Volume 3 - Telescope Specific Processing

AIPS++ Group, eds

October 8, 2002

Contents

Chapter 1

VLA reduction

Contributors: George Moellenbrock, Debra Shepherd,
Athol Kemball, & the NRAO AIPS++ user group
Version: 2002 June

1.1 Introduction

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

For fairly standard VLA continuum and spectral-line observations (including polarimetry), this document can stand on its own. Additional 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* – complete and detailed software documentation.

For complicated images or unconventional observing strategies, it may be necessary to consult the more detailed AIPS++ documentation listed above. AIPS-aips++ and MIRIAD-aips++ dictionaries are also available which provide mapping of common AIPS and MIRIAD tasks/adverbs to their counterparts in AIPS++.

The data reduction categories covered in this document are 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 GUI interface.

AIPS++ is a tool-based reduction package. This means that related functionality are gathered into entities called 'tools'. The user creates a working version of particular tool by logically associating a dataset with it, 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, the basic pattern of usage is as follows:

```
mytool:=sometool(data='mydata');          # Create a tool of type sometool
                                          #  and call it mytool

mytool.functionA(param1=1,param2='a');    # Execute functionA of sometool,
                                          #  specifying parameters

mytool.functionG();                       # Execute functionG of sometool,
                                          #  with no parameters

mytool.done();                            # End use of tool
```

In this example, a tool of type 'sometool' is created by associating it with a dataset ('mydata'), and giving it a name ('mytool'). Two different functions are executed in the tool using commands of the form `toolname.function(parameters)`. All tools have a 'done' function which undoes the explicit association of the tool functionality and the data.

Some tools have functionality which does not require that an explicit working version of the tool be created. Such functionality is accessed via 'global functions'.

The reduction sequences given below 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 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 during 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. Additionally, certain capabilities of the image tool are available from within the viewer, e.g., statistics. The viewer is also used to interact with images, e.g., to create CLEAN deconvolution regions or masks.

1.2 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).

1.3 Basic initialization

This section assumes that AIPS++ has been installed on your UNIX or LINUX system. First, create a .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 1.

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 aipsinit.csh file
                                      # that specifies setup parameters
aips++                                # start aips++ (stable version)
```

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 each value having AMP = 1 & phase = 0°.

You should get a *Glish* command prompt in the xterm window, a `Tool Manager` GUI window, and a `Logger` GUI window (see figure, above).

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

It is also convenient to start up a viewer tool to view and analyze images easily. To start the default viewer GUI from the *Glish* command line type:

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

Or, to start the default viewer from the tool manager window click on the `Tools in use` button. Select the `viewer` tool in the list and then the `Show` button. You will get a **Data Manager** tool which you can use to select any image and a viewer **Display Panel**:

You can put these off to the side until you are ready to use them. Note: once the aipsinit.csh file has been sourced, you can start the viewer directly from the UNIX prompt by typing `viewer`.

The remainder of this document uses *Glish* script sequences to execute data reduction steps. The *Glish* sequences can also be used as a guide for operation using the Tool Manager GUI interface. All parameter names and settings correspond exactly to those listed in the GUI interface (in general, only the immediately relevant parameters are listed in the glish sequences; others may be safely left set to default values). Consult the Getting Started document for general tips on the use of the Tool Manager.

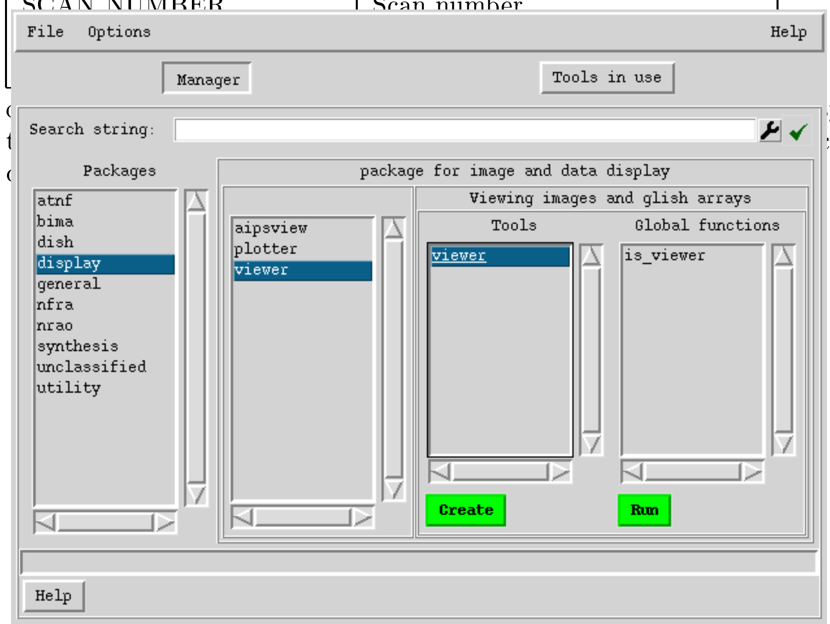
An important difference between working with the GUI interface and the *Glish* command line interface is that a *Glish* command to run a task or function begins with all values set to their default values. Thus, the user need only specify values that will be changed from their defaults. However, the GUIs remember the state of each input. Thus, if you change an input, it will be remembered

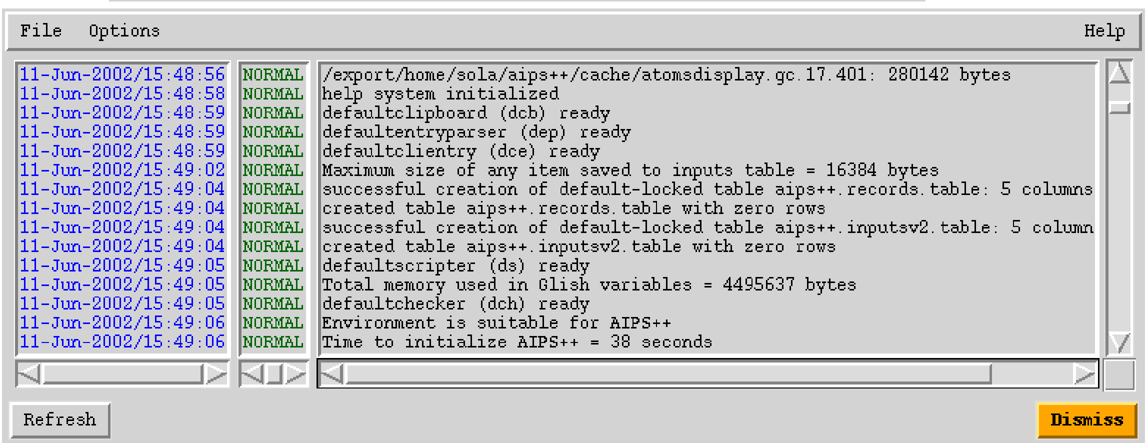
Table 1.2: Common Data Selection Parameters†

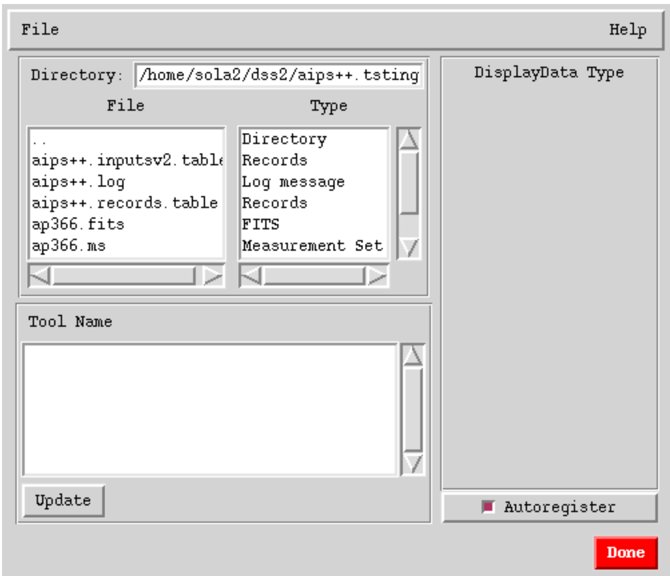
Parameter	Contents
ANTENNA1	First antenna in baseline
ANTENNA2	Second antenna in baseline
FIELD_ID	Field (source no.) identification
SPECTRAL_WINDOW_ID	Spectral window number (IF no.)
ARRAY_ID	Subarray number
OBSERVATION_ID	Observation identification
POLARIZATION_ID	Polarization identification
SCAN_NUMBER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

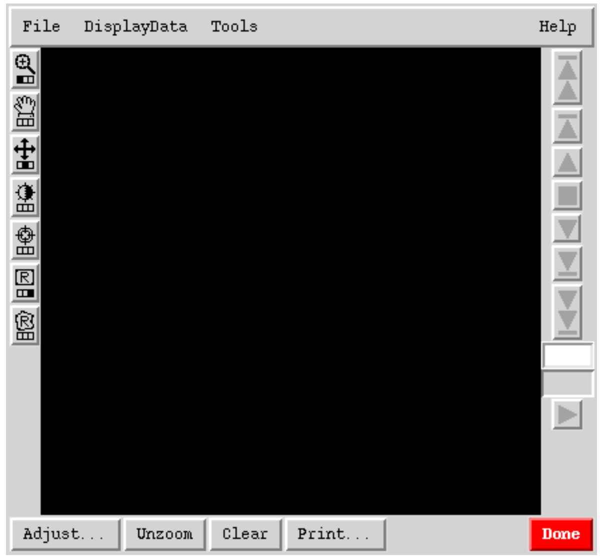
† Selection

ynthesis
ile these









the next time you use the GUI. If you are unsure of the default value for a parameter in a GUI window, use the "Wrench" icon to the right of the input line to reset the value to its default. Also, GUI inputs do not require quotes around inputs while *Glish* does.

Note: From the Tool Manager window, you can click on Options: 'Copy commands to scripter' and Options: 'Show scripter' to write equivalent *Glish* commands of your GUI-based actions to a Scripter GUI window. This is useful because you can copy *Glish* script into a file and use this as a record of your data reduction session (or just save the Scripter commands directly with the 'Save' button at the bottom of the Scripter GUI). You can also modify scripts to reduce similar data in a semi-automated fashion.

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';      # if filling from the VLA archive
include 'ms.g';             # if filling from a UVFITS file
include 'flagger.g';
include 'imager.g';
include 'msplot.g';
include 'calibrater.g';
include 'image.g';
include 'imagepol.g';
include 'viewer.g';
```

This is not required when working entirely from the GUI interface because

the tools are loaded automatically when they are selected.
What happens if something goes wrong?
First, check that your inputs are correct. If you are working with the GUI interface, there is 'pop-up' help available on most inputs – simply put the cursor on an item and a short help statement will pop-up on the screen.
You can get help from the Tool Manager window by 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 (click on Documentation in the upper left of the web page) or you can run a Search on key words you are interested in.

Also, you can get help on each tool and the tool inputs by clicking on the 'Help' button in the GUI tool screen.
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 that you can fill in and submit automatically within AIPS++.
If you manage to find yourself in a hung state (don't worry, this is rare), you can kill AIPS++ from the 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
```

You are now ready to restart AIPS++.

1.4 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

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

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.

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 $4{\times}4$ 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.

1.4.1 Summary of Data Reductions 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 mstool 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 msplottool 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.correctfunction 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.

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

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.

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 be applied prior to the solution.

1.5 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 sets.

The continuum polarimetry dataset 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. 3C286 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 (redshifed 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 3C286 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’ (`imgrC`, `calC`, etc.), and those associated with the spectral-line reduction have names ending in ‘S’ (`imgrS`, `calS`, etc.).

1.6 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.

1.6.1 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).

To run the vlafiller using a local tape drive, 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 file 2 and 3 on tape
                  project='TESTT');            # in current directory
```

To run the vlafiller using a remote tape drive:

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

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.

It is usually desirable to obtain a summary of the dataset's properties after filling. This can be done by constructing an ms tool, and running the ms.summary function as follows:

```
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).

1.6.2 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. This is achieved as follows:

continuum polarimetry case

```
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 2 for reference.

spectral-line case

```
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 2 for reference.

1.7 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.

1.8 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
```

1.8.1 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.

1.9 Basic Gain Calibration

1.9.1 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 can then be used to set the proper flux density for flux density calibrators. For sources that are recognized flux density calibrators (listed below), Imager.setjy will calculate the Perley-Taylor (1999) flux densities, Fourier transform the data (trivially) 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).

Recognized Flux Density Calibrators		
3C Name	*B1950 Name*	*J2000 Name*
3C286	1328+307	1331+305
3C48	0134+329	0137+331
3C147	0538+498	0542+498
3C138	0518+165	0521+166
−−	1934−638	−−
3C295	1409+524	1411+522

continuum polarimetry case

Set 3C286 (field 11) model according to Perley-Taylor (1999):

```
imgrC:=imager(filename='ap366.ms'); # Start imager tool

imgrC.setjy(fieldid=11);            # Calculate & set flux density for
                                    #  1328+307 (3C286)
```

spectral-line case

Set 3C286 (field 1) model according to user-specified value (if you don't want to use the standard values):

```
imgrS:=imager(filename='ngc5921.ms'); # Start imager tool

imgrS.setjy(fieldid=1,
            fluxdensity=[14.8009,0,0,0]);
                                      # Set flux density of 1331+305
                                      #  (3C286) to 14.8009 Jy
```

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.

1.9.2 Solving for complex gain, G

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 MS.

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 realized implicitly. For data that requires polarization calibration, it is important to 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 300 sec is chosen for the 21 cm spectral line data.

continuum polarimetry case

```
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=3,                     #  Choose reference antenna 3:
              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
```

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).

spectral-line case

```
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 on
              t=300.0,                   #  a 300-second timescale
              refant=15,                 #  Choose reference antenna 15:
              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.

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 is 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., the obtained 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.

1.9.3 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

```
calS.setdata(msselect='FIELD_ID==1');      # Select bandpass calibrator
                                           #  (Field 1 = 1331+305)

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=15,                   #  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=T);
```

1.9.4 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 as follows:

continuum polarimetry case

```
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 the solutions to
                                          #  the table ap366.fluxcal located
                                          #  on disk.

calC.plotcal(tablename='ap366.fluxcal');  # Inspect the solutions
```

spectral-line case

```
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 the
                                              #  solutions to the table
                                              #  ngc5921.fluxcal located on disk.

calS.plotcal(tablename='ngc5921.fluxcal',     # Inspect the solutions
             plottype='AMP',
             multiplot=F);
```

In both cases, 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.

1.9.5 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, simply create new calibration tables and correct the data again – the CORRECTED_DATA column will be over-written.

continuum polarimetry case

```
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.
```

spectral-line case

```
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.
```

To calibrate field 1, as well, `setdata(...)` and `setapply(type='G',...)` should be executed again, selecting FIELD_ID==1 in both, followed by calibrater.correct.

At this point, the data for both continuum and spectral-line are sufficiently calibrated to attempt initial total intensity imaging.

1.10 Instrumental Polarization Calibration

For the case of continuum polarimetry, 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 is appropriate. In fact, the D calibration could have been obtained at the same time as G by issuing an additional calibrater.setsolve execution before the solve was executed.

If the instrumental polarization calibrator is significantly polarized (as in the case of 3C286) a non-zero model for Stokes Q and U must be determined. This model can be obtained by imaging in polarization 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.

In this example, the polarization model of 3C286 is obtained by imaging the data after applying only the G (and P) calibration. Since 3C286 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. This operation is illustrated in the continuum polarimetry script.

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 nominally be averaged entirely 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.

[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.

[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 *Getting Results* for more information.

1.10.1 continuum polarimetry case (only)

First make IQUV images of the polarization calibrator 3C286 to determine the polarization model (a more detailed description of imaging is given in the next section).

```
imgrC.setdata(mode='none',              # Select continuum data for
              fieldid=11);              #  for 3C286.

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='3C286.cal1.im'); #  CORRECTED_DATA column
                                        #  in the MS.  Write the image
                                        #  3C286.cal1.im to disk.
```

The resulting image, 3C286.cal1.im, written to disk, has 4 planes, with I, Q, U, and V images. Using the Viewer, obtain statistics on each plane of the IQUV image (see the Image Analysis section of this document for details about Viewer operation). Record the maximum OR minimum value, which ever has the largest absolute value. Use the viewer arrow buttons on the right to display Q and U images – record statistics for each image. Assume V = 0. Use the Stokes I value reported by the Imager.setjy function run earlier. For this example:

```
I = 7.462  Q = -0.621  U = -0.593  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, solve for the D-term calibration:

```
stokes := [7.462, -0.621, -0.593, 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
```

Examine the D-term or “leakage” solutions with the calibrater.plotcal function, with `plottype='DRI'` (D-term real versus imaginary components):

```
calC.plotcal(plottype="DRI" ,
            tablename="ap366.dcal");
```

For VLA data like this example, see http://www.aoc.nrao.edu/∼gtaylor/calman/polcal.html for a description on how to plan for good polarization calibration and what to expect when solving for D-terms.

Now, the data can be corrected for both 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 3C286

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

1.11 Imaging

Imaging is done using the imager tool, which supports a broad range of imaging modes and deconvolution algorithms, as described in the synthesis imaging chapter and in the User Reference Manual. This section describes single-field imaging/deconvolution for the continuum and spectral-line datasets using a CLEAN region which is defined interactively.

continuum polarimetry case

For the continuum dataset, make a single-field, full-polarization (I, Q, U, & V), continuum image of the target source, 0957+561. This image is made with the Clark CLEAN algorithm as follows:

First, use the imager.advise function to get an estimate of what the image properties should be. Then set the image parameters and weight, and CLEAN the image using an interactive cleaning algorithm:

```
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.

                    # ADVISE FUNCTION RESULTS:
                    #  Maximum uv distance = 585844 wavelengths
                    #  Recommended cell size < 0.176041 arcsec
                    #  Recommended number of pixels = 360
                    #  Dispersion in uv, w distance = 293779, 94533.3 wavelengths
                    #  Best fitting plane is w = 0.340419 * u + 0.340608 * v
                    #  Dispersion in fitted w = 30797.4 wavelengths
                    #  Wide field cleaning is not necessary


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 to determine the resolution resulting from the chosen weighting scheme and see what artifacts to expect from beam sidelobes during CLEANing, or you can proceed to CLEAN the image. To make an image of the synthesized beam (e.g. a PSF image):

```
imgrC.makeimage(type='psf',            # Form the PSF image
                image='0957+561.psf');

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

The beam size and position angle will be reported in the logger window. To CLEAN the image:

```
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.
            interactive=T,           # Clean interactively, with the option
            npercycle=500);          #  to choose a new clean region every
                                     #  500 cleaning cycles.
```

During the CLEAN run, you will be asked to interactively define CLEAN regions (polygons or boxes) in a pop-up viewer. How to do this is described at the end of this section.

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. Note that the imager.setdata and imager.setimage functions need not be run again when restarting a CLEAN with the previously setup imager tool.

When you are finished CLEANing, use the viewer to display the image 0957+561.im and obtain statistics. The RMS should be ~ 0.5 mJy beam^{-1}. The peak in the image is ~ 26.2 mJy beam^{-1}.

spectral-line case

For the spectral-line dataset, total intensity images must be generated for NGC5921 for each spectral 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 often 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.

```
imgrS.setdata(mode='channel',        # Select channel data for field 3
              nchan=63,
              start=1,
              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.

                     # ADVISE FUNCTION RESULTS:
                     #  Maximum uv distance = 4798.4 wavelengths
                     #  Recommended cell size < 21.4931 arcsec
                     #  Recommended number of pixels = 180
                     #  Dispersion in uv, w distance = 1858.4, 759.892 wavelengths
                     #  Best fitting plane is w = -0.253496 * u + -0.574818 * v
                     #  Dispersion in fitted w = 172.198 wavelengths
                     #  Wide field cleaning is not necessary

imgrS.setimage(nx=256,               # Imaging parameters
               ny=256,
               cellx='10arcsec',
               celly='10arcsec',
               stokes='I',
               mode='channel',
               start=1,
               step=1,
               nchan=63,
               fieldid=3);

imgrS.weight(type='uniform');        # Uniform weighting

or

imgrS.weight(type='briggs',          # Robust weighting
             rmode='norm',
             robust=0.5);
```

At this point, you can examine the synthesized beam to see if you are happy with the resolution resulting from the chosen weighting scheme and see what artifacts to expect from beam sidelobes during CLEANing, or you can proceed to CLEAN the image.

To make an image of the synthesized beam (e.g. a PSF image):

```
imgrS.makeimage(type='psf',            # Form the PSF image
```

```
               image='ngc5921.psf');

imgrS.fitpsf(psf='ngc5921.psf');       # Measure the beam size
```

The beam size and position angle will be reported in the logger window.
To CLEAN the image:

```
imgrS.clean(algorithm='clark',       # Image and deconvolve
            niter=3000,              #  with Clark CLEAN
            gain=0.1,                #  Write the cleaned image to the file
            threshold='0Jy',         #  ngc5921.im on disk.
            model='ngc5921.mod',
            image='ngc5921.im',
            residual='ngc5921.resid');
```

When you are finished CLEANing, use the Viewer to display the image ngc5921.im and obtain statistics. The RMS in each channel should be $\sim$ 1.7 mJy beam^{-1}.

You should see 2 continuum source and the galaxy as you move through the cube. Note, in this example the entire inner quarter of the image is CLEANed. If desired, you can also use the interactive mask function as in the continuum polarimetry example.

To interactively define one or more CLEAN regions

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

2. To make a box-mask, click on the box region function. Use the mouse button to create a box. To make a polygon-mask, 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. Click on 'Add Region'

4. Double 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. 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:

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

9. When finished, 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, 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`.

1.12 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!

continuum polarimetry case

Form the model using the imager tool and fill the MODEL_DATA column in the MS for target source:

```
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" 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);
```

Now apply the P, G, D, and T solutions and write a new CORRECTED_DATA column for the target source in the MS:

```
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
```

CLEAN the self-calibrated image of 0957+561 to see how the self-calibration 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.
```

Use the viewer to examine the final source image 0957+561.im2. The RMS should be ~ 0.24 mJy beam^{-1} (versus 0.5 mJy beam^{-1} before self-calibration). The peak in the image is now ~ 29.4 mJy beam^{-1} (versus 26.2 mJy beam^{-1} before self-calibration). Self-calibration improved the dynamic range by a factor of 2.3 in this example. 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).

spectral-line case

In this example 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 as shown below.

Create an image with line-free continuum channels only. First, create an image from line-free channels 5–8:

```
imgrS.setdata(mode='channel',         # Select channel data for 4
              nchan=4,                #  continuum-only channels in
              start=5,                #  field 3 (target field)
              step=1,
              spwid=1,
              fieldid=3);

imgrS.setimage(nx=256,                # Imaging parameters
               ny=256,
               cellx='10arcsec',
               celly='10arcsec',
               stokes='I',
               mode='channel',        # Choose channel mode
```

```
               start=5,              #  Grid selected channels into one.
               step=4,
               nchan=1,
               spwid=1,
               fieldid=3);

imgrS.weight(type='uniform');        # Uniform weighting

or

imgrS.weight(type='briggs',          # Robust weighting
             rmode='norm',
             robust=0.5);

imgrS.clean(algorithm='clark',       # Image and deconvolve
            niter=1000,              #  with Clark CLEAN
            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');
```

Form the model using the imager tool and fill the MODEL_DATA column in the MS for target source using only continuum channels that were used to create the model:

```
imgrS.setdata(mode='channel',        # Select channel data for 4
              nchan=4,               #  continuum-only channels in
              start=5,               #  field 3 (target field)
              step=1,
              spwid=1,
              fieldid=3);

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

We want an incremental solution over the current G, so 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=5,                 #  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=15,
             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");
```

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 4
             nchan=63,                #  continuum-only channels in
             start=1,                 #  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.setapply(type="T",                # Arrange to apply T solutions
             t=0,
             table="ngc5921.tcal");

calS.correct();                        # Correct the data
```

CLEAN the self-calibrated image of ngc5921 to see if the image is improved:

```
imgrS.setdata(mode='channel',        # Select channel data for field 3
             nchan=63,
```

```
              start=1,
              step=1,
              fieldid=3)

imgrS.setimage(nx=256,               # Select imaging parameters
               ny=256,
               cellx='10arcsec',
               celly='10arcsec',
               stokes='I',
               mode='channel',
               start=1,
               step=1,
               nchan=63,
               fieldid=3);

imgrS.weight(type='uniform');        # Uniform weighting

or

imgrS.weight(type='briggs',          # Robust weighting
             rmode='norm',
             robust=0.5);

imgrS.clean(algorithm='clark',       # Image and deconvolve
            niter=3000,              #  with Clark CLEAN
            gain=0.1,                #  Write the cleaned image to the file
            model='ngc5921.mod2',    #  ngc5921.im2 on disk.
            image='ngc5921.im2',
            residual='ngc5921.resid2'
```

Use the viewer to examine the final source image ngc5921.im2. The RMS is ~ 1.4 mJy beam^{-1} (versus 1.6 mJy beam^{-1} before self-calibration). The peak of the continuum source is now 89.2 mJy beam^{-1} (versus 81.8 mJy beam^{-1} before self-calibration). Thus, self-calibration provided a marginal improvement to the image (the dynamic range improved by $\sim 20\%$).

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).

1.13 Image analysis

Extensive display and image analysis tools are available in AIPS++. See the Display and Image analysis chapters of *Getting Results* for a detailed discussion on the options available. A brief orientation on the use of the Viewer is given below.

The Viewer

Start the viewer by typing at the *Glish* command prompt:

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

The viewer Data Manager and Display Panel GUIs will appear. Using the Data Manager, 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.

If you are using the GUI interface, you can also click on the "Wrench" icon from an image entry line in a tool window and select `view` to view a single image. The viewer is described in detail in the chapter on Display of Data while the 'Adjust' options are described at Adjust

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

How the image, axes, and labels are displayed is controlled via the 'Adjustment' panel. 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 clicking on the 'Save' button in the 'Adjustment' panel. The displaydata state can then be restored later by clicking on the 'Restore' button.

To obtain statistics on all or part of an image, click on the `Tools: ImageAnalysis` button at the top of the Viewer Display. An `Image Analysis` GUI will be created. 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 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 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 straight-forward examples of a few image analysis tools that are useful for analyzing the example data sets:

continuum polarimetry case

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 *Getting Results*

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

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)

impol.complexlinpol(outfile="0957+561.cmplx");
                                           # Create a vector image of the
                                           # electric vectors of the polarization

impol.done();                              # Close the tool (no longer needed)
```

Load the relevant images into the viewer tool using the Tool Manager window (that you created early in the session):

- 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' tool and determine the RMS of the polarization intensity image using the 'Statistics' panel (for the 0957+561.im2 image, the RMS $\sim 1.5 \times 10^{-4}$). value yet. You will use this to set a mask expression in the 0957+561.cmplx vector image to get rid of polarization vectors that are noise.

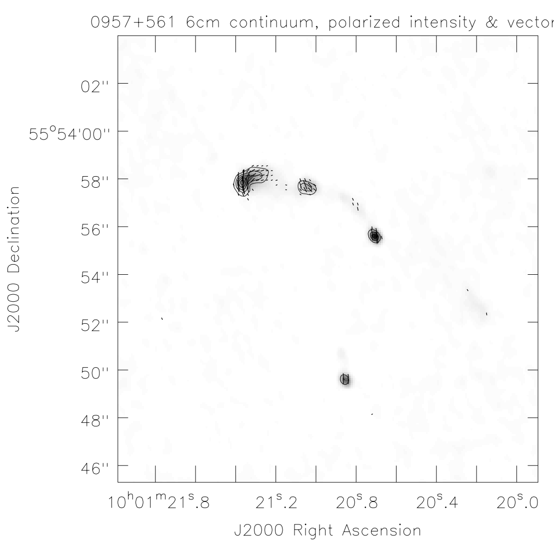
Click on the 'Adjust' button to bring up 3 Adjustment panels, one for each image. Then:

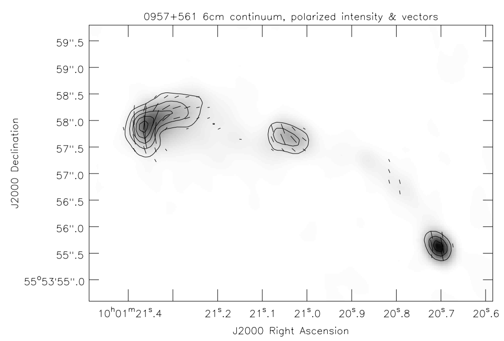
- Adjust the color map for 0957+561.im2 (Color palette of choice, Greyscale 1 for black & white Viewer, Greyscale 2 for output to a psfile).
- Adjust the size of the polarization vectors in 0957+561.cmplx using the 'amplitude scale factor', 'x-increment', and 'y-increment'.
- Set the mask expression in the 0957+561.cmplx image to be: '0957+561.pint' $>$ 0.00045 (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 angles, EVPA $= 0.5\tan^{-1}(U/Q) =$ $0.5*(\mathrm{R}-\mathrm{L}$ phase difference). First, it is necessary to specify the true ratio U/Q found for the polarization calibrator, 3C286. During the polarization calibration, we found for 3C286 ($I = 7.462 \;\; Q = -0.621 \;\; U = -0.593 \;\; V = 0$), thus, the measured R–L phase difference is: $\tan^{-1}(-0.593/-0.621) = 0.762338001$ radians or $\tan^{-1}(-0.593/-0.621)*180/\pi = 43.67875°$. Since U and Q are both negative, this is in the 3^{rd} quadrant. Thus, the angle is really $(-180+43.68) =$ $-136.32°$.

From the VLA calibration web page (http://www.aoc.nrao.edu/g̃taylor/calman/polcal.html) we find the true R-L phase difference for 3c286 is 66°. Thus, our phases should be rotated by $66 - 136.32 = 70.32°$ or $EVPA = 0.5*70.32°$. In the 'Adjust' panel for the 0957+561.cmplx image, enter `Extra rotation = 70.32/2`. To plot the Magnetic field vectors, enter `Extra rotation =` $70.32/2+90$.

Create postscript files of the resulting images:





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');
                                   # 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.

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.

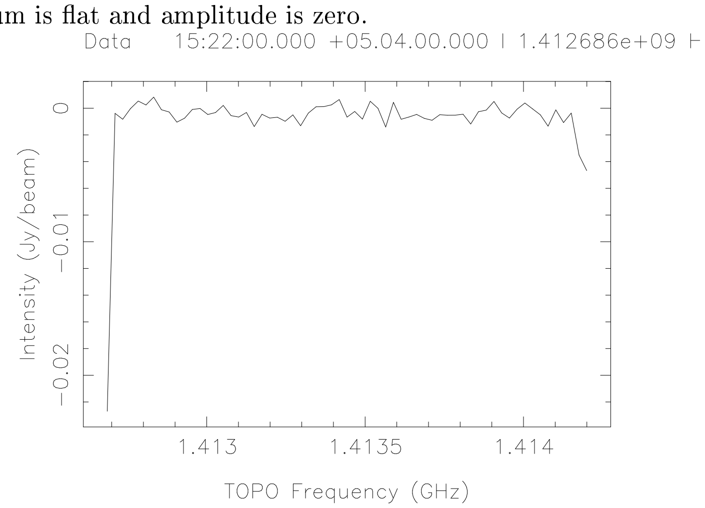
```
include 'imageprofilefitter.g';    # Create image profile fitter tool
```

```
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.
- Click on the 'Positioning' button (bulls-eye) to the left of the image.
- Use the left mouse button to 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 click in the region to get a spectrum.
- In the Profile Fitter window, click on 'Print' to print the spectral line plot to the printer or postscript file.

The spectrum below, 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.



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
                                          # 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');
```

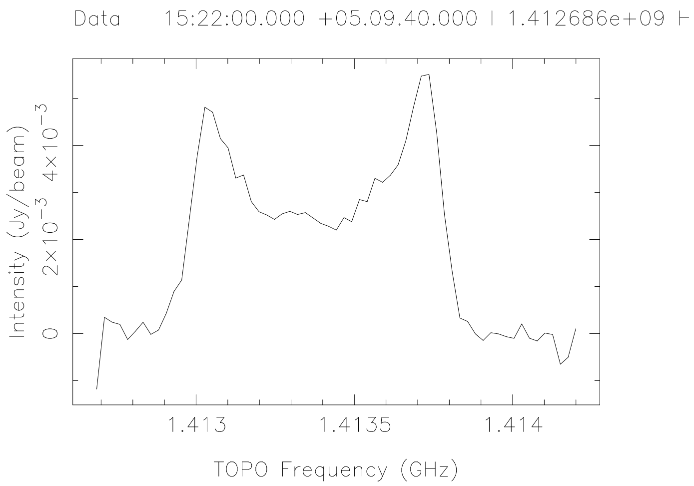
The moments command will create 3 images:

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



1.14 Cleanup

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

continuum polarimetry case

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

spectral-line case

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

1.15 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.

1.16 Appendix 1

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 Appendix 2

Detailed summary of the continuum polarimetry dataset (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                 Field          DataDescIds
   22-May-1998/22:40:05.0 - 22:41:10.0   0917+624       [1]
               23:05:30.0 - 23:14:00.0   0957+561       [1]
               23:26:55.0 - 23:47:30.0   0917+624       [1]
   23-May-1998/00:10:50.0 - 00:18:20.0   0957+561       [1]
               00:30:15.0 - 00:31:20.0   0917+624       [1]
               00:53:15.0 - 04:45:40.0   1328+307       [1]
Fields: 3
   ID   Name      Right Ascension  Declination   Epoch
   9    0917+624  09:21:36.23      +62.15.52.18  J2000
   10   0957+561  10:01:20.70      +55.53.52.65  J2000
   11   1328+307  13:31:08.28      +30.30.32.94  J2000
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>
```

Detailed summary of the spectral line dataset (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                 Field          DataDescIds
   13-Apr-1995/09:19:00.0 - 09:24:30.0   1331+30500002   [1]
               09:27:30.0 - 09:29:30.0   1445+09900002   [1]
               09:33:00.0 - 09:48:00.0   N5921          [1]
               09:50:30.0 - 10:23:00.0   1445+09900002   [1]
               10:26:00.0 - 10:43:00.0   N5921          [1]
               10:45:30.0 - 10:47:00.0   1445+09900002   [1]
```

```
Fields: 3
   ID   Name      Right Ascension  Declination   Epoch
   1    1331+3050000213:31:08.29      +30.30.32.96  J2000
   2    1445+0990000214:45:16.47      +09.58.36.07  J2000
   3    N5921     15:22:00.00      +05.04.00.00  J2000
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>
```

Chapter 2

GBT Standard Data Reduction Guide

J McMullin, J Braatz, B Garwood

2.1 Introduction

This chapter provides brief guidelines on how to reduce the currently supported observing modes at the GBT.

2.2 Example Session

An example data reduction session is shown to demonstrate the look and feel of data reduction in DISH. We calibrate and average position switched scans.

```
> . /home/aips++/gbt/aipsinit.sh  (BASH)
> source /home/aips++/gbt/aipsinit.csh (T/CSH)
>
> dish
- d.open('stjul13SP')
- d.gms()
Scan     Object   Proctype    SWState   SWtchsig   Procseqn   Procsize
  35         S8      Track    FSWITCH      FSW01          1          1
  36         S8      Track    FSWITCH      FSW12          1          1
  37         S8      Track    FSWITCH      FSW12          1          1
  38         S8      OffOn PSWITCHOFF   TPWCALSP          1          2
  40         S8      OnOff  PSWITCHON   TPWCALSP          1          2
  42         S8      Track       NONE   TPWCALSP          1          1
- d.calib(36)
- d.calib(37)
```

```
- avg\_spc:=d.aver([36,37]);
- d.plotscan(avg\_spc)
- d.header()
Proj : standards\_jul13    Src  : S8                Proc : Track
Obs  : Richard            RA   : 05:47:21.406      PType: SIMPLE
Scan : 36                 Dec  : -001.40.19.036    OType: LINE
Seq  : 1/1                Epoch: J2000             Swtch: FSWITCH
Date : 2002-07-13         Az   : -112.772          Swsig: FSW12
Time : 19:31:14.500 UT    El   : 23.733            Ints : 31

Tsys : 16.58              Trx  : 7.58              Tcal : 1.60
Tsys : 16.86              Trx  : 8.88              Tcal : 1.64

BW   : 4.995    (MHz)     Res  : 4.883    (kHz)
Expos: 54.549             Durat: 247.856
T
- d.setregion()
Use the left button to set location, right button to exit
region =  [15 180 296 681 810 1009]
T
- d.nfit(1)
T
- d.bshape()
T
- d.fileout('S8\_avgbl')
New file  S8\_avgbl  is created
T
- d.keep()
globalscan saved to  S8\_avgbl
T
```

2.3 Data Path

The following lists the most often used utilities. There are many more available functions in dish. Type "help('Refman:dish')" at the command line prompt to redirect your browser to the DISH command line documentation; 'd.info()' will also provide an alphabetical, table of all available DISH functions); typing d.help('function_name') will provide information on the function (d.help('function_name',T) will also automatically drive your web browser to the AIPS++ User Reference Manual entry on this function).

File Access	Scan Access	Editing	Calibration	Analysis	Display
import	getr	msplot	calib	(all operations)	plotr
open	getc	viewer	TPcal	e.g., aver	plotc
filein	getscan	clip	FScal	base	plotscan
fileout		*flag*	SRcal	etc	

2.4 Data and File Access

2.4.1 File Access

GBT data must be converted from their raw FITS files to an AIPS++ MeasurementSet before being accessed with **dish** functions. The command to fill raw GBT data is *import*. Once data have been filled, they can be accessed with the *open* function. Two additional file access functions, *filein* and *fileout*, are used to set the focus for where data is read from and written to.

Hint: The name of the MS should not begin with a numeral nor should it contain a '.'

1. *import* - Fills an AIPS++ MeasurementSet from raw GBT FITS files and makes the data accessible from **dish** via the *open* command.

   Example:

   ```
   d.import(projdir='/home/gbtdata/OHsurvey',outms='oh',startscan=100,stopscan=105)
   ```

   Parameters:

   - projdir - The project directory
   - outms - The name of the MeasurementSet to be created
   - outmsdir - The directory where outms will be written
   - startscan - First scan to fill
   - stopscan - Last scan to fill
   - backend - Backend to fill (e.g., 'SP' or 'SPECTROMETER')
   - calflag - Calibrate the data after filling (beta).

2. *open* - This function will load an existing MeasurementSet on disk into **dish** memory, that is, it can be used on datasets which have already been filled using the *import* function. This places it into the Results Manager, making it globally accessible in the Glish session (it can be manipulated from the command line, etc.). This also performs a *filein* on the MeasurementSet which indicates that subsequent commands/functions will use this MeasurementSet.

   Example:

   ```
   d.open('ohSP');
   ```

Parameters:

- file - The full path to the MeasurementSet file on disk
- access - The access permission on the MeasurementSet(r=read,w=write)
- new - Create a blank(new) MeasurementSet on disk
- corrdata - Look at the CORRECTED_DATA column (this will be a copy of the FLOAT_DATA (raw data) initially)
- filein - Point to this MeasurementSet for subsequent operations

3. *filein* - This function sets the scan group that will be examined and displayed. The *files* function will indicate the current settings for the selected filein and fileout. This allows the user to select between several open MeasurementSets in memory.

   Example:

   ```
   d.filein('ohSP')
   ```

   Parameters:

   - wsname - The name of the MeasurementSet in Glish (NOTE: if there are files with the same name, it will append the name with a numeric designator).

4. *fileout* - This function sets the scan group that will be written to. The *files* function will indicate the current settings for the selected filein and fileout. This allows the user to select between several open MeasurementSets in memory. *fileout* assumes that the function already exists and has write permission (e.g., you should have already created the file with *open*).

   Example:

   ```
   d.fileout('avg_spectra')
   ```

   Parameters:

   - wsname - The name of the MeasurementSet in Glish (NOTE: if there are files with the same name, it will append the name with a numeric designator).

2.4.2 Scan (Data) Access

Data in the MeasurementSet contains both the uncalibrated (raw or observed data) and the calibrated data. Initially, the calibrated data is simply a copy of the uncalibrated data. As the data is manipulated through the calibration routines (*calib, TPcal, FScal, SRcal*), the calibrated data becomes distinct.

The SDRecord (single dish record) is the data structure returned by the data access commands. The basic SDRecord has four sub-records which contain basic information on the observation:

- data - this contains the basic data (data, flags, weights, restfrequency, etc.)
- header - this contains the header information (scan number, project, bandwidth, tsys, etc.)
- hist - contains information on functions applied to data
- other - contains other non-standard information

See the Getting Results chapter on 'Single Dish Analysis' for a full breakdown of the SDRecord.

Several **dish** functions operate on a globally accessible SDRecord (spectrum) to help simplify the interface. In general, these functions will operate on the currently displayed spectrum or the contents of this globally accessible record (called *globalscan1*). Functions which take no arguments will in general access this spectrum and perform the operation on it.

1. *getr* - Retrieve uncalibrated data. This function will return a SDRecord (a Glish record with all of the astronomical information (e.g., data, header, history, etc)) of the result. Specifying only the scan and phase will yield an average over all available integrations. The function accesses the FLOAT_DATA column of the MeasurementSet (the raw data) which is never modified.

   Example:

   ```
   - s35_1 := d.getr(35,1,1,1) #retrieve scan 35, phase 1, integration 1,
                               #polarization 1 from the opened MS and
                               #store in variable s35_1
   - s35_1 := d.getr(35,1)     #retrieve scan 35, phase 1, average all
                               #integrations, show all polarizations
   ```

   Parameters:

   - scan - Scan number (required)
   - phase - Scan phase (required)
   - int - Scan integration (if more than 1). If this is not specified, it will average all integrations together.
   - pol - Polarization. If not specified, it will return all available polarizations.

2. *getc* - Retrieve calibrated data. This function will return a SDRecord (a Glish record with all of the astronomical information (e.g., data, header, history, etc)) with the result. Specifying only the scan will yield an average over all available integrations. The function accesses the CORRECTED_DATA column of the MeasurementSet (the calibrated/corrected data).

   Example:

```
- s35_c1 := d.getc(35,1,1)  #retrieve scan 35, integration 1,
                             #polarization 1 from the opened MS and
                             #store in variable s35_c1
- s35_c1 := d.getr(35)      #retrieve scan 35, average all integrations,
                             #show all polarizations
```

Parameters:

- scan - Scan number (required)
- int - Scan integration (if more than 1). If this is not specified, it will average all integrations together.
- pol - Polarization. If not specified, it will return all available polarizations.

3. *getscan* - Retrieve an individual row (phase, integration) from a scan.

   Example:

```
- s35_c1 := d.getscan(35,3) #retrieve scan 35, subscan 1
```

   Generally, you need to understand how the data was taken to effectively use *getscan*.

   Parameters:

   - scannum - Scan number (required)
   - subscan - Sub scan number (default=1)

Hint: Use d.qscan(scan_number) to obtain information about a scan

2.5 Data Inspection and Display

2.5.1 Flagging

Flagging data is done in two basic modes: 1) command line and 2) GUI. There are two basic command line flagging utilities, clip and flag.

clip is a simple function which will allow you to flag a currently viewed spectrum.

Example:

```
d.clip(0.4);    #clip all above 0.4
d.clip(0.4,[100:500]);  #clip above 0.4 in the channel range 100-500
d.clip(-0.3,clipdir='low'); #clip below -0.3
```

Parameters:

- cliplevel - Threshold for clipping

- channelrange - range to apply clip. Default is whole spectrum.

- clipdir - clip direction: 'high' (default) is above and 'low' is below.

flag is a more complete flagging utility allowing versatile flagging of data. Automated flagging based on data statistics is under development for single dish data.

Example:

```
d.open('stjul13SP');
d.gms();
d.plotc(36)
d.flag(36,channel=100:120)      #flag all subscans, polarizations of scan
                                #36 in the channel range of 100:120
d.plotc(36)                     #you should see the range masked
d.flag(36,flag=F);              #unset all flagging
d.plotc(36)                     #it's back!
d.flag(36,channel=500:550,polarization=1)
d.plotc(36)                     #you should see a gap just in the first
                                #polarization
d.flag(36,flag=F);
d.flag(36,1)                    #flag just the first integration
d.plotc(36)                     #should look okay though it doesn't
                                #include the first integration
d.plotc(36,1)                   #masked
d.plotc(36,2)                   #okay
```

Parameters:

- scan - scan number(s) to flag. If no scan is specified, it will flag all scans in the data according to the other parameters (e.g., all channels within a range for the MeasurementSet can be flagged in this manner).

- ints - integration number

- polarization - polarization number(s); if none specified, it will flag all polarizations.

- channel - channel (range) to flag; if none specified, it will flag all channels.

- flag - flag (T) or unflag (F) the data; the default is T.

autoflag

autoflag is a utility within AIPS++ which allows automated (based on statistics, etc) flagging of data without direct examination of each flagged row/scan. A few specific examples are provided below. See the

Flag based on statistics

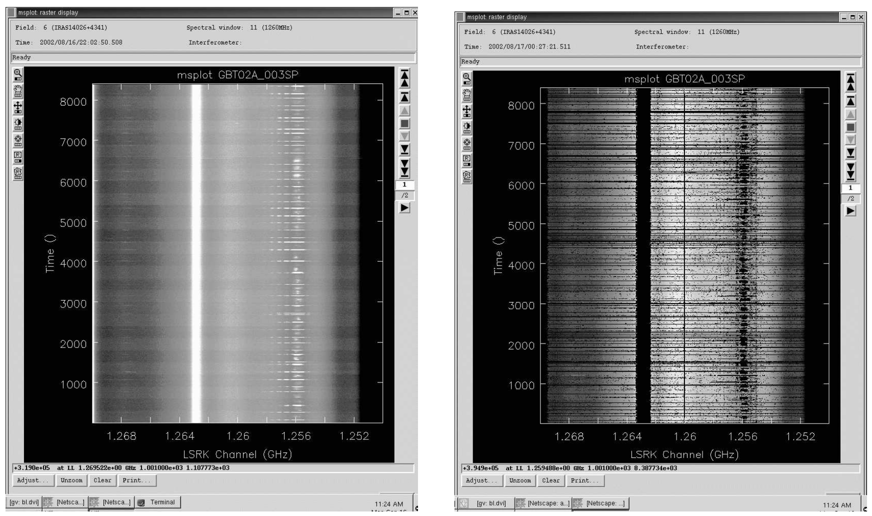


Figure 2.1: Flagged dataset using the following commands: af.setfreqmed() af.run(plotscr=F,reset=T), af.setfreqmed(), af.run(plotscr=F), af.settimemed(), af.run(plotscr=F).

```
include 'autoflag.g'
af:=autoflag('stjul13SP')       #Specify the MeasurementSet to operate on
af.setfreqmed(); #Use a sliding median filter in frequency
# default threshold is 5
af.run(plotscr=F);
af.done(); #important to release the MeasurementSet
```

Clip data using autoflag

```
include 'autoflag.g'
af:=autoflag('stjul13SP') #Specify the MeasurementSet to operate on

cliprec:=[=] #set up a record to contain the clipping info
cliprec[1]:=[expr='ABS XX',min=0,max=200]
#clip the XX correlation above 200 and below 0
af.setselect(clip=cliprec) #specify the clip record as the selection
af.run(plotscr=F) #execute (disable intermediate displays)
```

GUI modes of flagging are also enabled through the *viewer* and *msplot* tools.
For the viewer:

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

This will bring up two GUIs. The first is for selecting what to display. You select the MeasurementSet and choose the type of display (only 'Raster' is

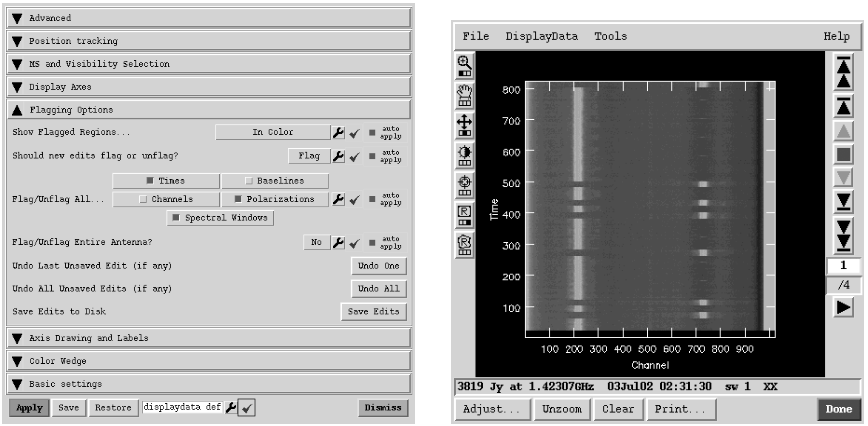


Figure 2.2: Display of the Adjust panel for the viewer with the flagging tab open alongside a viewer display. The data on the right has been flagged as bad. The selection will mark all times for the selected channels and all polarizations and spectral windows (from the Adjust panel selection)

.

available for MeasurementSets). This will then be displayed in the other display panel.

For msplot:

```
- include 'msplot.g'
- mp := msplot('your\_measurementset\_name');
```

This brings up a GUI which will allow you to select the type of display ("X vs. Y" or "Display data as image") and optionally select a subset of the data.

2.5.2 Display

1. *plotr* - Display the raw (observed data: in the FLOAT_DATA column) on the **dish** plotter.

   arameters:

   - scan - Scan number (required)
   - phase - Scan phase (required)
   - int - Scan integration (if more than 1). If this is not specified, it will average all integrations together.
   - pol - Polarization. If not specified, it will return all available polarizations.

2. *plotc* - Plots the calibrated (CORRECTED_DATA column) on the **dish** plotter.

   Parameters:

- scan - Scan number (required)
- int - Scan integration (if more than 1). If this is not specified, it will average all integrations together.
- pol - Polarization. If not specified, it will return all available polarizations.

3. *plotscan* - Plots an SDRecord to the **dish** plotter (as returned from functions like *getr, getc, getscan*.

   Parameters:

   - scanrec - A scan record (SDRecord, required)
   - overlay - Overlay the plot (T=true, F=false,default).

2.5.3 The dish plotter

The plotter is the standard **dish** display for spectral line data. It is built on the existing AIPS++ pgplotter tool and so has all of the basic PGPLOT functionality. In addition, the plotter features tools for:

1. changing the X-axis units
2. changing the reference frame for the X-axis
3. toggling header information display
4. toggling the overlay of subsequent plots
5. options for a) reversing the color scheme and b) providing a chalkboard draw facility
6. toggling the X-axis autoscale
7. unzoom (incrementally unzooms any zooms)
8. activate the gaussian fit tool
9. line identification (marks any molecular lines within the frequency ranges of the displayed plot

2.6 Calibration (SP,SPECTROMETER)

Calibration is applied to the data based on information in the STATE table of the MS, in particular, it utilizes the OBSMODE column to determine a procedure type, switching signal, and switching state. In addition, information on the procedure size and procedure sequence number are used.

The function applies a number of corrections:

1. determine elevation, frequency; solve for opacity,gain correction.

2. determine units used, solve for appropriate efficiency factor from table.

$$factor = \frac{e^{\tau(1/sin(elev))}}{\eta_x}$$

where η_x is the efficiency factor determined by the user selected units (η_l for TA*, $\eta_m b$ for TMB).

3. based on procedure, calibrate data. Typically, this entails calibrating "cal on" and "cal off" phases of signal (on source) and reference (off source) observations, where the "cal" is a known noise diode. e.g.,

$$V_{sig}(\nu) = \frac{(V_{sig,on}(\nu)w_{sig,on} + V_{sig,off}(\nu)w_{sig,off})}{w_{sig,on} + w_{sig,off}}$$

Similarly for the reference observation, leading the difference spectrum:

$$T_{diff}(\nu) = < T_{sys,ref} > \frac{V_{sig}(\nu) - V_{ref}(\nu)}{V_{ref}(\nu)}$$

where, $< T_{sys,ref} >$ is:

$$< T_{sys,ref}(\nu) > = < \frac{T_{cal}(\nu)V_{ref}(\nu)}{V_{ref,on}(\nu) - V_{ref,off}(\nu)} >$$

1. *calib* - Function will calibrate any **known** procedure. It determines the procedure sequence number and looks in the MeasurementSet for the other scans in the procedure. It then does the appropriate thing (e.g., if it is a map, it will calibrate all of the scans in the map, if it is a OnOff scan sequence it will calibrate both scans and then perform the (sig-ref)/ref. See the calib function documentation in the User Reference Manual for dish for more details on this.

   Parameters:

   - scan - The scan number or vector of scans
   - baseline - T or F to designate subtraction of a polynomial
   - range - The channel range over which to fit for a polynomial
   - order - The order of the fit for the polynomial
   - units - The output units: 1) TA*, 2) TMB, 3) Jy (not enabled)

2. *TPcal* - Function to calibrate any TPwCAL scans. Unlike *calib*, it will only calibrate the specified scan(s).

   Parameters:

   - scan - The scan number or vector of scans
   - baseline - T or F to designate subtraction of a polynomial

- range - The channel range over which to fit for a polynomial
- order - The order of the fit for the polynomial
- units - The output units: 1) TA*, 2) TMB, 3) Jy (not enabled)

3. *FScal* - Function to calibrate any FSWITCH scans. Unlike *calib*, it will only calibrate the specified scan(s).

   Parameters:

   - scan - The scan number or vector of scans
   - baseline - T or F to designate subtraction of a polynomial
   - range - The channel range over which to fit for a polynomial
   - order - The order of the fit for the polynomial
   - units - The output units: 1) TA*, 2) TMB, 3) Jy (not
   - flipsr - Flip the sense of Sig of Ref
   - fold - Fold the data
   - flipfold - Flip the sense of which is Sig and Ref in the folded data enabled)

4. *SRcal* - Function to perform a (SIG-REF)/REF calibration. Unlike the other calibration routines, *SRcal* uses the CORRECTED_DATA column (remember this is identical to the observed data in FLOAT_DATA if no calibration has been performed); this allows a layered approach to TP-wCAL scans which can be noise diode calibrated and then subsequently calibrated with *SRcal*. *SRcal* is ideally suited for situations with many SIGs per REF. *calib*, it will only calibrate the specified scan(s).

   Parameters:

   - scans - The scan number or vector of scans that are on signal
   - refscans - The scan number or vector of scans that are on reference - These are averaged and then applied to each signal scan
   - baseline - T or F to designate subtraction of a polynomial
   - range - The channel range over which to fit for a polynomial
   - order - The order of the fit for the polynomial
   - units - The output units: 1) TA*, 2) TMB, 3) Jy (not enabled)

2.7 Analysis

Each of the following operations has a GUI counterpart. We describe only the command line interface.

2.7.1 Averaging

The functionality for averaging is provided in the *aver* function.

aver - Average a group of scans, subscans.

Example:

```
- myavg := d.aver([303,305:308]) #average scans 303,305,306,307,308
- myavg := d.aver([159:165],'1/4')      #average every fourth subscan in
-                                       #scans 159-165
```

Generally, you need to understand how the data was taken to effectively use *getscan*.

Parameters:

- scanlist - a vector of scan numbers
- subscanlist - a vector of subscans. This can also be the string 'odd', 'even', or 'n/m'. 'odd' will average every odd subscan in the specified scans, 'even' will average every even subscan in the specified scans, and 'n/m' will select every nth subscan in a sequence of m (for example ('1/4' will select subscans [1,5,9,..]).
- weighting - 'NONE', 'RMS', 'TSYS'
- alignment - 'NONE', 'VELOCITY', 'XAXIS'

There is also a means of averaging individual scans using an accumulator. Three functions are relevant: sclear(), accum(), and ave().

Example:

```
d.sclear()    # clear the accumulator
d.getc(35)    # retrieve calibrated scan 35
d.accum()     # add the scan to the accumulator
d.getc(36)    # retrieve calibrated scan 36
d.accum()     # add the scan to the accumulator
d.ave()       # average scans in the accumulator
d.show()      # display the result on the plotter
```

Function sclear() takes no parameters, and clears the accumulator.

Function accum() also takes no parameters, and it adds the current contents of the globalscan1 spectrum into the accumulator.

Function ave() averages the contents of the accumulator, and the average is returned to the globalscan1 spectrum for further processing.

A UNIPOPS-like stack is also available. The stack is simply a vector of scan numbers which are intended to be averaged, or operated on in some other manner. It is possible to use Glish methods to accomplish the formation and manipulation of vector of scan numbers; nevertheless, these stack techniques provide another path for those familiar with the UNIPOPS syntax.

Four functions are used to manipulate the stack: empty(), addstack(), delete() and tellstack().

Example:

```
d.empty()                   # clear the stack
d.addstack(10,50,2)         # add even scans from 10 through 50 to the stack
d.delete(24)                # remove scan 24
d.delete(28)                # remove scan 28
d.tellstack()               # show the contents of the stack
d.aver(d.tellstack())       # average the scans listed in the stack
```

Function empty() takes no parameters, and simply clears the stack.

Function addstack(beg,end=beg,inc=1) is used to append new entries to the stack. Parameters:

- beg : the first scan numer to add
- end : the last scan number to add
- inc : the increment

So, addstack(10) appends scan 10 to the stack; addstack(15,18) appends scans 15, 16, 17 and 18 to the stack, and addstack(21,28,3) appends scans 21, 24 and 27 to the stack. If these three commands were executed on an empty stack, the result would be [10, 15, 16, 17, 18, 21, 24, 27].

Function delete(value) removes a given scan from the stack. So to remove scan 16 from the stack constructed above, execute d.delete(16).

Function tellstack() takes no parameters, and is used to list the entries in the stack, and also to pass the stack to other functions in AIPS++. For example, to average the entries in the stack, execute d.aver(d.tellstack()).

2.7.2 Fitting Baselines

Baselines can be removed during the calibration step; this is the most efficient means if the baseline regions are constant and there are many to perform. Additional functions for baselining are also provided in the *base* function.

base - Perform a polynomial or sinusoidal baseline fit to a scan.

Example:

```
mybl := d.base(order=2,action='subtract',range='[50:400],[1500:2000]')
                                        # This will perform a 2nd order poly.
                                        # fit to the currently displayed scan.
```

Generally, you need to understand how the data was taken to effectively use *getscan*.

Parameters:

- scanrec - an SDRecord. It will use the current display if none is specified.
- order - the order of the polynomial
- range - channel range for computing baseline. NOTE: currently this is specified as a string (e.g., range='[50:100],[800:1000]')

- action - 'subtract' (to subtract the baseline) 'show' (to display the fit)
- cursor - T (to set the region with a cursor; F is default)
- autoplot - T (to automatically plot the results; F is default)

One can also use fit a baseline with the UniPOPS-like functions.

```
d.nregion(50,100,900,1000)# Provide pairs of numbers as the channel range for the
                          # baseline region
d.setregion()             # This will prompt the user to set the baseline regions
                          # with a cursor
d.nfit(1)                 # Fit a first order polynomial
d.bshape()                # Show the baseline fit superposed on the spectrum
d.baseline()              # Subtract the baseline; this also calls the
                          # d.rms() function displaying the baseline region stats
d.show()                  # Show the result
```

2.7.3 Fitting Gaussians

Fitting Gaussians is best done with the GUI interface. Pressing the GFit button on the **dish** plotter will pull up the Profile Fitter window (if there are multiple polarizations, they need to be fit separately and so should be first displayed with the *getr* or *getc* functions or after use of the *getpol* function).

The appropriate sequence of events is:

1. Select the number of gaussian components. This provides enumerated buttons which allow you to set and get the properties of each component.
2. Get an estimate for each component:
   - Hit the Green "Estimate" button on the bottom of the window. This works best for single strong lines.
   - Hit the "Inter" button (under Estimates) to allow interactive (cursor) setting of the estimate gaussian parameters. If you select this (recommended), the message window in the lower left of the window will prompt you to "Put cursor at line center and click" followed by "Put cursor at line HWHM and click". After this is done, the flux, pos (center) and width entries are filled with the estimate. A yellow line will also appear showing the estimated gaussian. Each of these parameters can be edited or fixed.
   - Repeat this for each component. Select each enumerated button to toggle through the components.
3. Fit the profile. Hit the Green "Fit profile" button to fit all the components.

The Plotter will show the Estimate (yellow), the Fit (green) and the Residual (red). Each of these can be toggled on/off by going to the "Plot" menu in the Profile Fitter window and selecting/deselecting a plot.

The gauss fitting routine can also be used at the command line.

gauss - Fit gaussians to displayed spectrum.

Example:

```
- d.gauss(2)
defaultregionmanager (drm) ready for use
Creating (temp)image of shape [1023]
Gauss:  1
Center: 1.42037e+00   Height: 5.44724e+01    Width: 4.20960e-05
Cerr: 3.71902e-07   Herr: 1.30446e+00    Werr: 1.20558e-06
Gauss:  2
Center: 1.42037e+00   Height: 9.25963e+00    Width: 1.71623e-04
Cerr: 4.43034e-06   Herr: 1.10727e+00    Werr: 1.43272e-05
T
```

Parameters:

- ngauss - Number of gaussians to fit.
- guess - Estimates of the height, center and width
- pol - Polarization to fit

2.7.4 Moments and Statistics

There are several functions for obtaining statistical information on a given spectrum.

1. *stat* - Provide statistics on a spectrum.

   Example:

   ```
   - d.stat() # look at spectrum (full range)
   [peak=68.9537735, area=0.00448239934, min=-0.303991884, rms=5.74451895,
   scan=35, centroid=1.42037562, vpeak=1.4203805, startint=1.41901819,
   stopint=1.42400843]
   - d.stat(range='[700:800]'); # look at line
   [peak=68.9537735, area=914.18058, min=0.242817193, rms=16.1565856,
   scan=35, centroid=745, vpeak=744, startint=700, stopint=800]
   - d.stat(range='[550:650]') # look at line free region
   [peak=0.180041969, area=-1.33219323, min=-0.202106252, rms=0.0901912276,
   scan=35, centroid=0, vpeak=554, startint=550, stopint=650]
   ```

   Parameters:

- scanrec - SDRecord to smooth; default is last viewed
- range - Channel range to provide statistics over

2. *rms* - Display statistics on a region.

   This function will calculate statistics on the regions set via the *setregion* or *nregion* commands.

   Example:

```
- d.rms()
Using 1023 for calculating baseline stats.
== Feed  1  ===
RMS   =  3.24550665    Mean =  0.000737983237    Num points =  1023
max   =  48.9361229   min =  -12.9025164
nregion =  [1 1023]
== Feed  2  ===
RMS   =  3.29074012    Mean =  0.351124813    Num points =  1023
max   =  50.4430733   min =  -14.0051394
nregion =  [1 1023]
T
```

3. *stats* - Display statistics over a region.

   This function will calculate statistics on a spectrum. It will use the global echan and bchan parameters for the region if available; these can also be set within the function.

   Example:

```
- d.stats()
Feed : 1       bchan: 1        rms  : 5.744519       min  : -0.303992
Npts : 1023    echan: 1023     mean : 0.898234       max  : 68.953773

Feed : 2       bchan: 1        rms  : 6.180395       min  : -0.329462
Npts : 1023    echan: 1023     mean : 0.912738       max  : 73.913177

T
```

   Parameters:

   - quiet - Suppress write of information to screen
   - feed - Limit information to a single polarization
   - bchan - Beginning channel for region
   - echan - End channel for region

4. *moment* - Display statistics over a region.

   This function will calculate statistics on a spectrum. It will use the global emoment and bmoment parameters for the region if available; these can be set with the *bmoment* and *emoment* functions.

   Example:

```
- d.bmoment(700)
T
- d.emoment(800)
T
- d.moment()
[peak=68.9537735, area=914.18058, min=0.242817193, rms=16.1565856,
scan=35, centroid=745, vpeak=744, startint=700, stopint=800]
```

2.7.5 Regridding

smooth - Smooth the resolution of a spectrum
Example:

```
- smooth_spc:=d.smooth(type='HANNING',decimate=T);
- d.plotscan(smooth_spc)
```

Parameters:

- scanrec - SDRecord to smooth; default is last viewed
- type - "HANNING", "BOXCAR", "GAUSSIAN" (default is "HANNING")
- width - The boxcar smoothing width in channels.
- decimate - decimate when smoothing (default=F)

boxcar, *hanning*, and *chngres* are streamlined cases for the *smooth* function using type BOXCAR, HANNING, or GAUSSIAN respectively, and with decimation=T.
boxcar - Perform a boxcar smooth on a spectrum
Example:

```
- d.show()
- d.boxcar(5)
- d.show()
```

Parameters:

- smooth_width - The boxcar smoothing width in channels.

hanning - Perform a hanning smooth on a spectrum
Example:

```
- d.show()
- d.hanning()
- d.show()
```

chngres - Perform a gaussian smooth on a spectrum
Example:

```
- d.show()
- d.chngres(5)
- d.show()
```

Parameters:

- smooth_width - The gaussian smoothing width in channels.

2.7.6 Saving Spectra

Saving spectra to into a new or existing output file is done using the *fileout, lsoutfile, save,* and *keep* commands. *fileout* was described in the Section on File access.

save, keep - Save spectrum to output file.
Example:

```
- d.files()
Current filein is  :  N1SP
Current fileout is :  exampleout
T
- d.save()   #save currently viewed spectrum to output file.
```

Parameters:

- lv - An SDrecord. The default is last viewed.
- outf - Specify a new output file (this must exist and be writable).

keep does the same thing as *save* in the default case; it takes no arguments.
lsoutfile - List the scan numbers in the specified output file.
Example:

```
- d.fileout('exampleout')
New file  exampleout  is created
- d.save()
- d.lsoutfile()
159
```

2.7.7 Writing Spectra

writetofile - Write ASCII columned data of x and y to a disk file.
Example:

```
- d.writetofile('final.spc'); #default is to write currently viewed spectrum.
```

The columns that are written out are: 1) the x-axis in the currently displayed units and 2)-n) the data arrays for each polarization/receptor.
Parameters:

- outfile - The output file on disk.
- scanrec - Scan record to write (default is currently displayed)

2.7.8 Imaging

Imaging in **dish** is done with the *imager* tool in AIPS++. Currently, there is a single function *makeim* in **dish** to achieve this.
makeim - Make a spectral line data cube.
Example:

```
- d.makeim('final.spc'); #default is to write currently viewed
```

This function selects the appropriate data from the MeasurementSet, regrids the data and constructs the image. The data are converted into the coordinate system of the image and then added to the image using a gridding function (BOX, SF, or PB). It also constructs a coverage image which is simply the gridded weights; this is used to normalize the data for the sampling density - the final output image is the image name with a "_corr" attached (imname_corr).
Parameters:

- scan - The starting scan in a mapping procedure.
- start - The starting channel along the spectral axis.
- stop - The end channel along the spectral axis.
- step - The number of channels to average. Default=10
- imname - The output image name. Three images will be created:
  1. imname
  2. imname_weight
  3. imname_corr
- spwid - Spectral Window ID; this is set automatically but can be overridden here.

- nx - Number of grid points in x. This is automatically calculated by the program to be (map_extent_x/cellx) + buffer (several points, depends on convolution function). This may be over-ridden here.
- ny - Same as nx but for latitude.
- cellx - The cell size is automatically set by the frequency and the size of the telescope to be 1/2 * lambda/D. It may be over-ridden here.
- celly - The same for y direction of cell. displayed)

Moments

The **image** tool function *moments* offers traditional moment analysis (weighted sums of profiles) of images. It offers a wide range of moments and a wide range of methods with which to calculate them. The word 'moment' is used loosely here. It refers to collapsing an axis (the moment axis) to one pixel and setting the value of that pixel (for all of the other non-collapsed axes) to something computed from the data values along the moment axis. For example, take an RA-DEC-Velocity cube, collapse the velocity axis by computing the mean intensity at each RA-DEC pixel. This function offers many different moments and a variety of interactive and automatic methods to compute them. A quick summay of the available moments:

- -1 – the mean value of the spectrum

$$\frac{1}{n}\sum I_i$$

- 0 – the integrated value of the spectrum

$$M_0 = \sum I_i$$

- 1 – the intensity weighted coordinate (this is traditionally used to get 'velocity fields')

$$M_1 = \frac{\sum I_i v_i}{M_0}$$

- 2 – the intensity weighted dispersion of the coordinate (this is traditionally used to get 'velocity dispersion fields')

$$\sqrt{\frac{\sum I_i \left(v_i - M_1\right)^2}{M_0}}$$

- 3 – the median of I

- 4 – the median coordinate. Here we treat the spectrum as a probability distribution, generate the cumulative distribution, and then find the coordinate corresponding to the 50% value. This moment is not very robust, but it is useful for quickly generating a velocity field in a way that is not sensitive to noise. However, it will only give sensible results under certain conditions. The generation of the cumulative distribution and the finding of the 50% level really only makes sense if the cumulative distribution is monotonic. This essentially means only selecting pixels which are positive or negative. For this reason, this moment type is only supported with the basic method (see below – i.e. no smoothing, no windowing, no fitting) with a pixel selection range that is either all positive, or all negative

- 5 – the standard deviation about the mean of the spectrum

$$\sqrt{\frac{1}{(n-1)}\sum\left(I_i-\bar{I}\right)^2}$$

- 6 – the root mean square of the spectrum

$$\sqrt{\frac{1}{n}\sum I_i^2}$$

- 7 – the absolute mean deviation of the spectrum

$$\frac{1}{n}\sum\left|(I_i-\bar{I})\right|$$

- 8 – the maximum value of the spectrum

- 9 – the coordinate of the maximum value of the spectrum

- 10 – the minimum value of the spectrum

- 11 – the coordinate of the minimum value of the spectrum

```
- im:=image('g29im')  #g29im is the data cube
- im.moments(moments=[-1,0])  #get the average and integrated moments
Moment axis type is Frequency
```

```
**********************************************************************
You have selected the following methods
The basic method
Created /home/despina2/scratch/jmcmulli/g29im.average
Created /home/despina2/scratch/jmcmulli/g29im.integrated
Begin computation of moments
Finished image::moments
       0.48 real        0.08 user        0.02 system
T
- im2:=image('g29im_integrated')
- im2.view()
- im.summary();
- r:=drm.box(blc='1 1 1 5', trc='20 20 1 100') # create a region
- #using the channels 5 to 100; the first 2 coordinates are ra, dec,
- #the 3rd is stokes, and the 4th is the channel axis.
- im.moments(moments=0,region=r,outfile='g29im_lowvel') #create the
- #integrated intensity over channels 5-100
```

You can also use the GUI interface via:

```
- im:=image('g29im')            #g29im is the data cube
- im.momentsgui(); #use GUI to derive moments
```

See the User Reference Manual on the *moments* function for the most details moments and moment methods.

Source Fitting

Source fitting is achieved through the **imagefitter** tool. From the example above:

```
- imfit:=imagefitter('g29im_integrated')
- #Use the GUI to select a region (double click to derive source characteristics,
- #statistics, residuals, etc. The 'show' button will display the fit value.
```

Image math

Manipulation of images is done with the **imagecalc** tool. With this you can construct an image from expressions involving other images. For example,

```
im1 := imagecalc(outfile='x3', pixels='sin(x1) + min(x2) ')
```

where the image disk files *x1* and *x2* must pre-exist and the result is the image disk file called *x3*.

You can also construct a read-only image from expressions involving other images. The output image is never created, it is just evaluated each time you use it.

```
im2 := imagecalc(pixels='pi() - min(x2) + sqrt(x1)')
```

where the image disk files *x1* and *x2* must pre-exist. Note that by simply excluding the output file name (argment *outfile*) you have created the read-only image.

With the **image** *calc* function, you can change the pixel values of an image by replacing them with the result of the expression:

```
im1 := image('x2')
im2 := im1.calc('abs(x2) / min(x2)')
```

where the image disk file *x2* must pre-exist. Because the scalar expression is evaluated first, the images in the expression can be the same as the image which is being overwritten.

Note that for image file names with special characters in them (like for example), you should (double) escape those characters or put the file name in ouble quotes. E.g.

```
- im1 := imagecalc(pixels='test\\-im')            # Note double escape
- im2 := imagecalc(pixels='"test-im"')
```

When using *imagecalc*, you can use either image disk file names, or an associated **image** tool name in the expression. For example,

```
im1 := image('im1')
im2a := imagecalc('im2a', '2*im1')     # Use disk file names
im2b := imagecalc('im2b', '2*$im1')    # Use tool name
```

For information on image manipulation (concatenation, addition, etc), see the Getting Results on Image Analysis.

2.8 Exiting dish

To exit **dish**, type 'exit' at the Glish command line. If you have enabled the "Save state when done" option, the state of **dish** will be preserved for your next session (it will be loaded automatically). You can also manually save the state of **dish** using the File menu options of the main **dish** GUI.

Hint: It is usually useful to purge the logger before exiting.

Type dl.purge(n), where n are the number of logger lines to keep

2.9 Appendix

2.9.1 Index of commands

```
aver     Average a group of scans,subscans.
```

```
base     Perform a polynomial baseline fit to a scan.
baseline  Perform a polynomial baseline fit to spectrum in globalscan1
bdrop     Set value for dropping beginning channels of a spectrum in display
boxcar    Perform a boxcar smoothing
bshape    Show the baseline fit of order nfit to globalscan1
calib     Apply calibration to a scan (GBT only currently).
clearrm   Clear all variables from results manager, etc.
clip      Clip/Edit data (set flags)
done      Exits and destroys the current Dish tool.
edrop     Set value for dropping end channels of a spectrum in display
filein    Sets the scan group that will be manipulated.
fileout   Sets the scan group that can be written to.
files     Prints the currently set values for filein/out.
find      Do a grep on a string to find glish variables.
galactic  Provide the coordinates in the Galactic reference frame.
gauss     Fit gaussian profiles.
getc      Retrieve calibrated data (from the CORRECTED\_DATA column)
getr      Retrieve raw data (from the FLOAT\_DATA column)
getscan   Retrieve a scan (SDRecord) from a scan group.
getvf     Return the velocity and frequency for a specified channel.
getvfarray Return the velocities and frequences for the currently viewed
          spectrum.
gms       Print MeasurementSet summary to screen.
gui       Toggles mapping of DISH main gui (results manager).
hanning   Perform a hanning smooth
header    Print formatted header information to the command line window
help      Provide simple help on dish functions
history   Add a string or vector of strings to the history of a current SDRecord
imcal     Calibrate an imaging/mapping procedure
import    Import GBT data into an AIPS++ MeasurementSet
keep      Saves the current globalscan1 SDRecord to the designated fileout.
listscans List the scan numbers from the active scan group.
logcommand Writes command to scripter.
lsoutfile Lists scans in the fileout scangroup
makeim    Make an image from scans in a MeasurementSet.
message   Writes a command to the DISH GUI message entry.
mscal     Calibrate a MeasurementSet.
mult      Scales (multiplies) a scan by a value.
nfit      Set the order of the polynomial for baseline subtraction.
nogui     Unmaps frame to DISH main GUI.
nregion   Set the channel ranges for baseline subtraction.
open      Load a scan group from disk into DISH and perform a fil ein on it.
page      Clears the DISH plotter display.
peak      Obtain the peak amplitude, half-width and center in the displayed
          spectrum
plotc     Plot calibrated data (from the CORRECTED\_DATA column)
```

```
plotr     Plot raw data (from the FLOAT\_DATA column)
plotscan  Plot an SDRecord.
plotxy    Plot two vectors in the DISH plotter.
qscan     Query characteristics of scan.
reference See TPcal
restore   Retrieve the contents of the uni Record from disk.
restorestate Restore the state of DISH's GUIs.
rmadd     Add an SDRecord to the results manager.
rms       Display statistics on baseline regions.
save      Save an SDRecord to file.
savestate Save the current state of DISH.
saxis     Set the units of the plotters displayed x axis
scale     Multiply the array in an SDRecord by a value.
scanadd   Add scan1 and scan2.
scanbias  Adds a bias level to a scan.
scandiv   Divide scan 1 by scan 2.
scanmult  Multiply scan 1 by scan 2.
scanscale Scale a scan by a value.
scansrr   Performs a signal-reference/reference.
scansub   Subtract scan2 from scan1.
select    Select a subset of a scangroup.
setregion Set the channel range for use in baseline subtraction
show      Display the globalscan1 SDRecord
show1     Display one polarization of the globalscan1 SDRecord
signal    See TPcal
smooth    Smooth (hanning, boxcar or gaussian) an SDRecord.
stat      Generate statistics over a line region.
statefile Changes file for state saving.
stats     Generate statistics over a region.
store     Store the contents of the uni Record on disk.
summary   Print out a more detailed summary of scans from a scan group.
upr       Print the current value of the specified variable in the uni-record.
utable    Print channel and data columns to command window
writetofile Write X,Y axes to an ascii file on disk.
```

2.9.2 Observing Recipes (SP, SPECTROMETER)

In the following, the > designates a UNIX prompt, and a '-' designates the glish prompt.

Frequency Switched data: Pointed observations

This section details calibration of data taken in the following modes:

- Track:FSWITCH:FSW01
- Track:FSWITCH:FSW12

```
> source /home/aips++/gbt/aipsinit.sh           #load environment
> dish                                          #start AIPS++ DISH tool
...
- d.open('exampleSP')
T
- d.files() #show active files
Current filein is  :  exampleSP
Current fileout is :  not set
T
- d.gms()                                       #provide brief summary
Scan     Object   Proctype    SWState   SWtchsig   Procseqn   Procsize
  35         S8      Track    FSWITCH      FSW01          1          1
  36         S8      Track    FSWITCH      FSW12          1          1
  37         S8      Track    FSWITCH      FSW12          1          1
  38         S8      OffOn PSWITCHOFF   TPWCALSP          1          2
  40         S8      OnOff  PSWITCHON   TPWCALSP          1          2
  42         S8      Track       NONE   TPWCALSP          1          1
T
- #Note: This only lists the first scan in a procedural sequence of scans
- d.listscans() #shows all scans present
[35 36 37 38 39 40 41 42]
- d.plotr(35,1)                                 #plot the raw(observed) data
                                                #for the first phase of scan
                                                #35. Since no integration
                                                #value is specified, it averages
                                                #all phase 1 data (in this case
                                                #this is all data with CAL ON.
-
- d.calib(35)                                   #this will calibrate scan 35
                                                #according to the way the
                                                #the data were taken (i.e., the
                                                #observing procedure
- d.plotc(35)                                   #plot the calibrated data for
                                                #scan 35. Since no integration
                                                #was specified, all data is
                                                #averaged.
- d.plotc(35,1)                                 #plots the first integration of
                                                #the calibrated data for 35.
- d.plotc(35,3,2)                               #plots the third integration of
                                                #the second polarization for
                                                #scan 35.
- d.plotc(scan=35,int=3,pol=2)                  #same using the explicit names
                                                #for the arguments.
-
- #HINT: if you are uncertain as to the arguments for a function, simply type
- #      the function name.
```

```
- d.plotc
function (val scan, val int = F, val pol = F) {
(ok := public.plotscan(public.getc(scan, int, pol)));
return T}
- #NOTE: if you are unhappy with the calibration, you can simply reapply it
- #      using different attributes
- d.calib(35,baseline=T,range=[300:600,800:1000]);
                                                #Apply calibration but also
                                                #baseline the data using the
                                                #specified range (defaults
                                                #to first order but this can
                                                #also be specified.
- #This will over-write the calibration data with the new calibration data.
- d.plotc(35)                                  #re-examine the data
```

Total Power observations

This section details calibration of data taken in the following modes:

- Track:NONE:TPWCAL
- Track:NONE:TPWCALSP

```
> source /home/aips++/gbt/aipsinit.sh           #load environment
> dish                                          #start AIPS++ DISH tool
...
- d.open('exampleSP')
T
- d.files()                                     #show active files
Current filein is  :  exampleSP
Current fileout is :  not set
T
- d.gms()                                       #provide brief summary
Scan     Object   Proctype    SWState   SWtchsig   Procseqn   Procsize
  35         S8      Track    FSWITCH      FSW01          1          1
  36         S8      Track    FSWITCH      FSW12          1          1
  37         S8      Track    FSWITCH      FSW12          1          1
  38         S8      OffOn PSWITCHOFF   TPWCALSP          1          2
  40         S8      OnOff  PSWITCHON   TPWCALSP          1          2
  42         S8      Track       NONE   TPWCALSP          1          1
T
- #Note: This only lists the first scan in a procedural sequence of scans
- d.listscans()                                 #shows all scans present
[35 36 37 38 39 40 41 42]
- d.plotr(42,1)                                 #plot the raw(observed) data
                                                #for the first phase of scan
                                                #42. Since no integration
```

```
                                              #value is specified, it averages
                                              #all phase 1 data (in this case
                                              #this is all data with CAL ON.
-
- d.calib(42)                                 #this will calibrate scan 42
                                              #according to the way the
                                              #the data were taken (i.e., the
                                              #observing procedure
- d.plotc(42)                                 #plot the calibrated data for
                                              #scan 42. Since no integration
                                              #was specified, all data is
                                              #averaged.
- d.plotc(42,1)                               #plots the first integration of
                                              #the calibrated data for 42.
- d.plotc(42,3,2)                             #plots the third integration of
                                              #the second polarization for
                                              #scan 42.
- d.plotc(scan=42,int=3,pol=2)                #same using the explicit names
                                              #for the arguments.
-
- #HINT: if you are uncertain as to the arguments for a function, simply type
- #      the function name.
- d.plotc
function (val scan, val int = F, val pol = F) {
(ok := public.plotscan(public.getc(scan, int, pol)));
return T}
- #NOTE: if you are unhappy with the calibration, you can simply reapply it
- #      using different attributes
- d.calib(42,baseline=T,range=[300:600,800:1000]);
                                              #Apply calibration but also
                                              #baseline the data using the
                                              #specified range (defaults
                                              #to first order but this can
                                              #also be specified.
- #This will over-write the calibration data with the new calibration data.
- d.plotc(42)                                 #re-examine the data
- #NOTE: You can also individually calibrate each scan in the sequence with
- #      'd.TPcal(scan)'. There are numerous {\bf dish} functions for
- #       manipulating SDRecords (e.g., {\it scanscale (multiplies a scan
- #       by a factor, scandiv (divides two scans), etc.
```

Position Switched Observations

This section details calibration of data taken in the following modes:

- OffOn:PSWITCHON(OFF):TPWCAL

- OffOn:PSWITCHON(OFF):TPWCALSP
- OnOff:PSWITCHON(OFF):TPWCAL
- OnOff:PSWITCHON(OFF):TPWCALSP
- OffOnOff:PSWITCHON(OFF):TPWCAL
- OffOnOff:PSWITCHON(OFF):TPWCALSP

```
> source /home/aips++/gbt/aipsinit.sh           #load environment
> dish                                          #start AIPS++ DISH tool
...
- d.open('exampleSP')
T
- d.files()                                     #show active files
Current filein is  :  exampleSP
Current fileout is :  not set
T
- d.gms()                                       #provide brief summary
Scan     Object   Proctype    SWState   SWtchsig   Procseqn   Procsize
  35         S8      Track    FSWITCH      FSW01          1          1
  36         S8      Track    FSWITCH      FSW12          1          1
  37         S8      Track    FSWITCH      FSW12          1          1
  38         S8      OffOn PSWITCHOFF   TPWCALSP          1          2
  40         S8      OnOff  PSWITCHON   TPWCALSP          1          2
  42         S8      Track       NONE   TPWCALSP          1          1
T
- #Note: This only lists the first scan in a procedural sequence of scans
- d.listscans()                                 #shows all scans present
[35 36 37 38 39 40 41 42]
- d.plotr(39,1)                                 #plot the raw(observed) data
                                                #for the first phase of scan
                                                #38. Since no integration
                                                #value is specified, it averages
                                                #all phase 1 data (in this case
                                                #this is all data with CAL ON.
-
- d.calib(39)                                   #this will calibrate scan 38/39
                                                #(since both compose the
                                                #the OffOn procedure),
                                                #according to the way the
                                                #the data were taken (i.e., the
                                                #observing procedure
- d.plotc(39)                                   #plot the calibrated data for
                                                #scan 39. Since no integration
                                                #was specified, all data is
```

```
                                               #averaged.
                                               #The (sig-ref)/ref data is
                                               #only stored in the 'On' scans
                                               #data record; the 'Off' scan
                                               #contains only the calibrated
                                               #TPwCal data.
- d.plotc(39,1)                                #plots the first integration of
                                               #the calibrated data for 39.
- d.plotc(39,3,2)                              #plots the third integration of
                                               #the second polarization for
                                               #scan 39.
- d.plotc(scan=39,int=3,pol=2)                 #same using the explicit names
                                               #for the arguments.
-
- #HINT: if you are uncertain as to the arguments for a function, simply type
- #      the function name.
- d.plotc
function (val scan, val int = F, val pol = F) {
(ok := public.plotscan(public.getc(scan, int, pol)));
return T}
- #NOTE: if you are unhappy with the calibration, you can simply reapply it
- #      using different attributes
- d.calib(38,baseline=T,range=[300:600,800:1000]);
                                               #Apply calibration but also
                                               #baseline the data using the
                                               #specified range (defaults
                                               #to first order but this can
                                               #also be specified.
- #This will over-write the calibration data with the new calibration data.
- d.plotc(38)                                 #re-examine the data
```

Position Switched Observations - Many SIGs per REF

This section details calibration of data taken in the following modes:

- Track:NONE:TPWCAL
- Track:NONE:TPWCALSP
- OffOn:PSWITCHON(OFF):TPWCAL
- OffOn:PSWITCHON(OFF):TPWCALSP
- OnOff:PSWITCHON(OFF):TPWCAL
- OnOff:PSWITCHON(OFF):TPWCALSP
- OffOnOff:PSWITCHON(OFF):TPWCAL

- OffOnOff:PSWITCHON(OFF):TPWCALSP

```
> source /home/aips++/gbt/aipsinit.sh           #load environment
> dish                                          #start AIPS++ DISH
tool
...
- d.open('exampleSP')
T
- d.files()                                     #show active files
Current filein is  :  exampleSP
Current fileout is :  not set
T
- d.gms()                                       #provide brief summary
Scan     Object   Proctype    SWState   SWtchsig   Procseqn   Procsize
  35         S8      Track    FSWITCH      FSW01          1          1
  36         S8      Track    FSWITCH      FSW12          1          1
  37         S8      Track    FSWITCH      FSW12          1          1
  38         S8      OffOn PSWITCHOFF   TPWCALSP          1          2
  40         S8      OnOff  PSWITCHON   TPWCALSP          1          2
  42         S8      Track       NONE   TPWCALSP          1          1
T
- #Note: This only lists the first scan in a procedural sequence of scans
- d.listscans()                                 #shows all scans present
[35 36 37 38 39 40 41 42]
- d.plotr(39,1)                                 #plot the raw(observed) data
                                                #for the first phase of scan
                                                #38. Since no integration
                                                #value is specified, it averages
                                                #all phase 1 data (in this case
                                                #this is all data with CAL ON.
-
- d.calib(39)                                   #this will calibrate scan 38/39
                                                #(since both compose the
                                                #the OffOn procedure),
                                                #according to the way the
                                                #the data were taken (i.e., the
                                                #observing procedure
- d.plotc(39)                                   #plot the calibrated data for
                                                #scan 39. Since no integration
                                                #was specified, all data is
                                                #averaged.
                                                #The (sig-ref)/ref data is
                                                #only stored in the 'On' scans
                                                #data record; the 'Off' scan
                                                #contains only the calibrated
                                                #TPwCal data.
```

```
- d.plotc(39,1)                               #plots the first integration of
                                              #the calibrated data for 39.
- d.plotc(39,3,2)                             #plots the third integration of
                                              #the second polarization for
                                              #scan 39.
- d.plotc(scan=39,int=3,pol=2)                #same using the explicit names
                                              #for the arguments.

- #Now, if you wanted to use a different reference scan for this on, you could
- #do the following:
- d.TPcal(41)                                 #calibrate 'Off' scan 41.
- d.SRcal(39,41)                              #perform (sig-ref)/ref with
                                              #signal=39, reference=41
- d.SRcal(39,[38,41]);                        #perform (sig-ref)/ref with
                                              #signal scan=39, reference=
                                              #the average of 38 and 41.
                                              #in general:
-d.SRcal([vector of signal scans],
         [vector of reference scans])         #all reference scans will be
                                              #averaged and subtracted from
                                              #each signal scan.
```

Frequency Switched Observations - Mapping

This section details calibration of data taken in the following modes:

- RALongMap:FSWITCH:FSW01
- RALongMap:FSWITCH:FSW12
- DECLatMap:FSWITCH:FSW01
- DECLatMap:FSWITCH:FSW12

```
> source /home/aips++/gbt/aipsinit.sh         #load environment
> dish                                        #start AIPS++ DISH tool
...
- d.open('exampleSP')
T
- d.files()                                   #show active files
Current filein is  :  exampleSP
Current fileout is :  not set
T
- d.gms()                                     #provide brief summary
Scan     Object   Proctype    SWState   SWtchsig   Procseqn   Procsize
 303         S6      Track    FSWITCH      FSW01          1          1
 304     G250p8  RALongMap    FSWITCH      FSW01          1         41
T
```

```
- #Note: This only lists the first scan in a procedural sequence of scans
- d.listscans()                               #shows all scans present
[303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321
322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
341 342 343 344]
- d.plotr(304,1)                              #plot the raw(observed) data
                                              #for the first phase of scan
                                              #303. Since no integration
                                              #value is specified, it averages
                                              #all phase 1 data (in this case
                                              #this is all data with CAL ON.
-
- d.calib(304)                                #this will calibrate scan 304
                                              #according to the way the
                                              #the data were taken (i.e., the
                                              #observing procedure
- d.plotc(303)                                #plot the calibrated data for
                                              #scan 303. Since no integration
                                              #was specified, all data is
                                              #averaged.
- d.plotc(303,1)                              #plots the first integration of
                                              #the calibrated data for 303.
- d.plotc(303,3,2)                            #plots the third integration of
                                              #the second polarization for
                                              #scan 303.
- d.plotc(scan=303,int=3,pol=2);              #same using the explicit names
                                              #for the arguments.
-
- #HINT: if you are uncertain as to the arguments for a function, simply type
- #      the function name.
- d.plotc
function (val scan, val int = F, val pol = F) {
(ok := public.plotscan(public.getc(scan, int, pol)));
return T}
- #NOTE: if you are unhappy with the calibration, you can simply reapply it
- #      using different attributes
- d.calib(303,baseline=T,range=[300:600,800:1000]);
                                              #Apply calibration but also
                                              #baseline the data using the
                                              #specified range (defaults
                                              #to first order but this can
                                              #also be specified.
- #This will over-write the calibration data with the new calibration data.
- d.plotc(303)                                #re-examine the data
- #NOTE:In addition, if the SIG/REF signals are mixed (there is a temporary
- #     problem with the SPECTRAL PROCESSOR which causes some fraction of the
```

```
- #     scans to be flipped), you can reverse the sense of SIG and REF by
- #     doing:
- d.FScal(304,flipsr=T)                        #flips what is SIG and REF
-
- #NOTE:You can write a procedure to interactively view and flip your data.
-
- sns:=seq(304,344)                            #make a vector from 304-344
- for (i in sns) {                             #loop over these scans
+ d.plotc(i)                                   #plot scan
+ ans:=readline()                              #pause and prompt user for input
+ if (ans=='y') {                              #if 'y' typed, then
+    d.FScal(i,flipsr=T)                       #  flip SIG and REF sense
+ }
+ }
-
- d.makeim(304,600,900,imname='ralong')        #Make an image. This needs the
                                               #  first scan in the map
                                               #  the beginning (600) and end
                                               #  (900) channels to include.
                                               #  step determines the
                                               #  averaging over channel
                                               #  (none here) and imname
                                               #  specifies the output image
# NOTE: This actually creates three images on disk: 1) the original image,
#       a weights image, and an image corrected (divided) by the weights.
#       These are imname, imname\_weight and imname\_corr.
- im:=image('ralong\_corr')                     #Make an image tool - lots of
                                               #functions here!
- im.view()                                    #View the image
```

2.9.3 Observing Recipes (DCR)

OTF Mapping

Currently, the DCR tool is in a separate utility and must be included.

`include 'gbtcalutils.g';`

This creates a *gc* tool which four associated functions:

1. *setdata* takes the file name of a MeasurementSet on disk.
2. *contcal* calibrates the data
3. *makeimage* makes an image
4. *covercorr* corrects the image to the proper scale

See the Getting Results chapter on Continuum Single Dish Imaging for more details: docs/gettingresults/grvol3/node4.html.

```
- gc.setdata('cassaDCR');
successful readonly open of default-locked table cassaDCR/NRAO\_GBT\_GLISH: 21 col
umns, 120 rows
successful readonly open of default-locked table cassaDCR/NRAO\_GBT\_IF: 22 column
s, 240 rows
successful read/write open of default-locked table cassaDCR/FEED: 12 columns, 2
rows
successful read/write open of default-locked table cassaDCR/POLARIZATION: 4 colu
mns, 2 rows
successful read/write open of default-locked table cassaDCR/SPECTRAL\_WINDOW: 14
columns, 1 rows
successful readonly open of default-locked table cassaDCR/POINTING: 15 columns,
48000 rows
T
- gc.contcal()
successful read/write open of default-locked table cassaDCR: 28 columns, 192000
rows
T
- gc.makeimage()
successful readonly open of default-locked table cassaDCR: 28 columns, 192000 ro
ws
Starting server imager
Server started: /export/home/guthwine/aips++/linux\_gnu/bin/imager (AIPS++ versio
n: 1.7 (build #405))
Opening MeasurementSet cassaDCR
defaultcatalog (dc) ready
Starting imager::setdata
Selecting data
Performing selection on MeasurementSet
Selecting on field and spectral window ids
By selection 192000 rows are reduced to 48000
Finished imager::setdata
       0.96 real        0.69 user        0.19 system
successful readonly open of default-locked table cassaDCR/NRAO\_GBT\_GLISH: 21 col
umns, 120 rows
Cell size is  0.289495253arcmin
Map size is  117.705376  by  59.9669179
Grid sizes are  406   208
Defining image properties
nx = 406 is not composite; nx = 432 or 400 will be more efficient
ny = 208 is not composite; ny = 216 or 200 will be more efficient
Setting processing options
Starting imager::weight
Weighting MS: IMAGING\_WEIGHT column will be changed
Natural weighting
```

```
Sum of weights = 48000
Finished imager::weight
       32.6 real       23.58 user        6.11 system
Starting imager::makeimage
Calculating image (without full skyequation)
Making single dish image from corrected data
Image is : scanimage
Frequency = 8.9 GHz, synthesized continuum bandwidth = 0.08 GHz
Image polarization = Stokes I
Frequency = 8.9 GHz, synthesized continuum bandwidth = 0.08 GHz
Image polarization = Stokes I
Performing Single Dish gridding with convolution function SF
Gridding will use specified common tangent point:
     23:21:00.00         +58.32.00.00            B1950
Using prolate spheriodal wave function as the convolution function
Finished imager::makeimage
      10.25 real        6.77 user        0.49 system
Starting imager::makeimage
Calculating image (without full skyequation)
Making single dish coverage function
Image is : scanimage\_weight
Frequency = 8.9 GHz, synthesized continuum bandwidth = 0.08 GHz
Image polarization = Stokes I
Frequency = 8.9 GHz, synthesized continuum bandwidth = 0.08 GHz
Image polarization = Stokes I
Using prolate spheriodal wave function as the convolution function
Finished imager::makeimage
       8.12 real        6.58 user        0.52 system
Successfully closed empty server: imager
T
- gc.covercorr()
Starting server app\_image
Server started: /export/home/guthwine/aips++/linux\_gnu/bin/app\_image (AIPS++ ver
sion: 1.7 (build #405))
Selected bounding box :
    [1, 1, 1, 1] to [406, 208, 1, 1]  (23:28:23.897, +58.01.04.977, I, 8.900000e
+09Hz to 23:13:25.534, +59.00.59.643, I, 8.900000e+09Hz)
Creating new statistics storage lattice of shape [9]

Number points =   8.444800e+04       Sum      =   8.444801e+04
Mean          =   1.000000e+00
Variance      =   1.026810e+00       Std dev   =   1.013316e+00
Rms           =   1.423657e+00

Minimum value   0.000000e+00 at [1, 1, 1, 1] (23:28:23.897, +58.01.04.977, I, 8.
900000e+09Hz)
```

```
Maximum value   4.499732e+00 at [105, 182, 1, 1] (23:24:41.961, +58.54.05.639, I
, 8.900000e+09Hz)

Creating image ‘scanimage\_corr’ of shape [406, 208, 1, 1]
Created mask ‘mask0’
T
- #Scale images to Jy (e.g., 1.76 K/Jy)
- im:=imagecalc(outfile='scanimage\_tmb',pixels='(scanimage\_corr)/1.76');
- im.view();    #note the higher amplitudes
```

2.9.4 Optimized Use of AIPS++ in Green Bank

Starting AIPS++

Because AIPS++ is an actively developing software system, there are several versions available. These are called release, stable, weekly and daily. In Green Bank we maintain an additional version called gbt, which is just a link to the best version for reducing GBT data.

To start stable **dish**, issue the following commands (from the bash shell):

```
. /home/aips++/gbt/aipsinit.sh
dish
```

or for (t)csh:

```
. /home/aips++/gbt/aipsinit.sh
dish
```

To run one of the other versions, simply replace the word gbt in the path with the name of the version required. It is occassionally necessary to run weekly or daily to incorporate the latest bug fixes or enhancements.

Making AIPS++ Run Faster

You can run AIPS++ on any Linux system in Green Bank, but if the operation seems sluggish to you here are some hints on how to speed things up:

- 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 user name, e.g.

  ```
  user.aipsdir: /tmp/jbraatz\_aips++
  ```

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

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

The toolmanager GUI can then be restarted at a later time by:

```
- tm.gui()
```

- Run AIPS++ from the computer which hosts the disks with your data. If you keep your data on a disk local to your computer, you may get the best results by running on it. Alternatively, consider copying your data to a disk local to one of the fast computers, thalassa or despina, and running on that host.
- To achieve maximum performace be sure to access the disks not through the /home NFS link, but instead directly. In other words, do the following:

```
ssh despina
cd /export/home/despina3/scratch
mkdir my\_dir
cd my\_dir
aips++
```

Keep in mind that AIPS++ has to build its cache file structure the first time you start it with a new user.aipsdir, and each time you start a new version. So judge its startup speed on the second time you start it up, not the first. On etamin it should take about 14 seconds to start AIPS++ with the toolmanager GUI, and 7 seconds without it.

Where to Put Your AIPS++ Data

Make yourself a data directory starting at the following:

```
despina:             /export/home/despina3/scratch
thalassa:             /export/home/thalassa2/scratch
naiad:                 /export/home/naiad3/scratch
etamin:               /export/home/etamin/scratch
general use:       /home/aips++\_data (slow access from here)
```

Paging

If you find that error messages, code, and data listings go flying across your terminal window, and you'd like to page them instead (as with the more or less programs) then it may be that the PAGER environment variable is not set properly on your system. This seems to be the case for many of our systems in GB. You can remedy it by setting PAGER to "less" in your UNIX startup scripts. Contact Jim if you need help doing so. It makes glish nicer!

2.9.5 Example UniPOPS reduction in dish

This section shows an example data reduction session using **dish** in a UniPOPS fashion.

Sample Data Reduction Session

```
% ssh etamin
% cd /home/etamin/scratch
% mkdir myname
% cd myname
% . /home/aips++/gbt/aipsinit.sh
% aips++ -l uni2.g
- d.import('/home/gbtdata/tigerTeam\_02','tiger.ms',startscan=20,stopscan=33)
- d.lscans()
[20 21 22 23 24 25 26 27 28 29 30 31 32 33]
- d.reference(20)
- d.signal(21)
- d.temp()
- d.show()
- d.nfit(5)
# Use the plotter to set the x-axis units to pixels (channel number)
- d.nregion(0,3600,4100,8200)
- d.bshape()                  # to show the fit
- d.baseline()                # to subtract the baseline
- d.show()
- d.uscale(2.3)
- d.setYUnit('Jy')
- d.show()
- d.bdrop(2500)
- d.edrop(2500)
- d.show()
- d.sclear()
- d.getc(26)
- d.show()
- d.stats(bchan=1000,echan=2000)
- d.accum()
- d.getc(28)
- d.show()
- d.accum()
- d.stats(bchan=1000,echan=2000)
- d.ave()
- d.show()
- d.stats(bchan=1000,echan=2000)
```

About the Sample Data

The tigerTeam_02 data in the examples used throughout this document were taken with the GBT on 11 October 2001. The L-band receiver (1.15–1.73 GHz) was used with the intent to observe the HI profiles of an assortment of galaxies. The backend was the GBT spectrometer. The "OffOn" procedure in GO was used to drive the position-switched observations. Signal and reference data are recorded in separate, adjacent scans of equal duration.

Processing GBT Scans

The first step in processing data is to identify the input data file. The technique for doing so differs depending on if the data are in the raw GBT FITS format, or have been previously filled into the AIPS++ MeasurementSet format. To import data from FITS files, use the *import* command:

```
- d.import('/home/gbtdata/tigerTeam\_02','tiger.ms',startscan=20,stopscan=33)
```

This command creates a MeasurementSet called *tiger.ms*. If the desired MeasurementSet already exists on disk, use the dopen command to load the data:

```
- d.open('/home/aips++/data/demo/dishdemo/tiger.ms')
```

These data sets are specific to the Green Bank installation. For other sites, consult the AIPS++ system administrator to see how to access this data.

List the scans available in your new data set:

```
- d.lscans()
[20 21 22 23 24 25 26 27 28 29 30 31 32 33]
```

Calibrate a Sig/Ref pair as follows:

```
- d.reference(20)
- d.signal(21)
- d.temp()
- d.show()
```

You can also use the core **dish** functions to calibrate scans.

```
- d.calib(20)
or
- d.TPcal(20)
- d.TPcal(21)
- d.SRcal(21,20)
- d.plotc(21)
```

To calibrate and retrieve a frequency switched scan, only a single get is required:

```
# This example does not work with the sample data provided, but shows the
# sequence required for freq switched data.
- d.get(2005)
- d.show()
```

The red and green plots represent the two polarizations. These are currently handled together (use the *getpol* function to retrieve a single polarization. Currently you can show an individual polarization with the show1 function, and you can average to two polarizations using the avgFeeds function. The spectrum is initially shown with GHz units for the x-axis. These units can be changed using the button on the plotter labeled "GHz". Regions of the plot may be zoomed using the middle mouse button, and unzoomed using the unzoom button on the plotter.

To fit a baseline to the data, you must specify a range of the spectrum to use for the fit and the order of the polynomial. In this example we will describe the fit region using channel numbers, so first plot the spectrum against channel number (select the option labeled *pix* on the plotter x-axis selector button). Then, similar to UniPOPS, use the following procedure:

```
- d.nfit(5)
- d.nregion(0,3600,4100,8200)
- d.bshape()                    # to show the fit
- d.baseline()                  # to subtract the baseline
- d.show()                      # to show the baseline-subtracted data
```

The UNI-jr function setregion() can be used in place of the nregion function to enter the baseline region interactively using the mouse cursor.

If a calibration scaling factor is known, it can be applied:

```
- d.uscale(2.3)
- d.setYUnit('Jy')
- d.show()
```

The UNIPOPS-familiar bdrop and edrop parameters may be utilized in the plots:

```
- d.bdrop(2500)
- d.edrop(2500)
- d.show()
```

To recall the value of a UNIPOPS-type adverb, simply call the function with no value, as follows:

```
- d.bdrop()
2500
- d.nfit()
5
```

To fit a gaussian function to the data, use the “G fit” button on the plotter. Prior to fitting, an initial guess to the line shape must be given. This can be done automatically by pressing the green “Estimate” button, or interactively by pressing the gray “Inter” button under the “Estimates” title. Choose the latter for the sample data and follow the directions on the message line in the Profile Fitter GUI. Then press the green “Fit profile” button the do the actual fitting.

Averaging Scans

UNI-jr uses the UNIPOPs model for averaging scans. Data is stored away to the accumulator, and can then be averaged. First clear the accumulator:

```
- d.sclear()
```

then retrieve scans using (for example) get, getc, signal/reference/temp, getfs, or some other calibration and retrieval command, and accumulate them:

```
- d.getc(26)
- d.accum()
- d.getc(28)
- d.accum()
```

and finally average and show the result:

```
- d.ave()
- d.show()
```

Statistics and Other Analysis

The *stats* function will return statistics over a given channel range:

```
- d.stats(bchan=500,echan=1500)
```

The equivalent to the UNIPOPS *moment* command can be run as follows:

```
- d.bmoment(3900)
- d.emoment(4100)
- d.moment()
```

A Short dish UniPOPS Reference

This section lists the commands currently available in the UNI-jr package. Each function’s parameters are listed with it. If the parameter does not have an assignment (e.g. the *scan* parameter in the *signal* function) then it must be supplied when the function is called. If the parameter does have an assignment, then the assigned value shown here is the default, but it can be overridden by specifying a new value when the function is called.

The functions can either be called with parameters listed in order and no parameter name specified, or they can be called with parameter names specified and in any order. The following two calls are equivalent:

```
d.stats(F,2,1000,2000)
d.stats(bchan=1000,echan=2000,feed=2)


Loading data:
------------
d.import(projdir,outms=F,outmsdir=F,startscan=F,stopscan=F,backend=F)
                                        : Import (fill) data from GBT FITS files
d.open(filename)                        : Open an already filled data set
d.files()                               : Show input and output file settings
d.summary()                             : List some scan info (source, time)
d.ls()                                  : List files in directory

Retrieving and Calibrating Scans:
---------------------------------
d.lscans()                              : List available scans
d.dgetscan(scan,nphase)                 : Return an individual phase
d.getir(scan,phase)                     : Get an individual phase to memory
d.get(scan)                             : General purpose scan retrieval
d.signal(scan,tcal\_in=unset,pctTsys=0.01,verbose=F)
                                        : Retrieve a sig scan
d.reference(scan,tcal\_in=unset,pctTsys=0.01,verbose=F)
                                        : Retrieve a ref scan
d.temp()                                : Perform (s-r)/r for pos switch data
d.getc(scan)                            : Retrieve \& calibrate pos switch pair
d.getfs(scan,tcal\_in=unset,flipsigref=F,pctTsys=0.001,verbose=T)
                                        : Retrieve freq switch scan
d.getfsint(scan,intNo=1,tcal\_in=unset,flipsigref=F,pctTsys=0.01,verbose=F)
                                        : Retrieve freq switch integration

Averaging Scans:
----------------
d.accum()                               : Add scan to accumulator
d.ave()                                 : Average scans in accumulator
d.sclear()                              : Clear accumulator
d.avgFeeds()                            : Average all feeds

Plotting Scans:
----------------
d.page()                                : Clear the plotter
d.show()                                : Display scan in memory (globalscan1)
d.show1(ifeed)                          : Display 1 feed of scan in memory
d.showref()                             : Display ref scan
d.setYUnit(unit)                        : Set Y axis unit for plotting
d.saxis(strval)                         : Change units on X axis
d.bdrop(value=T)                        : Set bdrop parameter
```

```
d.edrop(value=T)                        : Set edrop parameter

Baseline Fit:
-------------
d.nfit(value=T)                         : Set order of polynomial to fit
d.nregion(...)                          : Set baseline region
d.setregion()                           : Set baseline region with mouse
d.baseline()                            : Subtract polynomial of order nfit
d.dcbase()                              : Subtract polynomial of order 0
d.bshape()                              : Show baseline fit of order nfit

Smoothing and Scaling data:
---------------------------
d.uscale(factor)                        : Scale the scan in memory
d.bias(factor)                          : Add a bias to scan in memory
d.hanning()                             : Hanning smooth scan in memory
d.boxcar(smooth\_width)                  : Boxcar smooth scan in memory
d.chngres(smooth\_width)                 : Change resolution of scan in memory

The Stack:
----------
d.addstack(beg,end=beg,inc=1)           : Add elements to the stack
d.empty()                               : Empty the stack
d.delete(value)                         : Delete one component from the stack
d.tellstack()                           : Show the stack

Analysis:
---------
d.stats(quiet=F,feed=0,bchan=0,echan=0) : Calculate stats over specified range
d.rms(printFlag=F)                      : Calc stats over the baseline region
d.bmoment(value=T)                      : Set bmoment parameter
d.emoment(value=T)                      : Set emoment parameter
d.moment(quiet=F)                       : Calculate moments of scan in memory
d.peak(quiet=F)                         : Calculate location and val of peak
d.galactic(quiet=F)                     : Calculate galactic coords from header

Scan Arithmetic:
----------------
d.push()                                : push a scan for processing
d.minus()                               : Perform sig = sig-ref
d.plus()                                : Perform sig = sig+ref
d.multiply()                            : Perform sig = sig*ref
d.divide()                              : Perform sig = sig/ref

Saving Data:
------------
```

```
d.fileout(outfile=F)                    : Set output file
d.keep()                                : Save a scan
d.lsoutfile()                           : List scans in keep file

Utility:
--------
d.lscans()                              : List available scans
d.utable()                              : Show data values from scan in memory
d.header()                              : Show header values for scan in memory
d.getvf(chan)                           : Show velocity, freq for given channel
d.getvfarray()                          : Get array or velocities, frequencies
d.uniput(name,value)                    : Store a value in memory slot
d.uniget(name)                          : Retrieve a value from memory slot
d.copy(fromhere,tohere)                 : Copy memory locations
d.upr(...)                              : Show internal variable values
d.store()                               : Store any state values (nregion, ...)
d.restore()                             : recall stored values
```

A Mapping between dish and UniPOPS

```
UNIPOPS                              {\bf dish}
-------                              ------

get 55                               d.get(55)
page                                 d.page()
show                                 d.show()
accum                                d.accum()
ave                                  d.ave()
sclear                               d.sclear()
nfit = 3                             d.nfit(3)
nregion = 1,128,380,512              d.nregion(1,128,380,512)
baseline                             d.baseline()
dcbase                               d.dcbase()
bshape                               d.bshape()
on 55                                d.signal(55)
off 56                               d.reference(56)
temp                                 d.temp()
factor=3.5; scale                    d.uscale(3.5)
factor=1.2; bias                     d.bias(1.2)
addstack(600,620,1)                  d.addstack(600,620,1)
addstack(631)                        d.addstack(631)
empty                                d.empty()
delete(609)                          d.delete(609)
tell stack                           d.tellstack()
table                                d.utable()
header                               d.header()
```

```
bmoment = 128                    d.bmoment(128)
emoment = 512                    d.emoment(512)
moment                           d.moment()
minus                            d.minus()
plus                             d.plus()
multiply                         d.multiply()
divide                           d.divide()
peak()                           d.peak()
chngfile                         d.chngfile()
keep()                           d.save()
hanning                          d.hanning()
nbox=5; boxcar                   d.boxcar(5)
newres=4.4; chngres              d.chngres(4.4)
```

2.9.6 Example Standards Data

The standards data set used for this example is:
/home/gbtdata/standards_apr27
and represents only Spectral Processor data.
The following details the relevant observing modes and scan numbers:

Scan	Procedure	Source	Switch State	Switch Signal	BE
5	OffOn	S171	PSWITCHOFF	TPWCAL	SP
6	OffOn	S171	PSWITCHON	TPWCAL	SP
7	OffOn	S171	PSWITCHOFF	TPWCAL	SP
8	OffOn	S171	PSWITCHON	TPWCAL	SP
9	Track	S171	FSWITCH	FSW01	SP
15	RALongMap	OrionA	FSWITCH	FSW01	SP

- All observations are at C-band, frequency=5010.9 MHz

2.9.7 Obtaining updates to AIPS++ releases

For the GBT users, updates to the AIPS++ releases are supported through a binary inhale. This option is only available for Linux users (Solaris is not supported). If the user has an AIPS++ release installed (from a CD or ISO image), he may 'update' this release to a newer version of the code through the following procedure:

```
1. On a linux system (binary inhale is not available for solaris), install
   aips++ v1.7 from a release CD.  Install the bare-minimum release.  If
   you want the up-to-date documentation, it can be retrieved below.

2. Change directory to the aips++ root directory (the directory
   you specified during the INSTALL procedure).
```

```
3. Make a symbolic link called linux_gnu that points to the linux
   directory (which contains the bulk of the release installation):

   % ln -s linux linux_gnu

4. Edit the .aipsrc file as follows:
   a. Comment out all lines referring to machines in Socorro (tarzan,
      cluster)
   b. Add the following line (replacing <username> with your username)

     account.manager: <username>

5. Source the aips++ initialization script:

   % source ./aipsinit.csh

6. Add the current directory to your PATH:

   % setenv PATH .:$PATH

7. Initiate the inhale:

   a. To retrieve only the code and binaries:

      % inhale binary

   b. To retrieve the data repository and documentation, as
      well as the code and binaries (be sure 'cvsup'
      is available on your machine):


      % inhale -cvsup <x> binary

      (where <x> is data, docs, or all, depending upon what you want)

   This will take up to several hours while a snapshot of the latest
   aips++ stable installation is retrieved (from Socorro), unpacked, and
   assembled.

Subsequent binary inhales require only steps 2, 5 and 7 to obtain new
stable versions of aips++.
```

Chapter 3

GBT Continuum Single dish Imaging

Tim Cornwell/Joe McMullin

3.1 Introduction

This chapter describes how to make continuum OTF single dish images. It should be read in conjunction with several other chapters in *Getting Results*, including the chapters which cover Display of data and Image analysis.

The examples here are worked from GBT data but the principles apply to other telescopes. In particular, this chapter provides a facile means of reducing some standard mode continuum images. Different modes/analysis can still be obtained through the direct use of the basic AIPS++ tools.

3.2 Filling single dish data into AIPS++

3.2.1 GBT

GBT data are read into an AIPS++ MeasurementSet using the gbtmsfiller `tool`.

```
filler := gbtmsfiller()
filler.setproject (/home/gbtdata/pnt_prime_13')
filler.fillall()
filler.done()
```

This will create a MeasurementSet called *pnt_prime_13_DCR* in the current directory.

This can also be done from the command line before entering the AIPS++ shell.

```
> gbtmsfiller project=/home/gbtdata/zeroSpacing01 minscan=101 maxscan=301
msrootname='orion_' fillrawpointing=True
```

This will create a MeasurementSet called orion_DCR in the current directory (it appends the name of the backend, in this case the GBT DCR).

3.2.2 Loading the continuum calibration utilities

The continuum calibration imaging utilities are currently packaged under a single script which can be optionally added to an aips++ session. This script is called 'gbtcalutils.g'. It can be loaded upon startup as:

```
> aips++ -l gbtcalutils.g
```

For a minimal startup (only opens the necessary AIPS++ utilities required for continuum processing), one can type:

```
> glish -l gbtcalutils.g
```

Or it can be loaded to an existing session of AIPS++:

```
- include 'gbtcalutils.g'
```

This script makes a tool, gc (Gbt Continuum imaging), which has several functions appropriate for calibrating and imaging GBT continuum data; the tool assumes that you already have available a MeasurementSet (MS) of the observed map.

3.3 Data examination and inspection

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.

```
include 'msplot.g'
msp := msplot('pnt_prime_13_DCR', edit=T)
```

The GUI panel controlling flagging in msplot allows global flag masks to be set, such as that for all polarizations if any one is bad, as well as access to the list of accumulated flags, reversal of flags already applied, and other related utility operations. Raster displays of data are possible also, and are recommended for spectral-line data editing.

You may also view and edit the data at a lower level: that of the tables themselves. To browse the MeasurementSet table using a graphical browser, do:

```
include 'table.g'
t := table('pnt_prime_13_DCR')
t.browse()
```

Note that editing of individual values using the browser is possible (but unlikely to be needed unless there is something structurally wrong with the tables).

3.4 Basic gain calibration

As noted above, AIPS++ data are stored in a MeasurementSet (MS). The observed data are stored in the DATA (complex) or FLOAT_DATA (real) column in the MAIN table of the MS; these are the raw data as loaded from the telescope archive or 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 data plane representation of the source or field model (MODEL_DATA). The latter two columns are populated at the filling stage (for GBT data these are initially just copies of the raw FLOAT_DATA column) or upon invocation of the imager tool on the MS. The observed data do not change during reduction, but the related columns do, and should be viewed in this context. When plotting the data using a `tool` such as msplot, the data sub-types (`observed, corrected and model`) can be selected individually.

An example data set checked into the AIPS++ system can be converted into a MS by using the following from the Unix prompt:

```
> gbtmsfiller project=/home/aips++/data/nrao/GBT/pnt_prime_13 minscan=2350 \
      maxscan=2424 msrootname='cygnus_' fillrawpointing=True
```

This creates a MS, cygnus_DCR. Another data set which has already been converted into a data set is also available: /aips++/data/demo/dishdemo/pnt_lowgreg_11_DCR. Either of these can be used in the following examples.

3.4.1 gc.setdata(msname)

setdata takes a single argument, the MeasurementSet name, and examines it for suitability (e.g., is it a GBT continuum map) and content, i.e., does it contain all of the requisite information to reduce automatically. In particular, it explores the data set to confirm that information on the polarization, bandwidth, and frequency are all present. In addition, it also obtains the extent of the map and calculates the appropriate cell size for gridding (hardwired to $\lambda/2D$).

3.4.2 gc.contcal(tcal=1.0,average=T,baseline=F,nfit=0,range=20)

contcal performs the basic calibration on the data. It has arguments for tcal (to over-ride the system determined value). The default is to average all polarizations (as available); this can be toggled off by setting average=F. Optionally, a baseline can be subtracted from each row of the data. The range argument determines the beginning and end percentage of each scan row that will be used for the baseline determination; the default is 20%.

The initial continuum maps obtained at the GBT are four phase data representing the states of noise-diode off and on for two separate receptors. *contcal* performs the following calculation for each OTF strip of the map:

$$T_{ANT} = T_{CAL} \times \frac{V_{ON} + V_{OFF}}{2\times < V_{ON} - V_{OFF} >} - < T_{CAL}/2. > \quad (3.1)$$

```
V_ON are the counts for the CAL ON phase
V_OFF are the counts for the CAL OFF phase
T_CAL is the K/counts conversion
```

3.5 Imaging

Imaging is done with the Imager tool, which supports a broad range of imaging modes and deconvolution algorithms, as described in the Imaging chapter of `Getting Results` and in the `User Reference Manual`.

The Imager `tool` requires initial selection of the data and initialization of the imaging coordinates.

A single-field, continuum GBT *unnormalized* image can be made using:

3.5.1 gc.makeimage(imname='scanimage',gridfn='SF',receptor=1)

makeimage performs the steps to select, regrid and construct the image. It accepts arguments to specify the output image, the gridding function to be used and which receptor's data to image. In addition, an image of the coverage (sampling) is made which is simply the image name with a "_weight" appended to it. The coverage image is simply the gridded weights. The receptor is a numeric designation for the polarizations to which each receiver is sensitive (e.g., XX, YY, etc). If contcal was executed with average=T, then this will be an average of these polarizations; otherwise, each can be imaged separately.

The process of making a single dish image is very straightforward. The data are converted into the coordinate system of the image (set via Imager.setimage), and then added to the image using a gridding function. A variety of convolution functions can be used: BOX, SF, or PB. The first is a simple nearest neighbor gridding, the second uses a prolate spheroidal wavefunction, and the last uses a primary beam model appropriate to the telescope being used. The primary beam is optimal in the least squares sense but degrades the resolution, whereas the prolate spheroidal function can avoid the resolution degradation but the noise level will be higher.

3.5.2 gc.covercorr(imname=F,wname=F)

The image from *makeimage* is not normalized by the sampling density. *covercorr* corrects for this by correcting for the coverage image to avoid nonsampled points (hardwired to the 10% level of the coverage map). If makeimage was previously executed, the values of "imname" (the source image) and "wname" (the weights image) are set. They may also be specified explicitly. The output image will be the name from "imname" with a "_corr" concatenated onto the end.

3.5.3 gc.plotsource(imname,nsource=1)

At this point, an image has been made which can be viewed simply by:

```
- im:=image('scanimage_corr')
- im.view();
```

plotsource also enables viewing of the contstructed images. It takes as an argument the name of the image to view and an optional parameter for the expected number of sources. The image module supports a findsources utility which identifies the positions of strong point sources in an image; it labels the position with 'Source x', where x is the number of the source found.

At this stage, use the viewer to manipulate the image in any way desired. The 'Adjust' button on the lower left provides the controls to the raster, contour and skycatalog information.

3.6 Example

The following is an example of how to calibrate and image a GBT continuum data set called pnt_lowgreg_11_DCR (contained in the standard AIPS++ data repository); the data is an OTF image obtained with the GBT DCR (digital continuum receiver). These functions are illustrated in the following example; note, for this example the logger is closed so all output is sent to the screen.

```
> cp -r /home/aips++/data/demo/dishdemo/pnt_lowgreg_11_DCR .
> glish -l gbtcalutils.g
NORMAL: gbtcalutils (gc) is ready for use
- field_names(gc)               # look at the available tools in gc
                                # gc = GBT continuum imaging
checkms setdata contcal makeimage covercorr plotsource debug
- gc.setdata('cassaDCR');
successful readonly open of default-locked table cassaDCR/NRAO_GBT_GLISH: 21 col
umns, 120 rows
successful readonly open of default-locked table cassaDCR/NRAO_GBT_IF: 22 column
s, 240 rows
successful read/write open of default-locked table cassaDCR/FEED: 12 columns, 2
rows
successful read/write open of default-locked table cassaDCR/POLARIZATION: 4 colu
mns, 2 rows
successful read/write open of default-locked table cassaDCR/SPECTRAL_WINDOW: 14
columns, 1 rows
successful readonly open of default-locked table cassaDCR/POINTING: 15 columns,
48000 rows
```

```
T
- gc.contcal()
successful read/write open of default-locked table cassaDCR: 28 columns, 192000
rows
T
- gc.makeimage()
successful readonly open of default-locked table cassaDCR: 28 columns, 192000 ro
ws
Starting server imager
Server started: /export/home/guthwine/aips++/linux_gnu/bin/imager (AIPS++ versio
n: 1.7 (build #405))
Opening MeasurementSet cassaDCR
defaultcatalog (dc) ready
Starting imager::setdata
Selecting data
Performing selection on MeasurementSet
Selecting on field and spectral window ids
By selection 192000 rows are reduced to 48000
Finished imager::setdata
       0.96 real        0.69 user        0.19 system
successful readonly open of default-locked table cassaDCR/NRAO_GBT_GLISH: 21 col
umns, 120 rows
Cell size is  0.289495253arcmin
Map size is  117.705376  by  59.9669179
Grid sizes are  406   208
Defining image properties
nx = 406 is not composite; nx = 432 or 400 will be more efficient
ny = 208 is not composite; ny = 216 or 200 will be more efficient
Setting processing options
Starting imager::weight
Weighting MS: IMAGING_WEIGHT column will be changed
Natural weighting
Sum of weights = 48000
Finished imager::weight
       32.6 real       23.58 user        6.11 system
Starting imager::makeimage
Calculating image (without full skyequation)
Making single dish image from corrected data
Image is : scanimage
Frequency = 8.9 GHz, synthesized continuum bandwidth = 0.08 GHz
Image polarization = Stokes I
Frequency = 8.9 GHz, synthesized continuum bandwidth = 0.08 GHz
Image polarization = Stokes I
Performing Single Dish gridding with convolution function SF
Gridding will use specified common tangent point:
     23:21:00.00        +58.32.00.00            B1950
```

```
Using prolate spheriodal wave function as the convolution function
Finished imager::makeimage
      10.25 real        6.77 user        0.49 system
Starting imager::makeimage
Calculating image (without full skyequation)
Making single dish coverage function
Image is : scanimage_weight
Frequency = 8.9 GHz, synthesized continuum bandwidth = 0.08 GHz
Image polarization = Stokes I
Frequency = 8.9 GHz, synthesized continuum bandwidth = 0.08 GHz
Image polarization = Stokes I
Using prolate spheriodal wave function as the convolution function
Finished imager::makeimage
       8.12 real        6.58 user        0.52 system
Successfully closed empty server: imager
T
- gc.covercorr()
Starting server app_image
Server started: /export/home/guthwine/aips++/linux_gnu/bin/app_image (AIPS++ ver
sion: 1.7 (build #405))
Selected bounding box :
    [1, 1, 1, 1] to [406, 208, 1, 1]  (23:28:23.897, +58.01.04.977, I, 8.900000e
+09Hz to 23:13:25.534, +59.00.59.643, I, 8.900000e+09Hz)
Creating new statistics storage lattice of shape [9]

Number points =   8.444800e+04       Sum      =   8.444801e+04
Mean          =   1.000000e+00
Variance      =   1.026810e+00       Std dev   =   1.013316e+00
Rms           =   1.423657e+00

Minimum value   0.000000e+00 at [1, 1, 1, 1] (23:28:23.897, +58.01.04.977, I, 8.
900000e+09Hz)
Maximum value   4.499732e+00 at [105, 182, 1, 1] (23:24:41.961, +58.54.05.639, I
, 8.900000e+09Hz)

Creating image `scanimage_corr' of shape [406, 208, 1, 1]
Created mask `mask0'
T
- #Scale images to Jy (e.g., 1.76 K/Jy)
- im:=imagecalc(outfile='scanimage_tmb',pixels='(scanimage_corr)/1.76');
- im.view(); #note the higher amplitudes
- #You can also modify/add header information to the image
- #See the images module for more details; below is an example of adding
- #information on the restoring beam
- rb:=[=];
- rb.major.value:=87.6;
```

```
- rb.major.unit :='arcsec';
- rb.minor.value:=87.6;
- rb.minor.unit :='arcsec';
- rb.positionangle.value:=0;
- rb.positionangle.unit :='deg'
- im.setrestoringbeam(beam=rb);
- im.restoringbeam(); #see restoring beam values
- im.setbrightnessunit('Jy/beam');
- im.brightnessunit(); #see brightness unit value
```

Chapter 4

GBT Spectral Line Reduction using UNI-jr

Jim Braatz

4.1 Sample Data Reduction Session

This section shows an example data reduction session using UNI-jr. It is intended to give the reader a feel for the look of UNI-jr commands.

```
% ssh etamin
% cd /home/etamin/scratch
% mkdir myname
% cd myname
% . /home/aips++/gbt/aipsinit.sh
% aips++ -l uni2.g
- import('/home/gbtdata/tigerTeam_02','tiger.ms',startscan=20,stopscan=33)
- lscans()
[20 21 22 23 24 25 26 27 28 29 30 31 32 33]
- reference(20)
- signal(21)
- temp()
- show()
- nfit(5)
# Use the plotter to set the x-axis units to pixels (channel number)
- nregion(0,3600,4100,8200)
- bshape()                  # to show the fit
- baseline()                # to subtract the baseline
- show()
- uscale(2.3)
- setYUnit('Jy')
```

```
- show()
- bdrop(2500)
- edrop(2500)
- show()
- sclear()
- getc(26)
- show()
- stats(bchan=1000,echan=2000)
- accum()
- getc(28)
- show()
- accum()
- stats(bchan=1000,echan=2000)
- ave()
- show()
- stats(bchan=1000,echan=2000)
```

4.2 Introduction

This document describes how to reduce and analyze GBT spectral line data. To ease the learning curve involved for users new to AIPS++, a set of functions has been developed which allow GBT data reduction using the UNIPOPS model. These functions are not an exact emulator of UNIPOPS, but are intended to resemble its data reduction philosophy. The command names are alike, where possible. As of June 2002, the UNI-jr code continues to be under development.

4.3 Prerequisites

You should have AIPS++ stable version 1.7 Build 257 (or later) installed on your system. The AIPS++ version is printed on the screen when the initialization script is executed. A sample data set, taken during the AIPS++ "Tiger Team" session, is available and valid as a test data set. Data taken with the GBT during or after November 2001 may also be viable. It is strongly recommended that GBT data be reduced in Green Bank. Reducing the data at one of the other NRAO sites is also an option. Contact Jim Braatz, Joe McMullin or Bob Garwood if you have troubles.

If you are in Green Bank, use the computer etamin, naiad or thalassa and its local disks. These will dramatically improve performance. Do so as follows:

```
ssh etamin
cd /home/etamin/scratch
mkdir myname
cd myname
```

4.4 About the Sample Data

The tigerTeam_02 data in the examples used throughout this document were taken with the GBT on 11 October 2001. The L-band receiver (1.15–1.73 GHz) was used with the intent to observe the HI profiles of an assortment of galaxies. The backend was the GBT spectrometer. The "OffOn" procedure in GO was used to drive the position-switched observations. Signal and reference data are recorded in separate, adjacent scans of equal duration.

4.5 Starting UNI-jr

Prior to running AIPS++, various environment variable must be set. If you are running the bash shell, do this:

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

and if you run csh, do this:

```
source /home/aips++/stable/aipsinit.csh
```

In Green Bank there is a link which points to the best AIPS++ version for reducing GBT data. This may be more recent than stable. It is recommended you source it instead, if in GB:

```
. /home/aips++/gbt/aipsinit.sh
```

Then start AIPS++ and UNI-jr as follows:

```
aips++ -l uni2.g
```

You will see the Glish prompt when the system is ready.

4.6 Processing GBT Scans

The first step in processing data is to identify the input data file. The technique for doing so differs depending on if the data are in the raw GBT FITS format, or have been previously filled into the AIPS++ MeasurementSet format. To import data from FITS files, use the *import* command:

```
- import('/home/gbtdata/tigerTeam_02','tiger.ms',startscan=20,stopscan=33)
```

This command creates a MeasurementSet called *tiger.ms*. If the desired MeasurementSet already exists on disk, use the dopen command to load the data:

```
- dopen('/home/aips++/data/demo/dishdemo/tiger.ms')
```

These data sets are specific to the Green Bank installation. For other sites, consult the AIPS++ system administrator to see how to access this data.

List the scans available in your new data set:

```
- lscans()
[20 21 22 23 24 25 26 27 28 29 30 31 32 33]
```

Calibrate a Sig/Ref pair as follows:

```
- reference(20)
- signal(21)
- temp()
- show()
```

You can also use the UNI-jr "get" function to retrieve and calibrate scans. It works as long as the header information associated with the scan is correct. For the position switched example above, the command sequence is:

```
- get(20)
- get(21)
- temp()
- show()
```

To calibrate and retrieve a frequency switched scan, only a single get is required:

```
# This example does not work with the sample data provided, but shows the
# sequence required for freq switched data.
- get(2005)
- show()
```

The red and green plots represent the two polarizations. These are currently handled together, but a developing enhancement to UNI-jr will allow the polarizations to be handled separately as they are in UNIPOPS. Currently you can show an individual polarization with the show1 function, and you can average to two polarizations using the avgFeeds function.

The spectrum is initially shown with GHz units for the x-axis. These units can be changed using the button on the plotter labeled "GHz". Regions of the plot may be zoomed using the middle mouse button, and unzoomed using the unzoom button on the plotter.

To fit a baseline to the data, you must specify a range of the spectrum to use for the fit and the order of the polynomial. In this example we will describe the fit region using channel numbers, so first plot the spectrum against channel number (select the option labeled *pix* on the plotter x-axis selector button). Then, similar to UNIPOPS, use the following procedure:

```
- nfit(5)
- nregion(0,3600,4100,8200)
- bshape()                    # to show the fit
- baseline()                  # to subtract the baseline
- show()                      # to show the baseline-subtracted data
```

The UNI-jr function setregion() can be used in place of the nregion function to enter the baseline region interactively using the mouse cursor.

If a calibration scaling factor is known, it can be applied:

```
- uscale(2.3)
- setYUnit('Jy')
- show()
```

The UNIPOPS-familiar bdrop and edrop parameters may be utilized in the plots:

```
- bdrop(2500)
- edrop(2500)
- show()
```

To recall the value of a UNIPOPS-type adverb, simply call the function with no value, as follows:

```
- bdrop()
2500
- nfit()
5
```

To fit a gaussian function to the data, use the "G fit" button on the plotter. Prior to fitting, an initial guess to the line shape must be given. This can be done automatically by pressing the green "Estimate" button, or interactively by pressing the gray "Inter" button under the "Estimates" title. Choose the latter for the sample data and follow the directions on the message line in the Profile Fitter GUI. Then press the green "Fit profile" button the do the actual fitting.

4.7 Averaging Scans

UNI-jr uses the UNIPOPs model for averaging scans. Data is stored away to the accumulator, and can then be averaged. First clear the accumulator:

```
- sclear()
```

then retrieve scans using (for example) get, getc, signal/reference/temp, getfs, or some other calibration and retrieval command, and accumulate them:

```
- getc(26)
- accum()
- getc(28)
- accum()
```

and finally average and show the result:

```
- ave()
- show()
```

4.8 Statistics and Other Analysis

The *stats* function will return statistics over a given channel range:

```
- stats(bchan=500,echan=1500)
```

The equivalent to the UNIPOPS *moment* command can be run as follows:

```
- bmoment(3900)
- emoment(4100)
- moment()
```

4.9 A Short UNI-jr Reference

This section lists the commands currently available in the UNI-jr package. Each function's parameters are listed with it. If the parameter does not have an assignment (e.g. the *scan* parameter in the *signal* function) then it must be supplied when the function is called. If the parameter does have an assignment, then the assigned value shown here is the default, but it can be overridden by specifying a new value when the function is called.

The functions can either be called with parameters listed in order and no parameter name specified, or they can be called with parameter names specified and in any order. The following two calls are equivalent:

```
stats(F,2,1000,2000)
stats(bchan=1000,echan=2000,feed=2)


Loading data:
------------
import(projdir,outms=F,outmsdir=F,startscan=F,stopscan=F,backend=F)
                                        : Import (fill) data from GBT FITS files
dopen(filename)                         : Open an already filled data set
files()                                 : Show input and output file settings
dsummary()                              : List some scan info (source, time)
ls()                                    : List files in directory

Retrieving and Calibrating Scans:
---------------------------------
lscans()                                : List available scans
dgetscan(scan,nphase)                   : Return an individual phase
getir(scan,phase)                       : Get an individual phase to memory
get(scan)                               : General purpose scan retrieval
signal(scan,tcal_in=unset,pctTsys=0.01,verbose=F)
                                        : Retrieve a sig scan
reference(scan,tcal_in=unset,pctTsys=0.01,verbose=F)
                                        : Retrieve a ref scan
```

```
temp()                                  : Perform (s-r)/r for pos switch data
getc(scan)                              : Retrieve \& calibrate pos switch pair
getfs(scan,tcal_in=unset,flipsigref=F,pctTsys=0.001,verbose=T)
                                        : Retrieve freq switch scan
getfsint(scan,intNo=1,tcal_in=unset,flipsigref=F,pctTsys=0.01,verbose=F)
                                        : Retrieve freq switch integration
calibTaStar(tau=0.012,eta_l=0.99)       : Calibrate to Ta*
calibTmb(tau=0.012,eta_m=0.98)          : Calibrate to Tmb
calibSnu(tau=0.012,eta_a=0.7)           : Calibrate to Jy

Averaging Scans:
----------------
accum()                                 : Add scan to accumulator
ave()                                   : Average scans in accumulator
sclear()                                : Clear accumulator
avgFeeds()                              : Average all feeds

Plotting Scans:
----------------
page()                                  : Clear the plotter
show()                                  : Display scan in memory (globalscan1)
show1(ifeed)                            : Display 1 feed of scan in memory
showref()                               : Display ref scan
setYUnit(unit)                          : Set Y axis unit for plotting
saxis(strval)                           : Change units on X axis
bdrop(value=T)                          : Set bdrop parameter
edrop(value=T)                          : Set edrop parameter

Baseline Fit:
-------------
nfit(value=T)                           : Set order of polynomial to fit
nregion(...)                            : Set baseline region
setregion()                             : Set baseline region with mouse
baseline()                              : Subtract polynomial of order nfit
dcbase()                                : Subtract polynomial of order 0
bshape()                                : Show baseline fit of order nfit

Smoothing and Scaling data:
---------------------------
uscale(factor)                          : Scale the scan in memory
bias(factor)                            : Add a bias to scan in memory
hanning()                               : Hanning smooth scan in memory
boxcar(smooth_width)                    : Boxcar smooth scan in memory
chngres(smooth_width)                   : Change resolution of scan in memory

The Stack:
```

```
----------
addstack(beg,end=beg,inc=1)             : Add elements to the stack
empty()                                 : Empty the stack
delete(value)                           : Delete one component from the stack
tellstack()                             : Show the stack

Analysis:
---------
stats(quiet=F,feed=0,bchan=0,echan=0)   : Calculate stats over specified range
rms(printFlag=F)                        : Calc stats over the baseline region
bmoment(value=T)                        : Set bmoment parameter
emoment(value=T)                        : Set emoment parameter
moment(quiet=F)                         : Calculate moments of scan in memory
peak(quiet=F)                           : Calculate location and val of peak
galactic(quiet=F)                       : Calculate galactic coords from header

Scan Arithmetic:
----------------
push()                                  : push a scan for processing
minus()                                 : Perform sig = sig-ref
plus()                                  : Perform sig = sig+ref
multiply()                              : Perform sig = sig*ref
divide()                                : Perform sig = sig/ref

Saving Data:
------------
chngfile(outfile=F)                     : Set output file
keep()                                  : Save a scan
kget(scan)                              : Retrieve a scan from the keep file
klscans()                               : List scans in keep file

Utility:
--------
lscans()                                : List available scans
utable()                                : Show data values from scan in memory
header()                                : Show header values for scan in memory
getvf(chan)                             : Show velocity, freq for given channel
getvfarray()                            : Get array or velocities, frequencies
uniput(name,value)                      : Store a value in memory slot
uniget(name)                            : Retrieve a value from memory slot
copy(fromhere,tohere)                   : Copy memory locations
upr(...)                                : Show internal variable values
store()                                 : Store any state values (nregion, ...)
restore()                               : recall stored values
```

4.10 A Mapping between UNIPOPS and UNI-jr commands

```
UNIPOPS                               UNI-JR
-------                               ------

get 55                                get(55)
page                                  page()
show                                  show()
accum                                 accum()
ave                                   ave()
sclear                                sclear()
nfit = 3                              nfit(3)
nregion = 1,128,380,512               nregion(1,128,380,512)
baseline                              baseline()
dcbase                                dcbase()
bshape                                bshape()
on 55                                 signal(55)
off 56                                reference(56)
temp                                  temp()
factor=3.5; scale                     uscale(3.5)
factor=1.2; bias                      bias(1.2)
addstack(600,620,1)                   addstack(600,620,1)
addstack(631)                         addstack(631)
empty                                 empty()
delete(609)                           delete(609)
tell stack                            tellstack()
table                                 utable()
header                                header()
bmoment = 128                         bmoment(128)
emoment = 512                         emoment(512)
moment                                moment()
minus                                 minus()
plus                                  plus()
multiply                              multiply()
divide                                divide()
peak()                                peak()
chngfile                              chngfile()
keep()                                save()
hanning                               hanning()
nbox=5; boxcar                        boxcar(5)
newres=4.4; chngres                   chngres(4.4)
```

Chapter 5

ATCA reduction

Mark Wieringa

5.1 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++.

5.2 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 5.1: 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 5.2: 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.

5.3 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.

5.4 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()
```

5.5 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.

5.6 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 $\pm$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
```

5.6.1 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.

5.7 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

$$\vec{V}_{ij} \; = \; B_{ij} \; G_{ij} \; D_{ij} \; P_{ij} \; T_{ij} \; \vec{V}_{ij}^{\;\mathrm{IDEAL}} \tag{5.1}$$

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 $\vec{V}_{ij}^{\,\mathrm{IDEAL}}$ 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 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.

5.7.1 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.

5.7.2 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.

5.7.3 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']`).

5.7.4 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.

5.8 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`.

5.8.1 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.

5.8.2 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.

5.9 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.

5.10 Acknowledgements

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

Chapter 6

Parkes Multibeam Reduction

Stacy Mader

6.1 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.

6.2 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)
```

6.3 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
```

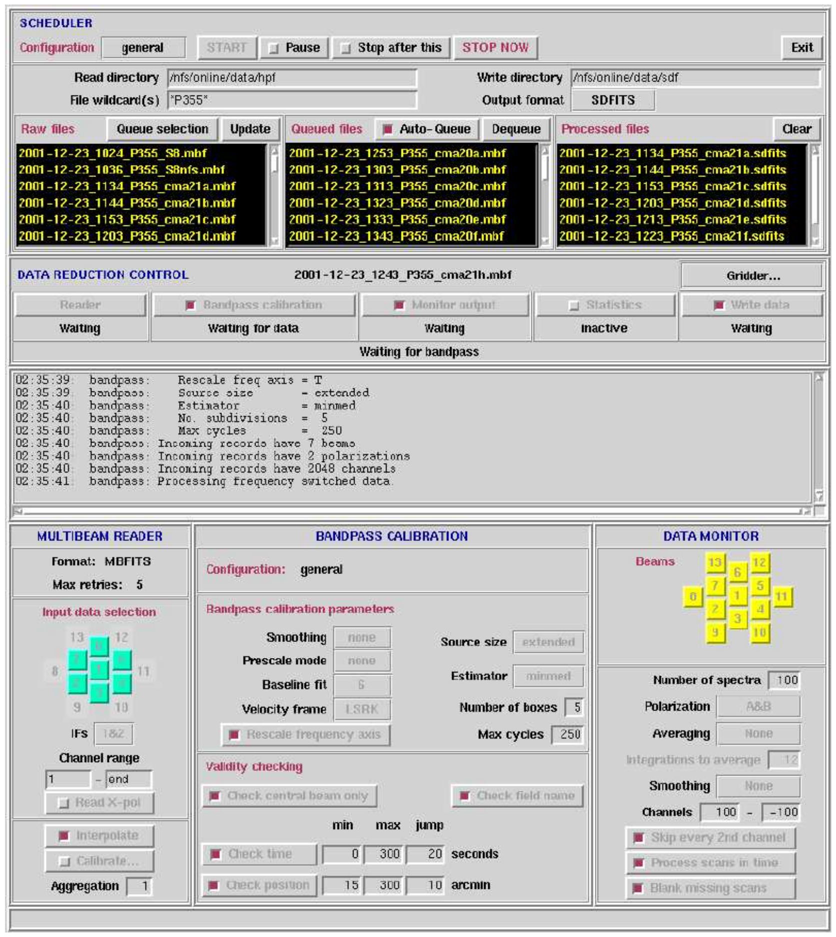


Figure 6.1: The Livedata GUI with all clients enabled.

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.

6.3.1 Scheduler

Figure 6.2 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.

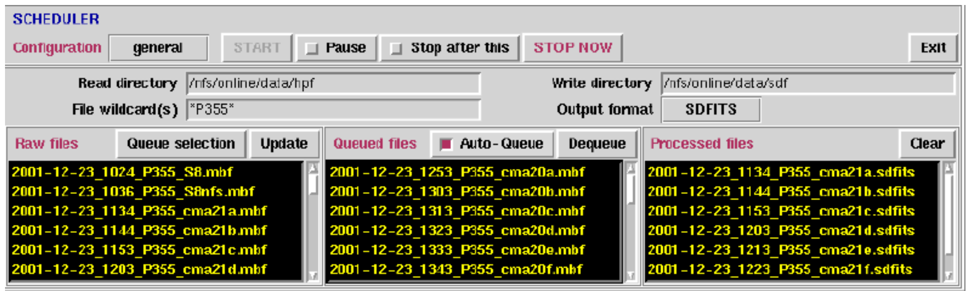


Figure 6.2: The Scheduler panel.

6.3.2 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 6.4. 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).

6.3.3 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.

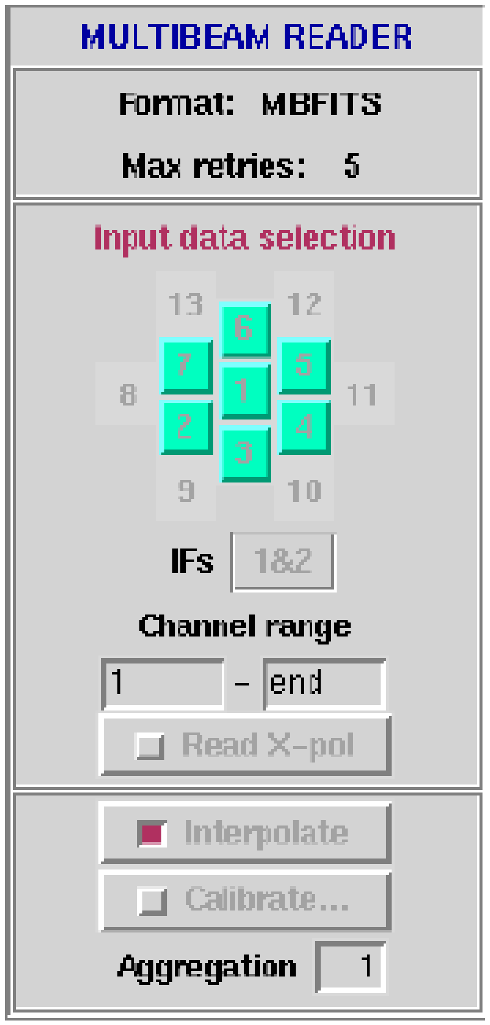


Figure 6.3: The Reader panel.

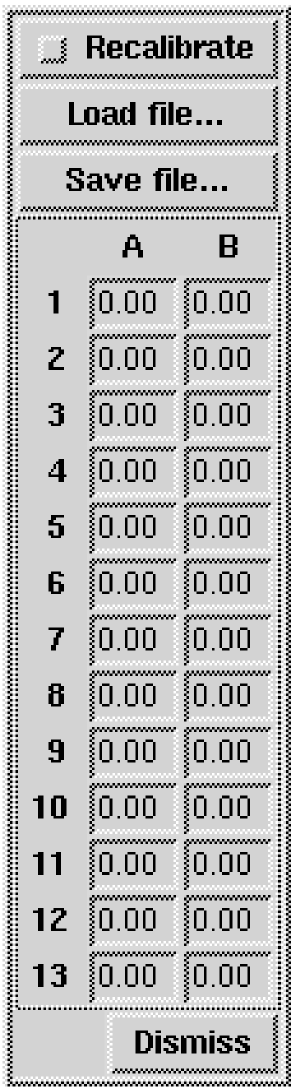


Figure 6.4: The Reader calibration panel.

    - 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.

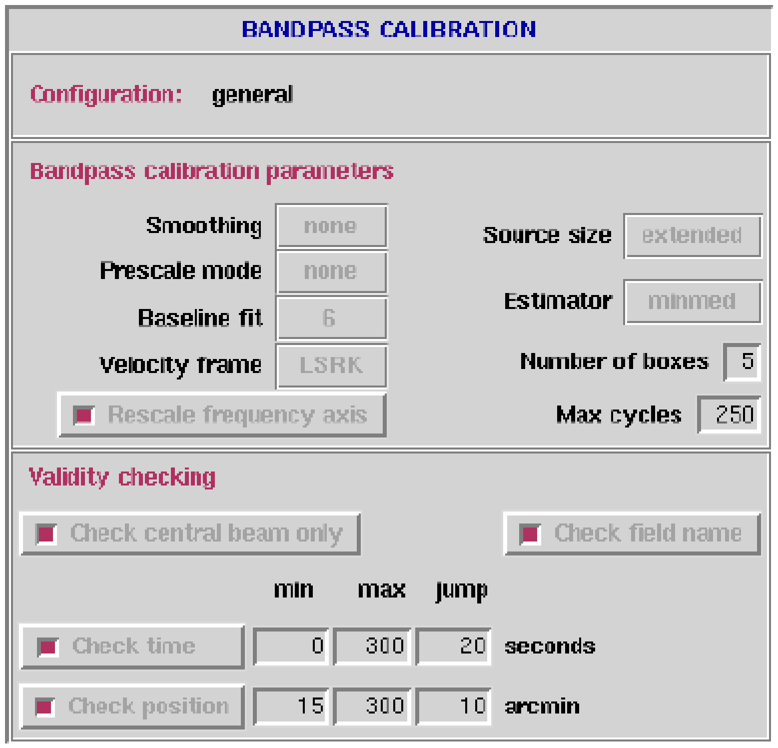


Figure 6.5: The Bandpass Calibration panel.

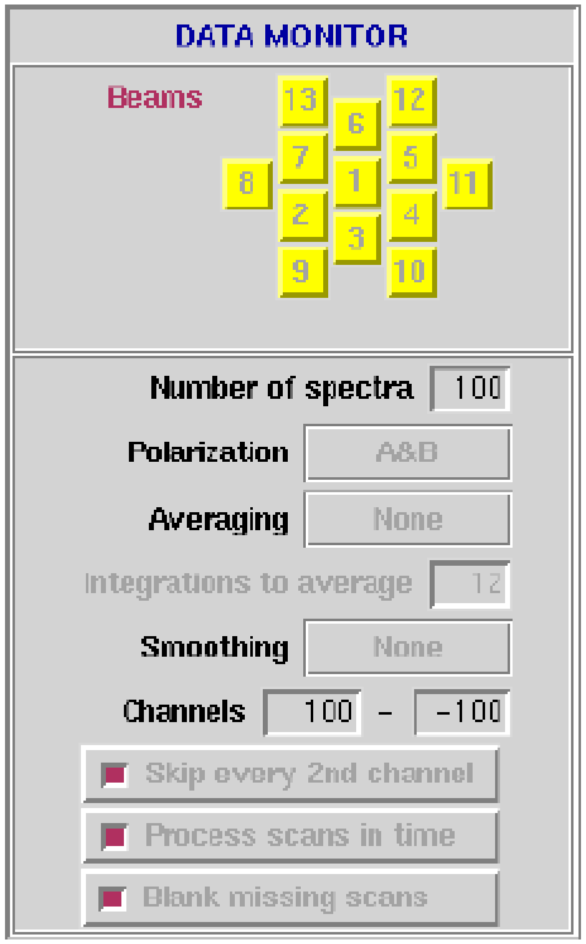


Figure 6.6: The Data Monitor panel.

6.3.4 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 6.7. 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.

6.3.5 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.

6.3.6 Writer

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

6.4 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.

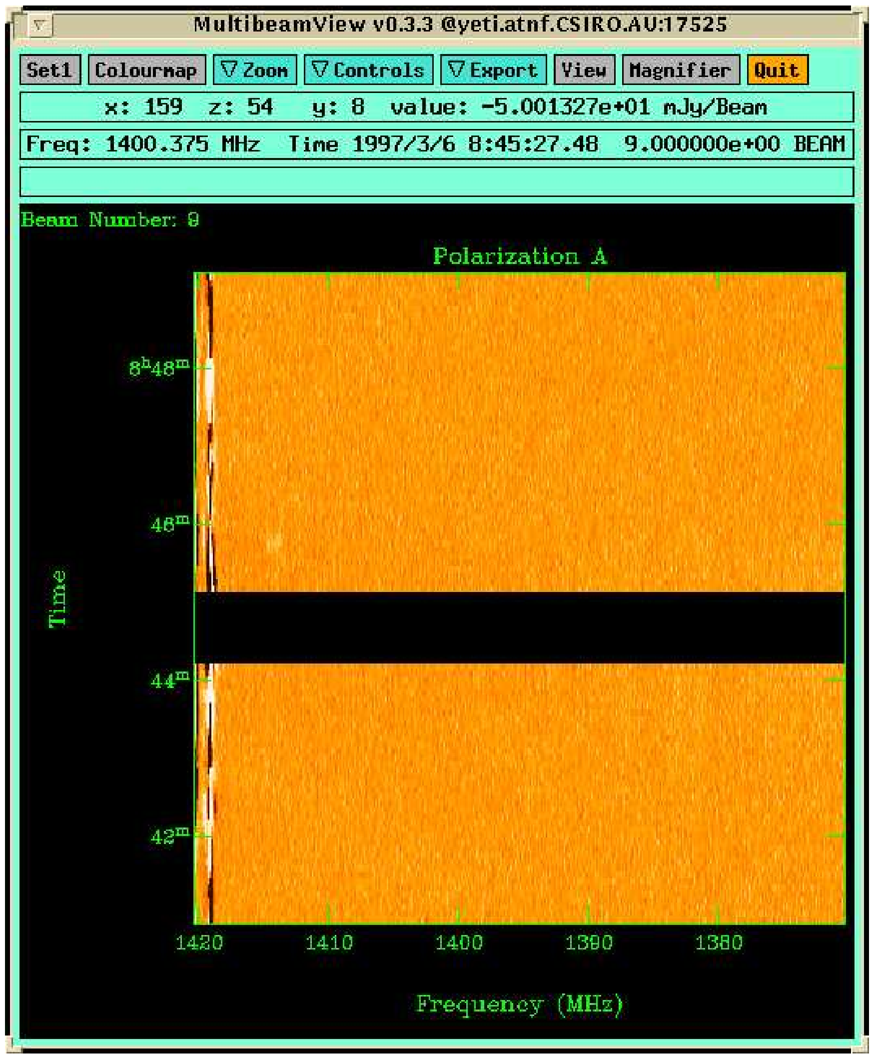


Figure 6.7: A MultibeamView panel displaying frequency vs time for beam 9, Polarization A. A faint galaxy can be seen near 1415 MHz at 08^h46^m.

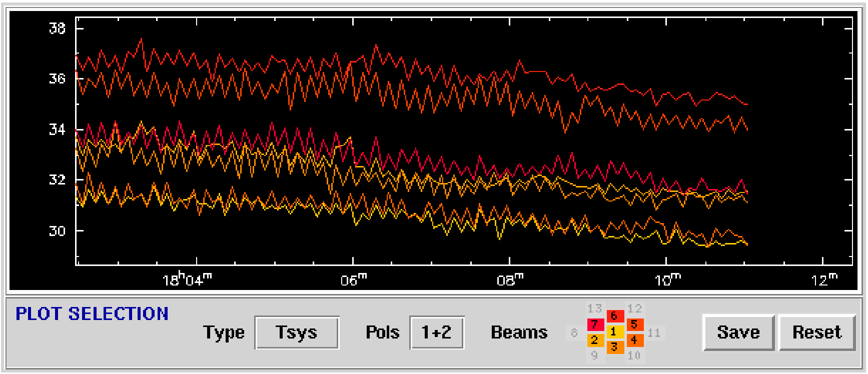


Figure 6.8: The Statistics window displaying T_{sys} for 7 beams.

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 6.9) 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

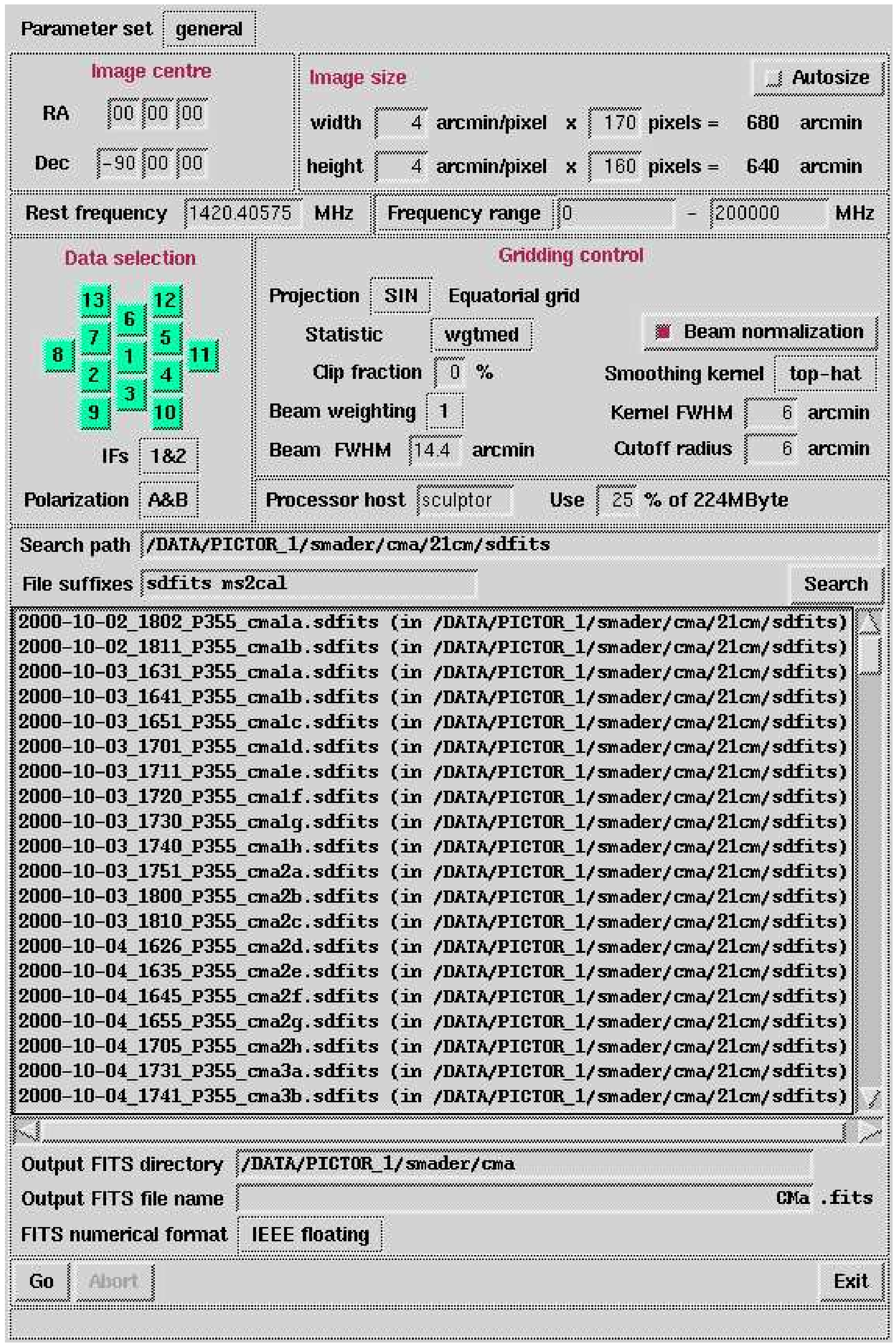


Figure 6.9: The default Gridzilla GUI.

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.

6.5 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 6.10. 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).

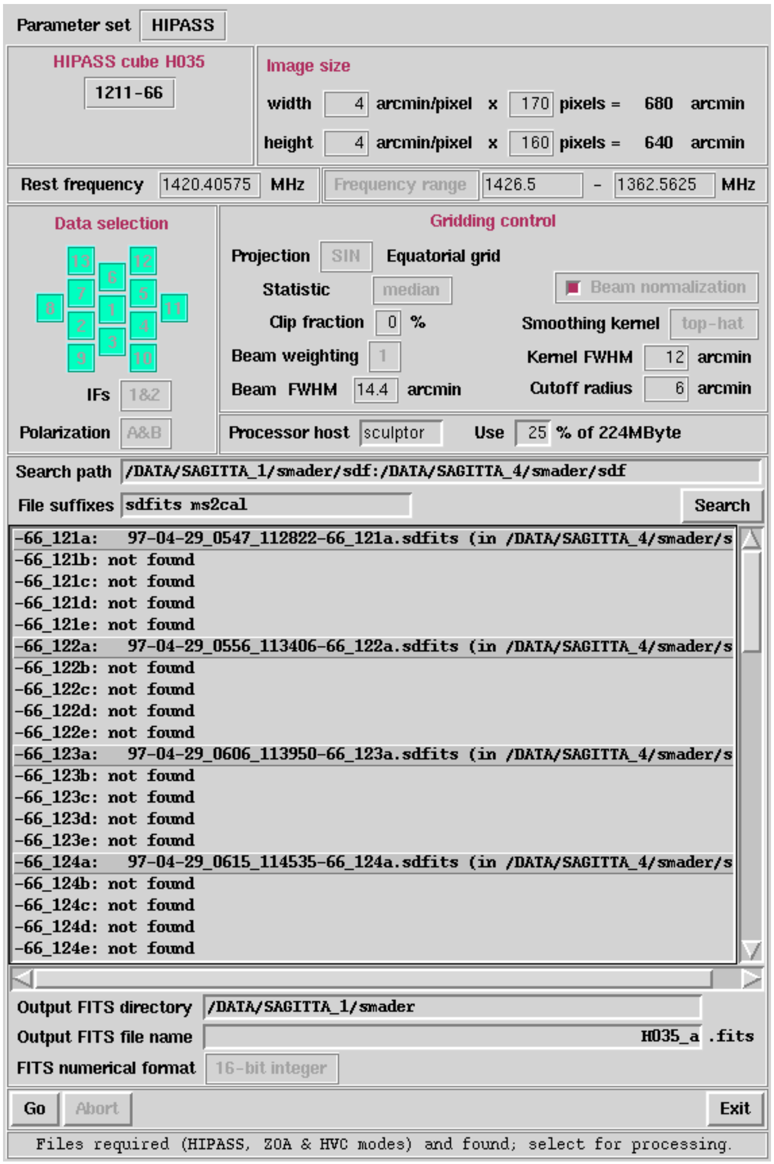


Figure 6.10: Using gridzilla to construct the HIPASS cube H035.

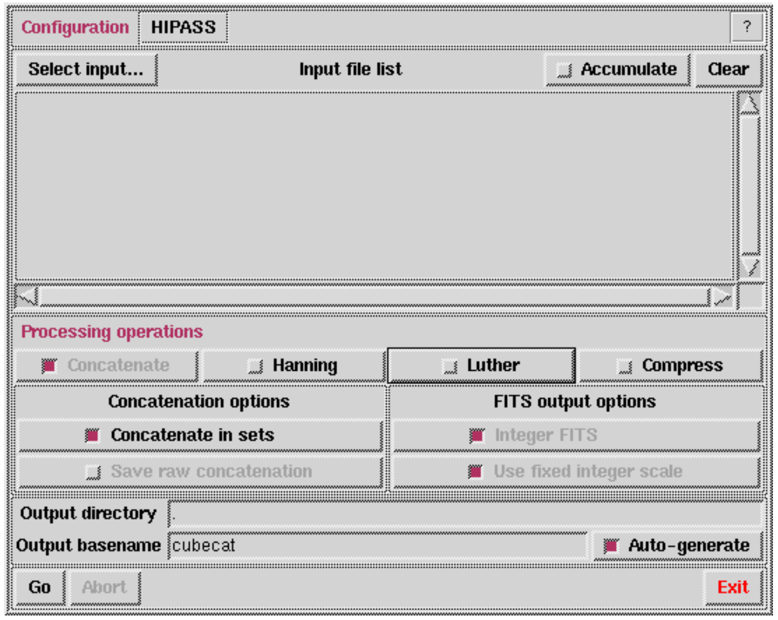


Figure 6.11: The default cubecat GUI.

6.6 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 6.11. 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 7

Reducing JCMT data in DISH

R. Phillips & J. McMullin

7.1 Setting Your Environment

7.1.1 Starting AIPS++

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

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

7.1.2 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.

7.1.3 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'
```

7.2 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
```

7.3 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');
```

7.4 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"

7.5 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
```

7.6 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

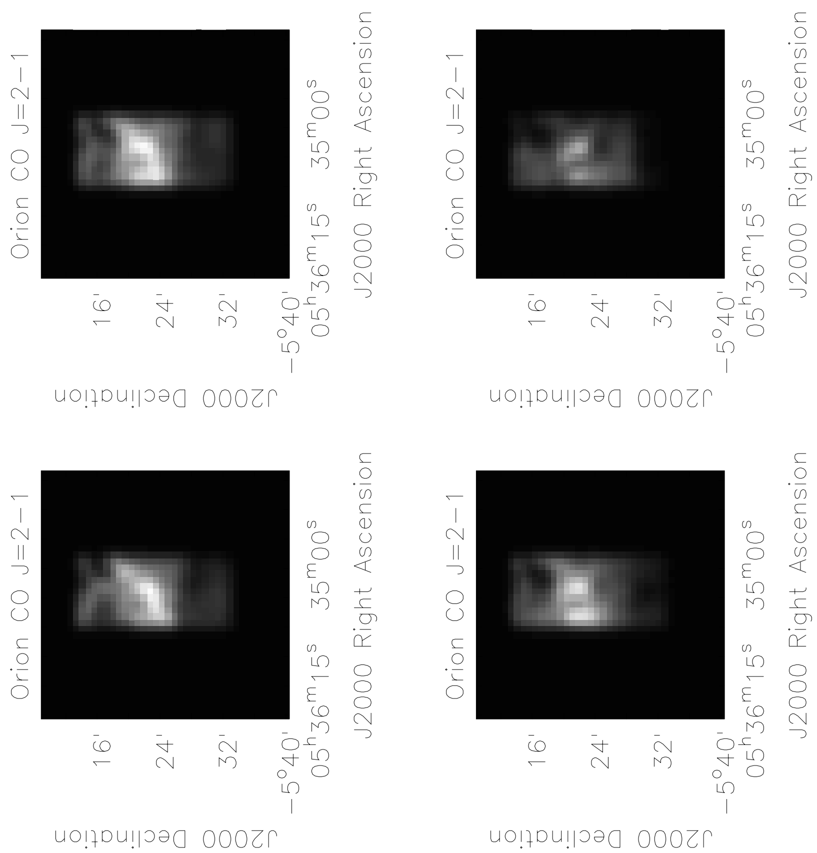


Figure 7.1: Viewer display of 4 channels of the data cube. Multiple planes can be viewed by selecting the "Canvas Manager" option from the "File" menu. Also see the Getting Results chapter 3.11.

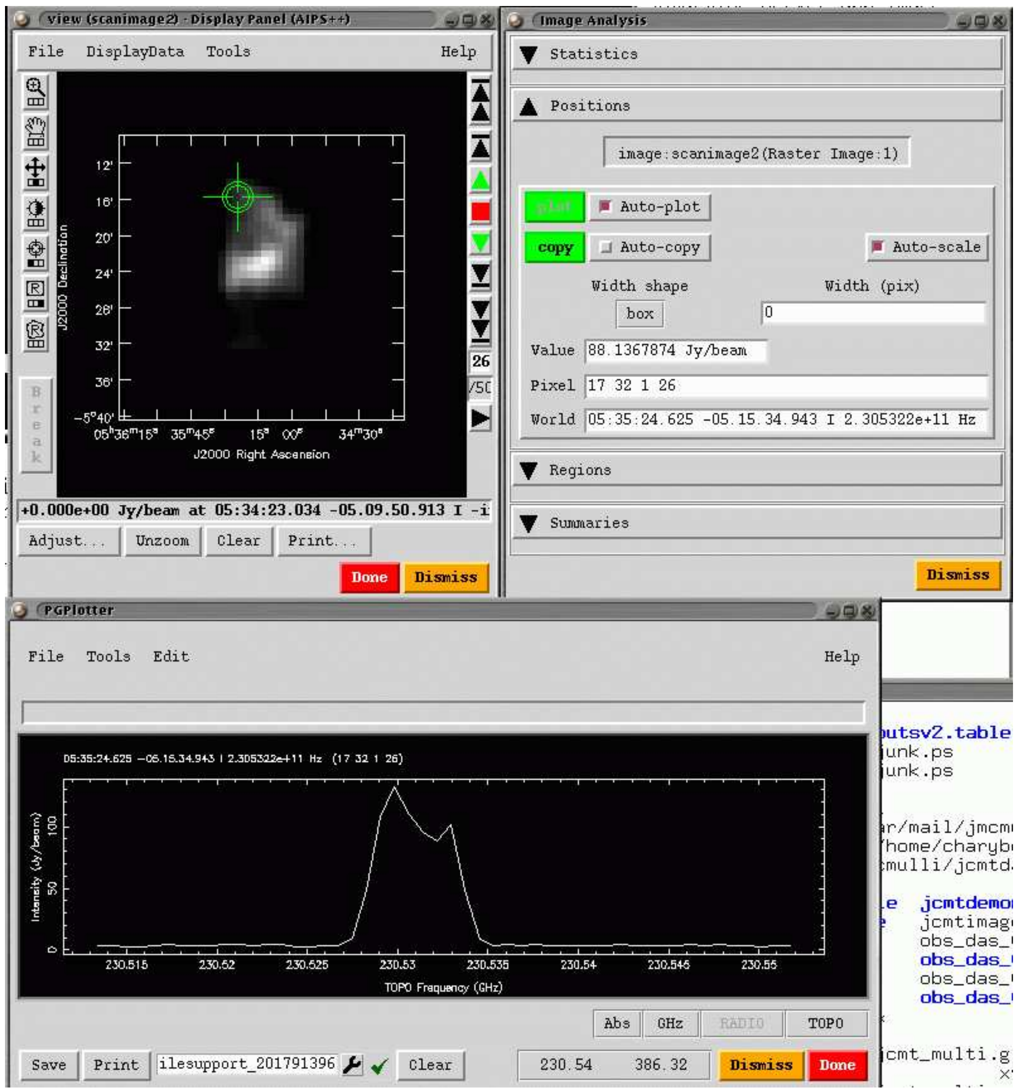


Figure 7.2: Viewer display of an image utilizing the Image Analysis Tool. Selecting the Positions Tool, choosing a position on the data cube (marked by the cross hairs), and selecting the "Auto-plot" feature allows the user to examine the gridded spectra of the data cube interactively. As each position 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 8

Arecibo Observatory Spectral Line Reduction with DISH

K. O'Neil and J. McMullin

8.1 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.

8.2 Setting Your Environment

8.2.1 Starting AIPS++

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

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

8.2.2 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.

8.2.3 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'
```

8.3 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.

8.3.1 SDFITS and AIPS++

8.3.2 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
```

8.3.3 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
```

8.4 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
```

8.5 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.

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.