in the MS contains a matrix of observed complex visibilities at a single time stamp, for a single baseline in a single spectral window. The shape of the data matrix is given by the number of channels and the number of correlations (voltage-products) formed by the backend instrument of the telescope (e.g. a correlator for an array, or a spectrometer for a single dish).

Table 3.1: Commonly accessed MAIN Table components (Data, Flags, & Weights *only*)†

Name	Format††	Comments
DATA	Complex(N_c, N_f)	Complex visibility matrix
		Observed DATA for synthesis arrays
WEIGHT	Float(N_c)	Weight for whole data matrix
FLAG	Bool(N_c, N_f^*)	Cumulative data flags
CORRECTED_DATA	Complex(N_c, N_f)	Corrected data created by calibrater or imager tools
MODEL_DATA †††	Complex(N_c, N_f)	Model data created by calibrater or imager tools
IMAGING_WEIGHT	Float(N_c)	Imaging weight, e.g. uniform, natural, robust
		Created by calibrater or imager tools

† Additional data attributes are described in AIPS++ Note 229

†† N_c= number of correlators, N_f= number of frequency channels.

† † † The MODEL_DATA column is created with each value having AMP = 1 & phase = 0°.

Table 3.2: Common Data Selection Parameters†

Parameter	Contents
ANTENNA1	First antenna in baseline
ANTENNA2	Second antenna in baseline
FIELD_ID	Field (source no.) identification
SPECTRAL_WINDOW_ID	Spectral window number (IF no.)
ARRAY_ID	Subarray number
OBSERVATION_ID	Observation identification
POLARIZATION_ID	Polarization identification
SCAN_NUMBER	Scan number
TIME	Integration midpoint time
UVW	UVW coordinates

† Selection of data from an MS currently appears in different guises in the various synthesis tools due to independent development of these tools. An effort to reconcile these differences is underway.

3.3 Basic initialization

To load the basic `tools` likely to be used in ATCA reduction, when working from the command line, the following initialization is recommended at the start of each AIPS++ or Glish command-line session:

```
include 'synthesis.g'
include 'atcafiller.g'
include 'ms.g'
include 'flagger.g'
include 'image.g'
```

This is not required when working from the Toolmanager GUI, in which case the `tools` are loaded automatically when they are selected.

3.4 Filling ATCA data into an AIPS++ MeasurementSet

ATCA data in on-line RPFITS format are read into an AIPS++ MS using the atcafiller `tool`.

The atcafiller tool can be used to read data from disk or CDROM. It allows selection on scan number, frequency band, frequency range and field name using the select function.

To run the atcafiller:

```
include 'atcafiller.g'
#
af := atcafiller('c999.ms', filenames="1999-01-01_0101.C999 1999-01-01_0202.C999",
```

```
                  options='reweight')          # Construct
af.select(firstscan=3);                        # data selection
af.fill()                                      # fill the actual data
af.done()                                      # destroy filler tool
#
m := ms('c999.ms')                             # access the MS
```

In this example, we construct the filler `tool` from from two input RPFITS files, specifying that we wish to apply lag reweighting to suppress ringing and sources. Then the **select** function is used to select which data we wish to read from the RPFITS files. Then we call the **fill** function to actually read and convert the data to the MS.

Data which have been exported from AIPS or Miriad as UVFITS files can be read into an AIPS++ MS using the general UVFITS filler ms.fitstoms. For example:

```
include 'ms.g'
m := fitstoms(msfile='atca.ms', fitsfile='/aips++/data/demo/ATCA.UV')
m.done()
```

3.5 Data examination and inspection

Once the data have been filled to a MeasurementSet, they can be examined or displayed using various tools. The Catalog tool provides an overview of AIPS++ data files at the highest level, as individual AIPS++ tables of different types (e.g. images, MS, etc) in a given directory. The Catalog tool can be started from Glish as:

```
include 'catalog.g'
dc.gui()
```

Here, `dc` is the default catalog tool. It can also be accessed via the Toolmanager interface from the `Tools in Use` button. A mouse click on an individual MS listed in the catalog GUI followed by a click on the Summary button, will write a summary of the data in the MS to the logger (including frequency and polarization settings and fieldnames). This information is useful during calibration.

When you click on the View button (to view the presently selected catalog item) you will launch the tablebrowser which provides facilities to browse and examine all data fields in the MS main and sub-tables.

Note that indices in the MS, such as FIELD_ID, are stored as integers beginning with zero when viewed at the low level provided by the tablebrowser. All AIPS++ synthesis tools correct for this artifact of c++ and present selection of indices beginning at one.

The primary graphical visibility data visualization tool in AIPS++ at present is msplot. msplot allows line and raster plots of the observed, corrected, and model data against a range of MS indices and parameters, as well as associated plots such as uv-coverage. Facilities are provided to support versatile data selection, range setting and plot iteration. Interactive editing is also supported, as described in the next section. It can be started from *Glish* via:

```
include 'msplot.g'
p := msplot(msfile='data.ms',
            edit=T);                # Allow editing as well as inspection
```

This will launch a gui for plot type and data selection. It can also be launched from the toolmanager via the `Synthesis` package, and ms module.

Consult the msplot documentation for more information on using this tool.

3.6 Data editing

Command-based, interactive and automated editing capabilities are supported in the flagger tool.

To start the flagger tool, type the following:

```
fg:= flagger(msfile='mydata.ms')        # start the flagger tool
```

Direct (command-based) editing

The flagger tool offers several directed editing (flagging) operations. In general, this involves selecting the data to be flagged (based in indices, timeranges, etc.), then running a flagging operation. The flagging commands all have the option to run them on a trial basis. To flag data from antenna=5, spw=1, field=2, within a timerange:

```
fg.setantennas(ants=5)                       # Select data including antenna 5
fg.setids(spectralwindowid=1,fieldid=2)      # Select spectral id and field id
fg.settimerange(starttime="24-FEB-2000/10:00:05",  # Select timerange
                endtime="24-FEB-2000/10:09:37");
fg.state();                                  # Review data selection
fg.flag(trial=T);                            # Trial run to see how many rows
                                             #   will be flagged
fg.flag();                                   # Flag it
```

To reset the data selection, type:

```
fg.reset();
```

To flag data from baseline 3-4, spw=2, field=1, within ±10s of a timestamp:

```
fg.setbaselines(ants=[3,4])                 # Select data for baseline 3-4
fg.setids(spectralwindowid=2,fieldid=1)     # Select spectral id and field id
fg.settime(centertime="24-FEB-2000/10:00:05",  # Select time and range
           delta="10s");
fg.state();                                 # Review data selection
fg.flag();                                  # Flag it
```

The logic in the data selection options is as follows. The two time-related data selection options (set by flagger.settimerange and flagger.settime) are logically OR'd together, and the two antenna-related data selection options (set by flagger.setantennas and flagger.setbaselines are also logically OR'd. The net time selections, net antenna selections, and all other selections are then AND'd together. Note that the data selection persists after the flagging operation flagger.flag is executed. To clear the selection, execute the flagger.reset function, or execute individual selection commands without arguments to clear only that particular selection. For example, to flag all data to antenna 11 for the same spectral window, field, and timestamp as for baseline 3-4 above:

```
fg.setbaselines();               # Clears baseline selection
fg.setantennas(ants=11);         # Select antenna 11
fg.state();                      # Review data selection
fg.flag();                       # Flag it
```

To unflag data based on a selection, use the flagger.unflag function. For example, to restore data for baseline 11-15 (that was flagged above):

```
fg.setantennas();                # Clears antenna selection
fg.setbaselines(ants=[11,15]);   # Select baseline 11-15
fg.state();                      # Review data selection (note
                                 #  other selections still in
                                 #  effect)
fg.unflag();                     # undo previous flags
```

Note that use of flagger.unflag may restore data that had been flagged by other means (e.g., by the on-line system, or by other criteria in previous use of flagger or msplot). In the near future, a flexible means of manipulating flagging versions will be made available in AIPS++.

Consult the flagger tool documentation for more information on the details of data selection.

Heuristic editing

Several simple heuristic editing methods are available in the flagger tool. These methods act on the data selected as described above, but add an additional heuristic constraint.

For example, it is usually desirable to flag the auto-correlation data from the VLA (otherwise, the imager tool may attempt to use it in image formation as if it were single dish data). To do this:

```
fg.reset();                  # Reset the data selection
fg.flagac();                 # Flag the auto-correlations
```

It is often the case that the VLA data near the beginning of scans (e.g. after field or frequency band changes is bad. To flag such data, use the flagger.quack function. For example, to remove 20 seconds of data at the beginning of scans (defined by gaps larger than 30 seconds) on field 3:

```
fg.reset();                  # Reset the data selection
fg.setids(fieldid=3);        # Limit quack to fieldid=3 only

fg.quack(scaninterval='30s',  # Trial run of quack
         delta='20s',
         trial=T);

fg.quack(scaninterval='30s',  # Quack it
         delta='20s');
```

Automated editing methods are available in the flagger and autoflag tools. These include clipping, median flagging and related methods.

For example, to clip data outside of a specified amplitude range with the flagger tool:

```
fg.setids(fieldid=2, spwid=[1,2]);    # Select field, spw
fg.setpol(pol=[1,4]);                 # Select polarizations 1,4
                                      #  (e.g., RR and LL)
fg.filter(column='DATA',              # Flag data if, DATA column
          operation='range',          #   Amplitudes are outside
          comparison='Amplitude',     #   1.12-1.34
          range=['1.12Jy','1.34Jy'],
          fullpol=T,                  # Extend flags to pols and
          fullchan=T);                #  and chans that weren't
                                      #  tested
```

The autoflag tool has more sophisticated algorithms. Examples of its use will be included here in the near future.

Finishing flagger

To finish the flagger tool:

```
fg.done();     # Done with flagging
```

3.6.1 Interactive editing

Interactive editing is possible on almost all graphical views of the data which are possible in msplot. To enable the editing option, it is necessary to set parameter `edit=T` when the msplot tool is created. The current set of flags will be saved (for possible restoration later). Select and plot data as in non-editing mode, then draw boxes with the left mouse button to select regions to flag and press the `Flag` button to flag it. As many flag boxes as desired may be drawn before executing the flags. Use the `Locate` button to obtain a listing (in the `logger`) of the data points within the box(es). Note that the listing may be very long if the region selected covers a large number of visibility points.

At this writing the channel and polarization flagging scope selections in `msplot` will only work if you have elected to plot all channels and/or polarization. This constraint will be relaxed in the near future.

Raster displays of visibility data are possible also, and maybe useful for spectral line data editing.

3.7 Basic Calibration Fundamentals

Calibration and imaging responsibilities are divided between the calibrater and imager tools. The calibrater tool handles visibility-plane calibration while the imager tool deals with image formation, deconvolution and image-plane calibration.

In a typical synthesis observation, a number of calibrators of known structure (usually point sources, and at least one of known flux density) are observed in addition to the scientific target(s) in order to obtain a measure of the systematic properties of the instrument and, if relevant, the corrupting media above it. The effects of these systematic properties are then removed from the data for the scientific target. This technique is often referred to as *cross-calibration* and its success depends implicitly on the relationship between: 1) the true brightness distribution of the observed radio sources (what we are after); 2) the formal response of the instrument and its variety of imperfections of varying importance (what we are calibrating); 3) and the data that is actually recorded (what we have measured).

The AIPS++ Measurement Equation, originally defined by Hamaker, Bregman, & Sault (1996a, 1996b), is an explicit description of this relationship. The Measurement Equation (in the visibility plane) for a spectral line polarization calibration is given by

$$\vec{V}_{ij} \;=\; B_{ij}\; G_{ij}\; D_{ij}\; P_{ij}\; T_{ij}\; \vec{V}_{ij}^{\,\mathrm{IDEAL}} \tag{3.1}$$

where:

$\vec{V}_{ij}$ = cross-correlations between two polarizations for each of two feeds (i,j) which characterize an individual baseline (e.g. the measured visibilities).

$\vec{V}_{ij}^{\,\mathrm{IDEAL}}$ = visibilities measured by an ideal interferometer (e.g. no instrumental errors or calibration effects). $\vec{V}_{ij}^{\,\mathrm{IDEAL}}$ = is directly related to the Fourier Transform of the true polarized sky brightness, $\vec{I} = (I,Q,U,V)$.

T_{ij} = Complex gain effects which are polarization-independent, e.g. tropospheric effects, high-frequency opacity corrections, antenna gain as a function of elevation, baseline corrections.

P_{ij} = Parallactic angle.

D_{ij} = Instrumental polarization response. "D-terms" describe the polarization leakage between feeds (e.g. how much the R-polarized feed picked up L-polarized emission, and vice versa).

G_{ij} = Electronic gain response due to components in the signal path between the feed and the correlator. This complex gain term G_{ij} includes the scale factor for absolute flux density calibration, and may include phase and amplitude corrections due to changes in the atmosphere (in leiu of T_{ij}).

B_{ij} = Bandpass response.

Note that the Measurement Equation is a matrix equation. $\vec{V}_{ij}$ and $\vec{V}_{ij}^{\text{IDEAL}}$ are 4-vectors with elements for each correlator product, and the T_{ij}, etc., are 4×4 matrices that 'corrupt' the correlator products according to the properties of the specific term. The order of the corrupting terms from right-to-left, for the most part, is the order in which the errors affect the incoming wavefront. AIPS++ hardwires the Measurement Equation order and all of the specific algebra associated with the individual terms. As such, it is usually possible to ignore the fact that the Measurement Equation is a matrix equation, and treat the different terms as labeled 'components' (black boxes) along the signal path.

In practical calibration, it is often best to begin the calibration process by determining solutions for those terms which affect the data most. Thus, the user would normally determine gain solutions (G) first (applying pre-computed parallactic angle corrections P if doing polarization calibration), then bandpass solutions (B) and/or polarization solutions (D). Errors which are polarization-independent (T) can be determined at any point in the calibration process. T has nearly identical form to G but it located at a different point in the Measurement Equation. A practical advantage to this factorization is the ability to store polarization-independent effects which are characterized by a certain timescale or a particular parametrization (e.g., tropospheric opacity) as a separate calibration factor, thus isolating them from electronic gain effects which may vary on a different timescale and/or with different systematics. Although most data reduction sequences will probably proceed in this order, the user has complete freedom and flexibility to determine the order in which calibration effects are solved for. Self-calibration, the process by which an improved model of the target source is used to obtain improved solutions for the calibration, is also not limited to just the electronic gain (G) corrections, as has been traditionally the case. Improved solutions for, say, the instrumental polarization (D) may be obtained by self-calibration as well. In general, decision-making along the calibration path is a process of determining which effects are dominating the errors in the data and deciding how best to converge upon the optimal set of calibration solutions (for all relevant components) and source models which best describe the observed data.

Once the calibration solutions have been determined, the AIPS++ software applies the solutions according to the order defined by the Measurement Equation. The individual calibration components represented in the Measurement

Equation are described in detail in the Synthesis Calibration chapter of `Getting Results`. The generalization of the Measurement Equation to image plane effects is also described there.

The observed data, $\vec{V}_{ij}$, are stored in the DATA column in the MAIN table of the MS; these are the raw data as loaded by the filler tool or imported from a UVFITS file. Associated with the DATA column are related columns to hold the most recent version of the calibrated data (CORRECTED_DATA), and the latest visibility plane representation of the source or field model, $\vec{V}_{ij}^{IDEAL}$, (MODEL_DATA). The latter two columns are filled in by the calibrater and imagertools, respectively. The actual calibration information is stored in separate calibration tables. The observed DATA column does not change during reduction, but the related columns do. When plotting the data using a tool such as msplot, the data sub-types (`observed, corrected, and model`) can be selected individually.

3.7.1 Setting the calibration source model

When solving for a visibility-plane calibration component, the Calibrater `tool` requires that the calibrator model be set first. This is stored in the MODEL_DATA column, and used by the Calibrater tool to form χ^2 when solving for calibration components. In general, this process requires transforming a source model from the image-plane to the visibility-plane, including any image-plane calibration effects. For this reason, the AIPS++ functions for converting source models to the visibility plane are attached to the Imager `tool`.

For the ATCA, the default source model is usually either 1934638 or point sources of unit flux density. If you have used 0823500 as your primary calibrator, or given 1934638 another name, you'll need to set the flux density for it by hand (see below).

To compute source models of this type:

```
imagr:= imager('atca.ms');         # Create an imager tool
imagr.setjy(fieldid=1);            # Compute source model for primary calibrator OR
imagr.setjy(fieldid=1,spwid=1,fluxdensity=5.6);    # Set flux density explicitly for source
imagr.setjy(fieldid=1,spwid=2,fluxdensity=5.8);
imagr.done()
```

In these examples, the source is specified through its `fieldid` argument. Running the Imager.setjy function without arguments will compute the default point source flux density for all recognized standard sources for all spectral windows in the data. The Imager.setjy function also allows selection by field id. and spectral window id. as required, as well as the specification of an arbitrary Stokes (I, Q, U, V) for individual sources where necessary. The standard sources recognized are [(3C286, 1328+307, 1331+305), (3C48, 0134+329, 0137+331), (3C147, 0538+498, 0542+498), (3C138, 0518+165, 0521+166), (1934-638), and (3C295, 1409+524, 1411+522)]. At the present time, the flux density is computed for the frequency mid-point of each spectral window using the Perley-Taylor (1995) spectral index coefficients by default. All unrecognized sources, will be set to unit Stokes I flux density. (Note that the MODEL_DATA for all sources is already set to unity when this column if first created, so you don't normally need to do this again here.)

Arbitrary source models can be set using the Imager.ft method, which will transform images and source model components, as required.

3.7.2 Solving for complex gain, bandpass and leakage

The individual calibration effects in the ME, which are expressed as Jones matrices, are described in the Synthesis Calibration chapter of `Getting Results`, and the reader is referred there for a more detailed discussion. Each Jones matrix has a specific type; the complex electronic gain is the `G Jones` component, the bandpass gain is the `B Jones` component and the polarization leakage is the `D Jones` component.

For ATCA data with 4 polarizations you will want to solve for G, D and B. If you have spectral-line data with only 1 or 2 polarizations, then you need solve for G and B only. You can solve for more than one spectral window at a time. Solving for `D Jones` with XX/YY only data will not do anything useful, but generally won't harm either.

First solve for `G, D and B Jones` using the primary calibrator:

```
cal := calibrater('atca.ms')                     # Create a calibrater tool
cal.setdata(msselect='FIELD_ID==1')              # Select the primary calibrator (in this case id=1)
cal.setapply(type='P', t=10.0);                  # Correct for parallactic angle with 10 sec sampling.
cal.setsolve(type='G', t=30.0, table='cal.G')    # Solve for G Jones in 30 sec intervals
cal.setsolve(type='B', t=1.e6, table='cal.B')    # Solve for B Jones with an interval larger than your observation
cal.setsolve(type='D', t=1.e6, table='cal.D')    # Solve for D Jones with an interval larger than your observation
cal.solve();                                     # Do the actual solution
cal.plotcal(tablename='cal.G');                  # Plot the calibration solution obtained
cal.done()                                       # Destroy the calibrater tool
```

Now solve for the `G Jones` term using the secondary calibrator(s), e.g.:

```
cal := calibrater('atca.ms')                               # Create a new calibrater tool
cal.setdata(msselect='FIELD_ID==2')                        # Select the secondary calibrator (in this case id=2)
cal.setapply(type='P', t=10.0)                             # Correct for parallactic angle with 10 sec sampling.
cal.setapply(type='B', table='cal.B')                      # Apply B Jones solved for on primary
cal.setapply(type='D', table='cal.D')                      # Apply D Jones solved for on primary
cal.setsolve(type='G', t=60.0, table='cal.G',append=T) # Solve for G Jones in 60 sec intervals; append output
cal.solve()                                                # Do the actual solution
cal.plotcal(tablename='cal.G',plottype='PHASE',fields=2);  # Plot the calibration solution obtained
```

Generally you should use the above scheme, but if you want or need to (e.g. if you used 0823-500 as your primary calibrator), you can solve for D (and/or B) using the secondary instead of the primary, and just use the latter for the absolute fluxscale. If you want to solve for D using the secondary the process would look like this:

```
cal := calibrater('atca.ms')                     # Create a calibrater tool
cal.setdata(msselect='FIELD_ID==1')              # Select the primary calibrator (in this case id=1)
cal.setapply(type='P', t=10.0);                  # Correct for parallactic angle with 10 sec sampling.
cal.setsolve(type='G', t=30.0, table='cal.G')    # Solve for G Jones in 30 sec intervals
cal.setsolve(type='B', t=1.e6, table='cal.B')    # Solve for B Jones with an interval larger than your observation
cal.solve();                                     # Do the actual solution
cal.done()                                       # Destroy the calibrater tool
```

Now solve for the `G Jones` and `D Jones` terms using the secondary calibrator, e.g.:

```
cal := calibrater('atca.ms')                                 # Create a new calibrater tool
cal.setdata(msselect='FIELD_ID==2')                          # Select the secondary calibrator (in this case id=2)
cal.setapply(type='P', t=10.0)                               # Correct for parallactic angle with 10 sec sampling.
cal.setapply(type='B', table='cal.B')                        # Apply B Jones solved for on primary
cal.setsolve(type='G', t=60.0, table='cal.G',append=T) # Solve for G Jones in 60 sec intervals; append output
cal.setsolve(type='D', t=1.e6, table='cal.D')                # Solve for D Jones using secondary
cal.solve()                                                  # Do the actual solution
cal.plotcal(tablename='cal.G',plottype='PHASE',fields=2);  # Plot the calibration solution obtained
```

Note that the `G Jones` term, although designated as electronic gain, will include an atmospheric contribution, as solved for here. If separable, atmospheric corrections (e.g., WVR corrections) can be separately set or solved for as `T Jones`. The inclusion of a parallactic angle correction `P Jones` is required for all ATCA reduction, see the following section.

3.7.3 Correcting the gains for polarization of the calibrators

The gain solutions were generated with an unpolarized unit point source model for all calibrators unrecognized by Imager.setjy. In practice calibrators often have noticable linear polarization of up to 10% or so. This is not a problem for telescopes with circular feeds, but for telescopes with linear feeds, like the ATCA, this causes a small time variation in the gains proportional to the percentage polarization of the calibrator. The calibrator polarization can be determined from the variation of the X and Y gain solutions with changing feed position angle. After determining the polarization we can correct the gains for this effect. Note that this only works correctly if the calibrator has been observed over a sufficient range in position angle. Generally 5 or more scans with a span of at least 90 degrees in position angle is sufficient. The quality of the polarization solution can be checked by looking at the fit error, it should generally be smaller than 0.5% in Q and U. The correction is performed using the Calibrater.linpolcor function.

```
cal.linpolcor(tablein='cal.G', fields="0405-385 2254-367");
```

This corrects the calibration table entries for sources 0405-385 and 2254-367 and updates the table. If you add a `tableout` argument, it will write the updated table to a new file. (Note: using double quotes like in the example above produces a vector of values, split at each space, another way to specify the fields argument would be: `fields=['0405-385','2254-367']`).

3.7.4 Establishing the flux density scale

For point sources of unknown flux density, but for which valid solutions have been derived against a unit point source model, the absolute flux density can be established by scaling the mean gain moduli against those of the known amplitude calibrators recognized by Imager.setjy. This scaling is achieved by adjusting the `G Jones` calibration table, after determining the amplitude calibration factors, as:

```
cal.fluxscale(tablein='cal.G', reference='1934-638',     # scale the gains
              transfer="0405-385 2254-367");
cal.done();
```

In this case, the amplitude scale is set with 1934638 as the reference. The calibration table `cal.G` will be updated with the scaled `G Jones` calibration factors, and will be used going forward in the reduction.

3.8 Imaging

Imaging is performed using the Imager `tool`, which supports a broad range of imaging modes and deconvolution algorithms, as described in the Synthesis Imaging chapter in `Getting Results`.

3.8.1 Correct the data

Before imaging, the data need to be corrected by applying the Jones matrices determined in the calibration:

```
cal:= calibrater('atca.ms')
cal.setapply(type='P', t=10.0)           # Apply parallactic angle
cal.setapply(type='G', table='cal.G')    # Apply the G Jones solutions
cal.setapply(type='D', table='cal.D')    # Apply the D Jones solutions
cal.setapply(type='B', table='cal.B')    # Apply the B Jones solutions
cal.correct()                            # Correct the data
cal.done()
```

This action fills the CORRECTED_DATA column, which can be used directly in further imaging.

3.8.2 Basic imaging

The Imager tool, which can be used to make images for each field or source, requires initial selection of the visibility data and initialization of the imaging coordinates. Imaging, deconvolution and image restoration operations are then possible, as described in the Imaging chapter of `Getting Results` or in more detail in the Imager documentation. A single-field, continuum ATCA image can be made as follows:

```
imagr := imager('atca.ms')                           # Create an imager tool
imagr.setdata(fieldid=3, mode='channel', nchan=25,  # Select field id. 3, 25 channels
              start=5, step=1, spwid=2)             # and spectral window id. 2
#
imagr.setimage(nx=256, ny=256, cellx='0.4arcsec',   # Set imaging parameters
               celly='0.4arcsec', stokes='IQUV',    # use mfs mode to combine all selected channels
               mode='mfs', nchan=1)
#
imagr.clean(algorithm='clark', niter=500, gain=0.1, # Image, and deconvolve using the Clark CLEAN.
            threshold='0.0Jy',
            model='1331-g1.model',                  # Save model
            image='1331-g1.restored',               # Save restored
            residual='1331-g1.residual')            # Save residual
imagr.done();
#
im:= image('1331-g1.restored')                      # View restored image
im.view()
#
im2 := image('1331-g1.residual');                   # View residual image
im2.view()
```

Note that the deconvolved image model is written to the image file *1331-g1.model*. If non-zero on entry to the **clean** function, it will be subtracted before imaging. This is how you restart the deconvolution. In the above example we are not restarting so the model image is made for you.

The imaging parameters may be selected automatically using Imager.advise (not recommended for ATCA data at this time). For further information on any imaging parameters or options, please consult the full Imager documentation.

3.9 References

Hamaker, J.P., Bregman, J.D., and Sault, R.J., 1996a, *A&AS*, **117**, 137.

Sault, R.J., Hamaker, J.P., and Bregman, J.D., 1996b, *A&AS*, **117**, 149.

3.10 Acknowledgements

This chapter is based on the VLA chapter, with modifications where appropriate.

Chapter 4

Parkes Multibeam Reduction

Stacy Mader

4.1 Introduction

This chapter describes how to use AIPS++ in reducing Multibeam data from the ATNF's Parkes Radiotelescope. Data reduction is controlled via the graphical user interfaces (GUIs) Livedata, Gridzilla and Cubecat.

4.2 Configuring AIPS++

The source code for AIPS++ is available (at the ATNF) in the directory /nfs/aips++/stable. To setup paths for glish libraries, etc., perform the following:

```
% source /nfs/aips++/stable/aipsinit.csh (for csh/tcsh shells)
% . /nfs/aips++/stable/aipsinit.sh       (for korn/bash shells)
```

4.3 Livedata

Livedata is the on-line and off-line processing software for data taken with the Parkes multibeam receiver, though has the capability of processing data from other telescopes. The principal task of Livedata is to remove the bandpass which is the dominant component of every raw spectrum. Livedata also calibrates the spectra, applies doppler tracking, smoothes and baselines the spectra, and has an excellent capability for visualising calibrated spectra. To startup Livedata, type:

```
% livedata   ..or..
% glish -l livedata.g
```

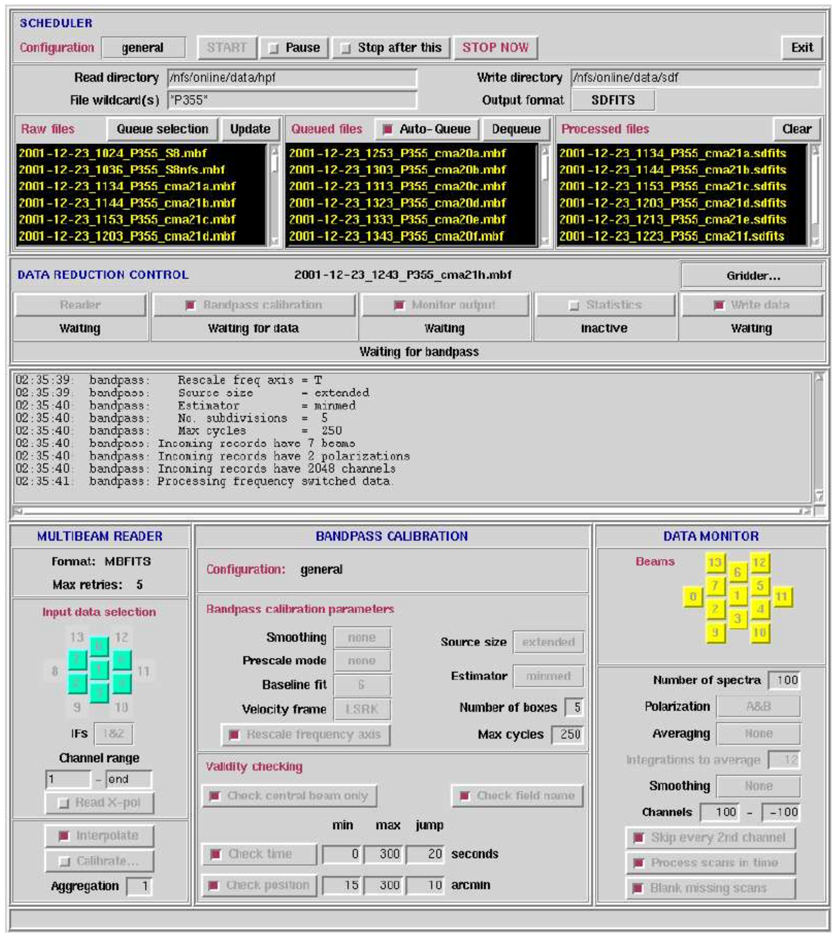

Figure 4.1: The Livedata GUI with all clients enabled.

Livedata directs and regulates the data flow between the following seven clients:

```
Scheduler
       -> Reader
              -> Bandpass calibrator
                                  -> Monitor
                                  -> Statistics
                                  -> Writer
                                             -> Gridder
```

The flow of data is from left to right. All clients other than the reader may be disabled or short-circuited, i.e. any line(s) of the above may be bypassed other than the first. When selected, each client has a control panel on the Livedata GUI. By default, only the reader client is enabled at startup. A detailed description of each client and input arguements (via the GUI) are made below.

4.3.1 Scheduler

Figure 4.2 shows the Scheduler, which provides for realtime (live) and off-line data reduction. Realtime reduction differs from off-line reduction in that files newly created by the correlator may be discovered automatically (auto-queued) and that several attempts are made to read files which may be incomplete.

- **Configuration:** general, HIPASS, HVC, ZOA
- **Read Directory:** Input file directory (default $MB_HPF_SOURCE_DIR).
- **File wildcard(s):** Wildcard specification(s) for input files.
- **Write Directory** Output file directory (default uses $MB_MSCAL_DESTINATION_DIR).
- **Output format:** 'SDFITS' (recommended) or 'MS2'.
- **Auto-Queue:** Automatically check for new files which may appear in the input directory and add them to the processing queue. When the end of a data file is reached, LiveData will wait for new data to be appended.

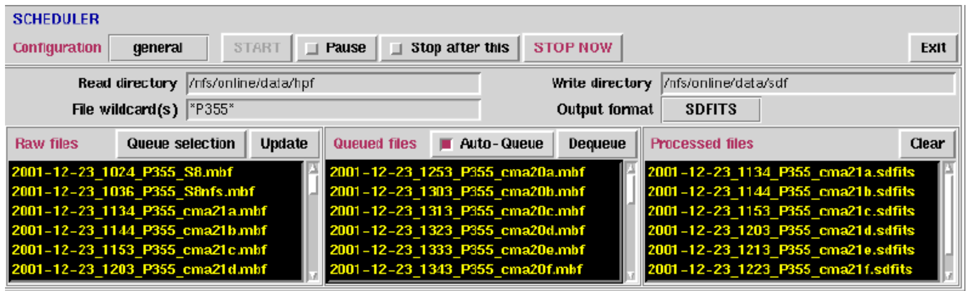

Figure 4.2: The Scheduler panel.

4.3.2 Multibeam Reader

This client reads data from sdfits (usually .sdfits) or rpfits (usually .rpf .mbf or .hpf) files or an AIPS++ Measurement Set (.ms2cal). The input data, which may be selected on a beam-by-beam basis, is packaged as a Glish record for the next client in the chain. The Reader client is enabled by default whereas the others (see below) are disabled.

- **Input data selection:** Mask of beams selected subject to their presence in the data.
- **IFs:** select IF's to read 1&2, 1, or 2.
- **Channel range:** Start/End spectral channel; zero or negative value specifies an offset from the last channel, can also be specified as 'end' or 'last'. Spectral inversion may be achieved by setting endChan < startChan.
- **Read X-pol:** Read cross-correlation data in addition to auto-correlation data. There is presently no option to process or write out cross-correlation data.
- **Interpolate:** When telescope is scanning, apply position interpolation when position and data timestamps are not aligned (rpfits format only).
- **Calibrate:** Apply flux calibration? Once selected, a panel will appear as shown in Figure 4.4. For each beam and polarization, you can enter and apply calibration scale factors to auto-correlation or cross-polarization data. Set to zero to use values stored in raw data headers. For each modified beam and/or polarization factor, T_{sys}, σ and the spectrum are multiplied by the entered factor. For non-zero values, values present in the raw data header are divided by themseleves and then multiplied by the new calibration factors. Options are provided to load/save a calibration file.
- **Aggregation:** Number of scans to aggregate in one Glish record (for efficiency).

4.3.3 Bandpass Calibration

This client does most of the work in calibrating Multibeam data. It operates in several modes depending on the method of observation:

- Scanning mode: as used in HIPASS and ZOA surveys.
- Frequency switching mode.
- Position switching mode: where each beam of the Multibeam receiver is pointed at the source in turn.
- Extended source modes such as used for High Velocity Clouds.

Each of these modes uses a separate bandpass calibration strategy. These are based on robust statistical estimators, particularly median estimators, which allow rejection of RFI without human intervention.

- **Bandpass calibration parameters:**
  - **Smoothing function:** 'Tukey', 'Hanning' or 'none'.
  - **Prescale mode:** Method of prescaling spectra before averaging, 'none', 'mean', or 'median'.
  - **Baseline fit:** Method of post-bandpass residual spectral baseline removal:
    - -1: for no post-bandpass fit,
    - 0: for constant offset (i.e. median),
    - 1: for robust linear fit,
    - >1: for robust (iterative) polynomial fit.
  - **Velocity frame:** Shift spectra to this velocity frame: 'BARY', 'LSRK', 'LSRD', 'LSR', 'GEO', 'TOPO', or 'GALACTO'.
  - **Rescale frequency axis:** The Doppler shift may be applied in either of two ways:
    - **T:** predominantly by scaling the frequency axis parameters but also by shifting the spectrum (via FFT) by a fraction of a channel so that the new reference frequency is an integer multiple of the original channel spacing. This method is more accurate and satisfies gridzilla's requirements, for which it must be used if Doppler tracking was enabled when the observations were made.
    - **F:** by shifting the spectrum (via an FFT) without changing the frequency axis parameters (the HIPASS/ZOA method).
  - **Source size:** Source extent, determines extent of baseline for bandpass estimation, 'compact' or 'extended'.
  - **Estimator:** Statistical estimator used for averaging spectra:
    - median: Median of whole scan.

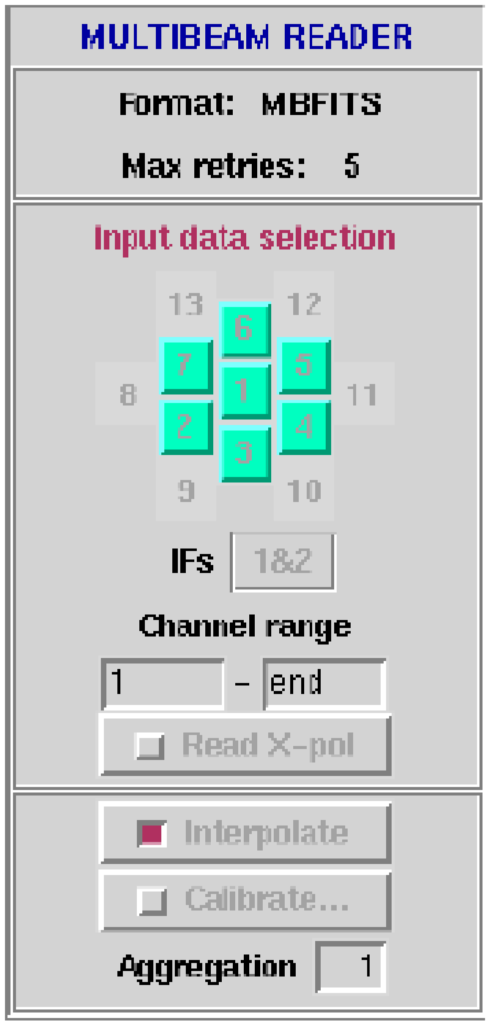

Figure 4.3: The Reader panel.

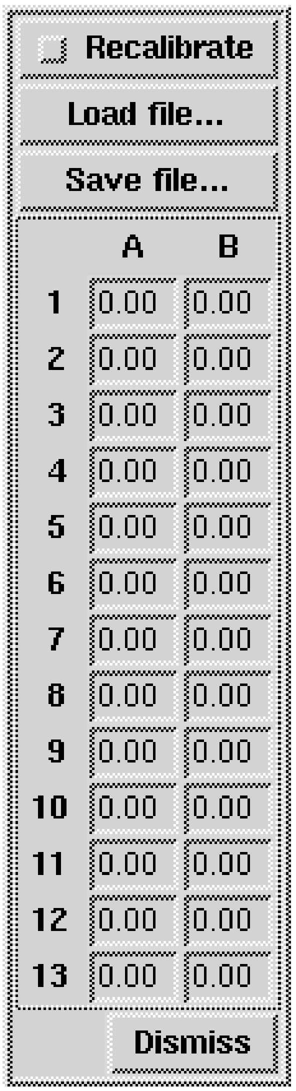

Figure 4.4: The Reader calibration panel.

    * mean: Mean of whole scan.
    * minimum: Minimum of whole scan.
    * minmed: Minimum of the medians of a number of sub-scans.
    * medmin: Median of the minima of a number of sub-scans.
    * none:

  - **Bandpass interval:** The period, in cycles, at which the bandpass is recalculated.
  - **Number of precycles/postcycles:** to store in the buffer and search for valid bandpass spectra.

- **Validity checking:**
  - Check central beam only: For faster processing only do checking (see below) on the central beam.
  - Check field name: Check that the field name hasn't changed between successive integrations?
  - Check time: min/max/jump; these parameters define the allowable time offset between the integration being calibrated and those used for bandpass estimation. Default values for min, max and jump are 0, 300 and 20 seconds respectively.
  - Check position: min/max/jump; these parameters define the allowable position offset between the integration being calibrated and those used for bandpass estimation. Default values for max, min and jump are 15, 300 and 10 arcmin respectively.

Regarding the number of pre– and postcycles, the buffer must be large enough to contain the current scan, the previous and following scans, plus the first integration of the scan after the following scan. For example, if scan 2 is being processed, the buffer must contain all of scans 1, 2, and 3 plus the first integration of 4, the presence of which told us that scan 3 had finished. The reference spectrum for 2 is computed once when the first integration of scan 4 is read, and the first integration of scan 2 is then written out. This space is then available for the second integration of scan 4; each successive integration of scan 4 may overwrite the integration of scan 2 which has just been sent out.
In cases where the buffer size **(precycles + postcycles + 1)** is insufficient (usually in the MX, or beam–switching mode), a warning is issued stating **both** pre– and postcycles should be increased by at least the present value of (precycles + postcycles + 1) plus an overrun value which is stated in the log message. Generally, the pre– and postcycle values can be calculated by: **3*nint**, where nint is the number of integrations specified during observering. By not incrementing the values of the pre– and postcycle, the signal to noise ratio of the reference spectrum becomes degraded.
For the HIPASS, HVC and ZOA options, the pre– and postcycle values are set to 24, which implies a maximum of (24+24+1=49) scans are held in the buffer.

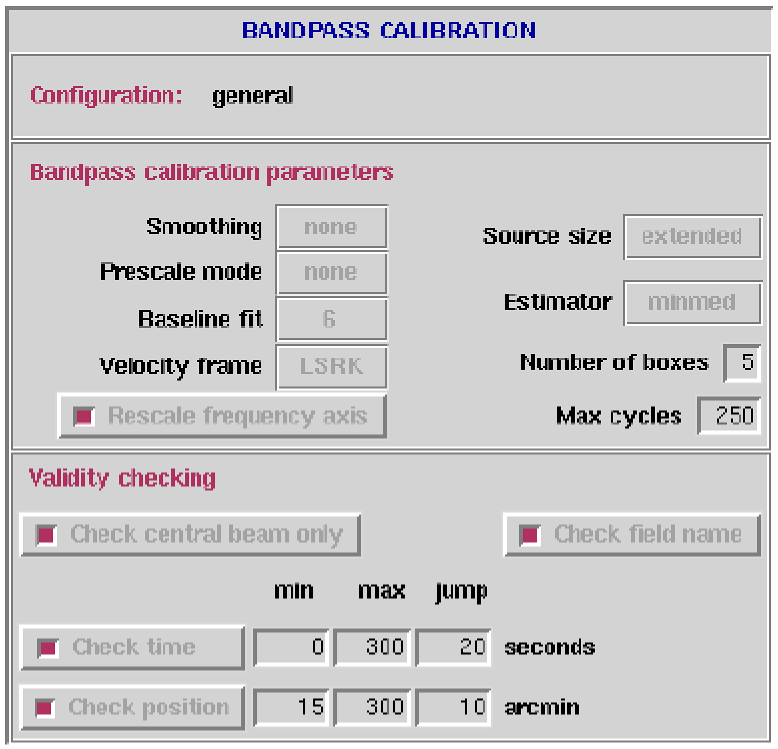

Figure 4.5: The Bandpass Calibration panel.

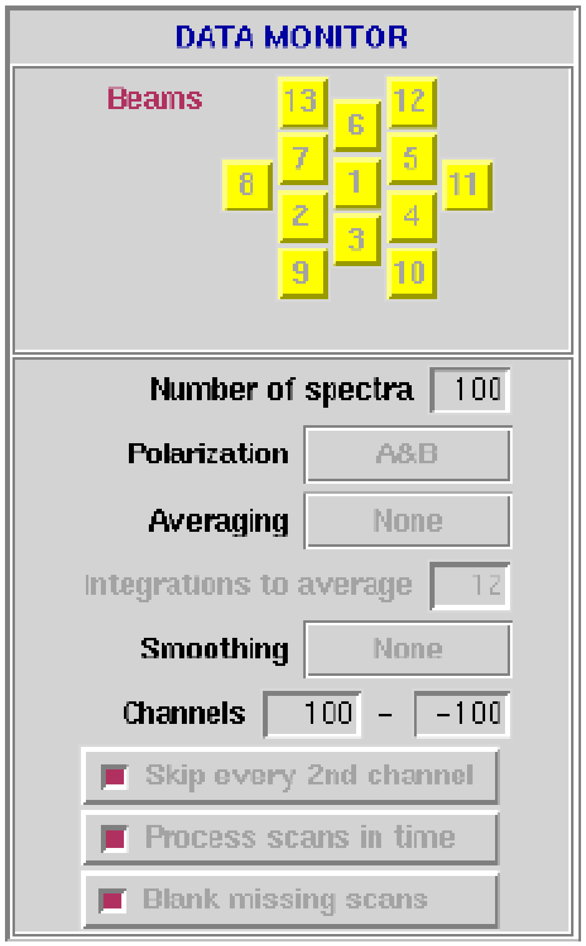

Figure 4.6: The Data Monitor panel.

4.3.4 Data Monitor

The data monitor interfaces to MultibeamView, a specially modified version of the Karma package "kview". In fact, it invokes MultibeamView twice, once for each polarization, to provide two panels displaying frequency versus time with various image enhancement options. This provides visual inspection for each pair of polarizations of each beam, one pair at a time as data arrives. An example ("waterfall") panel is shown in Figure 4.7. It is anticipated that MultibeamVIew will be replace by a native AIPS++ imaging tool in the near future.

- **Beams:** Mask of beams present in data.
- **Number of spectra:** Number of spectra to display.
- **Polarization:** A&B, A only, B only, or A-B.
- **Averaging:** Time averaging method:
  - **None** ...no averaging.
  - **Mean, Median, Maximum or RMS** calculate the relevant statistic for each channel over N integrations; one spectrum is displayed for every N integrations.
  - **Data-median** subtracts the median value for each channel computed over N integrations; the number of spectra displayed is unaffected.
- **Integrations to average:** Number of 5 second scans to use in time averaging. Ignored for timemode "None".
- **Frequency smoothing:** None or Hanning.
- **Channels:** Start/End spectral channel: zero or negative value specifies an offset from the last channel, can also be specified as 'end' or 'last'.
- **Skip every 2nd channel:** If true, skip every second channel of input spectra.
- **Process scans in time:** If true, process scans in time (i.e. output frames have a time axis).
- **Blank missing scans:** If true and "process scans in time" is false, missing scans are not blanked, else missing scans are always blanked.

Changing the MultibeamView displays:

- **Intensity:** After some data has appeared on the plots, set the displayed intensity range for each waterfall by selecting **IScale for Dataset 2** from the **Intensity** menu. This will bring up the **Intensity Zoom** window, and show a histogram of intensity values. Select the intensity range by clicking on the histogram, using the left mouse button for the lowest intensity and the right button for the highest intensity (or you can click on the 95**Close** makes the changes as well as closing the window.

- **Display Colour:** Choose a colour map by selecting **Pseudocolour (8 bit)** from the **Intensity** menu. The **pseudoCmapwinpopup** window displaying a list of colour maps on the RH side will appear. Choose a colour map. Move the dot around in the LH field to change the contrast and darkness.

- **Resizing:** Next, you will have to resize the window so you can get as much of the data on the window as possible. Under the Zoom menu, select the **Zoom Policy** option. The **zoomPolicyPopup** window displaying several options will be presented, but all you have to do is deselect the **Integer X Zoom** and **Integer Y Zoom** boxes. Now click on **Close** to exit the window. The data should now fill a larger part of the waterfall display. Do this for the other polarisation window.

- **Axis Labels:** To display the Frequency (X Axis) and Time (Y Axis) annotations, click on **Overlays**, select the **Axis Labels** option and the **dressingControlPopup** window will appear. Select the **Display Axis Labels** box and **Close** the window. The waterfall should now display Frequency vs Time. Do this for the other polarisation as well.

- **Select Different Beams:** To display different beam outputs, click on the **View** button which will bring up the **View Control** window. Now click on the **Movie** button where you will be presented wit the **Animation Control** window with buttons like **Previous Frame** and **Next Frame**. By clicking on one of these two buttons, the waterfall display will show the appropriate beam (as defined next to the **Number of frames:** line. Similarly, if you move the mouse cursor onto the waterfall display, the currently selected beam is shown in the top RH corner. Click on the **Close** to dismiss the View and Animation Control windows after you are finished.

Obtaining 1-D Profiles within MultibeamView:

To obtain a 1-dimensional spectral profile:

- Click the **View** button.

- Change **profile axis: y** to **profile axis: x**.

- Select **line** from the **profile mode** menu.

The profile changes according to the position of mouse cursor on the XZ (Frequency-Time) image. To Zoom a profile:

- Use the spacebar to freeze the profile you want to zoom.

- Click and drag in the profile window according to the area you want zoomed.
- Back in the main Livedata window, use the spacebar to unfreeze the profile frame.
- Use the **Unzoom** button in the profile window to unzoom. Axis labels can also be added.

4.3.5 Statistics

The statistics client reads through an AIPS++ Measurement Set and computes basic statistical measures for each beam and polarization.

- **Type:** The statistic to be plotted: T_{sys}, mean, median rms, quartile, minima, maxima.
- **Pols:** Polarizations selected for plotting: 1, 2, 1&2, 1+2, 1-2.
- **Beams:** Beams selected for plotting.
- **Save:** Save the current plot.

4.3.6 Writer

Data is written to an AIPS++ Measurement Set by the writer client. These measurementsets are converted to either SDFITS or MS2 format.

4.4 Gridder

Each Multibeam integration consists of a spectrum taken at a particular point on the sky. The gridder (gridzilla) takes the spectra from a collection of bandpass-calibrated measurementsets and writes a 3-plane cube with the total number of spectra or scans per pixel in the first plane, the number of these occurring at nighttime in the second plane, and the number during daytime the third plane.

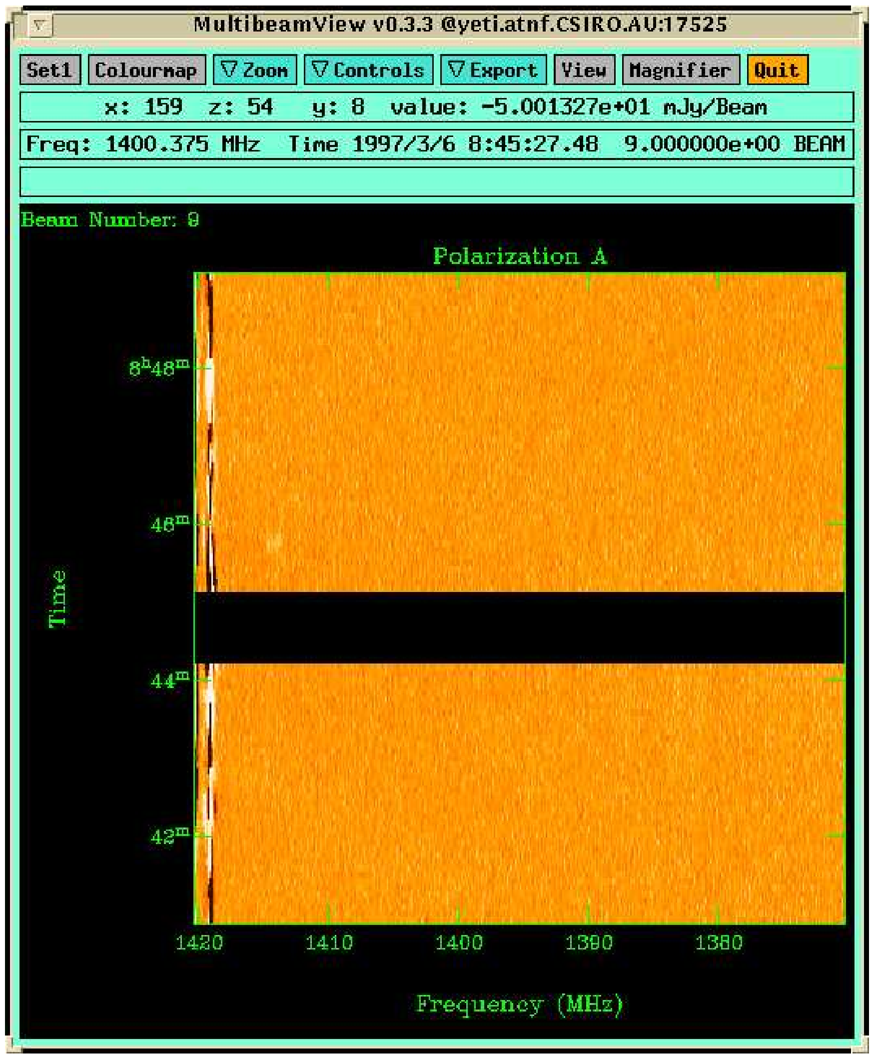

Figure 4.7: A MultibeamView panel displaying frequency vs time for beam 9, Polarization A. A faint galaxy can be seen near 1415 MHz at 08^h46^m.

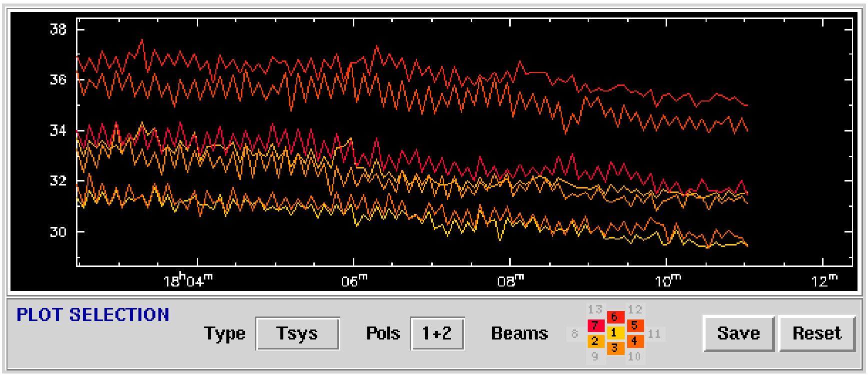

Figure 4.8: The Statistics window displaying T_{sys} for 7 beams.

Typically each point in the sky is sampled many times in separate observations and the gridder combines these using robust statistics to automatically eliminate RFI. Being the most computationally intensive client, the gridder is usually used separately for off-line data reduction, but by pressing the **Gridder** button on the Livedata GUI, SDFITS files are transferred straight to the gridder. In this section, we assume reduction is off-line. To start up gridzilla:

```
% glish -l gridzillarc.g
```

Both the gridzilla (Figure 4.9) and logger windows appear. The GUI contains several panels and entry widgets, each which is described below.

- **Parameter set:** Options are **general**, **HIPASS**, **HVC** and **ZOA**. All except the **general** option has field centres on a fixed grid. For options HIPASS, HVC and ZOA, the gridder runs one of the various 'coverage' scripts to determine the files required for the standard HIPASS, HVC or ZOA cubes. Each script is passsed several arguments according to the survey and field centre selected. An example is given below.

- **Image centre and size:** It is possible to manually input the center coordinates of a region to be gridded and the pixel resolution of the output FITS cube(s). If the **Autosize** widget is selected, the **Image centre** panel disappears and gridzilla automatically determines the image center from the input data. For HIPASS, HVC and ZOA options, the field centres lie on a fixed grid and if selected, both panels are set to default values. For example, if the ZOA option is selected, the Image centre panel displays a menu widget showing ZOA galactic longitudes and the Image size panel shows the output image size (width/height) defaulting to 4 arcmin/pixel.

- **Rest frequency and Frequency/Velocity range:** You can input the line **Rest frequency**, in MHz. The start/end spectral range can be set

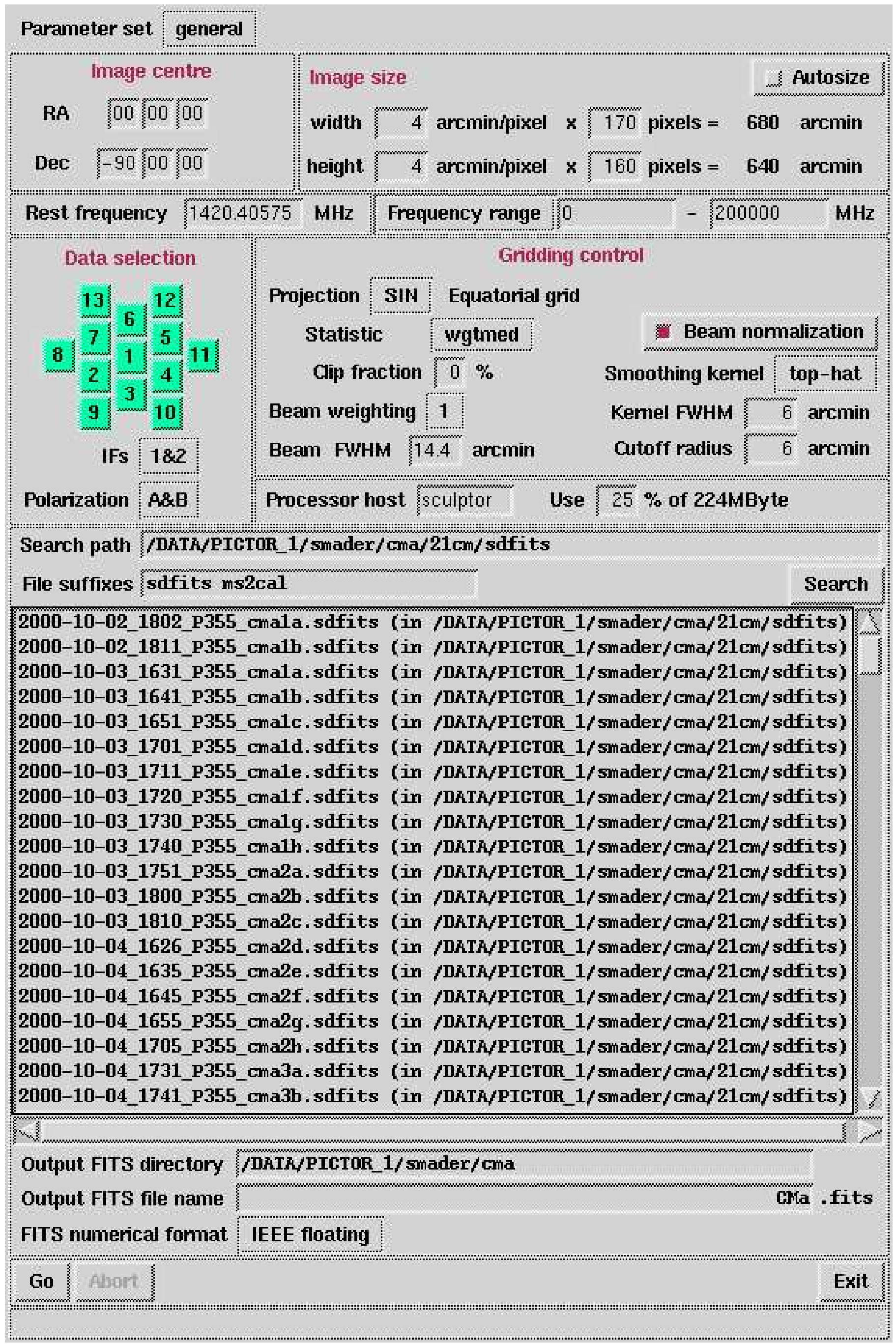

Figure 4.9: The default Gridzilla GUI.

to either **Frequency** (MHz) or **Velocity** (km s^{-1}). By changing from velocity to frequency or vice-versa, the conversion is calculated and displayed.

- **Data selection:**
  - **Input beams:** You can select/deselect input for each beam. If a beam is deselected, its number is grayed out.
  - **IFs:** 1&2, 1, or 2.
  - **Polarization:** A&B, A only, B only, A+B, or A-B.

- **Gridding control:**
  - **Projection:** Output map projection (equatorial grid): SIN, NCP.
  - **Statistic:** Statistical estimator, 'wgtmed', 'median', 'mean', 'rms', or 'quartile'. 'wgtmed' is the weighted median of the values as opposed to 'median' which is the median of the weighted values. Quartile is wgtmed$\{|X - wgtmed(X)|\}$ and measures the inter-quartile range of the pixel values.
  - **Clip fraction:** Percentage of data to discard in the smoothing operation.
  - **Beam weighting:** Beam weighting is based on the beam response calculated for the distance of each spectrum from the pixel.
    * 0: No beam weighting.
    * 1: Weight spectra by beam response.
    * 2: Weight spectra by square of beam response.
  - **Beam FWHM:** FWHM in arcmin of the smoothing kernel.
  - **Beam normalization:** Apply beam normalization?
  - **Smoothing kernel:** top-hat or gaussian.
  - **Cutoff radius:** The smoothing cutoff radius should always be 6 arcmin for HIPASS, HVC and ZOA cubes.

- **Processor host:** You have the option to select the host on which to run gridzilla. The default is to run the client on the startup host. You can also enter a percentage of the processor memory to make available to the gridder client.

- **Search path:** Directory search path for data files. Default is to search for files with suffuxes (see next item) in the '.', '..' and '/DATA/MULTI_1/mscal' directories. Any matches are listed. Any number of valid directories can be entered, seperated by a colon (':').

- **File suffixes:** List files with defined suffixes. Default is to list 'sdfits' and 'mscal' files in paths defined at **Search path**. If additional files are added to **Search path**, the **Search** button can be used to update the input file list.

- **Output FITS directory:** Output FITS directory. Default uses PATH.

- **Output FITS name:** Output FITS file name. The extension is automatically set to 'fits'. If a file with the current file name exists, it will be overwritten. Default file name is 'gridzilla'. The client determines the size of the gridding problem and the number of passes required to stay within the memory limits. If the entire problem can be completed within one pass, the output file name will just be 'gridzilla.fits'. However, if the problem cannot fit within a single pass, the cube is output into multiple FITS files with '_001.fits', '_002.fits', ... , '_00n.fits' appended to the entered file name.

- **FITS numerical format:** Small means 16-bit FITS integer on scale -8 to 32 Jy, else IEEE floating.

4.5 Creating HIPASS cubes: an example.

Although each standard HIPASS, ZOA and HVC cube is created by respective team members, those who wish to create new cubes (using different livedata processing options) can do so by reading on.
During the data taking process at Parkes, each scan is archived onto CDROMs as a raw (HPF) and processed (SDFITS) file. For each CDROM, there is a size file which lists files archived onto it. In order to identify which scans are located on which CDROM(s), a 'coverage' script is available which allows you to pass parameters enabling you to list the required CDROMs to extract scans from. The HIPASS script is 'coverage.pl' and for the HVC and ZOA surveys, they are 'coverage-hvc.pl' and 'coverage-zoa.pl' respectively. For ATNF sites, these scripts are located in /nfs/atapplic/multibeam/bin and each requires the $MB_CATALOG_PATH environment variable to be set where the size files are located:

```
For csh/tcsh shells:
% setenv MB_CATALOG_PATH '/nfs/atapplic/multibeam/archive'

For sh/bash shells:
% set MB_CATALOG_PATH '/nfs/atapplic/multibeam/archive'
```

Say you wanted to create the standard HIPASS cube H035. To identify which scans are on which CDROMs, the coverage script has the following parameters:

```
# t = file type: "SDF" or "HPF"
```

```
# c = cube number (overrides d,s,e)
# d = declination band
# s = start sequence
# e = end sequence
# m = scan letters required (eg "abcde")
# a = auto recurse in read dir eg. for CDs.
# r = read dir
# w = write dir
# x = read ext
# y = write ext
# q = quiet (0,1,2)
# b = (anything) means include bad scans - USE WITH CAUTION!
```

The following command will list all SDF (processed) CDROMs you need to make cube H035:

```
coverage.pl -t SDF -c 035
```

The script goes through all size files and lists the required CDROMs, which in this case, are 68, 71, 156, 157, 160, 161, 166, 196, 280, 296, 360, 408 and 416. You can then configure the coverage script to automatically copy the required files from each loaded CDROM to disk by creating a shell script containing the following:

```
#wait for CDROM to mount:
sleep 10
#now identify scans in cube H035 and copy to disk:
coverage.pl -t SDF \
            -c 035 \
            -r /cdrom/cdrom0 \
            -w /path/to/diskspace \
            -x sdfits \
            -y sdfits \
            -a 0 \
#eject and load another CDROM if required
eject cdrom
```

Once all scans have been copied from CDROM to disk and assuming you have entered the correct path(s) to the SDFITS files in the **Search path** entry box of the gridzilla GUI, selecting the HIPASS option at the **Parameter set** widget brings up the list of standard cubes. Each shows central coordinates and cube name as in Figure 4.10. The central position of cube H035 is shown as 1211-66 (12h 11m 00s, -66d 00m 00s J2000).
Gridzilla invokes the coverage script again and goes through all size files. After the script finishes, the log window lists all scans required to make up the cube and also if each scan was found in the search path(s). In this example we see

that out of all the 'abcde' scans in the cube, only the 'a' scans where found. For this reason, the output FITS file name becomes 'H035_a.fits' instead of H035.fits (which indicates all SDFITS files were found). At this stage you can create the partial cube or load more scans with the coverage shell script.
Once you select all scans and press the **Go** button, the gridzilla log window lists parameters of the output cube such as velocity/frequency range and central pixel coordinates. Depending on the amount of available memory, gridzilla may combine all scans into a single cube, but if not, several "cubelets" will be made. You can concatenate them with **Cubecat** (see below).

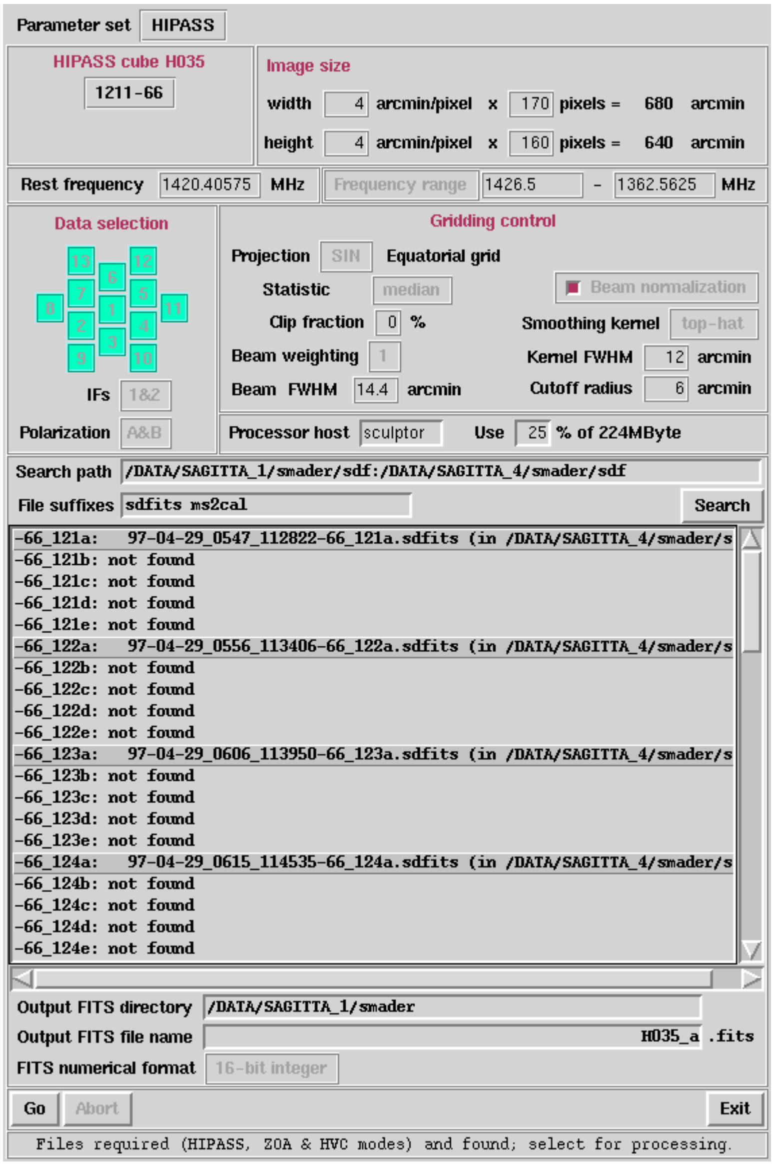

Figure 4.10: Using gridzilla to construct the HIPASS cube H035.

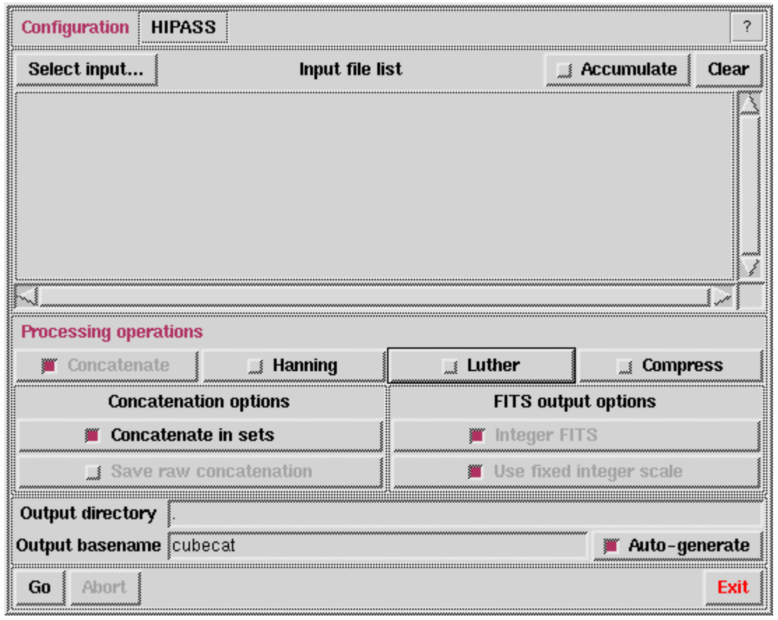

Figure 4.11: The default cubecat GUI.

4.6 Cubecat

Due to memory allocation, gridzilla may not create a single cube. If gridzilla outputs may cubelets, you can concatentate them together with cubecat. Cubecat also allows for post–processing options such as continuum source ripple removal. To start up cubecat:

```
% glish -l cubecat.g
```

The default GUI is displayed in Figure 4.11. Cubecat is self-documenting. For information on what a particular widget does, simply place the mouse cursor on the widget for an explaination. A more detailed description is available by pressing the right-most button on the mouse whilst over the widget. The type and amount of detail available can be altered via the **"?"** widget located at the top RH corner of the main GUI.

- **Configuration:** general, HIPASS, HVC or ZOA
- **Select input:** Brings up a seperate GUI to allow you to enter the directory where your input cubes are located. There is an option **Stay up** which keeps the GUI visible. Under the **Options** list, you can list files alphabetically, time modified and access time. Other options are listed.

- **Accumulate:** If true, add additional files to the input list, otherwise overwrite it.
- **Processing operations:**
  - **Concatenate:** If true, concatenate all files in the input list. Other options include:
    - **Concatenate in sets:** If true, the input list will be split into multiple concatenation sets based on the first four letters of the input file name.
    - **Save raw concatenation:** If true, and concatenation and smoothing and/or luther are selected, then save the raw concatenated file.
  - **Hanning:** If true, Hanning smooth the output. Hanning smoothing is performed on the 3rd spectral axis of the output cube.
  - **Luther:** If true, run luther on the FITS file. In the vicinity of bright continuum sources, there is a gradual rise in flux towards the high–velocity end of the bandpass. As a result, bandpass removal within livedata is over–estimated such that there are negative baselines for low–velocities and positive baselines for high velocities. The LUTHER algorithm makes a polynomial fit to continuum sources in a data cube and then subtracts this fit from each of the spectra in the gridded cube. More details of the LUTHER algorithm can be found in Barnes et al. (2001), MNRAS, **322**, 486–498.
  - **Compress:** If true, compress the output FITS file using gzip. Other FITS output options include:
    - **Integer FITS:** If true, write 16-bit integer FITS data format (with range -8 to +32 Jy) instead of 32-bit IEEE floating. Integer format output is half the size and is also amenable to compression.
    - **Used fixed integer scale.** If true, set the FITS integer scale to -8 to +32 Jy. Otherwise the scale is adjusted to encompass the minimum and maximum values.
  - **Output directory:** Output directory. Default uses PATH.
  - **Output basename:** Output file name; used only if there is one input file or there are several to be concatenated.
  - **Auto-generate:** If true, generate an output file name based on the (first) input cube filename and processing options. For example, if the Hanning option is selected, '-hanning' is appended to the filename.

Chapter 5

Reducing JCMT data in DISH

R. Phillips & J. McMullin

5.1 Setting Your Environment

5.1.1 Starting AIPS++

To start AIPS++, issue the following commands (from the bash shell):

```
> . /aips++/stable/aipsinit.sh
> aips++
```

5.1.2 Making AIPS++ run faster

Here are several hints for speeding up your performance, if the operation/startup of AIPS++ seems sluggish.

- Store cache files on the computer you run from by adding the following line to your .aipsrc file:

  ```
  user.aipsdir: /tmp/username_aips++

  where username is your username, e.g.,
  user.aipsdir: /tmp/rphillips_aips++
  ```

- If you do not want to have the toolmanager GUI available on starting AIPS++, you can suppress it and speed up the startup time by including the following lines in your .aipsrc file:

  ```
  toolmanager.fullgui: F
  toolmanager.gui.auto: F
  ```

The toolmanager isn't directly needed for any DISH operations. Should you want it, it can be used at a later time by typing:

```
tm.gui();
```

- Run AIPS++ from the computer which hosts the disks with your data. Typically the data I/O over NFS is slow and the above bypasses this as much as possible

AIPS++ builds its cache file structure the first time it is started with a new user.aipsdir, and each time you start a new version. Judge its startup speed on the *second* time you start it up, not the first.

5.1.3 Paging

If you find that error messages, code, and data listings scroll off of your terminal window, you might need to assign the **PAGER** environment variable. You can remedy this by setting **PAGER** to 'less' in your UNIX startup scripts.

```
export PAGER='less'
```

5.2 Filling gsd data to an AIPS++ MeasurementSet

```
> GSDfiller
Name of input GSD file: jcmtdemo.dat
Input file : jcmtdemo.dat
Output file: jcmtdemoms
GSD version is 5.3
Using JCMT position from Observatory Table
GSDDataSource::fillSampleData called
n spectral windows is 4
-- filling MAIN Table
-- filling SPECTRAL_WINDOW sub-Table
-- filling ANTENNA sub-Table
-- filling DATA_DESCRIPTION sub-Table
-- filling FEED sub-Table
-- filling FIELD sub-Table
[0.897304580145, 0.536422981884]
[0.897699235533, 0.535695727179]
-- filling OBSERVATION sub-Table
-- filling POINTING sub-Table
-- filling POLARIZATION sub-Table
-- filling PROCESSOR sub-Table
-- filling SOURCE sub-Table
```

```
241800000000
Frequency: 241800000000
Frequency: 241800000000
241800000000
Frequency: 241800000000
Frequency: 241800000000
241800000000
Frequency: 241800000000
Frequency: 241800000000
241800000000
Frequency: 241800000000
Frequency: 241800000000
-- filling STATE sub-Table
-- filling SYSCAL sub-Table
-- filling WEATHER sub-Table
```

5.3 Loading Data into DISH

At this point, you have made an AIPS++ MeasurementSet called 'jcmtdemoms'. This MeasurementSet differs slightly from the standard and so a utility function is available to correct this.

```
> . /aips++/stable/aipsinit.sh
> glish -l jcmtutils.g #this starts up a DISH tool
#and loads some jcmt utility functions
...
Time to initialize DISH =  10.392246
DISH tool is --> d
JCMT   tool is --> jcmt
Glish version 2.7.
- jcmt.fixms('jcmtdemoms');
- d.open('jcmtdemoms');
```

5.4 Analyzing Data

The analysis of JCMT data is similar to other single dish telescopes. Please see the other Getting Results chapter titled "Single Dish Analysis"

5.5 Imaging JCMT data

Imaging is enabled through the imager tool in DISH. It can also be executed as a script from the command line. The following is an example with comments; this assumes that the MeasurementSet has been 'patched' as shown in Section 3.

```
include 'imager.g'#load the imager tool
#
#define the data to be used
thems:='/home/charybdis2/jmcmulli/jcmtdata/obs_das_0064ms'
im:=imager(thems); #create an imager tool

#set which data will be used; in this case each pointing is represented
#as a separate field. For this 2048 channel data, we select 500 channels
#of data from 850 to 1350 (850+500)
im.setdata(fieldid=[1:1593],spwid=1,nchan=500,start=850,step=1);

im.setoptions(gridfunction='SF')#set the gridding function to be SF
im.weight('natural'); #use natural weighting of the data

#the following defines the phase center or center of the mapped field
#this isn't really necessary for single dish data but is a required
#argument.
row:=800;
ptab:=table(spaste(thems,'/POINTING'));
direcs:=ptab.getcol('DIRECTION');
thedir:=direcs[,1,row];
global mydir:=dm.direction('J2000',spaste(thedir[1],'rad'),
       spaste(thedir[2],'rad'));

#set the image characteristics. In this case we construct a 40x40 map
#using 0.79' x 0.79' pixels. We construct a data cube with 50 planes
#which represent 10 channel averages from channel 850 to channel 1350.
im.setimage(nx=40,ny=40,cellx='0.79arcmin',celly='0.79arcmin',stokes='I',
       doshift=T,spwid=1,mode='channel',nchan=50,start=850,step=10,
       phasecenter=mydir);

#make the data cube/image. It is saved on disk as 'scanimage'
im.makeimage(image='scanimage',type='singledish');

#now view it
myimage:=image('scanimage'); #make an image tool in AIPS++ based on scanimage
myimage.view(); #calls up the viewer for seeing the image
```

5.6 To Do List for JCMT/DISH

- Enable filler merge capability for multi-spectral window data
- Enhance gsd-¿MS filler so patch script isn't needed
- Enable DSB scale for plotter

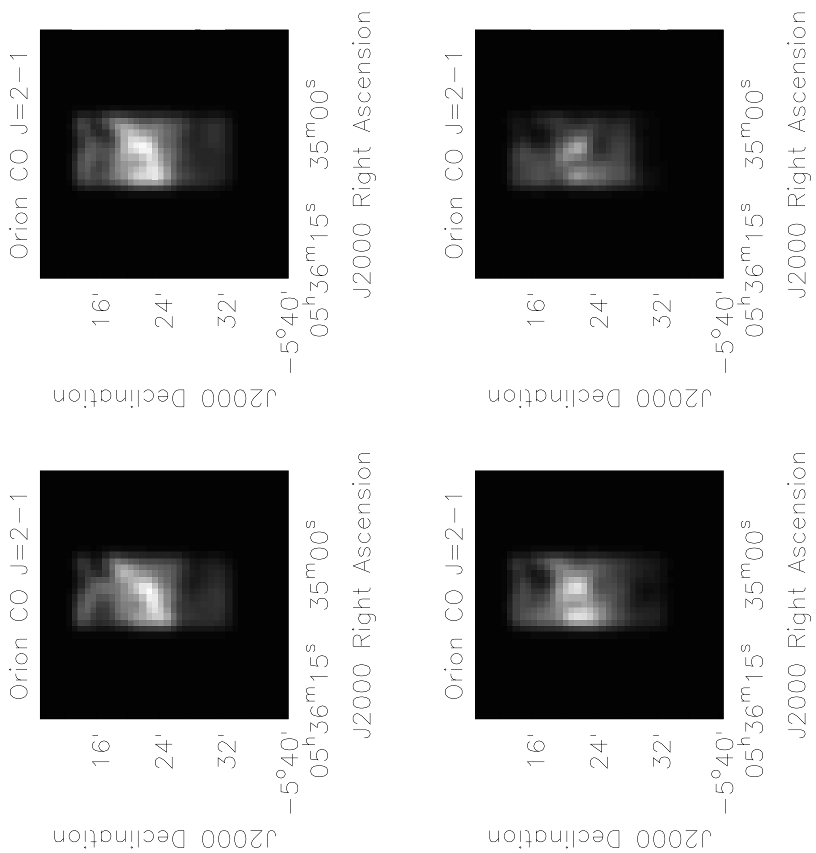

Figure 5.1: Viewer display of 4 channels of the data cube. Multiple planes can be viewed by selecting the "Canvas Manager" option from the "File" menu. Also see the Getting Results chapter 3.11.

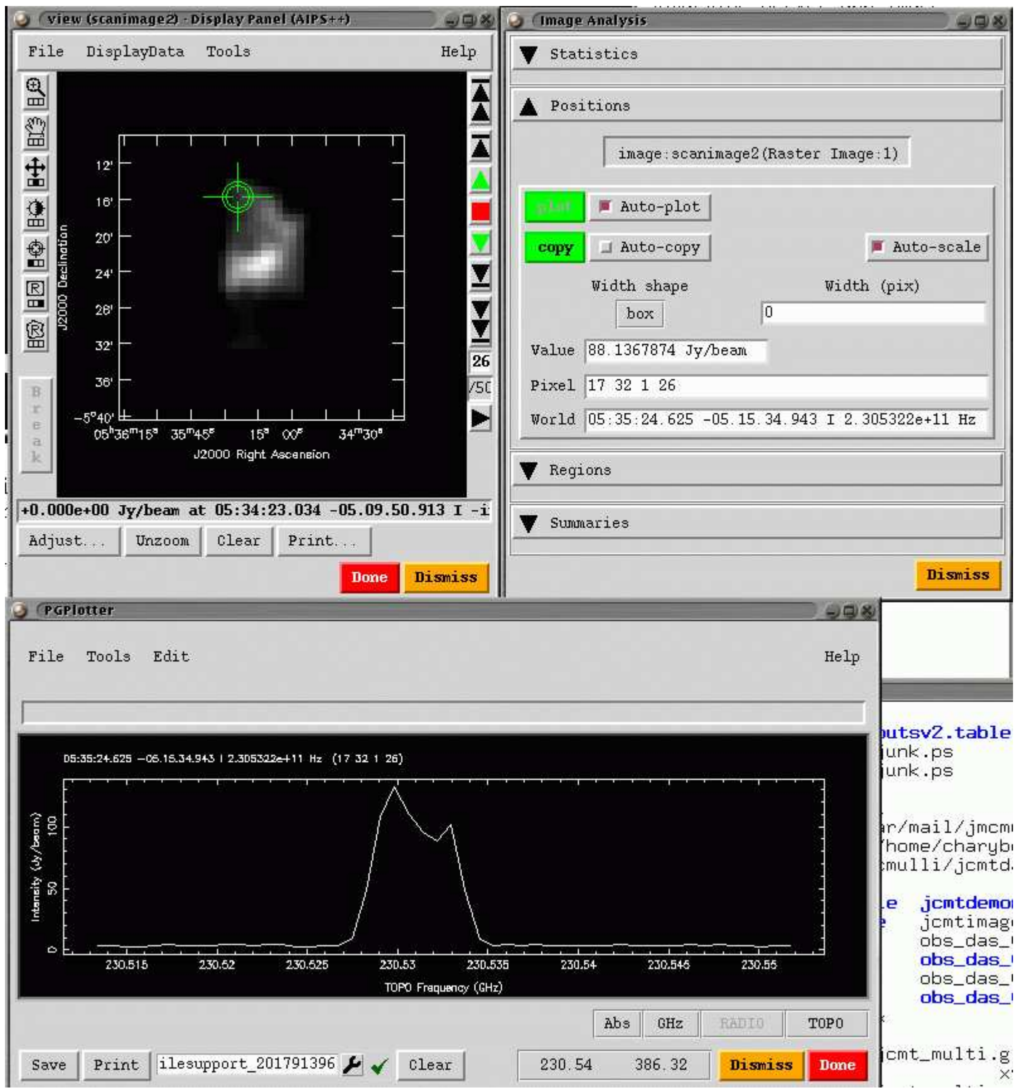

Figure 5.2: Viewer display of an image utilizing the Image Analysis Tool. Selecting the Positions Tool, choosing a position on the data cube (marked by the cross hairs), and selecting the "Auto-plot" feature allows the user to examine the gridded spectra of the data cube interactively. As each position is selected, the spectrum for that position is displayed in the lower plot.

- Develop Frequency Switch Calibration procedure for JCMT data
- Refine JCMT imaging to support standard baseline removal
- Develop a Grid Spectra map utility
- Enable multi-panel image display – *Done.*

Chapter 6

Arecibo Observatory Spectral Line Reduction with DISH

K. O'Neil and J. McMullin

6.1 AIPS++ Versions Available at Arecibo

The are a number of different version of AIPS++ available at Arecibo:

- **Stable:** This is the version which should be the most reliable aips++ version available at Arecibo. On the other hand, this is often the least recent version available, as it is only updated every month.
- **Weekly:** This version exists for those people who wish to run an up-to-date version of aips++, and who are willing to accept a few bugs/problems with the version. As its name indicates, this version is updated once a week and is used mainly be folks working on developing aips++ software.
- **Daily:** This version should only be used by aips++ software developers. Because it is updated daily, it can be extremely unreliable.

6.2 Setting Your Environment

6.2.1 Starting AIPS++

To start AIPS++, issue the following commands (from the c shell):

```
> source /pkg/aips++/stable/aipsinit.csh
> aips++ -l naic.g
```

6.2.2 Making AIPS++ run faster

Here are several hints for speeding up your performance, if the operation/startup of AIPS++ seems sluggish.

- Store cache files on the computer you run from by adding the following line to your .aipsrc file:

  ```
  user.aipsdir: /tmp/username_aips++

  where username is your username, e.g.,
  user.aipsdir: /tmp/rphillips_aips++
  ```

- If you do not want to have the toolmanager GUI available on starting AIPS++, you can suppress it and speed up the startup time by including the following lines in your .aipsrc file (located in your home directory):

  ```
  toolmanager.fullgui: F
  toolmanager.gui.auto: F
  ```

  The toolmanager isn't directly needed for any DISH operations. Should you want it, it can be used at a later time by typing:

  ```
  tm.gui();
  ```

- Run AIPS++ from the computer which hosts the disks with your data. Typically the data I/O over NFS is slow and the above bypasses this as much as possible

AIPS++ builds its cache file structure the first time it is started with a new user.aipsdir, and each time you start a new version. Judge its startup speed on the *second* time you start it up, not the first.

6.2.3 Paging

If you find that error messages, code, and data listings scroll off of your terminal window, you might need to assign the **PAGER** environment variable. You can remedy this by setting **PAGER** to 'less' in your UNIX startup scripts.

```
export PAGER='less'
```

6.3 Converting AO Data into SDFITS

Arecibo data must first be converted from its native format into single-dish FITS format, through use of the "sdtest" program (/usr/S2local/bin/sdtest). Single-dish fits data are stored as FITS binary tables with a single row for each spectrum: A single dump usually generates 4 or 8 spectra (four subcorrelators with either one or two polarizations per board). If data are taken with different numbers of channels per spectrum, the file size will be as if they all had the largest number of channels. If the number of channels increases during the conversion, then sdtest will resize the entire file, which may take a while.

Before running the sdtest program you need to know the following:

- Where is your data located (i.e. at /share/olcor/... or /proj/a9999/...)
- What was the system temperature for both channel A & B ? (This is only used for the initial data headers, and will be replaced once the data is calibrated.)
- Whether the receiver was set to linear or circular polarization
- The total number of scans/cycle (i.e. its four if you took: ON, OFF, CAL ON, CAL OFF)

Once you feel confident you know these things, login to an Arecibo computer and then go to whatever directory you wish your files to reside in. Type sdtest to run the SDFITS conversion program. The program will will ask a few questions and then start converting your data.

Note that you can also run this program "on-the-fly" to continuously update a datafile with the online data via: sdtest -c.

In that case, when you are finished you must quit out of the conversion program by typing

```
^C
```

(possibly a few times).

See http://www.naic.edu/ãstro/data/aips++/ for more complete details.

6.3.1 SDFITS and AIPS++

6.3.2 SDFITS Example 1

A data file with only two observations, each of a different galaxy (UGC 3489 and UGC 4431). Each cycle consisted of ON, OFF, CAL ON, CAL OFF

```
sdtest
Enter Input (AO) filename: /proj/a9999/corfile.01jan00.a9999.1
Enter Output File: 01Jan00a.fits

Channel A system temp? :35
```

```
Channel B system temp? :35

Linear or Circular inputs? :C
Total number of scans in a cycle:4
Scan 930900005, UTC=2000-01-05T12:06:42, AST=2000-01-05T08:06:42
Object: U3489
nter Command (UPPER CASE) or Object Name: (cr will keep U3489
  QUIT to stop converting,
  DRIVE to finish entire file with same object name,
  UNTIL <num|scan>  to continue for num or until reaching scan,
  NEWCYCLE <num> to change the number of scans in a cycle to num,
  SKIP  [num | scan] to skip this scan, or num scans, or until scan
  ASKFREQ to prompt for frequencies each time,
  ASK to ask basic questions again.): U3489
Scan 930900009: U3489-1.  RA 11:12:24.40 Dec +31:24:41.0
Scan 930900010: U3489-2.  Offset 5400.000000 (5400.165333)
Scan 930900011: U3489-3.  Offset 5400.000000 (5399.659220)
Scan 930900012: U3489-4.  Offset 5400.000000 (5399.719945)
Object:U3489
Enter Command (UPPER CASE) or Object Name: (cr will keep U3489
  QUIT to stop converting,
  DRIVE to finish entire file with same object name,
  UNTIL <num|scan>  to continue for num or until reaching scan,
  NEWCYCLE <num> to change the number of scans in a cycle to num,
  SKIP  [num | scan] to skip this scan, or num scans, or until scan
  ASKFREQ to prompt for frequencies each time,
  ASK to ask basic questions again.): U4431
Scan 930900009: U4131-1.  RA 07:59:11.43 Dec +31:48:23.6
Scan 930900010: U4131-2.  Offset 5400.000000 (5400.165333)
Scan 930900011: U4131-3.  Offset 5400.000000 (5399.659220)
Scan 930900012: U4131-4.  Offset 5400.000000 (5399.719945)
9632 records translated
```

6.3.3 SDFITS Example 2

A data file with three observations, two of one galaxy (U5361) and one of another (U5592). Each cycle consisted of ON, OFF, CAL ON, CAL OFF

```
sdtest
Enter Input (AO) filename: /proj/a9999/corfile.01jan00.a9999.4
Enter Output File: 01Jan00d.fits

Channel A system temp? :9.2
```

```
Channel B system temp? :9.2

Linear or Circular inputs? :L
Total number of scans in a cycle:4
Scan 930900061, UTC=2000-01-05T15:30:13, AST=1999-11-05T11:30:13
Object: U5361
Enter Command (UPPER CASE) or Object Name: U5361
Enter Command (UPPER CASE) or Object Name: (cr will keep U5361
  QUIT to stop converting,
  DRIVE to finish entire file with same object name,
  UNTIL <num|scan>  to continue for num or until reaching scan,
  NEWCYCLE <num> to change the number of scans in a cycle to num,
  SKIP  [num | scan] to skip this scan, or num scans, or until scan
  ASKFREQ to prompt for frequencies each time,
  ASK to ask basic questions again.): U5361
Scan 930900061: U5361-1.  RA 09:55:03.25 Dec +25:12:10.6
Scan 930900062: U5361-2.  Offset 8955.000000 (8948.313686)
Scan 930900063: U5361-3.  Offset 8955.000000 (8948.299047)
Scan 930900064: U5361-4.  Offset 8955.000000 (8948.302798)
Scan 930900065, UTC=2000-01-05T15:42:43, AST=1999-11-05T11:42:43
Object: U5361
Enter Command (UPPER CASE) or Object Name: (cr will keep U5361
  QUIT to stop converting,
  DRIVE to finish entire file with same object name,
  UNTIL <num|scan>  to continue for num or until reaching scan,
  NEWCYCLE <num> to change the number of scans in a cycle to num,
  SKIP  [num | scan] to skip this scan, or num scans, or until scan
  ASKFREQ to prompt for frequencies each time,
  ASK to ask basic questions again.):
Scan 930900065: U5361-1.  RA 09:55:03.25 Dec +25:12:10.2
Scan 930900066: U5361-2.  Offset 8955.000000 (8948.189632)
Scan 930900067: U5361-3.  Offset 5400.000000 (5399.836752)
Scan 930900068: U5361-4.  Offset 8955.000000 (8948.473693)
Scan 930900069, UTC=2000-01-05T15:57:01, AST=1999-11-05T11:57:01
Object: U5361
Enter Command (UPPER CASE) or Object Name: (cr will keep U5361
  QUIT to stop converting,
  DRIVE to finish entire file with same object name,
  UNTIL <num|scan>  to continue for num or until reaching scan,
  NEWCYCLE <num> to change the number of scans in a cycle to num,
  SKIP  [num | scan] to skip this scan, or num scans, or until scan
  ASKFREQ to prompt for frequencies each time,
  ASK to ask basic questions again.): U5592
Scan 930900069: U5592-1.  RA 10:21:31.63 Dec +22:32:51.0
Scan 930900070: U5592-2.  Offset 5400.000000 (5401.432019)
Scan 930900071: U5592-3.  Offset 5400.000000 (5400.222629)
```

```
Scan 930900072: U5592-4.  Offset 5400.000000 (5400.127029)
14448 records translated
```

6.4 Available AIPS++ Routines

The following provides details on the available routines to apply gain/temperature corrections for Arecibo data, as well as details of the "ao.import" routine:

```
These files are designed for the import and calibration of
data taken at Arecibo Observatory.

A typical session for this may go something like:

% aips++ -l naic_start.g
- ao.import('U3564.sdfits')
- aoname:='U3564_ms1'
- ao.gaincorr()
- ao.plot('average1')

After the above commands, the calibrated data is shown in the dish pgplotter
screen.  Note that the variable "aoname" is assumed to be the name of a measurement
set as it appear in the dish results manager (i.e. aoname:='U3564_ms1' and NOT
aoname:='U3564_ms')

KNOWN PROBLEMS:

(1) The use of a hybrid to convert a receiver from linear <-> circular has not been
dealt with (in a large part due to a lack of knowledge of what happens to
the cal values).
(2) The baseline routines need to be updated to allow baseline fitting to the cals.
Plus, the "gaincorr" routine needs baseline fitting.
(3) The baseline routines do not yet allow for the user to interactively
select (i.e. with a mouse) their ranges
(4) The user cannot yet calibrate their data into units of T_main_beam
     (and probably never will be allowed to....)

PLANNED UPGRADES:

- Make routines more generic.  This means having the  routines automatically
recognize the ON, OFF, CAL_ON, CAL_OFF
- No calibration for mapping data is in place yet
- Others?
----------------------------------------------

The primary files are:
```

```
(1) ao.import

Description:  This program imports sdfits data and converts it into
a measurement set.  In the process, the program adds
a couple 'missing' columns to the data.

Syntax: ao.import(sdfitsname)
Where sdfitsname is the full name of the sdfits file,
and which is assumed to be of the form <name>.<ending>.
The output measurement set will have the name
<name>_ms

Subroutines called: ao.fixpnt

(2) ao.gaincorr

Description:
This routine reads in data from a measurement set,
does (ON - OFF)/OFF for the individual data dumps,
and then applies the gain/temp correction to the
individual (ON - OFF)/OFF data dumps.  The output
is a measurement set whose data column contains the
calibrated data.  The measurement set will be named
msname_on_off.  Additionally, the calibrated data will
be averaged and the average will be written to the
dish results manager as X records, where X is the
number of boards x the number of polarizations of the scan.

Note that the data is currently assumed to be in the
form of ON, OFF, CAL_ON, CAL_OFF.  The temperature
is determined from the CAL measurements, and the
gain correction is obtained from the 'standard' AO
gain curves, as described in the subroutines below.
After the routine is run, the tys and tcal calculated from
the noise diode data is placed into the data headers.

Syntax: ao.gaincorr([name=aoname,onscan=0,calscan=0,tsys=0,gainval=0,offscan=0,weight='none',convert='T'])
Here name is the name of the measurement set which contains
the data of interest.  As with all these routines, it defaults to th
definition of the aoname variable.  The other options are:
  onscan = Scan number of the ON source observation
   Default is the first scan of the data set
  calscan = First scan number of affiliated noise diode observation
   Default is onscan+2
  tsys = array of tsys values.  This must be a X by 6 array, where X
```

```
   is the number of (polarizations)x(boards).
   Default is to calculate this value from the cals.
   Note entering a value here will override any entry for the
  "calscan"
  gainval = array of gain values.  This must be a X by 4 array, where X
                   is the number of (polarizations)x(boards).
                   Default is to calculate this value from the gain curve.
  offscan = Scan number of the OFF source observation
                   Default is onscan+1
  weight = weighting scheme for the averaging of the scans.  The
  options are "none" (the default), "rms" (rms across each spectra
  is used for weighting), and "tsys" (the weight for all channels in
  each spectra is the same value).
  convert = convert the data to Janskies (value of 'T', the default),
  System temperature (value of 'Tsys') or leave as raw data (value of 'F')
Subroutines Called:
ao.tsys, ao.gain, ao.onoff, ao.avg, get_scan

(3) ao.gaincorr2

Description:
This routine will read in data, average the on and off scans,
do (on - off)/off, apply the gain curve, and then multiply
the result by the tsys (obtained from the associated CAL
observations.  The output is sent to the dish results
manager as X records, where X is the number of boards x the
number of polarizations of the scan.

Note that the data is currently assumed to be in the
form of ON, OFF, CAL_ON, CAL_OFF.  The temperature
is determined from the CAL measurements, and the
gain correction is obtained from the 'standard' AO
gain curves, as described in the subroutines below.
After the routine is run, the tys and tcal calculated from
the noise diode data is placed into the data headers.

This routine allows the user to fit a baseline to the
data in place of using an off scan.  Note that for the moment
the system temperature is always found via (CAL_ON-CAL_OFF(/CAL_OFF).

Syntax: ao.gaincorr2([name=aoname,onscan=0,calscan=0,tsys=0,gainval=0,offscan=0,fitbase='F',weight='none',convert='T',order=2,range='F'])
Here name is the name of the measurement set which contains
the data of interest.  As with all these routines, it defaults to th
definition of the aoname variable.  The other options are:
  onscan = Scan number of the ON source observation
   Default is the first scan of the data set
```

```
  calscan = First scan number of affiliated noise diode observation
   Default is onscan+2
  tsys = array of tsys values.  This must be a X by 6 array, where X
   is the number of (polarizations)x(boards).
   Default is to calculate this value from the cals.
   Note entering a value here will override any entry for the
  "calscan"
  gainval = array of gain values.  This must be a X by 4 array, where X
                   is the number of (polarizations)x(boards).
                   Default is to calculate this value from the gain curve.
  offscan = Scan number of the OFF source observation
                   Default is onscan+1
  fitbase = if set to true (T, t, or 1), the routines will fit a baseline
   to the averaged on scan, and use that baseline in place of an
   'off' spectra.  The fitted baseline will be a polynomial, of
   the order, and across the range, set with the "order" and "range
   keywords.
  weight = weighting scheme for the averaging of the scans.  The
  options are "none" (the default), "rms" (rms across each spectra
  is used for weighting), and "tsys" (the weight for all channels in
  each spectra is the same value).
  convert = convert the data to Janskies (value of 'T', the default),
  System temperature (value of 'Tsys') or leave as raw data (value of 'F')
  order   = this is the order of the polynomial used for fitting a baseline to
  the data.  This option is used only with the baseline fitting option.
  range   = the range of data (in channel numbers) to which a baseline is fit,
  when using the baseline fitting option (otherwise, this keyword is
  ignored).  If, for example, you wish to fit a base line to channels
  500 through 1500, you would type: range='[500:1500]'
Subroutines Called:
ao.tsys, ao.gain, ao.onoff, ao.avg, get_scan

(4) ao.calmany

Description:
        This routine allows the user to run the above ao.gaincorr function
on a list of objects (scans), provided the scans all lie within
the same measurement set.  For a more complete description of the
routine, see the ao.gaincorr description.

Syntax: ao.calmany([name=aoname,numpairs=1,pattern=4,onscan=0,calscan=0,tsys=0,gainval=0,offscan=0,
weight='none', convert='T'])
The entries are identical to those of the ao.gaincorr routine, bar:
  numpair = number of dataset (objects) to reduce
  pattern = the number of scans in a given pattern (i.e. a set which
consists of an ON, OFF, CAL_ON, CAL_OFF would have pattern=4).
```

```
Subroutines Called: ao.gaincorr,ao.tsys, ao.gain, ao.onoff, ao.avg, get_scan


(5) ao.calmany2

Description:
        This routine allows the user to run the above ao.gaincorr2 function
on a list of objects (scans), provided the scans all lie within
the same measurement set.  For a more complete description of the
routine, see the ao.gaincorr2 description.

Syntax: ao.calmany([name=aoname,numpairs=1,pattern=4,onscan=0,calscan=0,tsys=0,gainval=0,offscan=0,
fitbase='F', weight='none', convert='T',order=2, range='F'])
The entries are identical to those of the ao.gaincorr routine, bar:
  numpair = number of dataset (objects) to reduce
  pattern = the number of scans in a given pattern (i.e. a set which
consists of an ON, OFF, CAL_ON, CAL_OFF would have pattern=4).

Subroutines Called: ao.gaincorr,ao.tsys, ao.gain, ao.onoff, ao.avg, get_scan



----------------------------------------------

The subroutines (all of which can be called separately) are:

(1) ao.avg

Description:
This routine takes in a typical AO measurement set,
determines the total number of boards, polarizations,
and dumps, and then averages all the dumps for
each board/polarization.

The output will be X records which are placed in the
dish results manager, where X is the number of
scans X the number of polarizations

Syntax:
ao.avg([name=aoname,scan=0,weight="none"])
If no scan number is given, the average is done on the
first scan in the given measurement set.  The weight
keyword describes the  weighting scheme for the averaging of
the scans.  The options are "none" (the default), "rms" (rms
across each spectra is used for weighting), and "tsys" (the
weight for all channels in each spectra is the same value).
```

Subroutines called: ao.numdumps, ao.getscan

(2) ao.gain

Description:
This routine determines the gain correction for the
individual dumps of the entered scan number/measurement
set. The coefficients for the gain curves are taken from the
gain.datRX files in the AOgains directory (X is the AO receiver
number). If no gain.dat file is available, the program provides
an error message and returns a gain value of 1.

The result is output as an X by 4 array, where X is the
(number of boards) x (number of polarizations). The
columns of the output array are:
column 1: The unique number to describe the board/pol
column 2: The center freq. of the board, used for the gain corr.
column 3: The polarization i.d.
column 4: The gain correction (in K/Jy)

Syntax:
ao.gain([name=aoname,scan=0])
        If no scan number is given, the gain is determined from the
        first scan in the given measurement set. Note that if you
enter this command as, say, "gain:=ao.gain([,scan])",
The resultant gain will be placed into the "gain" variable.

Subroutines called: ao.getgainval,ao.convertdate,ao.convertmjd,ao.numscans,ao.getscan

(3) ao.onoff

Description:
This routine will calculate the (ON-OFF)/OFF for all the dumps,
boards, and polarizations in the measurement set. Additionally,
if desired, the routine will convert the data from raw counts
to Jy using the entered gain and Tsys values. The output of
the routine is a table.

Syntax:
ao.onoff([name=aoname,onscan=0,convert='T',temp=0,gainval=0,offscan=0])
As with all the routines, if no name is given, this function defaults
to the name given to the aoname variable, while the default for the "onscan"
id simply the first scan in the measurement set. Additionally, if the "convert"
variable is set to "T" or "Tsys" (or "t", "1", or "tsys"), the
program will quit unless a nonzero value of the temperature and gain

is entered. Additionally, the user may enter a scan number to be used for the OFF source scan. If no value is entered, the routines will default to offscan=onscan+1

Subroutines called: ao.numscans

(4) ao.tsys

Description:
This routine will calculate the system temperature from a pair of ON+OFF noise diode observations. The system temperature is calculated per channel and then the mean value is returned. The temperatures of the noise diodes are found in the cal.datRX files in the AOgains directory (X is the number of the AO receiver).

```
The result is output as an X by 6 array, where X is the
        (number of boards) x (number of polarizations).  The
        columns of the output array are:
                column 1: The unique number to describe the board/pol
                column 2: The center freq. of the board, used for the gain corr.
                column 3: The polarization i.d.
column 4: The noise diode temperature used in the calculations
column 5: The (mean) temperature of the given board/polarization
                column 6: The standard deviation for the temperature measurement
```

Syntax:
ao.tsys([name=aoname,calscan=0])
Here the calscan number should be the scan of the CAL_ON observation. Again, if no scan number is given the routine assumes the first scan in the measurement set is the CAL_ON.

Subroutines called: ao.getcalval,ao.convertdate,ao.convertmjd,ao.numscans,ao.getscan

(5) ao.convertdate

Description:
This routine determines the number of days which have passed in a year based off the entered date. That is, if the entered date is 01 January, 2001, the returned day would be 1 (as January 01 is the first day of the year), while if the entered date is 29 January, 2001, the returned day would be 29.

Syntax:
ao.convertdate(day,month,year)
The day, month, and year should be entered as whole numbers. The year should be given completely (i.e. 2002 and not 02).

Subroutines called: none

(6) ao.convertmjd

Description:
This routine converts the mean Julian date into the month, day, and year.
The result is returned as a 3 component array with:
column 1: day
column 2: month
column 3: year

Syntax:
```
        ao.convertmjd(mjd)
```

Subroutines called: none

(7) ao.converttime

Description:
        This routine converts the number of seconds since 1970.0 into
a day and year.

Syntax:
```
ao.converttime(tnow)
Typically, tnow:=time()
```

Subroutines called: none

(8) ao.getscan

Description:
This routine determines the scan number of the first scan in a measurement
set, and returns it.

Syntax:
```
ao.getscan([name=aoname])
```

Subroutines called: none

(9) ao.getcalval

Description:
        This routine returns the temperature of a given noise diode with a given
configuration. The temperatures of the noise diodes are found in the
cal.datRX files in the AOgains directory (X is the number of the AO receiver).

```
Syntax:
ao.getcalval(rcvnum, freqin, calnum[, day=-1, year=-1, pol=0])
rcvnum = AO receiver numbers (1-12)
freqin = the frequency of interest for the diode temperature
calnum = configuration number for the noise diode
day, year = the day (1-366) and year for which you want the cal values.
    (these default to the present date if no value is entered)
pol = polarization of interest for the diode temperature
    (this defaults to pol=0 of no value is entered)

Subroutines called: ao.converttime

(10) ao.getgainval

Description:
        This routine returns the gain correction for a given receiver at a given
azimuth, zenith angle, and date.  The coefficients for the correction
are found in the gain.datRX files in the AOgains directory (X is the number of
the AO receiver).  If no gain.datRX file is found, an error message is printed
and the program returns a value of 1.

Syntax:
ao.getgainval(rcvnum,freqin[,az=90,za=12,day=-1,year=-1,pol=0])
rcvnum = AO receiver number
freqin = the frequency of interest for the diode temperature
az = azimuth angle of interest (equal to 90 deg. if not entered)
za = zenith angle of interest (equal to 12 deg. if not entered)
day, year = the day (1-366) and year for which you want the cal values.
     if not entered, this defaults to the current date
pol = polarization of interest for the diode temperature.  Default is 0.

Subroutines called: ao.converttime

(10) ao.numdumps

Description:
This routine determines the number of data dumps within a given scan number,
and returns this value.

Syntax:
ao.numdumps([name=aoname,scan=0])
Here name is the name of the measurement set of interest, and scan is
the scan number of the scan of interest.

Subroutines called: none
```

(12) ao.numscans

Description:
        This routine determines the (number of boards) x (number of polarizations)
within a given measurement set, and returns this value.

Syntax:
        ao.numscans([name=aoname,scan=0])
        Here name is the name of the measurement set of interest, and scan is
        the scan number of the scan of interest.

Subroutines called: none

(11) ao.base

Description:
This function will calculate (On-OFF)/OFF using an input baseline as the OFF.
It can also convert the data into Tsys, or Jy if desired.
This will be expanded in the near future to accommodate
a much wider variety of baseline fitting options.

Syntax:
        ao.base(basel,[name=aoname, onscan=0, convert='T', temp=0, gainval=0)
Here, basel is the name of the baseline (in the dish results manager).
This variable MUST be given. Name is the name of the measurement set
of interest. onscan is the scan number of the scan of interest. Convert
can be set to 'T' (convert to Janskies, the default), 'Tsys' (convert to system
temperature, or 'F' (leave as raw data), and temp and gain are the temperature and
gain arrays, usually obtained from the ao.tsys and ao.gain routines.

Subroutines called: ao.numscans

(12) ao.gainone, ao.gaintwo, ao.gainthree

Description:
        These functions obtain the gain for the variety of possible
equations

Syntax:
        ao.gainone(az,za,coef,zac)
        ao.gaintwo(az,za,coef,zac)
        ao.gainthree(az,za,coef,zac)
Here az, za are the azimuth and zenith angle (in degrees)
coef is the array of coefficients for the equation, and
zac is the zenith angle at which the gain curve turns over.
All the entries are usually supplies by the ao.gain routine.

```
Subroutines called: none

(12) ao.plot

Description
This routine just calls the d.plotscan() routine.

Syntax:
ao.plot(name)
Here name is the name of an sdrecord (i.e. ao.plot('average1')).

Subroutines called: none

(13) ao.fixpnt

Description:
This routine corrects for the absence of a REF_FREQ, which enables viewing in msplot.
It also adds a column entitled "NRAO_GBT_STATE_ID" which is required by the uni2.g
routines.  Note that this column (NRAO_GBT_STATE_ID) is simply a mirror of the DATA_DESC
column.  This routine is called by the import command.

Syntax:
ao.fixpnt([name=aoname])

Subroutines called: none
```

6.5 Sample Data Reduction Session

This example assumes that an AIPS++ installation is available in directory /pkg/aips++/stable and that the AIPS++ sample data is available in the data repository at /pkg/aips++/data/demo/dishdemo/U3294.sdfit. Modify the first few lines of this example appropriately if your installation has a different path.

```
% source /pkg/aips++/stable/aipsinit.csh
% cp /pkg/aips++/data/demo/dishdemo/U3294.sdfit .
% aips++ -l naic.g
...
Time to initialize DISH =  11.978507
DISH tool is --> d
AO   tool is --> ao
Glish version 2.7.
- ao.import('/pkg/aips++/data/demo/dishdemo/U3294.sdfit');
T
- #this automatically opens up the file
- ao.gaincorr('U3294_ms1');
```

After the above commands, the calibrated data is placed in the dish results manager. It can be plotted with the "ao.plot" command.

Chapter 7

BIMA Data Reduction

D.M. Mehringer and The LAUGH Group[1]

7.1 Introduction

David Mehringer, Ray Plante, and Anuj Sarma
This version of the BIMA AIPS++ Cookbook is a much abridged version of the full version which can be found at http://monet.ncsa.uiuc.edu/aips++/forBIMA/cook/cook.html. The abridged version essentially describes only BIMA-specific issues in AIPS++ while the full version has useful chapters on using AIPS++ in general. If you discover errors or have suggestions, please email them to dmehring@astro.uiuc.edu.

7.1.1 Starting AIPS++

To start AIPS++, issue the following commands if you are using c shell:

```
> source /appl/aips++/stable/aipsinit.sh
> aips++
```

or, if you are using bash shell:

```
> . /appl/aips++/stable/aipsinit.sh
> aips++
```

7.1.2 Exiting AIPS++

There are two ways to **exit** from AIPS++:

- In the command line interface (*Glish* window), type after the "–" prompt:

```
exit
```

[1] Laboratory for Astronomical Imaging AIPS++ Users Group and Help: members were H. Dickel, D. Fong, D. Mehringer, R. Plante, A. P. Sarma

and press return

- From the **Tool Manager GUI,**
  select "File ⇒ Exit AIPS++"

7.2 Manipulating AIPS++ tools

Hélène. R. Dickel

7.2.1 How do I create a tool?

The **Tool Manager window is used to create tools. Tools and global functions are organized first by *Packages* and then by *Modules*, as illustrated by the graphical lists that make up the Tool Manager. The** `mirfiller tool can be used to import Miriad visibility data into` AIPS++.

`In the BIMA Data Reduction section of the` AIPS++ `cookbook, we use a short hand for selecting tools and functions in the` **Tool Manager. It looks like this:**

Package	***Module***	***Tool***		***Constructor***
`bima`	`mirfiller`	`mirfiller`	`Create`	`mirfiller`

Don't understand this table? See § 7.2.1

Here's what it means:

> **Tip: A *Constructor* is a special type of function that returns a tool as its value; that is, it creates a tool.**

1. **Under the Packages list, click on "bima."**
2. **Under the Modules list, click on "mirfiller."**
3. **Under the Tools list, click on "mirfiller."**
4. **Hit the "Create" button below the Tools list. This will cause the Tool Manager to morph into a new GUI for constructing a** `mirfiller tool.`

Now that you have the constructor interface up, you need to enter values for its parameters. Here are two most important parameters you need to provide and what they mean:

Constructor: `mirfiller`		
Parameter	***Sample Value***	***What it means***
`toolname`	`mymirfiller`	**The name by which you want to refer to this tool; this is helpful when you want to refer to the tool from the Glish command line or if you have more than one tool of the same type.**
`mirfile`	`3c273`	**The name of the Miriad dataset.**

In the BIMA Data Reduction section of the AIPS++ cookbook, a table like the one above is used to summarize the inputs you need to give to a function. Usually, only the most important parameters are listed; all others can be left to their default.

> **Tip:** The first time you start AIPS++, the Tool Manager will display default values for all function parameters; however, if change a parameter value and then run the function, the new value will be remembered. To see how to return values to their default, select "Default" from the wrench menu to the right of the input (see § ??).

Most likely, the "toolname" parameter will already be set to "mymirfiller;" however, if you want to change it, move your cursor over the "toolname" input, erase its contents and type in the new value.

> Tips for entering parameter values:
>
> - Here are two quick ways to erase the text in an input:
>   - Select "Clear" from the wrench menu to the right of the text input (see § ??).
>   - Triple-click on the text; that is, place your cursor over the text and click the left button three times in rapid succession. The text will now be highlighted or "selected". (If the field contains just one word, than double-clicking is enough.) Next, hit the backspace key or just start typing. The selected text will disappear, and the new text will be entered in its place.
> - Text that appears on top of a raised box represents a menu. (In our example, the `preview` and `defpass` parameter values are presented as menus.) Click on the box to see the other choices.
> - A yellow box containing the label "<unset>" means that no value is set. When the function is run, it will choose an appropriate default value.

When all the parameters have been set, you can execute the constructor and create the `mirfiller tool by clicking the green button labeled ''Create.''`

`When the constructor finishes its work, the GUI will morph again to show the interface to the tool's functions.`

> **Tip:** `One way to tell that the constructor has finished its work and the tool is ready for use is when the button on the upper right, which used to have only the tool's name (e.g. mirfiller),` *changes* `to` toolname:toolname-value `(e.g.,` mirfiller:mymirfiller`).`

7.2.2 How do I get rid of a tool?

`You can shut down a tool from its function view (where the functions appear in a list on the right; if you do not see this, press the ''mirfiller:mymirfiller" button along the top of the` **Tool Manager.**

Now press the "Done" button at the bottom, right hand corner of the window. The Tool Manager will morph back into its tool selection view. The "mirfiller:mymirfiller" button is now gone.

Note that the "Dismiss" button does not shut down the tool. It simply returns to the tool selection view. The "mirfiller:mymirfiller" button would remain.

7.3 Importing BIMA Data in AIPS++

Ray Plante

7.3.1 How do I import my Miriad *uv*-data into AIPS++

Here's a summary of how to import your data it using the GUI:

- **Create a** `mirfiller` tool:

Package	*Module*	*Tool*		*Constructor*
bima	mirfiller	mirfiller	Create	mirfiller

Don't understand this table? See § 7.2.1

- Enter the constructor parameters:

Constructor: mirfiller		
Parameter	*Sample Value*	*What it means*
toolname	mymirfiller	The name by which you want to refer to this tool; this is helpful when you want to refer to the tool from the Glish command line or if you have more than one tool of the same type.
mirfile	3c273	The name of the Miriad dataset.

When you have filled in the above-mentioned values, press **Create**.

> **Tip:** During its construction, your mirfiller tool will open the Miriad dataset and print a summary of its contents to the logger. It will also set up the default data selection depending on what the dataset contains.

- Run the fill function to copy the data to an AIPS++ measurement set:

Function: fill		
Parameter	*Sample Value*	*What it means*
msfile	3c273.ms	The name of the output measurement set to fill.
verbose	True	Setting this to true will send messages on **mirfiller**'s progress to the logger window.

After filling in the above values, press **Go**.

- Delete the tool when the filling is done.

How do I do the same thing in *Glish*?

If you have the scripter running, you can just look the scripter window. You'll see exactly what was run by the GUI.

Note that AIPS++ will usually write Glish commands to the scripter in their most verbose form. Here's a more compact form:

```
include 'mirfiller.g'
mymirfiller := mirfiller("3c273");
mymirfiller.fill("3c273.ms", verbose=T);
mymirfiller.done();
```

Parameters left to their default values do not have to be included. Plus, if you give the parameter values in the order that they appear in the GUI (or the documentation), you don't have to give the parameter names.

7.3.2 How do I import my FITS *uv*-data into AIPS++

Loading of FITS *uv*-data into AIPS++ can be done using the ms tool via its fitstoms constructor. Here a summary of how to import FITS data using the GUI:

- Create an ms tool by selecting the following in the **Tool Manager window.**

Package	*Module*	*Tool*		*Constructor*
general	ms	ms	Create	fitstoms

Don't understand this table? See § 7.2.1

- **Choose the** fitstoms constructor. To do this, click on fitstoms on the left hand side of the ms tool under 'ms all constructors.' Now fill in the values for the parameters that appear on the right hand side. Sample values are given in the table below.

Constructor: fitstoms		
Parameter	*Sample Value*	*What it means*
toolname	myms	The name by which you want to refer to this tool.
msfile	3c273	Name of the *output* AIPS++ measurement set to be created.
fitsfile	3C273.FITS	Name of your *input* UVFITS file.

When you are done filling in the above inputs,
press **Create**.

- Delete the tool when you are done.

> **Tip:** This is done by clicking on the red **Done** button on the bottom right in the ms tool function view (see § 7.2.2).

The *glish equivalent* of the above procedure is the following:

```
include 'ms.g';
myms := fitstoms(msfile="3c273.ms", fitsfile="3C273.FITS");
myms.done();
```

7.4 Inspecting UV data

David Fong

7.4.1 Overview

This section describes how one can inspect UV data by means of the following questions:

7.4.2 How do I plot the *uv*-data using the imager tool?

7.4.3 How do I examine the *uv*-data using msplot?

7.4.3 How do I list the spectral parameters of the *uv*-data?

7.4.2 How do I plot the *uv*-data using the imager tool?

The imager tool contains the functions for imaging. If you wish to have a quick look at the *uv*-data before imaging, then follow these steps. If you wish examine the *uv*-data in greater detail, then use msplot (see § 7.4.3).

- Create an imager tool
  In the **Tool Manager window, select the following:**

Package	*Module*	*Tool*		*Constructor*
synthesis	imager	imager	Create	imager

Don't understand this table? See § 7.2.1

- **Enter the constructor parameters:**

Constructor: Imager		
Parameter	*Sample Value*	*What it means*
toolname	myimager	**The name by which you want to refer to this tool; this is helpful when you want to refer to the tool from the Glish command line or if you have more than one tool of the same type.**
filename	cloud.ms	**Name of the AIPS++ measurement set.**

When you are done filling in the above inputs, press Create.

When the constructor is finished, the GUI will morph into the imager tool window.

> **Tip:** You can tell that the constructor is finished and the GUI has morphed into the **Imager** tool window by looking at the button near the upper right of the **Tool Manager** that used to say **"imager" before you hit the Create button — this button near the upper right will now say "imager:myimager."**

- Make the plots.

  1. If you wish to plot the uv-coverage:

     – select ⇒ plotting (under the ''Function group'' menu)

     > **Tip:** The ''Function group'' menu is on the left side of the **Tool Manager** near the middle. **Left-clicking on this menu button with your mouse gives you a drop-down menu — hold down the left mouse button and move down the drop-down menu until "plotting" is highlighted — click on "plotting" to select it, and the words "plotting functions" together with a highlightable box of choices will appear below the "Function group" menu button.**

     – select ⇒ plotuv (below plotting functions)

– press **Go** --- after a while, a new pgplot screen will pop up and display the *uv*-coverage plot.

2. If you wish to plot amplitude vs. *uv*-distance:

   – select ⇒ plotvis (below plotting functions)

   – press **Go**

7.4.3 How do I examine the *uv*-data using msplot?

msplotprovides more selection options for plotting *uv*-data. The GUI-based tool can make plots, displays or listings of various parts of an MS (see Reference manual: general.ms.msplot for a full description).

- Create an msplottool
  In the **Tool Manager window, select the following:**

Package	*Module*	*Tool*		*Constructor*
general	ms	msplot	Create	msplot

Don't understand this table? See § 7.2.1

- **Enter the constructor parameters:**

Constructor: msplot		
Parameter	*Sample Value*	*What it means*
toolname	mymsplot	**The name by which you want to refer to this tool.**
msfile	cloud.ms	**Name of the measurement set.**
edit	False	**False option only allows inspection of the *uv*-data.** **True option allows interactive flagging of the *uv*-data.**

When you are done entering the above parameters, press Create.

When the constructor is finished, a new msplot GUI window will start up.

Example 1: A common plot to make is X vs Y, where the X and Y axes can be time, UV-distance, amplitude, phase, etc.

- select ⇒ 'Plot X vs Y'
  (default option for the menu bar the on the first row, next to 'Show')

  For 'Plot X vs Y', the X and Y axes to be plotted are chosen by selecting the options found under the menu bars on the second row labeled 'X Axis' and 'Y Axis'.

- select ⇒ uvdist (under X axis menu)

- select ⇒ amplitude
  (first under 'Y axis' menu, next under 'data' submenu, and then under 'value' subsubmenu)

- use the buttons in the third row, namely, 'X, Y plot limits,' 'Data selection,' 'Spectral selection,' and 'Polarization selection' to specify the data you want to plot. Selecting one of these buttons will open a panel below the button --- this panel has additional input parameters to control data selection

- press **Go**

> **Tip:** The green **Go** button is on the upper right of the msplot gui, next to the red **Stop** button.

Example 2: Another common plot to create is the UV-coverage.

- select ⇒ 'Plot UV Coverage' (under the menu bar the on the first row, next to 'Show')

- press **Go**

To individually plot the UV-coverage for each baseline:

- select ⇒ 'Antenna 1' and 'Antenna2' (under 'Iteration selection' menu on the first row)

> **Tip:** 'Antenna 1' and 'Antenna 2' are the elements forming a baseline.
> 'Antenna 1' = antenna # 1 to $(n-1)$, and
> 'Antenna 2' = antenna # 2 to n.

- press **Go**

> **Tip:** Use the 'Plot flagged data points' toggle switch under the 'Options' menu in the msplot gui to control whether flagged *uv*-data points are shown.

How do I list the spectral parameters of the *uv*-data?

- select ⇒ 'Summarize MS in logger'
  (found under the menu bar on the first row, next to 'Show')

- press **Go**

  The spectral parameters will be listed in the **Logger** window.

7.5 Editing UV data

Anuj Sarma

7.5.1 Overview

This section will describe how to flag/edit your *uv*-data.

7.5.2 How do I flag *uv*-data?

Flagging and editing of the *uv*-data can be done using the **autoflag** tool.

To use **autoflag**, go to the 'Manager' view in the **Tool Manager GUI** and make the following selections:

- Create an autoflag tool:

Package	*Module*	*Tool*		*Constructor*
synthesis	autoflag	autoflag	Create	autoflag

Don't understand this table? See § 7.2.1

> **Tip:** When you press the green Create button, you will get an additional button to the right of the 'Manager' and 'Tools in use' buttons, and the view of the constructor shown below.

- Enter the constructor parameters:

Constructor: autoflag		
Parameter	*Sample Value*	*What it means*
toolname	myautoflag	The name by which you want to refer to this tool; this is helpful when you want to refer to the tool from the Glish command line or if you have more than one tool of the same type.
msname	bllac.ms	Name of the AIPS++ measurement set that you wish to flag.

When you are done entering the above constructor parameters, press Create.

> **Tip:** To create an AIPS++ measurement set from raw BIMA data (e.g., 'bllac.ms' mentioned above), see § 7.3.1 on page 150.

The GUI will morph into the autoflag tool window when the constructor is done.

Tip: You can tell the constructor is done when the button on the upper right in the Tool Manager GUI, which used to say "autoflag," changes to "autoflag:myautoflag."

- **In the list under 'all functions' on the left of the autoflag tool, click on setselect. On the right, you will get a list of options. You can choose to flag based on one or more of these options. For a detailed description of each of these options, see the User Reference Manual for** `autoflag.setselect`**.**

  Tip: As of June 12, 2002 (version 1.7 build 419) the options for the setselect gui are not activated when chosen — obviously a bug in the GUI! If this bug has still not been removed when you use this manual, see the following § 7.5.3 on page 158 to use the glish environment to carry out all the processes described here.

  When you are done filling in all the desired options in setselect, press Go.

- **Having made the selections on which you wish to flag, the next step is to run autoflag and apply the flags to the data.**
  Look in the list on the left under 'all functions' again, and this time
  click on run.

  You will get the following view, go ahead and make the selections shown in the *Sample Values*:

Function: `run`		
Parameter	***Sample Value***	***What it means***
`globparm`	`[=]`	**Record of global method parameters. Leave at default.**
`plotscr`	`F`	**Number of plots per window in flagging report. 'F' means "do not print this report." Default is to print report.**
`plotdev`	`F`	**Number of plots per page in hardcopy flagging report. 'F' means "do not generate this report."**
`devfile`		**Name of hardcopy report. Leave blank if you don't want hardcopy. (Default is flagreport.ps/ps.)**
`reset`	`False`	**Reset pre-existing flags. Pull-down menu.**

When you are done making the above selections for run, press Go.

The data for 'bllac.ms' are now flagged.

> **Tip:** Do not flag the target source data until after the calibration has been performed on the phase calibrator and the calibration tables have been copied over to the target source (see Chapter 7.6 on page 161). Otherwise, the bimacalibrator tool with exit with an error. This is due to a bug in the AIPS++ code for heterogenous spectral windows and is under investigation.
>
> If you are flagging a calibrator, make sure you run the `reavg` routine in bimams after you flag and before you calibrate, so that the wideband data are recalculated after the flags have been applied. This is described in § 7.5.4.

7.5.3 How do I carry out all the above steps to flag *uv*-data using glish?

All the steps described in the last section can be done in glish.

> **Tip:** Bring up the scripter GUI when you are going through the steps in the last section. (See § ?? on page ?? if you have forgotten/don't know how to bring up the scripter.) All the glish commands used during the gui operations will then be written to the scripter. Compare the scripter output to the glish commands below.

Follow the steps below to flag your data using glish. The procedure followed is identical to that in the last section.

- First, include the autoflag script

```
include 'autoflag.g'
```

- Then, create the autoflag tool. To do this, attach the tool to the dataset you wish to flag. In this example, we have named the dataset 'bllac.ms.'

```
af:=autoflag('bllac.ms')
```

- The next step is to select the data to be flagged. You can choose to flag based on a variety of criteria, all of which are listed in the User Reference Manual for `autoflag.setselect`. Below, we choose to flag all data for antenna 3, and clip all data that are outside the amplitude range (0,25).

```
af.setselect(ant=2, clip=[expr='ABS YY', min=0, max=25]);
```

Tip: We specify ant=2 to flag all data for antenna 3. If you think this is weird, I do too. Some things in AIPS++ are 0-numbered, others are 1-numbered. The only way to tell is to look at the messages later and see which antenna was flagged; in this case, it will say that it flagged ANT3 (which is antenna 3). This is obviously a bug, and will hopefully be corrected in the future.

Also, to flag all data that are *outside* the amplitude range (0,25), we specify the expression given above for the clip parameter.

Once you have run the above by hitting return, you should get the following list. Anything other than an 'F' signifies that you chose to flag based on that criterion, and the value you used to determine how to flag (except for 'dtime,' which is related to 'timeslot' and has the value dtime=10 as default). For example, ant=2 because you set that in setselect above; similarly for clip.

```
 spwid      = F
 field      = F
 fq         = F
 chan       = F
 corr       = F
*ant        = 2
 baseline   = F
 timerng    = F
 autocorr   = F
 timeslot   = F
 dtime      = 10
 quack      = F
*clip       = [expr=ABS YY, min=0, max=25]
 flagrange  = F
 unflag     = F
```

- Finally, run the autoflagger.

```
af.run(plotscr=F, plotdev=F);
```

Tip: To prevent additional annoying windows from popping up, plotscr and plotdev have been set to 'F' above; if you want these windows to come up, leave these out. For further details, see the User Reference Manual for `autoflag.run`.

Two cautionary notes: Do not flag the target source data until after the calibration has been done on the phase calibrator and the calibration tables have been copied over to the target source (see Chapter 7.6 on page 161). Otherwise, the bimacalibrator tool will exit with an error. This is due to a bug in the AIPS++ code for heterogenous spectral windows, and is under investigation.

If you are flagging a calibrator, make sure you run the `reavg` routine in bimams before you calibrate, so that the wideband data for the calibrator are recalculated after the flags have been applied. This procedure is described in § 7.5.4 below.

7.5.4 How do I re-generate the wideband data after flagging?

After flagging the calibrator data, you will have to re-generate the wideband data, in order to carry over the flags to the sideband averages. To do this, use the reavg routine in the bimams tool.

To use reavg go to the 'Manager' view in the Tool Manager GUI and make the following selections:

- Create a `bimams` tool:

Package	*Module*	*Tool*		*Constructor*
bima	bimams	bimams	Create	bimams

Don't understand this table? See § 7.2.1

> Tip: When you press the green Create button, you will get an additional button named 'bimams' to the right of the 'Manager' and 'Tools in use' buttons, and the view of the constructor shown below.

- Enter the constructor parameters:

Constructor: `bimams`		
Parameter	*Sample Value*	*What it means*
toolname	mybimams	The name by which you want to refer to this tool; this is helpful when you want to refer to the tool from the Glish command line or if you have more than one tool of the same type.
msname	bllac.ms	Name of the flagged AIPS++ phase calibrator measurement set for which you want to re-generate the wideband data.

When you are done entering the above constructor parameters, press Create.

The GUI will morph into the `bimams` tool window when the constructor is finished.

Tip: Again, you can tell the constructor is done when the button on the upper right in the Tool Manager GUI, which used to say "bimams," changes to "bimams:mybimams."

- **Click on the 'Function group' button on the left of the bimams tool GUI. From the drop-down list, select 'spectral window'. A list of spectral window functions will appear below the button.**

- **Click on reavg in the list of 'spectral window functions' below the 'Function group' button.**

- **Enter values for the function parameters:**

Function: `reavg`		
Parameter	***Sample Value***	***What it means***
`out`		**Output dataset — Leave blank. If left blank, it will modify the input dataset.**
`dosort`	`True`	**Sort data when doing subqueries. Pull-down menu. Default is `True`. Best bet is to leave at default. Set to False only if sure that data is already in order required by `reavg`.**

  When you are done entering the above function parameters, press Go.

The above procedure will re-generate the wideband data. You are now ready to calibrate the data.

The above procedure to re-generate the wideband data can be carried out in glish using the following:

```
include 'bimams.g';
mybimams:=bimams('bllac.ms');
mybimams.reavg();
mybimams.done();
```

7.6 Calibration of uv data

Anuj Sarma

7.6.1 Overview

This section will answer the following questions concerning the calibration of your uv data:

[A work in progress. Future updates will include interactive editing of gain tables, self-calibration, and bandpass correction.]

7.6.2 How do I calibrate the data?

Calibration of BIMA data is done using the `bimacalibrater` tool. Before using this tool, make sure that the following steps have been completed.

- First, apply the linelength correction to the data in miriad. A procedure to apply the linelength correction is under development in AIPS++. Watch this space for updates!
- Load the data into AIPS++ using the `mirfiller tool (see § 7.3.1).`
- `The data may be edited and flagged using` **autoflag** `(§ 7.5.2). Flag only the calibrator data at this stage. Do not flag the target source data; due to a bug in AIPS++, the calibration routines will not work if you flag the target source data before the calibration. (The target source data should be flagged after the calibration routines below have been run.)`
- `After flagging the calibrator data, use the reavg routine in the bimams tool to recalculate the wideband data for the calibrator. This is described in § 7.5.4.`

Now, you are ready to start calibrating the data. Below, we describe the steps to calibrate BIMA data based on observations of a phase calibrator. In the example below, the phase calibrator is named "bllac" (hence the AIPS++ measurement set is bllac.ms) and the target source is named "cepa."

Go the the 'Manager' view in the Tool Manager and make the following selections.

- Create a `bimacalibrater` tool:

Package	*Module*	*Tool*		*Constructor*
`bima`	`bimacalibrater`	`bimacalibrater`	`Create`	`bimacalibrater`

Don't understand this table? See § 7.2.1

Tip: When you press the green Create button, you will get an additional button named 'bimacalibrater' to the right of the 'Manager' and 'Tools in use' buttons, and the view of the constructor shown (in part) below.

- **Enter the constructor parameters:**

Constructor: `bimacalibrater`		
Parameter	*Sample Value*	*What it means*
`toolname`	`mybimacalibrater`	**The name by which you want to refer to this tool; this is helpful when you want to refer to the tool from the Glish command line or if you have more than one tool of the same type.**
`targets`	`cepa.ms`	**The name of the target source dataset.**
`phcals`	`bllac.ms`	**The name of the phase calibrator dataset.**
`pbcals`		**The name of the passband calibrator — leave blank.**
`fcals`		**The name of the flux calibrator — leave blank.**
`targetnames`	`narrow`	**Your nickname for the target source.**
`phcalnames`	`wide`	**Your nickname for the phase calibrator.**

When you are done entering the above parameters, press Create.

The GUI will morph into the bimacalibrater tool window when the constructor is done.

Tip: You can tell the constructor is done when the button on the upper right in the Tool Manager GUI, which used to say "bimacalibrater," changes to "bimacalibrater:mybimacalibrater."

- **In the list under 'all functions' on the left of the** `bimacalibrater` **tool, click on setjy. On the right, you will get a list of parameters that need to be set. An example is given below.**

Function: setjy		
Parameter	*Sample Value*	*What it means*
sources	wide	**Phase calibrator nickname.**
fluxdensity	[1,0,0,0]	**Flux density for the phase calibrator in Jy. For now, you have to fill in this value by yourself, e.g., we have filled in a value of 1.0 for the I-flux. A routine to get this value automatically from a calibrator table is under development.**

When you are done filling in all the desired options in setjy, press Go.

> **Tip: The setjy procedure will set the specified flux values for the phase calibrator in each spectral window. Hence, this will take some time. During this time, the 'Running: abort?' button next to the 'Go' button will turn to and remain red. When the function is done, the button will no longer be highlighted, and the logger window will show the message: "Finished imager::setjy."**

- **Now, you have to set the parameters to solve for the gains. Again, in the list under 'all functions' on the left hand side, click on setsolve. Enter the values for the parameters on the right hand side. Sample values are given below.**

Function: setsolve		
Parameter	*Sample Value*	*What it means*
sources	wide	**Phase calibrator nickname.**
type	G	**Component type = G-Jones. This is a pull-down menu.**
t	600	**Solution interval in seconds.**
phaseonly	True	**Only solve for phases. Pull-down menu.**
refant	2	**Reference antenna. Note** bimacalibrater **has 1-based numbering, so antenna 2 means antenna 2, *not* antenna 3.**
table	gcal	**Output calibration table name.**

When you are done filling in all the desired options in setsolve, press Go.

- **Solve for the selected calibration components that you set above. In the list under 'all functions' click on solve.**

Function: solve		
Parameter	*Sample Value*	*What it means*
sources	wide	Phase calibrator nickname.

press Go.

- To look at the solutions, click on plotcal under 'all functions' and enter values for sources (wide) and tablename (gcal), and choose the plottype (AMP or PHASE) in the list of parameters that comes up on the right. Pressing Go will give you the desired plot.
- Transfer the calibration components to a new table. To do this, click on transfer under 'all functions' on the left hand side of the bimacalibrater tool.

> Tip: Transfer of the calibration components to a new table is required because a BIMA target source and calibrator may differ in the number of spectral windows. Transfer to a new table ensures that the AIPS++ gain table is set up to be consistent with the number of spectral windows in the target source.

Function: transfer		
Parameter	*Sample Value*	*What it means*
outtable	gcal.xfer	Output table in which to write calibration solutions.
intable	gcal	Input table generated by solve step.
spwmap	[ ]	Array describing the spectral window mapping from input window to output window. Should be able to leave at default. The function is intelligent enough to figure out how spectral windows need to be mapped from input to output.
calibratees	narrow	Nicknames of sources to which the solutions will be applied, e.g., if you have more than one source, you can include them all here with their given nicknames (or roles, e.g., targets). If you specify more than one source here, then all must have the same spectral window configuration, otherwise run transfer multiple times.

When you are done setting the above inputs,
press Go.

- Next, set up how the calibration components will be applied to the target source. On the left hand side under 'all functions' click on setapply. Sample inputs are given below.

Function: `setapply`		
Parameter	*Sample Value*	*What it means*
`sources`	`narrow`	Target source nickname.
`type`	`G`	Component type = G-Jones; pull-down menu.
`t`	`0.0`	Interpolation interval in seconds. Leave at default (0.0).
`table`	`gcal.xfer`	Input calibration table. Should be the output table given by `transfer`.
`select`		Leave at default, which is blank.

When you are done setting the above inputs,
press Go.

- Finally, apply the calibration to the target source. Click on correct under 'all functions' on the left hand side, and fill in the inputs on the right hand side, like the sample below.

Function: `correct`		
Parameter	*Sample Value*	*What it means*
`sources`	`narrow`	Nicknames of target source(s) where calibration components are to be applied.

press Go.

The calibration is now done. Remember, you must flag the target source data after calibration and before imaging (meaning *now*, see Chapter 7.5) — this will continue to be the procedure until the bug in AIPS++ is removed.

7.6.3 How do I carry out the above steps to calibrate the *uv*-data using glish?

All the steps described in the last section can be done in glish. First, make sure you are in the glish environment (which is started by typing "glish" at the command line prompt) and that you have loaded the data into AIPS++, flagged only the phase calibrator data, and regenerated the wideband data for the flagged phase calibrator, as described in the last section.

- Before using the bimacalibrater tool, define the following two quantities. Such a pre-definition is given here as an example; it

will allow you to ignore the code and make changes at the top of the file when using glish scripts.

```
RefAnt := 2;
IntTime := 600;
```

> **Tip: The bimacalibrater tool has 1-based numbering, hence `RefAnt := 2` means we are choosing antenna 2 as the reference antenna, and not antenna 3.**

- **Now, include the bimacalibrater script:**

```
include 'bimacalibrater.g';
```

- **Create the bimacalibrater tool. To do this, specify the target source name and the name of the phase calibrator. You can also specify aliases for the target and calibrator, using "targetnames" and "phcalnames."**

```
bc := bimacalibrater(targets='cepa.ms',targetnames='narrow',
                        phcals='bllac.ms', phcalnames='wide');
```

- **As mentioned before, you have to manually set the flux density for the calibrator for now. An automatic way to do this is under development.**

```
bc.setjy(fluxdensity=[1.0,0,0,0]);
```

- **Set the parameters to solve for the gains. The solutions will be written into a table named "gcal." You have to specify the reference antenna and the integration time; in this example, we have set these quantities in the pre-definition above, so we just call these quantities below.**

```
bc.setsolve(type='G',refant=RefAnt,table='gcal',t=IntTime,phaseonly=T);
```

- **Now solve for the selected calibration components above.**

```
bc.solve();
```

- **You may now examine the "gcal" table using `bc.plotcal`. If things look OK, transfer the calibration components from one table to another. As mentioned before, this is required because a BIMA target source may have a different number of spectral windows from the calibrator.**

```
bc.transfer(outtable='gcal.xfer',intable='gcal');
```

- **Set up how the calibration components will be applied to the target source.**

```
bc.setapply();
```

- **Apply the calibration to the target source:**

```
bc.correct();
```

- **The calibration is now complete. Close the tool.**

```
bc.done();
```

Copyright

The AIPS++ consortium may be reached by email at aips2-request@nrao.edu. The postal address is: AIPS++ Consortium, c/o NRAO, 520 Edgemont Rd., Charlottesville, Va. 22903-2475 USA.