Miriad

Multichannel Image Reconstruction, Image Analysis and Display

User Guide

Bob Sault and Neil Killeen

26 May 2011

See http://www.atnf.csiro.au/computing/software/miriad

Contents

Table of Contents **i**

1 General Information **1-1**

2 The User Interface **2-1**

2.1 Introduction 2-1

2.2 The `miriad` Front-End 2-1

2.3 *MIRIAD* on the World Wide Web 2-5

2.4 Bug Reports and Giving Feedback 2-5

2.5 Indirect Parameter Input 2-7

2.6 Host Command-Line Interface 2-7

2.7 Writing Shell Scripts 2-9

3 Plotting Concepts **3-1**

3.1 Introduction 3-1

3.2 The ATNF Visualisation Suite 3-1

3.3 Using X Windows 3-1

3.4 PGPLOT Plotting Devices 3-2

4 *MIRIAD* Datasets **4-1**

4.1 Listing Datasets – PRTHD 4-1

4.2 Inside Datasets 4-1

4.3 Scratch Files 4-2

4.4 *MIRIAD* Data Disks at Epping 4-3

5 Visibility Data Concepts **5-1**

5.1 *MIRIAD* Visibility File Format 5-1

5.2 *MIRIAD*'s Calibration Model 5-1

5.3 On-the-Fly Calibration Correction 5-2

5.4 Channel Selection, Averaging and Doppler Correction – UV Linetypes 5-3

5.5 Selecting UV Data in *MIRIAD* 5-5

5.6 Spectral Window Selection . . . 5-8
5.7 Selection and Multi-Source/Multi-Frequency Datasets . . . 5-9
5.8 Polarization/Stokes Handling (stokes and select) . . . 5-10

6 Image Data Concepts **6-1**
6.1 Image Datasets . . . 6-1
6.2 Image Coordinate Systems . . . 6-1
6.3 Image Region of Interest (region) . . . 6-1

7 Reduction Strategies **7-1**

8 Getting Data In and Out of *MIRIAD* **8-1**
8.1 `ATLOD`: Reading RPFITS Files . . . 8-1
8.2 FITS Tapes and *MIRIAD* . . . 8-5
8.3 Reading Visibility FITS Files . . . 8-5
8.4 Writing Visibility FITS Files . . . 8-8
8.5 Reading and Writing FITS Images . . . 8-9
8.6 Archiving and Transporting *MIRIAD* Datasets . . . 8-9

9 Generating Dummy Visibility Data **9-1**
9.1 Making a Fake Observation . . . 9-1
9.2 Generating Visibilities from a Model Image . . . 9-3

10 Flagging, Manipulating and Examining Visibility Data **10-1**
10.1 Correcting Visibilites . . . 10-1
10.1.1 Initial "fixes" to millimetre data . . . 10-1
10.2 Flagging Visibilities . . . 10-2
10.3 Other Flagging Tasks . . . 10-10
10.4 Listing Visibilities . . . 10-12
10.5 Copying, Concatenating and Averaging Visibility Datasets . . . 10-13
10.6 Plotting Visibilities . . . 10-14
10.7 Examining Visibility Variables . . . 10-16
10.8 Modifying Visibility Datasets . . . 10-17

11 Calibration, the ATCA, and *MIRIAD* **11-1**
11.1 Some Theory . . . 11-1
11.1.1 Polarisation . . . 11-1
11.1.2 Atmospheric opacity and system temperature . . . 11-2
11.1.3 Antenna gain considerations and flux calibration . . . 11-3

11.1.4 Instrumental phase . . . 11-3
11.1.5 Atmospheric phase and phase calibration considerations . . . 11-3
11.2 Calibration Methods . . . 11-4
11.3 Determining Calibration Solutions . . . 11-5
11.4 Determining Gains and Bandpass Functions – MFCAL . . . 11-5
11.5 Determining Gains and Polarimetric Properties – GPCAL . . . 11-6
11.6 Copying Calibration Tables – GPCOPY . . . 11-8
11.7 Displaying Calibration Tables – GPPLT . . . 11-8
11.8 Deleting Calibration Tables – DELHD . . . 11-8

12 Calibration Strategies 12-1
12.1 Introduction . . . 12-1
12.2 Preparing your Data . . . 12-1
12.3 Calibration . . . 12-3
12.3.1 An alternative approach to calibration . . . 12-12
12.3.2 Information on planets . . . 12-13
12.4 The Interpolation Process . . . 12-13
12.5 High precision polarimetric calibration . . . 12-16
12.6 Combining Pre- and Post-August 1994 Data . . . 12-16

13 Imaging 13-1
13.1 Introduction . . . 13-1
13.2 Spectral Line Imaging Considerations . . . 13-2
13.3 Weighting . . . 13-2
13.4 Computation . . . 13-3

14 Image Deconvolution 14-1
14.1 Introduction . . . 14-1
14.2 Deconvolution with CLEAN . . . 14-2
14.3 Deconvolution with maximum entropy algorithms . . . 14-6
14.4 Multi-frequency deconvolution – MFCLEAN . . . 14-8
14.5 Other deconvolution tasks . . . 14-10
14.6 Restoring an image . . . 14-10

15 Self-Calibration 15-1
15.1 Computation . . . 15-2

16 Spectral Line Data Reduction 16-1
16.1 Velocities in *MIRIAD* . . . 16-1

16.2 ATCA Spectral Line Correlator Configurations 16-2
16.3 Spectral Line Processing and RPFITS files 16-2
16.4 Spectral Line Processing and FITS files 16-2
16.5 Listing and Plotting Velocity Information 16-3
16.6 Recomputing Velocity Information 16-3
16.7 Setting the Rest Frequency 16-4
16.8 Velocity Linetype 16-4
16.9 Continuum Subtraction 16-5
16.10 Imaging and Image Velocity Definitions – INVERT and VELSW 16-8

17 Displaying Images **17-1**

17.1 Introduction 17-1
17.2 The ATNF Visualisation Software 17-1
17.3 PGPLOT Device Tasks 17-1

18 Image Analysis **18-1**

18.1 Image Statistics and Histograms 18-1
18.2 Listing Image Values 18-2
18.3 Source Fitting and Positions 18-2
18.4 Copying, Reordering and Regridding Images 18-4
18.5 Image Arithmetic – MATHS 18-5
18.6 Smoothing Images 18-7
18.7 Modifying Images by Models 18-8
18.8 Polarimetric Analysis Tasks 18-8
18.9 Miscellaneous Analysis Tasks 18-11

19 Displaying Spectral Cubes **19-1**

19.1 Introduction 19-1
19.2 The ATNF Visualisation Software 19-1
19.3 Displaying spectra only 19-2
19.4 Overlaying spectra on images 19-2

20 Analysing Spectral Cubes **20-1**

20.1 Introduction 20-1
20.2 Moment Analysis 20-1
20.3 Smoothing the Velocity Axis 20-2
20.4 Velplot 20-2
20.5 Modelling galactic disks 20-8

20.6 Zeeman Analysis 20-10

21 Primary Beams and Mosaicing 21-1

21.1 Primary Beams and Primary Beam Correction 21-1

21.2 Mosaicing 21-2

21.3 Mosaicing Observing Strategies 21-2

21.4 Visibility Processing 21-3

21.5 Summary of Imaging Strategies 21-4

21.6 The Joint Approach 21-4

21.7 The Individual Approach 21-10

21.8 Combining Mosaics and Single Dish Data 21-11

22 Reducing 12-mm ATCA observations 22-1

22.1 Loading data into $\mathcal{MIRIAD}$: ATLOD 22-1

22.2 Calibration 22-1

23 Reducing 3-mm ATCA observations 23-1

23.1 Some theory 23-1

23.1.1 Atmospheric opacity and system temperature 23-1

23.1.2 Antenna gain considerations and flux calibration 23-2

23.1.3 Instrumental phase 23-2

23.1.4 Atmospheric phase and phase calibration considerations 23-2

23.1.5 Flux density calibrators 23-3

23.2 Practical loading and calibration 23-3

23.2.1 Loading data into $\mathcal{MIRIAD}$: ATLOD 23-3

23.2.2 Initial “fixes” to millimetre data 23-4

23.2.3 Additional monitoring variables 23-5

23.2.4 Bandpass and antenna gain calibration 23-6

23.2.5 Flux density bootstrapping 23-8

23.2.6 Information on planets 23-9

23.3 Handling large spectral bandwidths 23-9

24 ATCA Bin Mode Observations 24-1

24.1 ATCA Correlator Bin Modes 24-1

24.2 High time resolution bin-mode observations 24-1

24.3 Pulsar bin-mode observations 24-1

24.4 Tasks Specific to Pulsar Bin-Mode Observations 24-2

A Setting Up Your Account A-1

B Image Items B-1

C UV Variables C-1

D Preparing Your Data in AIPS D-1

E Using xmtv and xpanel E-1

E.1 Using xmtv . . . E-1

E.2 The Control Panel . . . E-2

F Glossary F-1

G Trouble Shooting G-1

G.1 CAVEATS . . . G-1

G.2 Error messages . . . G-1

G.3 Command not found . . . G-2

G.4 Not a directory . . . G-2

H Bibliography H-1

Chapter 1

General Information

This manual is also available, in hypertext form, on the World Wide Web at

`http://www.atnf.csiro.au/computing/software/miriad`.

This also contains information on retrieving and installing *MIRIAD*.

This manual serves as a reference guide for the *MIRIAD* package. It is assumed that the reader has some familiarity with the underlying operating system. Before proceeding further you need to read Appendix A on how to set up your account in order to use *MIRIAD*.

The manual is split into two basic parts. The first part (Chapters 2 to 6) introduces you to the *MIRIAD* package and its general concepts. The second part (Chapters 7 to 22), which uses a cookbook approach, takes you through the steps of putting your data on the local disk and reducing them.

The cookbook will describe how to reduce data from both the CABB and pre-CABB correlator eras. By default, it should be assumed that the methods described in the cookbook apply equally well to data from both correlators. However, whenever data from CABB requires extra or different treatment, the manual will clearly present the information as such.

If this is your first encounter with the *MIRIAD* package, we recommend that you read through Chapters 2 and 4 carefully.

For those interested, an overview of some of the history and design decisions of *MIRIAD*, the 'standard reference' is: Sault R.J., Teuben P.J., Wright M.C.H., 1995, "A retrospective view of Miriad" in *Astronomical Data Analysis Software and Systems IV*, ed. R. Shaw, H.E. Payne, J.J.E. Hayes, ASP Conf. Ser., **77**, 433-436.

Please send any comments or suggestions to
`miriad@atnf.csiro.au`.

Recommended Reading

The following books tend to concentrate on the techniques of radio astronomy, and particularly radio interferometry.

- Synthesis Imaging in Radio Astronomy II, G.B Taylor, C.L. Carilli, & R.A. Perley eds, Astronomical Society of the Pacific Conference Series Volume 180.
- The Fourier Transform and its Applications, R.N. Bracewell.
- Interferometry and Synthesis in Radio Astronomy, A.R. Thompson, J.M. Moran & G.W. Swenson.
- Radio Astronomy, J.D. Kraus.

- Radiotelescopes, W.N. Christiansen & J.A. Högbom.
- Tools of Radio Astronomy, K. Rohlfs & T.L. Wilson.

A general description of the science that can be done with radio telescopes is given in the following books.

- Galactic and Extragalactic Radio Astronomy, G.L. Verschuur & K.I. Kellermann eds.
- An Introduction to Radio Astronomy, B.F. Burke & F. Graham-Smith.

The following are more specific to the ATCA and the practicalities of observing and data reduction.

- Journal of Electrical and Electronics Engineering, Australia: Special Issue – The Australia Telescope, Volume 12, Number 2, June 1992.
- Australia Telescope Compact Array User's Guide
  http://www.narrabri.atnf.csiro.au/observing/users_guide/html/atug.html

A large amount of can be found on the ATNF web sites

http://www.atnf.csiro.au
http://www.narrabri.atnf.csiro.au

Acknowledgements

Most of this manual was written by Bob Sault (rsault@atnf.csiro.au) and Neil Killeen (nkilleen@atnf.csiro.au). Other parts of this manual were written by Peter Teuben (teuben@astro.umd.edu – University of Maryland), Keith Jones (jones@galaxy.physics.uq.oz.au – University of Queensland) and Tom Oosterloo (oosterloo@nfra.nl – Netherlands Foundation for Research in Astronomy). Jamie Stevens (Jamie.Stevens@csiro.au)

Chapter 2

The User Interface

2.1 Introduction

This chapter describes the user interfaces to *MIRIAD*. Each *MIRIAD* task has a number of parameters which can be specified. With some parameters, you must assign values to them when you run the task. With other parameters, if they are not set, a default value will be used, which may or may not be sufficient to run the task successfully.

There are two approaches to running tasks: the command-line approach and the front-end (or shell) approach. With the command-line approach, you give the *MIRIAD* command directly at the system prompt, in the same way as any other operating system command, such as `ls`. With the front-end approach, you interact with another program, which aids you in forming the command-line to a *MIRIAD* task.

First we will describe the `miriad` front-end, then command-line interface will be discussed Finally we discuss using scripts with *MIRIAD*.

If this is your first encounter with *MIRIAD*, we suggest you work your way through the examples starting in the next section.

2.2 The `miriad` Front-End

For involved tasks, giving all the parameters are the hosts command line becomes far too cumbersome. In this case, you can use the `miriad` front-end, which has a mentality somewhat similar to the *AIPS* user interface. To invoke `miriad`, type:

```
% miriad
```

The usual system prompt will be replaced by the `miriad%` prompt:

```
miriad%
```

and `miriad` will read a keyword file, `lastexit`, with the values of all parameters saved when you last exited `miriad` (see EXIT below). This file will be created/read in from your current working directory.

We shall now describe all `miriad` commands in more detail. A summary of the commands, and their syntax, is given in Table 2.2 at the end of this section. Commands not known to `miriad` are simply passed to your host operating system. This means standard commands, such as `ls`, are still usable and valid.

INP, GO and TASK

To inspect the inputs of a task, as well as to select the task, *e.g.* `histo` type

```
miriad% inp histo
```

`miriad` will show the parameters of the task along with the values, if any, previously set. For example, if, the first time you run `miriad`, you type:

```
miriad% inp histo
```

`miriad` will reply by writing

```
Task:   histo
in       =
region   =
range    =
nbin     =
```

and will replace the `miriad%` prompt with a task prompt

```
histo%
```

indicating that you have chosen the task `histo`. You can also choose the task `histo` without using `inp` by typing:

```
miriad% task histo
```

`miriad` will then replace the `miriad%` prompt with the `histo%` prompt, but the inputs will not be printed out. Either way a parameter can be set with:

```
histo% in=gauss
```

Retyping:

```
histo% inp
```

will result in `miriad` replying:

```
Task:   histo
in       = gauss
region   =
range    =
nbin     =
```

Tasks are run either by typing `go` *taskname* at the `miriad%` prompt (advisable only if you know you like the inputs) or by typing `go` at the *taskname%* prompt. Thus, in the above example, typing:

```
histo% go
```

would result in the task `histo` running with `in=gauss` as the only assigned parameter (all the other parameters would be set to their default values).

Any task can be run regardless of the chosen task, merely by typing:

```
miriad% go itemize
```

`miriad` then executes the task `itemize` (using whatever input parameters it finds from the `lastexit` file or a previous run) and changes the default task and prompt to:

```
itemize%
```

Generally `miriad` waits for the task to finish before it resumes. The `-b` can be used to tell the operating system to put the task in the background, and allow `miriad` to be ready immediately for new commands. However the output of the task just started may still return output to the terminal. You can start as many jobs as the operating system will allow.

HELP

The help command is generally used to get information about a task. The general format of the command is:

`miriad% help` [`-w`] *taskname* [`-k` *keyword*]

which will give you information about the keyword of the given task. If the `-k` *keyword* is omitted, information is given about all parameters of the task. If the task is omitted, information is given about the current task. For example:

```
miriad% help histo
```

will give you help information about the histogramming task, `histo`.

The `-w` flag causes output to be directed to the `firefox` web browser.

In addition to tasks, `help` will give you information about a few other select topics. These include information on tasks in general (`help tasks`), on `miriad` itself (`help miriad`) as well as a number of commonly used keywords (`device, line, options, region, select, server, stokes`).

By default, help information is sent to your terminal. However you can direct it to a web browser by setting the **MIRPAGER** environment variable to a browser of your choosing. You can also set **MIRPAGER** (or environment variable **PAGER**) to your favourite text paging program (the default is `more`, but you may prefer `less`). Note that the web version of the documentation usually contains extra links to other relevant material.

EXIT and QUIT

`exit` exits `miriad`, and, if any parameter values have been changed from the previous time you ran `miriad`, all parameter values are saved in a keyword file `lastexit`. The parameter values from this file are read in automatically when you next run `miriad`. `quit` exits `miriad` without saving the present parameter values in `lastexit`. This command is useful if you have changed parameter values you do not wish to save.

Command-Line Editing and ER

`Miriad` includes a reasonably sophisticated command-line editor. The up- and down-arrow keys allow scrolling through the history of previous commands, and the left- and right-arrow keys allow cursor motion within a line. Pressing the TAB key causes an attempt to perform file-name completion. Various control keys allow extra editing. A summary is given in Table 2.1. For copious information, see the GNU Readline library manual.

Table 2.1: `miriad` Command Line Editing Commands

Key	Description
UpArrow	Previous command
DownArrow	Next command
LeftArrow	Move cursor left
RightArrow	Move cursor right
DEL	Delete character to left of cursor
TAB	File name completion
^TAB	List possible completions
`^A`	Move to beginning of line
`^D`	Delete character underneath cursor
`^E`	Move to end of line
`^U`	Delete to beginning of line

The `er` command can be used to perform command-line editing on the current value of a parameter. It also obeys the left- and right-arrow keys, the DEL key etc. For example, to edit the current value of the parameter `select`, use:

```
miriad% er select
```

The `er` command is not as sophisticated as the normal command-line editor.

Interrupting a Task

Control-C can usually be used to interrupt the execution of a task, and revert back to the `miriad` shell.

UNSET

It is often convenient to assign the original default value to the parameter. This 'assignment' is accomplished with the command `unset`. For example, if the task `histo` had been run with inputs:

```
in       = gauss
region   =
range    = 0,1
nbin     = 10
```

but you wanted to run histo with the default for `nbin`, you would type:

```
histo% unset nbin
```

Multiple parameters can be unset on the same line. Thus, to assign both `range` and `nbin` to their default values, you would type:

```
histo% unset range nbin
```

Then, typing:

```
histo% inp
```

will result in `miriad` replying

```
Task:   histo
in       = gauss
region   =
range    =
nbin     =
```

SAVE and LOAD

As noted above, when you exit `miriad`, all your inputs will be saved in the keyword file `lastexit`. However, you can save and reload your current parameters at any time. The command `save` writes the current parameter values to a keyword file (*taskname*`.def` if the task *taskname* has been selected) or to a user-specified file. The parameter settings in this keyword file (or a user-specified file) can be retrieved using the command `load`. Note that `save` writes out **all** parameter values, not just those for the specific task chosen. Example:

```
histo% save
histo% save histo1.pars
```

In the first case the parameters are saved in a file `histo.def`, in the second case they are saved in a user-specified file `histo1.pars`.

TPUT and TGET

When you invoke a task, `miriad` saves all the parameters associated with this task in a keyword file, in the directory given by the **MIRDEF** environment variable (if **MIRDEF** is not set, the current directory will be used). The parameters will be saved in a file with name *taskname*`.def`. Additionally you can force parameters to be saved using '`tput` *taskname*'. If the taskname is omitted, the parameters for the current task are saved. The `tget` command is used to retrieve the parameters at a later stage. For example, to save and then restore parameters, use

```
miriad% tput histo
histo% tget histo
```

`tput` and `tget` normally expect the keyword files to be in the directory given by the **MIRDEF** environment variable. This is unlike `lastexit` and the `save` and `load` commands, which expect the keyword files to be in the current directory. The `-l` flag to `tput` and `tget` changes them to write/read the keyword file from the current directory.

2.3 *MIRIAD* on the World Wide Web

A hypertext version of this manual, as well as other *MIRIAD* information, is available on the World Wide Web. Its URL is

`http://www.atnf.csiro.au/computing/software/miriad`

2.4 Bug Reports and Giving Feedback

Needless to say, *MIRIAD* is not perfect. The `mirbug` command can be used if you encounter a bug, have a question about a task, or have a suggestion. The format of the command is

`% mirbug` *taskname*

Table 2.2: `Miriad` shell command overview

Command	Syntax	Comments
=	key = value	assignment (see also SET)
cd	cd directory	change directory
er	er key	command line editing of a parameter.
exit	exit	exit `miriad`, and save parameters in lastexit
go	go [-b] [taskname]	start up task
help	help [-w] [taskname] [-k key]	on line help on task
inp	inp [taskname]	overview inputs
source	source cmdfile	read commands from a command file
load	load [keyfile]	load task parameters from keyword file
quit	quit	quit `miriad`, and does not save lastexit
save	save [keyfile]	save task parameters to keyfile
set	set key value	assigment
setenv	setenv var value	set an environment variable
task	task taskname	set new default taskname
tget	tget [-l] [taskname]	load parameters for a particular task
tput	tput [-l] [taskname]	save parameters for a particular task
unset	unset key1 key2	unset variables (blank them)
unsetenv	unsetenv var	unset an environment variable
view	view [taskname]	edit task parameters in a keyfile

Table 2.3: `Miriad` Environment Variables

Environment Variable	Comments
VISUAL	Editor for editing keyword files and bug reports.
EDITOR	Editor for editing keyword files and bug reports.
MIRPAGER	Web browser or pager for reading help documents
PAGER	Pager for reading documents
MIRDEF	Directory for keyword files
TMPDIR	Directory for scratch files (see Section 4.3).

Here *taskname* is the task in question. Use the name "`general`" if the bug report is not about a specific task. The `mirbug` command places you in an editor (given by the `EDITOR` environment variable), with a template of a bug report, which includes the parameters most recently used for that task. After you finish editing the template, `mirbug` mails the report.

2.5 Indirect Parameter Input

It is possible that the length of a parameter value can become large (indeed quite huge). In this case, it is convenient to store the parameter value in a file and then to use the file name in place of the parameter value.

To do this, the file name should be preceded by an 'at' sign (`@`). For example, assuming the file `name` contains the text `chicyg` , then

```
% histo in=@name
```

is equivalent to

```
% histo in=chicyg
```

These `@` files can be nested, and multiple `@` files can appear on a line (separated by commas). Indeed an `@` file and other parameter values can be intermixed, separated by commas.

The `@` file can contain many lines. An end-of-line is treated like a comma.

As described in Section 2.6, parameter input can also be made easier by using the `-f` flag when *MIRIAD* commands are given directly from the shell.

2.6 Host Command-Line Interface

MIRIAD tasks can always be run by specifying their parameters on the host command line. A parameter is specified by equating its keyword to its value. For example, `out=gauss` sets the parameter `out` to the 'value' `gauss`. As an example, we shall first create an image dataset, `gauss`, with the task `imgen`. At the system prompt type

```
% imgen out=gauss
```

In this case, default values are used for all parameters, other than `out`. The names of all parameters, and their default values, are described in a help file (see below) which is always available on line.

Next, we will run a histogram task, `histo`, with the input image dataset `gauss` that we just created:

```
% histo in=gauss
```

Several parameters can be given on the command line, separated by spaces. Often a parameter value consists of several numbers and/or strings. These values should be separated by commas. For example:

```
% histo in=gauss range=0,1 nbins=10
```

Strings will often be file names. These are the names used by the host computer, and so they should obey the file-naming rules of the computer. Some tasks support wildcard expansion for file names.

Whether strings are in lower or upper case is usually significant. That is,

```
% histo in=gaus
```

is very different to

```
% HISTO IN=GAUS
```

Generally lower case is preferred.

A parameter value cannot contain a space. Often a parameter value will be composed of one or more strings out of a fixed set. In this case, the strings can be abbreviated to the minimum number of characters required for uniqueness.

The host's shell may try to interpret special characters in your commands – most notably the UNIX shells treat asterisks and brackets as special. This will often result in the host shell issuing cryptic messages and failing to invoke the *MIRIAD* task. For example, giving the command

```
% histo in=gaus*
```

will probably result in

```
No match.
```

Similarly

```
% histo in=gaus region=quarter(1)
```

will result in

```
Badly placed ()'s.
```

You can avoid this by escaping the special character or quoting the text containing the special characters. For example:

```
% histo in=gaus "region=quarter(1)"
```

The command-line interface is quite appropriate for tasks that take relatively few parameters, or in shell scripts. However, it is cumbersome when a task has many parameters. Here the use of a keyword file comes in handy:

Keyword files are files containing `parameter=value` pairs, one per line, and can be created with an editor (most front-ends to be discussed in the next sections create such `.def` files by themselves) and passed to a *MIRIAD* task using the `-f` command line switch, or flag. Hence the previous example would be equivalent to:

```
% histo -f histo.def
```

where the file `histo.def` would contain:

```
in=gaus
region=quarter(1)
```

Unknown keywords are simply ignored (though you will get a warning). The file should not use the same keyword twice. As parameter values do not go through the host's shell, special characters do not need to be escaped or quoted.

Help on a *MIRIAD* task can be obtained on the host command line with the command `mirhelp`. Example:

```
% mirhelp histo
```

This will provide description of the task parameters and their default values.

The `mirhelp` command can be used to save help on a MIRIAD task as a text file, or to get a print-out of a help file. In UNIX, you would use the normal shell output redirection and piping. For example, to redirect the help on task `histo` to a file `histo.hlp`, use:

```
% mirhelp histo > histo.hlp
```

Alternately, to print out this help file, use

```
% mirhelp histo | lw80
```

where `lw80` is your favourite print command.

2.7 Writing Shell Scripts

As MIRIAD commands can be invoked directly from the command line, scripts and command procedures to run a sequence of MIRIAD commands can be developed using the normal host's facilities. This is a somewhat advanced topic – you will probably want to be familiar with the shell scripts and MIRIAD before you attempt to develop your own script.

There are numerous books written of shell programming or the like – a manual like this cannot be expected to cover the subject in the depth that these books go into. Instead a simple annotated example will be given using the C-Shell commonly used on UNIX systems. To aid description, line numbers are given on the left side of each line (these line numbers are *not* part of the shell script).

```
 1:  #!/bin/csh
 2:
 3:  # Delete any datasets called "multi.uv".
 4:
 5:  rm -rf multi.uv
 6:
 7:  fits in=MULTI.UV op=uvin out=multi.uv
 8:
 9:  foreach srcnam ( 1934-638 0823-500 vela )
10:    uvaver vis=multi.uv "select=source(${srcnam})" out=${srcnam}.uv
11:  end
12:
13:  mfcal vis=1934-638.uv interval=10 refant=3
14:  gpcal vis=1934-638.uv interval=10 options=xyvary refant=3
15:  gpcopy vis=1934-638.uv out=0823-500.uv
16:
17:  gpcal vis=0823-500.uv interval=10 \
18:                 options=nopol,xyvary,qusolve refant=3
19:
20:  gpboot vis=0834-500.uv cal=1934-638.uv
21:  uvplt vis=0823-500 stokes=i,q,u,v axis=real,imag device=0823.ps/ps
22:
23:  lpr 0823.ps
```

- Line 1: C-shell scripts *must always* start with the rather cryptic `#!/bin/csh`. This allows the system to determine the appropriate interpreter for the script (i.e. the C-shell).
- Line 3: Comments are introduced by a hash character.
- Line 5: Delete a MIRIAD dataset called `multi.uv`. The `r` flag (recursive) is needed to delete a directory (which is how MIRIAD stores a dataset). The `f` (force) flags causes `rm` to work in non-interactive mode.

- Line 7: Execute the *MIRIAD* task `fits`. This reads a visibility FITS disk file (`MULTI.UV`), and saves it as a *MIRIAD* dataset (`multi.uv`).
- Lines 9 and 11: This is the C-shell's equivalent of a DO-loop. The values listed inside the brackets (source names in this instance) are progressively assigned to the control variable of the loop (`srcnam` here). The net result is that this loop will be executed three times, with `srcnam` being successively set to `1934-638`, `0823-500` and `vela`. The "things" inside the brackets are separate by spaces – the C-shell generally likes spaces to separate different components of a command.
- Line 10: The body of the loop. The `uvaver` command will be executed three times. The shell substitutes the string `${srcnam}` with the current value of the `srcnam` variable. Thus the `select` parameter being successively set with `source(1934-638)`, `source(0823-500)` and `source(vela)` on the three times through the loop. The output dataset of the command also changes. Note that the `select` parameter is quoted (`"select=..."`). This is as brackets (`()`) are special to the shell. Quoting them prevents this interpretation. Note however that the `${srcnam}` is still treated as special, even though it is in quotes (software does not have to be consistent!).
- Lines 13-15: Execute various *MIRIAD* commands on data-sets produced by line 10.
- Lines 17-18: Yet another command. However this one was too long to fit on one line. In the C-shell, a line can be continued onto the next by ending it with a backslash.
- Lines 20 and 21: More of the same. Generate a postscript output file, `0823.ps`, using task `uvplt`.
- Line 23: Spool this postscript file to the laser printer.

On UNIX systems, after having developed a script, you will need to change the "file mode" of the script to indicate that the script is executable. For example, to mark the shell script `calibrate` as executable, use the UNIX command

```
% chmod +x calibrate
```

Chapter 3

Plotting Concepts

3.1 Introduction

Display software for *MIRIAD* data falls into two main camps – those that are part of the ATNF Visualisation Suite (not strictly part of *MIRIAD*) and those that use a PGPLOT device (line plots plus some image display capability). Generally the Visualisation suite is better for interactive work and the PGPLOT software is better for quantitative work. This chapter will give an overview of using the PGPLOT software.

Historically there is a third camp – the "TV" software. There is only one useful task left which uses this software, and so the "TV" software will be described with that task. Unfortunately, the "TV" software can only be used with 8-bit colour-depth displays, which limits its usage with modern computers which usually operate with far greater colour depths.

3.2 The ATNF Visualisation Suite

The KARMA visualisation suite is a set of X-windows-base interactive tools that can display a wide variety of images and cubes – in particular *MIRIAD*'s image format. You select images, display options, etc, via pull-down menus (rather than setting parameter values). More details can be found in the on-line documentation available on the ATNF web pages (http://www.atnf.csiro.au/computing/software/karma/) or in a gzipped postscript document on the ATNF ftp server (usermanual.ps.gz under pub/software/karma, which is also available through the ATNF web pages).

KARMA replaces the ATNF Visualisation suite, details of which can be found in the on-line documentation available on the ATNF web pages (http://www.atnf.csiro.au/computing/software/visualisation/) or in a gzipped postscript document on the ATNF ftp server (vizdoc.ps.gz under pub/software/sutra, which is also available through the ATNF web pages).

See Appendix A to set-up your account to use the visualisation software.

To run a tool, type the program name, no arguments are required. You do not need to be running the `miriad` front-end to use the tools.

3.3 Using X Windows

The ATNF Visualisation software and some of the PGPLOT devices work via X-windows. For these to work, you must ensure that you have setup your environment to allow communication with your X server. In particular

- You should be working at a workstation running X windows!

- You do not need to run the display tasks on the same computer as the X-windows server. However if you are running on a different machine, then you will need to tell the display software which machine is running your X server.

  If you have connected through SSH, then the SSH client should have, or can be made to automatically configure X-windows forwarding to your local screen. To ensure that X-forwarding is enabled, use the SSH connection command:

  `% ssh -X user@`*remotehost*

  Different SSH clients may use `-Y` or `-x` to enable X-forwarding; check your man page to determine which is appropriate for your installation.

  If you have connected in another way – `telnet` for example – then you will need issue the command

  `% setenv DISPLAY` *localhost*`:0`

  to the machine you are running *MIRIAD* tasks on. Here *localhost* is the name of the host machine running X windows. Normally this will be you local workstation. Do NOT just type `localhost` as this will not work.

- X windows has a simple security feature to prevent arbitrary people drawing plots on your screen. To authorise a particular machine to send commands to your X server, you will need to issue the command

  `% xhost +`*remotehost*

  to your local host. Here *remotehost* is the name of the host machine that you want to grant permission to. For *MIRIAD* applications, this will be the host you are running the plotting task on.

3.4 PGPLOT Plotting Devices

Tasks which use the PGPLOT software always use the `device` keyword to allow you to select a plotting device. Only a brief review of PGPLOT, from a users perspective, is given here. A more complete description, both for users and programmers, can be found in the PGPLOT manual.

PGPLOT devices are specified in the standard PGPLOT manner, namely

`device=`*name*`/`*type*

The *name* part either gives a device name (usually for graphics devices) or a file name (usually for hardcopy devices, such as postscript printers). In the case of a file name, any normal file name can be given. However if it contains a `/` character, then the entire file name should be enclosed in double quotes (`"`). Once a file has been created, you will generally have to issue operating system commands to spool this to the appropriate output plotter, etc. The *name* part can often be left blank – it defaults to something sensible. The most common case where you do not let it default is when you are specifying a disk plot file.

The *type* part tells PGPLOT what sort of graphics or hardcopy device is being used. Minimum match is used. Some possible types are:

`xs` This is a re-sizable and persistent X window.

`xw` This is a transient X window.

`xd` This is the `pgdisp` X windows server (deprecated).

`ps` Postscript. PGPLOT will write the plot as a disk file in postscript form. At ATNF sites, you can print this postscript file with the command:

  `% lp` *filename*

`vps` Vertical postscript. The only difference between this and `ps`, is that `vps` generates a plot in portrait mode, whereas `ps` is in landscape mode.

`cps` Colour postscript in landscape mode.

`vcps` Colour postscript in portrait mode.

`null` The null device. Useful for debugging when you do not have a display.

`re` A VT100 with RETROgraphics card.

`v603` A V603 terminal.

`krm3` A Kermit 3 IBM-PC terminal emulator.

`tek` A TEK 4010 compatible terminal (or window).

Examples of PGPLOT devices are:

- `device=4/xs` directs output to the persistent X window number 4 (you can have many of these windows)
- `device=/xw` directs output to the transient X window.
- `device=plot.ps/ps` create a postscript plot file called `plot.ps`
- `device=/tek` assumes you are working at a Tektronix 4014 terminal. Graphics output is directed to your terminal.

Another useful pseudo-PGPLOT device is `?`. This causes a complete list of PGPLOT types to be printed, and then the task will prompt you to give a PGPLOT device.

When plotting in an X window, you must meet the conditions which allow PGPLOT to communicate with your X server.

PGPLOT has two X windows servers, and as of PGPLOT V5.0, one of which (`pgdisp`) is deprecated. The PGPLOT device types `/xs` and `/xd` are now mediated by a server (called `pgxwin_server`). This server is automatically started whenever either of these devices is invoked. The `/xs` windows are permanent and resizable whereas the fixed size `/xw` windows disappear as the application terminates. This server allows multiple windows and you can choose to write to whichever one you like (see above example). The resources for these windows are managed through the standard X windows resource file.

The deprecated server, `pgdisp`, which creates a resizable persistent X window is started by hand with a command such as

```
% pgdisp &
```

from a workstation window (usually the console window). This window is written to with the keyword `device`, see below). This window can be used for line graphics as well as grey scales. By default, the `pgdisp` window only has 16 colour levels, which is insufficient for most image display applications, but fine for line graphics.

If you start it up with the command

```
   % pgdisp -lineColors 128  &
```

then the window will be allocated 128 colours which is sufficient for imaging applications. This window is much slower than the windows mediated by the `pgxwin_server`. In addition you can only have one.

Chapter 4

MIRIAD Datasets

4.1 Listing Datasets – PRTHD

MIRIAD uses the term 'dataset' to refer to an image, a cube or a set of visibility data. A *MIRIAD* dataset is made from a host-system directory, *i.e.*, the host operating system sees a 'directory' whenever *MIRIAD* sees a 'dataset'.

The task `prthd` provides an astronomical summary of a dataset. For visibility datasets, it will tell you the number of visibilities, the number of spectral-channels, the number of spectral-windows (IFs), the polarisations present and some information about frequencies. Note that if the visibility dataset contains multiple source or frequencies, it will only give you information about the source and frequency of the first record. For images, `prthd` will give dimension and axis information, minimum and maximum, etc. The input to `prthd` is the dataset name via the keyword `in`.

PRTHD	
in=gauss	Input dataset.

4.2 Inside Datasets

Generally the user does not need to understand much about the internal structure of a dataset. A dataset contains several kinds of data, called 'items', . Items can be quite small (e.g. a single number), intermediate in size (e.g. the `history` item, use to store history information), or very large (e.g. the `image` item in an image dataset, which is used to store the pixel data, or the `visdata` item in a visibility dataset, which contains the correlations). Large items are stored as a host system file, whereas all small items are stored in a common file which is rather inappropriately called `header`. Indeed, there are a number of instances where the word "header" is used where "item" or "small item" would be a more appropriate description.

The implementation of datasets as directories does complicate some manipulations of your datasets, since your favourite image, etc, is not just a file anymore. On the other hand, as the host system sees a *MIRIAD* dataset as a normal directory, all the usual host commands to manipulate directories can be used. On UNIX a `-r` switch often has to be used with the command, to indicate that the operation is to be applied 'recursively' (i.e. to all files in the directory). For example, to delete a dataset, use

% `rm -r` *dataset*

If you have aliased `rm` to prompt you before deleting a file (as is common in a number of the standard login scripts at Epping), you will be prompted before deleting each individual file within a dataset. This can become somewhat tedious, so you might want to make another alias to delete without prompting. For example, insert

```
alias rrm 'rm'
```

in your `.cshrc` file. Similarly to copy a dataset, you would use

% cp -r *dataset1 dataset2*

Generally the user is insulated from this internal organisation of a dataset and can always think of them as a whole. However there are a few *MIRIAD* utilities to manipulate at the item level. These tasks do not contain any astronomical knowledge. Consequently they may seem somewhat crude. These tasks include:

- `itemize`: List the items in a dataset.
- `delhd`: Delete an item from a dataset.
- `puthd`: Add an item which consists of a single-value to a dataset.
- `copyhd`: Copy an item from one dataset to another.
- `gethd`: Print the value of an item. This is most useful in scripts.

Note the rather inappropriate use of 'hd' in the preceding names, where something suggesting items (rather than header) would have been more appropriate.

As an example, consider the `itemize` task, which lists the items in a dataset. For the test image dataset, `gauss` (created with `imgen` in Chapter 2) `itemize` will tell us of the following items:

```
% itemize in=gauss
Itemize: version 1.1 4-mar-91
naxis    = 2
naxis1   = 256
naxis2   = 256
crpix1   = 129
crpix2   = 129
cdelt1   = -4.848137e-06
cdelt2   = 4.848137e-06
crval1   = 0
crval2   = 0
history    (text data, 38 elements)
image      (real data, 65536 elements)
```

Here the item `naxis` consists of a single integer, having the value of 2. The item `image` is a larger item (being the pixel data) consisting of 65536 (256×256) real numbers.

Note for images that many items have FITS-like names (although they are lower case, and the units can be different from the FITS standard). A list of the items in an image and visibility dataset are given in Appendices B and C.

4.3 Scratch Files

Some *MIRIAD* tasks create scratch files to hold temporary results. These files do not show up in directory listings, but they do take up real space (these files are guaranteed to be deleted if a task dies). *MIRIAD* will place these scratch files in the directory pointed to by the environment variable `TMPDIR`. If this is not set, the current default directory is used.

4.4 *MIRIAD* Data Disks at Epping

As datasets are normal directories, they can be created wherever you can create directories. Though it is possible to create them in your home directory, given the quotas enforced on home directories you are unlikely to be able to store much data there.

Epping users can use the `/DATA` disks (shared with *AIPS* users) to store their data for a few days to a few weeks. To do so, you should create a directory `/DATA/`*host_x*`/`*username*, where *host_x* is a machine name and a number (*e.g.* `ATLAS_1`) and *username* is your login name. For example

```
% mkdir /DATA/ATLAS_1/rsault
```

creates a directory for `rsault` on the first data disk of machine atlas.

All the disks attached to all the different workstations are accessible from any machine. That is, the disks are cross-mounted between the computers. However there is a right way and a wrong way to use this cross-mounting – there is a large network penalty (particularly to write) to a remote disk. For example, there is a large overhead if you write to a disk attached to APUS when you are logged into CARINA. The penalty is smaller if you read from a network mounted disk, but is still substantial. In general you should attempt to keep your data on the disks of one workstation, and use that workstation for all your compute and I/O intensive work.

There are two sorts of disks – public and bookable. Public disks can be used by anyone at anytime. However their contents *may be automatically cleared* at 5:00 am on Mondays. The bookable disks provide a private allocation of disk space, which is not subject to destruction on Monday morning. Note, however, when your booking has expired, your data can be deleted from a bookable disk without warning. Disks can be booked by e-mailing 'bookings'.

Chapter 5

Visibility Data Concepts

5.1 *MIRIAD* Visibility File Format

A visibility dataset in *MIRIAD* logically consists of two parts – a stream of variables and calibration tables. The stream of variables consists of parameters that are known at the time of the observation. They describe the changing state of the telescope. Variables include the observing frequency, the observing centre, the source name, (u, v) coordinates, the baseline number, the polarisation parameter being measured, and the actual measured correlations. The calibration tables are parameters that are derived after the observation. They are usually deduced from observations of a calibrator. Generally, *MIRIAD* tasks give a higher level view of the internals of a dataset, but it can be useful sometimes to examine these building blocks of the dataset. A list of all the variables and calibration tables is given in Appendix C.

Because source names, frequencies, observing centre etc., are stored as variables, *MIRIAD* does not distinguish between a single-source file and a file with multiple sources or frequencies, or a mosaiced observation. For example, a visibility dataset containing multiple sources does not differ structurally from one containing a single source. However, the variables `source, ra` and `dec` will change several times within a dataset which contains multiple sources, whereas they would remain constant in a dataset with only a single source.

5.2 *MIRIAD*'s Calibration Model

MIRIAD's suite of calibration tasks generally produce or use items which contain the relevant calibration tables and parameters. The tasks to manipulate these calibration tables generally have names starting with `gp`, though there are a few exceptions such as `selfcal` and `mfcal`. The calibration parameters are antenna-based ones, with the calibration data being used to derive a model of the response of each antenna. The response of an antenna to radiation is modelled by four sets of parameters:

- Antenna gains. These are complex-valued gains which vary with time, but not frequency. These are predominantly atmospheric in origin, though there is a significant instrumental component (the two cannot be distinguished readily). *MIRIAD* stores these in the `gains` item of a visibility dataset.
- Delay factors. These are antenna-based parameters which give a time delay, which will probably be a sum of atmospheric and instrumental components. This term is assumed to vary with time, but not frequency. It is stored in the `gains` item.
- Antenna bandpass functions. These are complex-valued gains modelling the instrumental bandpass. These vary with frequency but not necessarily with time (recent MIRIAD additions have allowed the optional use of time-dependent bandpass solutions). These are largely instrumental in origin. These are stored in the `bandpass` item (and some associated items).
- Antenna leakages. These are complex-valued terms (which do not vary with time or frequency) which model the leakage of one polarisation into another. These are entirely instrumental in origin.

Current experience suggests that leakages may vary moderately with frequency. These are stored in the `leakage` item.

Ignoring leakage (which is discussed in too much detail in Chapter 12), we model the composite gain function of an antenna (representing both atmospheric and instrumental terms) as

$$g(t)g_P(\nu)\exp(i2\pi\tau(t)(\nu-\nu_0))$$

Here $g(t)$ is the frequency-independent part of the antenna gain, $g_P(\nu)$ is the bandpass function, $\tau(t)$ is a delay term. The delay is calculated with respect to a reference frequency, ν_0. For dual polarisation systems the two polarisation bands are assumed to have independent gains and band passes, although the delay is common.

Note that $\mathcal{MIRIAD}$ datasets *cannot* contain more than one set of calibration tables – you cannot have multiple versions of calibration tables. Running a calibration task twice will result in the first calibration table being overwritten. This may be inconvenient if the dataset has two sets of data that need to be calibrated separately.

5.3 On-the-Fly Calibration Correction

Many tasks will apply calibration corrections 'on-the-fly'. This means that often there is no need to form a calibrated dataset. Normally this calibration correction step is performed by default. However, the correction stage can be disabled using some fairly standard switches in the `options` parameter. These switches are:

`nocal`: Do not apply antenna gain and delay corrections.

`nopol`: Do not apply polarisation leakage corrections.

`nopass`: Do not apply bandpass function corrections.

For example, to disable gain and bandpass correction in `uvspec`, use:

```
% uvspec options=nopass,nocal
```

It is always possible to form a calibrated dataset by using either of the visibility copying tasks (`uvcat` or `uvaver`). Whether or not you choose to do this may be often a matter of personal taste. However there are three *VERY* important situations where you should form a calibrated copy of your dataset (*i.e.* where 'on-the-fly' calibration is inadequate).

- A still significant, but decreasing, number of tasks do not perform 'on-the-fly' calibration. Tasks which do apply calibration corrections will invariably have `nocal, nopol` and `nopass` options. Additionally all tasks which are performing 'on-the-fly' calibration will issue messages when they are performing these steps.
- Many $\mathcal{MIRIAD}$ tasks allow 'on-the-fly' averaging of spectral channels (using the `line` parameter with a width greater than 1 – see section 5.4) in addition to bandpass correction. However the technique used when simultaneously bandpass correcting and averaging is only approximately correct. For arcane reasons, rather than applying the bandpass and then averaging the channels, the tasks correct the average of the channels with the average of the bandpass. This can be significantly in error if the bandpass functions vary significantly over the range of channels being averaged. Tasks performing this dubious operation issue warning messages to alert you of your possible folly. If this is a real problem, then it is best to form a copy of the bandpass-corrected data, and then use these corrected data for channel averaging.

- The self-calibration tasks produce gain tables in the same format as the calibration tables derived from observations of calibrators. Thus if you self-calibrate a dataset containing your initial calibration, you will overwrite your initial calibration – a generally undesirable step. Additionally the self-calibration tasks do not apply some calibration corrections (e.g. bandpass and sometimes leakage corrections). Generally it is best to self-calibrate a dataset which has had the initial calibration applied.

5.4 Channel Selection, Averaging and Doppler Correction – UV Linetypes

Many *MIRIAD* tasks support on-the-fly selection and averaging of the channels to be processed. However, before launching into a description of this, we will review some *MIRIAD* history. *MIRIAD* was originally designed for a telescope which simultaneously measured both spectral and continuum data using separate correlators. Spectral data are generally narrowband, and the frequencies are defined to high precision. Doppler tracking is often employed. Continuum, or wideband, data have a much larger bandwidth, and frequency tolerances are not as great.'Channel-0' data (data formed by averaging all spectral channels together) are also treated as continuum data.

For ATCA use, there is no real distinction between wideband (continuum) and spectral data, and the *MIRIAD* distinction is not really relevant. All ATCA data (even channel-0 data) are treated as spectral data.

All the same, a *MIRIAD* visibility dataset can, in principle, contain multiple spectral and wideband correlators. The spectral data is described by a set of 'spectral windows' ('IFs' or 'IF channels' in *AIPS* terminology); each window consists of a number of channels separated by a fixed increment in sky frequency (though this increment can vary with time). Similarly there can be several measured wideband correlations, simply called wideband channels.

It is quite common, when analysing, plotting or mapping visibility data, that you will want to perform some averaging and selection of the desired channels, and you might wish to examine either the spectral or the wideband data. For the spectral data, if the channel number does not correspond reasonably directly with velocity (*e.g.* if Doppler tracking was not used), then it might be desirable to resample the spectral data at equal increments in velocity.

The ability to select a range of wideband or spectral channels, to perform averaging, and to resample in velocity is provided by the 'line' parameter – also called the linetype. If your data contain multiple spectral windows, you should also refer to Section 5.6 for more information on spectral channel selection.

The linetype parameter consists of a string followed by up to four numbers. Defaults will be used for any trailing part of the linetype specification that is missing. The string can be one of:

`channel` This gives raw or averaged spectral channels. This is generally the default if spectral data are present. As all ATCA data are treated as spectral data, this will be the most commonly used linetype.

`wide` This gives raw or averaged wideband (continuum) data. This is the default if only wideband data are present. Probably this will be of no interest to ATCA users.

`velocity` This gives spectral data, that have been resampled at equal increments in radio velocity (or equivalently frequency). The resampling operation is a weighted average of spectral channels. See Section 16.8 for more information.

`felocity` This is like `velocity`, but allows the velocity parameters to be given using the optical definition. Note, however, that the resampling operation is still in equal increments in *frequency (or, equivalently, radio velocity)*. Because of the difference between the radio and optical velocity definitions, equal increments in radio velocity are not quite equal in optical velocity, and visa versa. The velocity increment that you give is used as the optical velocity increment of the first channel.

The accompanying four numbers are used to specify the range of input channels selected and averaged to produce the output channels. The four numbers are:

nchan, start, width, step

For `channel` and `wide` linetypes, *start, width* and *step* are channel numbers (channels are numbered from 1 to N), whereas for `velocity` and `felocity` linetypes these values are in km s^{-1} (the velocity is relative to the rest frame – usually LSR). These values are

nchan The number of output channels produced. Generally it defaults to the maximum number of channels that can be produced from the input data. A value of zero can also be used to give you the default.

start For `channel` and `wide` linetypes, the *start* value is the first input channel to be selected. For `velocity` and `felocity` linetypes, *start* is the centre velocity of the first output channel to be formed. The default value is 1 channel.

width This value determines the width of the selected channels. For `channel` and `wide` linetypes, this gives the number of input channels to average together to produce a single output channel. For `velocity` and `felocity` linetypes, this gives the velocity width (in km s^{-1}) of the output channels. The default value is again 1 channel.

step This parameter gives the increment between channels. For `channel` and `wide` linetypes, this gives the increment between selected input channels. For `velocity` and `felocity` linetype this gives the velocity increment between the output channels. This defaults to the same value as *width*.

For example

```
line=channel,10
```

selects 10 output channels, being input spectral channels 1 to 10. Similarly

```
line=channel,10,8,1,2
```

again selects 10 output channels, starting at input spectral channel 8, and skipping every second input channel. If you wanted to average together every pair of channels (rather than skipping it), you would use something like

```
line=channel,10,8,2,2
```

Finally a linetype of:

```
line=velocity,10,1.5,1.0,3.0
```

would return 10 'velocity' channels with velocities *centred* at 1.5, 4.5, 7.5, etc. km s^{-1}. Each channel would have a width of 1 km s^{-1}.

When using `velocity`, `felocity` or `channel` linetypes on datasets with multiple spectral windows, `window` selection, as described in the following sections, may be useful.

Some tasks require two linetypes, the first being the linetype of the data, and the second the linetype of a single reference channel (see *e.g.* `invert`). When specifying a reference linetype, you do not give the *nchan* (it is always 1) or *step* (it makes no sense for a single channel).

5.5 Selecting UV Data in *MIRIAD*

It is common to wish to process only a subset of the possible visibility data. In *MIRIAD*, the visibility data to be selected are usually specified by one parameter – the `select` keyword. This parameter is constructed from a number of subcommands, each subcommand selecting or rejecting visibility data which satisfies some condition. The subcommands, which can be abbreviated to uniqueness, are as follows:

`time`(*t1*,*t2*) This selects visibilities observed between times *t1* and *t2*. Times are given in UT. The second time (*t2*) is optional. If missing, it is assumed to be 24 hours after *t1*.

The time is composed of a date and 'time-of-day' portions, either of which can be omitted (but not both at the same time!). When the date is omitted, then the selection matches data, for the given time-of-day, regardless of the day. This is most useful for files containing only a single day's data. When the time-of-day is omitted, then 00:00 is assumed.

The times *t1* and *t2* are given in the form:

yymmmdd.fff

or

yymmmdd:hh:mm:ss.s

Here *yy* is the year, *mmm* are the first three letters of the month's name, *dd* is the day of the month, *fff* is a fraction of a day, *hh* is the hour (24-hour clock), *mm* are the minutes, and *ss.s* are seconds and fractions of a second. The 'time-of-day' portion can be abbreviated. For example, the seconds part can be omitted. Indeed (provided a date is given) the 'time-of-day' portion can be totally omitted if desired.

Note that only a two-digit year is given. These two digits give a year in the century 1940 – 2040.

For example:

```
90jan12:12:30
```

is 12:30 on 12 January, 1990, whereas

```
78jun03.5
```

is midday on 3 June, 1978, and finally

```
00apr01
```

is April Fools Day in the year 2000.

To give some examples:

```
select=time(91jan05:10:50,91jan05:17:20)
```

will select data observed on 5th January, 1991, between 10:50 UT to 17:20 UT.

The form where no date is given:

```
select=time(10:50,17:20)
```

will select data observed on any day between 10:50 and 17:20.

To give an example where only one time is given, the following

```
select=time(91jan03)
```

will select all data observed on 3rd January, 1991. This form is only useful for a file containing several days of data.

`antennae(`*a1,a2,...*`)(`*b1,b2...*`)` This selects according to the antennas and baseline. Here *a1,a2,...* is a list of antennas to select. The second list, *b1,b2,...* is optional. If present, only baselines corresponding to the antennas pair (a1,b1), (a1,b2), ..., (a2,b1), ..., etc are selected. For example, to select all visibilities using antennas 1 or 3, use

```
select=antennae(1,3)
```

To select all visibilities using baselines with antennas 1 and 2, or 3 and 2, use

```
select=antennae(1,3)(2)
```

`uvrange(`*uvmin,uvmax*`)` This selects visibilities whose $u-v$ radius is in the range *uvmin* to *uvmax*. If only one value is given, then *uvmin* is taken as zero. The units of *uvmin* and *uvmax* are kilowavelengths.

`uvnrange(`*uvmin,uvmax*`)` This is the same as the `uvrange` subcommand, except that the units are nanoseconds.

`ha(`*h1,h2*`)` This selected data based on the hour angle of the data. *h1* and *h2* are the hour angle (in decimal hours).

`elevation(`*e1,e2*`)` This selects data based on the elevation at which the data were observed. *e1* and *e2* give the elevation range in degrees.

`lst(`*l1,l2*`)` This selects data based on the LST of the observation. l1 and l2 are the start and end LST values, and can be given in the form hh:mm:ss or as decimal hours.

`visibility(`*n1,n2*`)` This selects visibilities numbered *n1* to *n2* inclusive.

`increment(`*inc*`)` This selects only every *inc*'th visibility.

`ra(`*hh:mm:ss1,hh:mm:ss2*`)` Select visibility data within a particular range of right ascension. Right ascension is given in *hh:mm:ss* format, or as decimal hours.

`dec(`*dd:mm:ss1,dd:mm:ss2*`)` Select visibility data within a particular range of declination. Declination is given in *dd:mm:ss* format, or as decimal degrees.

`dra(`*p1,p2*`)` For data files containing several pointing centres, this selects visibilities whose pointing centre is offset, in RA, from the main pointing centre, by between *p1* and *p2* arcseconds.

`ddec(`*p1,p2*`)` For data files containing several pointing centres, this selects visibilities whose pointing centre is offset, in DEC, from the main pointing centre, by between *p1* and *p2* arcseconds.

`pointing(`*p1,p2*`)` For data files containing rms pointing error data, this selects visibilities with the rms pointing error in the range *p1* to *p2* arcseconds. If only one number is given, *p1* is taken as 0. The rms pointing error of a visibility is taken as the maximum of the rms azimuth and rms elevation pointing errors of the two antennas.

`source(`*srcnam1,srcnam2,...*`)` For data files which contain multiple sources, this selects data according to source name. Several source names can be given, separated by commas. The source name can include an asterisk, which acts as a wildcard.

`frequency(`*loval,hival*`)` This selects data according to the sky frequency of the first channel. The *loval* and *hival* values give the permissible range, in GHz, of frequencies to accept. If only a single value is given, then this is used as a mid-frequency, and frequencies within a tolerance of 1% are accepted.

`amplitude(`*x,y*`)` Any correlation, where the amplitude is between *x* and *y*, is processed. If only one value is given, it is assumed to be *x*, and *y* is assumed to be infinity.

`window(`*w1,w2,...*`)` This can be used when the dataset contains multiple spectral windows (or IF bands in $\mathcal{AIPS}$ terminology). This is discussed in Section 5.6.

`shadow(`*d*`)` This selects data that would be shadowed by an antenna of diameter *d* meters. If *d* is zero, then the actual diameter of the antennas (if known) is used. If some data is shadowed, it is advisable to use an antenna diameter value greater than the physical antenna size (*e.g.* 20% larger).

`auto` This selects auto-correlation data only. So the negated form (`-auto`) selects cross-correlation data only.

`on` This is intended for use with beam switched single dish observations, and selects based on the 'on' variable in the visibility dataset. The 'on' variable indicates whether the telescope is pointing on or off source.

`bin(`*n1,n2*`)` This selects visibilities whose pulsar bin number is in the range *n1* to *n2* inclusive. If *n2* is omitted, just bin *n1* is selected.

`polarization(`*a,b,c,...*`)` This selects based on the polarization/Stokes type of the visibility. The possible values for *a,b,c,* etc. are mnemonics for the polarization type, as discussed in Section 5.8.

`or` This is discussed below.

As noted before, all subcommand names can be abbreviated to the minimum number of characters needed to keep them unambiguous (minimum match).

Each subcommand can be prefixed with a plus or minus sign (`+` or `-`). A plus sign means to select the data given by the following subcommand, whereas a minus sign means to discard the data. If neither a plus nor a minus sign is present, a plus sign is assumed. For example

```
select=uvrange(0,10)
```

means to select all visibilities between 0 and 10 kilowavelengths, whereas

```
select=-uvrange(0,10)
```

selects all data *except* visibilities between 0 and 10 kilowavelengths.

Several subcommands can be combined on the same line, separated by commas. When combining several subcommands of *different* types, the visibility must be selected by all the subcommands to be accepted (a logical AND). When combining several subcommands of the same type, then the visibility is accepted if it is selected after sequentially examining each of the subcommands (a logical OR). For example:

```
select=uvrange(0,10),uvrange(20,30)
```

selects data with a $u-v$ radius in the intervals 0 to 10 kilowavelengths as well as 20 to 30 kilowavelengths. The following uses a 'select then discard' approach to selecting the same $u-v$ ranges as above:

```
select=uvrange(0,30),-uvrange(10,20)
```

The following selects the same $u-v$ ranges, but only for the baseline between antennas 1 and 3.

```
select=uvrange(0,10),uvrange(20,30),antennae(1)(3)
```

The following selects all baselines, with the exception of 1-2, 5-7 and 6-7:

```
select=-antennae(1)(2),-antennae(5,6)(7)
```

Another way of combining subcommands, is with the `or` subcommand. This allows you to OR together two 'clauses' of selection commands. For example, to select spectral windows 1 and 2 for the times 0:00 to 1:00, and spectral windows 4 and 5 for times 2:00 to 3:00, use:

```
select=time(0:00,1:00),window(1,2),or
       time(2:00,3:00),window(4,5)
```

By combining the various subcommands and the `or` subcommand, quite complex selection criteria can be developed. For complex selections, an `@` file (as described in Section 2.5) may be preferred.

For example, consider a file, `select.data`, containing the text:

```
time(1:00,2:00),window(1,2),or
time(2:00,3:00),window(1,2),-uvrange(50,100),-antennae(1)(3),or
time(3:00,4:00),window(1,2,3,4)
```

Then

```
select=@select.data
```

will use windows 1 and 2 for time 1:00 to 2:00, and windows 1, 2, 3 and 4 for times 3:00 to 4:00. For time 2:00 to 3:00, it uses windows 1 and 2, but omits data for baseline 1-3 which has a $u-v$ range of 50 to 100 kilowavelengths.

There are a few limitations on the use of `amplitude`, `polarization` and `window` selection. Some of these limitations are mentioned in following sections, whereas others are not. Few of these limitations affect normal practice, and the selection software will inform you if there is an problem.

5.6 Spectral Window Selection

A $\mathcal{MIRIAD}$ visibility dataset can contain simultaneous observations at multiple spectral bands, or spectral windows. These are called 'IF-bands' in $\mathcal{AIPS}$. When there are multiple spectral windows, channels are still numbered from 1 to N, where channel 1 is the first channel of the first spectral window, and channel N is the last channel of the last spectral window. That is, channel numbering takes no regard of spectral window boundaries.

When there are multiple spectral windows, it is often convenient to be able to select based on the spectral window, rather than a channel range. This is achieved with `select=window`, which selects data from a set of spectral windows. The general form is

`select=window(`*w1,w2,...*`)`

where *w1, w2,* etc, are spectral window numbers. For example, selecting the data from spectral windows 1, 2 and 3 would be achieved with

```
select=window(1,2,3)
```

Spectral window selection and `velocity/felocity` linetypes are quite complementary. The different spectral windows might correspond to different spectral lines with different rest frequencies, and so it is desirable to select only a subset of spectral windows (those corresponding to a given spectral line) in forming the output velocity channels.

Spectral window selection is not particularly complementary to `channel` linetype processing – both select a range of input channels. When `line=channel` and `select=window` are used together, the total apparent number of input channels is equal to the number of channels in the selected spectral windows, and the *start* channel number of the `line` parameter is relative to the first selected spectral window. For example, if there are multiple spectral windows, then

```
select=window(2)
line=channel,1
```

would select just the first channel of the second spectral window. Similarly

```
select=window(2)
line=channel,10,8,1,2
```

would select 10 channels, being every second channel starting at the 8th channel of the second spectral window.

There are some restrictions on the use of spectral window selection.

- For `channel` linetype, the selected spectral windows must be contiguous. For example, you cannot select just windows 1 and 3. This restriction is relaxed by some tasks, most notably `uvcat`.

- For `channel` linetype, `window` selection cannot be used with `or` selection.

- For `velocity/felocity` linetype, if both `window` and `or` selections are used, then there should be a window selection within each `or` clause. Additionally only one of the `or` clauses should be satisfiable for a given visibility. For example, consider the following:

  ```
  select=time(0:00,1:00),window(1,2),or
         time(0:00,2:00),window(3,4)
  ```

  Here data in the times 0:00 to 1:00 could have been selected by either of the two clauses. There is no guarantee as to which of the two window subcommands will be used during this time interval.

5.7 Selection and Multi-Source/Multi-Frequency Datasets

Having no distinction between single- and multi-source files has the advantage that all visibility tasks can manipulate any visibility dataset. However manipulating a dataset with, say, several sources can require more care on the part of the user. For example, it rarely makes sense to make an image using data from several sources. Generally the user is given reasonable flexibility to perform whatever s/he deems appropriate – but this has some disadvantages. For example, consider determining calibration for an observation where the calibrator (a point source) and the program source are within the one dataset. In deriving the antenna gains, you would want to select only the data from the calibrator, whereas when imaging you would select only the data from the program source. The visibility selection mechanism (see Section 5.5) handles this situation. But to forget to select the appropriate data (calibrating with data including the program source, or imaging including the calibrator) would result in a mess.

Thus, for example, if the calibrator was $0823-500$, and the program source was `vela`, one would use

```
select=source(0823-500)
```

and

```
select=source(vela)
```

to select the appropriate source.

The selection criteria most appropriate for datasets containing multiple sources, frequencies and mosaiced pointing centres are `source, frequency, window, ra, dec, dra` and `ddec`.

Other examples of using the `select` keyword are given below:

SELECT	
select=window(2)	Select only data from the 2nd IF
select=freq(4.74)	Select data where the frequency of the first channel is 4.74 GHz (±1%)
select=ra(05:30,06:00)	Select visibilities with RA between 5:50 and 6:00 hours.
select=dra(0),ddec(0)	Select data from only the central pointing (delta RA and DEC of 0) of a mosaiced observation.
select=source(vela)	Take data for source vela only
select=window(2),freq(4.74)	Select data from the 2nd IF where its frequency is 4.74 GHz.

5.8 Polarization/Stokes Handling (stokes and select)

The visibility datasets possibly contain correlations for different polarizations and Stokes parameters. There are two basic ways that polarization characteristics can be measured: firstly, two orthogonal feeds can be present on the antennas, and the four different polarization correlations from a baseline can be taken and recorded. Alternatively only one feed may be present, which can be rotated, or there are not enough correlators to calculate all four polarization correlations simultaneously. The first approach, used by the ATCA, allows all polarization parameters to be measured simultaneously. The second forces a 'time-sharing' approach, where the feed or correlator must be switched between measuring one polarization to another. With the four simultaneous measurements, it is possible to convert a visibility from raw polarization parameter to a Stokes parameter. With the time-shared mode, this is not directly possible. These two different scenarios mean that two very different suites of software are needed to obtain Stokes parameters.

In $\mathcal{MIRIAD}$, each different polarization/Stokes measurement is treated as a separate visibility (this differs from the $\mathcal{AIPS}$ approach – which does not really support the time-shared approach to polarization measurement). So if a baseline measures four simultaneous polarization correlations, then four 'visibilities' will be produced for this baseline, per integration interval. In this case, each visibility will be tagged (with a $u-v$ variable `pol`) to indicate the polarization type.

There are two ways the user can determine which polarizations he or she wants to process, either with the `select` parameter, or with the `stokes` parameter.

The `select` approach selects visibilities purely on the basis of their polarization or Stokes parameter. It is a normal part of the $u-v$ selection process, as described in the previous section. The `select` mechanism cannot convert from raw polarization parameters to Stokes parameters. It just selects visibilities, in the file, based on their polarization. The general form is:

`select=polarization(`*a*,*b*,*c*,...`)`

where *a*,*b*,*c* etc can be one of the mnemonics

- For Stokes parameters: `i, q, u, v`
- For circular polarization parameters: `rr, ll, rl, lr`
- For linear polarization parameters: `xx, yy, xy and yx`

Using the `stokes` approach is superficially similar, in that it causes only certain polarization/Stokes parameters to be processed. However, with the `stokes` approach, the software can perform conversion between raw polarizations and Stokes parameters if required. It also insists that all the requested polarization/Stokes parameters are calculable, at a given time, before it will allow any of them to be processed. For example, if the user requests Stokes I and V, and the dataset contains linear polarization data, then all four of `xx, yy, xy` and `yx` polarizations must be present for I and V to be returned.

The general form of the `stokes` approach is:

`stokes=`*a,b,c,...*

where *a,b,c* etc, can be one of the mnemonics given above (`i, q, u, v, rr, ll` etc).

The `stokes` can also take the values `ii, qq` and `uu`:

- Mnemonic `ii` returns Stokes-I *given the assumption that the source is unpolarised*. For example *MIRIAD* would not return any data if you requested `stokes=i` and the dataset contains only, say, `xx` (`xx` is not Stokes-I for a polarised source). How if you requested `stokes=ii` in this situation, the `xx` data would be passed through, because you have told *MIRIAD* that the source is unpolarised. Many tasks use `stokes=ii` as their default.
- Mnemonic `qq` and `uu` returns Stokes-Q and Stokes-U *given the assumption that the parallactic angle is 0*. These quantities are rarely of astronomical interest, but they are useful to investigate some instrumental effects.

The `select` mechanism is usually used for time-shared polarization measurements, whereas the `stokes` mechanism is usually used where there are simultaneous measurements. However this is not a hard and fast rule. The two approaches generally cannot be used at the same time. Some tasks will prohibit the use of the `select` approach altogether (they will give error messages if you attempt to), whereas others are more lenient. The rules which determine whether you can or cannot use the `select` approach are quite arcane and can be file dependent. Check the documentation for each task, especially if both `select` and `stokes` parameters are present. If `polarization` selection is not allowed, then `increment` selection will also be prohibited (due to some arcane quirk).

Chapter 6

Image Data Concepts

6.1 Image Datasets

Most of the items which are used to build up an image follow a FITS-like convention. For example the number of axes in an image is given by the `naxis` item, whereas the number of pixels along each axis is given by item `naxis`*i*. Three exceptions are the `image`, `mask` and `history` item. The `image` item contains the pixel data, stored as floating point numbers. The `mask` item, if present, contains a bit-mask used for pixel blanking – *MIRIAD* does not use magic value blanking like *AIPS*. Finally the `history` item contains a text file describing the history of the dataset.

A complete list of the items that are potentially contained within an image dataset is given in Appendix B.

6.2 Image Coordinate Systems

MIRIAD correctly handles a variety of image coordinate systems (the mapping between pixel and physical coordinates) in a correct fashion. The coordinates (and their description within the dataset) are treated in a fashion similar to *AIPS* (see Eric Greisen's *AIPS* memo No. 27, "*Non-linear coordinate systems in AIPS*").

6.3 Image Region of Interest (region)

Most image-related tasks can process a subset of the pixels in an input image. Depending on the task, the selected pixels may either be a fairly arbitrary region, or only a regular subimage of the input image.

The task parameter, `region`, which gives the region-of-interest consists of a combination of subcommands. Each subcommand specifies either a subregion or the units of the coordinates used in subsequent subcommands.

The subregions selected by multiple subcommands are effectively 'OR-ed' together to form the overall region. That is, the overall region selected is the 'union' (not intersection) of the subregions.

For comparatively simple regions, combining subcommands is quite adequate. However for complex regions, a cursor-based program, `cgcurs`, may be the most convenient for generating the subcommands.

Region specification is composed of one or more of the following subcommands. Each subcommand can be abbreviated to uniqueness, and subcommands are separated by a comma.

`images(`*z1,z2*`)` This selects the image planes *z1* to *z2* inclusive. *z2* is optional, defaulting to the same value as *z1*.

`quarter(`*z1,z2*`)` This is somewhat like the `images` command, except that it selects only the central quarter of each plane. Both *z1* and *z2* are optional.

`box(`*xmin,ymin,xmax,ymax*`)(`*z1,z2*`)` This subcommand selects the pixels within a box whose corners are *xmin, ymin,xmax* and *ymax*. *z1* and *z2* are optional, and are the same as in the `image` subcommand. If the `(`*z1,z2*`)` part is missing, a default is used (generally all planes are selected).

`polygon(`*x0,y0,x1,y1,x2,y2,...*`)(`*z1,z2*`)` This gives the vertices of a polygon.

> `box(`*xmin,ymin,xmax,ymax*`)`

is equivalent to

> `poly(`*xmin,ymin,xmax,ymin,xmax,ymax,xmin,ymax*`).`

z1 and *z2* are the same as with the `images` and `boxes` subcommands.

`mask(`*name*`)` This selects pixels according to the mask given by the mask item in the dataset *name*.

The units used for the coordinates are controlled by the following subcommands:

`abspixel` Subsequent coordinates are given as absolute pixel values (`i.e.` values ranging from 1 to $NAXISi$ - see `prthd` or `itemize`). This affects image coordinates and the coordinates along the third dimension. This is also the default.

`relpixel` Subsequent image coordinates are relative to the reference pixel, as defined by the header of the map of interest. Use `prthd` to see the reference pixel is (*CRPIXi*).

`relcenter` Subsequent image coordinates are relative to the central pixel of the image. This is somewhat like the `relpixel` command, but used the image centre, rather than the reference pixel.

`arcsec` Subsequent image coordinates are given in arcseconds, and are relative to the reference pixel of the map of interest.

`kms` Subsequent coordinates in the third dimension are given in km s^{-1}.

For example, to specify a 21×21 region, centred on the reference pixel, use:

```
region=relpix,box(-10,-10,10,10)
```

or to give a 10×10 region in the lower left corner of the image, use

```
region=box(1,1,10,10)
```

If there are multiple maps in the file, use

```
region=box(1,1,10,10)(1,2)
```

to select the first 2 maps.

The region-of-interest specifications can become rather involved, when complex regions are used. As with visibility data selection, `@` files (see Section 2.5) are a convenient way to store these.

There are some warnings for those accustomed to the `select` visibility data selection method:

- The subcommands of a region of interest are logically OR'ed together, whereas in the visibility data selection it is a logically AND. For $u - v$ `select` the logical OR must be explicitly given by use of the `or` subcommand.
- The subcommands do not allow a plus or minus prefix, like the visibility data selection does.

Chapter 7

Reduction Strategies

Here we give an overview of the reduction process, and review some of the basic decisions that you will have to make during the reduction of the data. We now consider some questions that you should ask yourself before the reduction process.

- *Is this observation more than a single-pointing, continuum experiment?* Hopefully the answer to this is obvious! This manual contains special chapters addressing the reduction of spectral line (Chapter 16), mosaic (Chapter 21) and pulsar-bin mode (Chapter 24) observations. You are encouraged to review appropriate chapters *before* you start reducing your data.

- *Will I want to deconvolve my images?* Deconvolution is the process of removing artifacts due to the incomplete sampling in the $u-v$ plane. You will want to deconvolve for observations where the source is stronger than a few times the noise limit. That is, unless you are doing a detection experiment, you are likely to want to deconvolve. Deconvolution is addressed in Chapter 14.

- *Will I want to self-calibrate the data?* Self-calibration is the process of determining the antenna-based gain function from the source itself. For this to be possible, your signal needs to be about 5 to 10 times stronger than the thermal noise when integrated over the self-calibration solution interval (typically 15 seconds to 5 minutes). For the old ATCA correlator in continuum mode, this means a source which contains at least 100 to 200 mJy on most baselines. For the CABB correlator, self-calibration is possible with a 15 second solution interval using sources with as little as 10 mJy of flux density on most baselines. Self-calibration is discussed in Chapter 15.

- *As I have a continuum experiment, do I want to use multi-frequency techniques?* Even in continuum mode, the ATCA produces multiple channels of data. As the fractional bandwidth can be quite significant, you may be making a significant approximation if you average all these channels into a single channel (the so-called 'channel-0' dataset). The result will be poorer $u-v$ coverage and bandwidth smearing. If you are interested in high dynamic range imaging, or if a good beam is important, and you are observing at 21, 13 and possibly 6 cm with the old correlator, or at any cm wavelength with CABB (and even at 12mm), then it is best *not* to average your data into a single channel. Rather you can calibrate and form a single image directly using the multichannel data. This practise is known as multi-frequency synthesis.

  Multi-frequency synthesis can be taken a few steps further with the ATCA. As two IFs can be measured simultaneously, these two IFs can be combined in the imaging stage to further enhance both sensitivity and $u-v$ coverage. As the ATCA can frequency switch rapidly, it is possible to time share between different frequency settings. This involves a trade-off between tangential and radial $u-v$ gaps.

- *Should I average my data in time or frequency?* The reason to average your data is to reduce your disk consumption and to increase the speed of the various processing programs. If these practical considerations are important, you may consider averaging your data in time and frequency. In this case, you probably want to do this as early as possible in the reduction process – after the initial flagging.

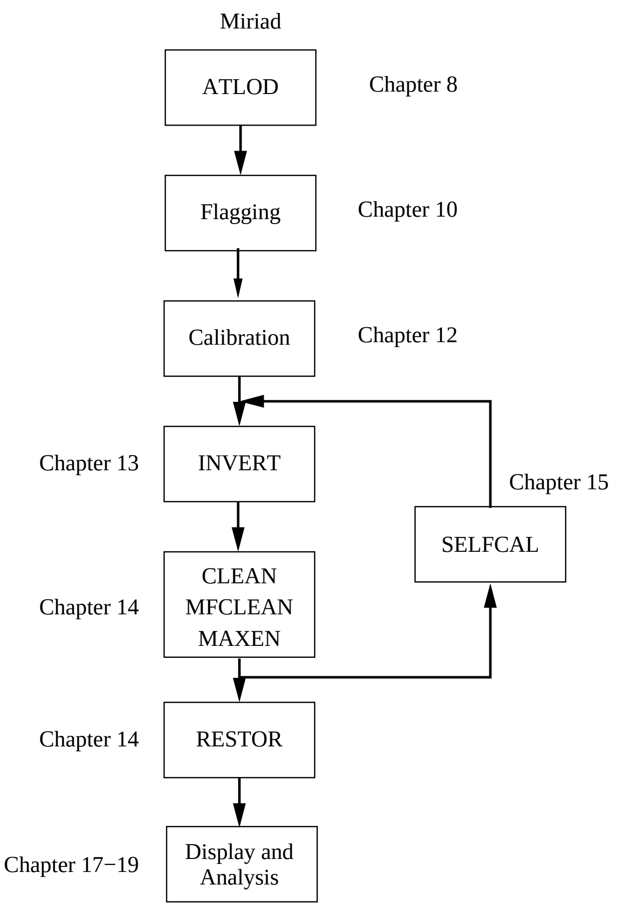

Figure 7.1: ATCA Data Reduction Strategy

Frequency averaging is generally only applicable for continuum experiments and only if you are *not* going to be doing multi-frequency synthesis. You might also think twice about averaging if interference was a problem during the observation. You may have to go back and do some more flagging later.

Averaging in time can be performed when observing with short arrays (i.e. when the 6 km antenna is not used). A conservative rule is that you should not average longer than $90/d$ seconds, where d is the array length in km. If you observe with a short array and are not interested in long baselines for the program source, you probably still want to use antenna 6 when determining the initial calibration. Despite this, d is the array length of interest for the program source, not the calibrators. Nor should you average for longer than the antenna-gain solution interval (unless you have finished calibrating, and are not going to self-calibrate). If you have a strong source, and the phase stability is poor, you should not average in time.

Both averaging in frequency and time are performed by `uvaver` (see Section 10.5).

Figure 7.1 gives a flow chart of the normal steps involved in reduction of ATCA data. Note that this is somewhat abbreviated by necessity – additional flow-charts in subsequent chapters give more detail and describe some of the variations.

We will now consider each step in the flow-chart in turn.

1. Data from the ATCA is written in "RPFITS" format – an ATNF-specific format – and a special task (`atlod`) is needed to read it. Loading your data into *MIRIAD* is described in Chapter 8.

2. Data editing (flagging) can be very time consuming, especially if you do are affected by interference. The *MIRIAD* flagging tasks are described in Chapter 10.

3. The actual calibration steps can be the most confusing steps, as there are a large variety of paths that can be followed. Only the most frequently trodden paths are described in Chapter 12.

4. The task `invert` takes a visibility dataset and forms either a single image or an image cube (for spectral-line observations). It also produces a point-spread function (dirty beam), ready for deconvolution. See Chapter 13

5. The main deconvolution tasks are `clean`, `maxen` and `mfclean`. Task `clean`, which is the most commonly used, attempts to decompose your image into a number of delta functions. It can be slow for images with a large amount of extended emission. An alternative to `clean` is `maxen`, a maximum entropy-based deconvolution task. It tends to be less robust and more difficult to run correctly. A third alternative is `mfclean`. This is a derivative of the CLEAN algorithm, and simultaneously determines a flux density and spectral index image. It is only appropriate for multi-frequency synthesis experiments when more than one observing band has been used, or for CABB data that has a large fractional bandwidth.

   The deconvolution tasks are described in Chapter 14.

6. As noted above, self-calibration is useful for determining antenna gains directly from a strong program source. It is described in Chapter 15.

7. The deconvolution tasks produce an output image that is in units of flux density per pixel. That is, the outputs are CLEAN component images. Task `restor` converts these to flux density per CLEAN beam, and adds back the residuals. This is covered in Chapter 14.

8. At last, you are ready to display and think about your images!

Chapter 8

Getting Data In and Out of *MIRIAD*

8.1 ATLOD: Reading RPFITS Files

ATCA data will be initially in RPFITS format. This needs to be converted to *MIRIAD*'s format, by task `atlod`, before any further processing can be done. We will discuss the various input parameters to *MIRIAD* `atlod`.

- The `in` parameter gives the name of the input UNIX file, or files, to load. Many files can be given, and wildcards are supported.

  Almost all ATCA RPFITS files can now be downloaded directly from the AT Online Archive (ATOA). Any requested RPFITS files can be saved onto your local disk, and `atlod` can be directed to load this data from there.

  RPFITS files from the old correlator are commonly archived on CDROM or DVD: a CDROM/DVD appears much like a disk to the operating system, and `atlod` can load RPFITS files directly off these. To use a CDROM/DVD drive, first check that the drive is not in use (an orange light flashing). To load a disc, press the Open/Close on the front of the drive, place the disc in and again press the Open/Close button. After about 10 seconds, the operating system will mount the CDROM/DVD, and it can be accessed. CDROM/DVDs written at the ATCA will have the RPFITS data in the directory

  ```
  /media/cdrom/DATA
  ```

  You can do a directory listing to see what you have – its like a normal file system!

  You can set the `in` parameter to the RPFITS files in this directory. For example, to select all files belonging to project C561, use

  ```
  in=/media/cdrom/DATA/*.C561
  ```

  When you are done with a CDROM/DVD drive you eject it with the command

  ```
  eject cdrom
  ```

  (the Open/Close button will not work).

  RPFITS files are (or at least were once) commonly written to exabyte tapes. Unlike other *MIRIAD* tasks, `atlod` can read exabytes directly. In this case you should use the physical device name as the input name. This is usually displayed on the exabyte drive. On the Epping Solaris systems, the device name is usually `/dev/rmt/0lbn` or similar (this is the so-called non-rewinding, raw, BSD compatibility interface). On older Solaris systems, it may be `/dev/nrst4` or something similar (non-rewinding, raw interface).

An alternative to directly reading from exabyte is to load the data to disk with the Unix commands `ansiread` or `ansitape`, which are available on a number of the ATNF systems. See the UNIX `man` pages for more information.

- The `out` parameter gives the name of the output MIRIAD visibility dataset. There is no default.
- The `ifsel` parameter makes `atlod` extract only the IFs that are selected here from the RPFITS files. The IFs to extract should be specified as a comma-separated list. This can be very useful if you are interested only in a specific frequency, or if the data has two or more IFs with different numbers of channels that would normally need to be read in with `options=noif`.
- The `options` parameter gives miscellaneous processing options. Several values can be given, separated by commas.

  Some of the options described below control on-the-fly corrections that `atlod` makes to the data, while others can reduce the amount of disk space that the dataset will consume. It is very important therefore to pass the correct options to `atlod`, to save yourself time later doing corrections that could have been done more easily right at the start.

  At frequencies above 10 GHz, the opacity of the atmosphere can be significant: in mediocre conditions, the signal loss can be 20% at low elevations. With high-frequency data, an atmospheric opacity correction should be applied to the data; `atlod` will perform this correction if it is passed the `opcorr` option. This option can be safely included for data of any wavelength, as it does very little (and certainly does not harm the data) at low frequencies, and is completely ignored at 3mm wavelengths, where it is not appropriate.

  For high frequency data taken with the old ATCA correlator, it is also generally desirable to load both observing bands together, to allow joint calibration. A shortcoming of MIRIAD is that it is poor at handling multiple observing bands when the polarisation products differ between the bands. However the old ATCA correlator configurations that combined narrowband and wideband observations generally did measure different polarisation products in the two bands. As polarimetry with the ATCA 3-mm system was not possible until recently, the simplest approach to ensuring the polarisation products of the two bands are the same is by discarding polarisation products that are not needed. The `atlod` option `nopol` causes `atlod` to discard any XY and YX products.

  A few sets of permutations of the options will be appropriate with most observations. These include:

  - For CABB data: `options=birdie,rfiflag,xycorr,opcorr`
  - For old-correlator continuum: `options=birdie,xycorr` or `options=birdie,reweight,xycorr`. See below for a discussion on whether to use `reweight` or not.
  - For old-correlator non-polarimetric line: `options=birdie` or `options=birdie,compress` or `options=birdie,hanning` or `options=birdie,compress,hanning`. See Section 16.3 for more on using `atlod` with spectral line data and Section 24.2 for high time resolution bin-mode data.
  - For old-correlator 3mm data: `options=birdie,nopol,reweight`

  Possible options are:

`birdie:` The old ATCA correlator suffered from self-interference at frequencies that were a multiple of 128 MHz, while the CABB correlator displays self-interference spikes at frequencies related to the 640 MHz sampler clock. The birdie option flags out the channels affected by this self-interference. This option is strongly recommended.

Additionally, in the 33 channel/128 MHz mode, the birdie option also discards some edge channels and every other channel. This operation does not incur a sensitivity penalty, as correlator channels are not independent in this mode. The net result is that the output consists of either 13 or 14 good channels.

For CABB data, this option also flags 100 channels on each edge of the band, and any part of the 20cm or 13cm band that is not within the recognised frequency range of the receiver.

`xycorr:` In polarimetric correlator configurations, the ATCA makes an on-line measurement of the phase difference between the X and Y channels. Although it is generally only a few degrees (assuming the observation was correctly set-up), this phase difference should be corrected when doing polarimetric work. Since a hardware upgrade in November 1992, the recommendation has been that the on-line measurements should be applied, without averaging, using the `xycorr` option in `atlod`. For data prior to November 1992 – seek expert advise.

`reweight:` In the 33 channel/128 MHz mode, a Gibbs phenomena (see Albert Bos in the *Indirect Imaging* proceedings) affects the ATCA. This introduces a non-closing error into the data. The effect is moderately subtle, and is not significant for dynamic ranges of less than about 500. The 'reweight' option reweights the visibility spectrum in the lag domain to eliminate this problem.

This option is recommended for high dynamic range work (more than about 500). However, the option also reduces the effectiveness of the birdie option to reject self-interference. This can be a significant effect for 20cm continuum observations, where the usual observing band straddles the 1408 MHz birdie. Particularly at 20cm, the reweight option is *not recommended* where dynamic ranges of less than 500 are expected.

This option is *not* required for CABB data.

`compress:` Normally the correlation data are stored as 32-bit floating point numbers. Alternatively, the `compress` switch can be used to instruct `atlod` to store the correlations as 16-bit integers, with a scale factor associated with each spectrum. This approximately halves the disk space occupied by a dataset, and so may be very advantageous for large spectral-line observations.

`hanning:` Hanning smooth the data, and discard every second channel. This can be useful for spectral line experiments. Task `atlod` will refuse to Hanning smooth data with 33 channels or less.

This is unlikely to be beneficial for CABB data, as each CABB channel is independent, and boxcar smoothing (or channel binning) is preferred.

`bary:` Use the barycentre (often called heliocentre and usually used for extra-galactic work) as the rest frame when calculating velocity information. The default is to compute velocity information relative to the LSR frame (which is usually used for Galactic work).

`hires` This option may be appropriate for observations using the ATCA's high time resolution bin mode. See Chapter 24 for more information.

`opcorr:` This option corrects the visibility data for atmospheric opacity. It is since October 2003 that there is sufficient information in the visibility dataset to allow this. This options *should not* be used for 3mm data. Generally it is relevant for 12mm observations only. See Sections 22.1 and 23.2.2 for more information.

`samcorr:` Correct the measured visibilities for bad sampler statistics. Normally the samplers adjust their various threshold levels to ensure optimal operation. However various transients can cause the sampler levels to be wrong. All is not lost – you can correct the data for the imperfect sampler levels after the fact. Use the `samcorr` (sampler-correct) option for this, and is the recommended approach for data observed before December 1993. Since December 1993, sampler correction has been done automatically on-line. For this data, the `samcorr` option is (quietly) ignored.

`relax:` This option is generally not recommended. Normally `atlod` flags data if it has not previously read the appropriate calibration record or if the information in the calibration record suggest the data are bad. Although calibration records generally precede the data, there are some instances when this is not the case, and the data are still good. In this case, the `relax` option causes `atlod` to be more lenient, and not flag the data.

`unflag:` Normally `atlod` will discard a visibility record (i.e. not write it to disk) if all the data in it are flagged bad. Flagging usually indicates that there was significant reason to suspect the data – for example the telescope was not on source or was off-line, etc. The `unflag` option causes all data to be saved to disk, although flagged data is still marked as such. This option is generally not recommended, although the disk-space penalty for using it is often modest.

`noauto` and `nocross` cause `atlod` to discard any autocorrelation or cross-correlation data respectively. It's fairly unlikely that you will want to discard the cross-correlation data!

`noif:` When there are multiple frequency bands being observed simultaneously, *MIRIAD* normally maps the frequency bands to its spectral windows (IF axis in *AIPS* terminology), and writes them out in a single record. Alternatively, using the `noif` switch causes `atlod` to write the simultaneous frequencies as sequential records, making it appear somewhat like a frequency-switch. You will need to use the `noif` option if you have two spectral windows which sample different polarization parameters, or that have different numbers of channels.

`nopflag`: If at least one polarization of a set of 2 or 4 polarimetric spectra are bad, `atlod` normally flags all of the polarizations. With this option only the nominally bad spectrum is flagged.

`caldata`: `atlod` will usually mark as bad any scans that are labelled as a reference pointing calibration or a paddle measurement. Normally, this data is of little use as corrections from these scans are applied on-line. However, occasionally, the data in these scans may be of use, and including this option causes `atlod` to keep this data instead of discarding it, although it will still be flagged as bad.

`rfiflag`: At low frequencies, the large bandwidth of the CABB correlator makes it impossible to avoid most of the known RFI. Including this option causes `atlod` to automatically flag regions of known interference as bad. It is also possible to specify flagging regions manually using the file `rfiflag.txt` in the current directory, or the default version in the `MIRCAT` directory. Each line in this file should have two numbers representing the lower and upper frequencies to flag in MHz.

- `nscans` gives two numbers which are the number of scans to skip over (before processing), and the number of scans to save. The default is to save all scans.
- `nfiles`, like `nscans`, gives two numbers, which are the number of files to skip and process. This parameter is only really useful when reading from exabyte. The default is to process the first file only. Note, however, that the mechanism used to skip files is rather inefficient. If you wish to skip a large number of files, you should do this with the Unix command `mt`. A complication is that every RPFITS file appears as three files to `mt` – so you will want to skip three times as many tape files as RPFITS files. For example, to skip 10 RPFITS files, you would use the Unix command

```
mt -f /dev/nrst4 fsf 30
```

  See the Unix `man` page on `mt` for more information.

Typical inputs for `atlod` are given below.

ATLOD	
in=/dev/nrst0	Input is either exabyte name
in=2010-02-04$_1$844.C007	or RPFITS file.
out=0823.uv	Output visibility data set.
options=birdie,rfiflag,xycorr,opcorr	Normal CABB data options, or
options=birdie,reweight,xycorr	Normal old-correlator continuum mode options, or
options=birdie,hanning,compress	possible old-correlator spectral line options.
nfiles=0,3	Skip 0, then read 3 files (use only for tape reading).
nscans	Unset to save all scans

MIRIAD's `atlod` saves a number of on-line measurements as visibility variables. These measurements may by helpful in analysing and flagging the data. They can be plotted and listed with task `varplt` (see Section 10.7). These on-line parameter, and its MIRIAD variable name, are described below.

`xyphase`: This is the on-line measurement of the XY phase. It is important to examine this measurement if you are doing polarimetry.

`xyamp`: This is the amplitude of the correlation between the X and Y polarization channel of a given antenna.

`xsampler, ysampler`: These variables give the sampler statistics for the X and Y polarization channels. There are three numbers per antenna per IF, which reflect the sampler levels. They should have values of 17.3, 50 and 17.3. Thus for six antennas, the `xsampler` variable will consist of 18 numbers per IF. These values are not available for CABB data.

`xtsys, ytsys`: These give the system temperature, in Kelvin, for the X and Y polarization channels. Unfortunately *MIRIAD* tasks were originally developed around a single polarization model of a telescope, and so most tasks that concern themselves with system temperature do not handle dual polarization systems. The partial workaround used in *MIRIAD* has been to store the geometric mean of the X and Y system temperatures in the variable `systemp`. It is this variable that *MIRIAD* tasks use when they need to use the system temperature value.

`axisrms, axismax`: These give the antenna rms and maximum tracking errors in a particular cycle. The units are arcseconds.

`airtemp,pressmb,relhumid,wind,winddir`: These give various meteorological measurements at the observatory as a function of time.

`smonrms`: This gives a measure indicative of the seeing conditions at mm wavelengths, and is obtained by a two-element interferometer observing a communications satellite at 30 GHz. This number is the RMS of the difference between the expected path length difference and that observed, and is due to the atmospheric conditions above the Narrabri site. It may not however directly represent the atmosphere along the line-of-sight to the object currently being observed. A smaller number here means a more stable atmosphere, and better conditions for mm observing.

8.2 FITS Tapes and *MIRIAD*

MIRIAD has a single task, `fits`, to read and write FITS files – both visibility and image data. One significant shortcoming of `fits` is that it cannot work directly from a tape device – the FITS file *must* be on disk. The *MIRIAD* utility, `tpcp`, is needed to copy FITS files between disk and tape, whereas the UNIX command `mt` might be needed to position the tape. See the appropriate *MIRIAD* help or UNIX `man` pages on these commands. However, to copy from tape to disk, one would use

% `tpcp` *device fits-file*

Here *device* is the tape device. Typically it is something like `/dev/nrst0`. Note that you should always use the "raw, non-rewinding devices" – that is devices whose names start with `nr`. The output is *fits-file*.

To write a FITS file to tape, use

% `tpcp -b 2880` *fits-file device*

This causes an unblocked FITS file to be written (record length of 2880 bytes).

To move the tape to the end-of-tape, use

% `mt -f` *device* `eom`

To skip forward n files, use

% `mt -f` *device* `fsf` n

and to rewind, use

% `mt -f` *device* `rewind`

8.3 Reading Visibility FITS Files

For visibility data, the current version of `fits` can cope with both single- and multi-source files, with single or multiple antenna configurations (earlier versions of `fits` were more limited). Flagging (FG) tables are copied (if present), but the flagging specified by these tables *is not performed*. An additional MIRIAD task, `fgflag` (see Section 10.3), must be invoked to apply the flagging tables. Task `fits` does not read or use AIPS calibration tables (SN, CL, BP, and BL). Consequently if you want to preserve your AIPS calibration, you will need to apply the relevant tables with AIPS `SPLIT` before writing the FITS file.

`fits` can handle linearly and circularly polarized correlations, as well as correlations that have been converted to Stokes parameters. In addition, it knows that the ATCA produces linearly polarized data. Because AIPS does not handle linear polarizations well, it is common practice to label the linear polarizations of ATCA data as if they were circular. If `fits` finds data from the ATCA which is nominally circularly polarized, it issues a warning, and relabels the correlations as linearly polarized.

Alternately, you could correct the polarization labelling explicitly within AIPS by changing the reference value of the Stokes axis to -5. Do this with `PUTHEAD`. It is necessary to do this after `SPLIT` because it puts them back to circulars !

AIPS/PUTHEAD	
keyword = 'crval2'	Select the Stokes axis
keyvalue=-5	Change to -5 (XX)

By default the AIPS task to write FITS files, `FITTP`, writes to the directory `/DATA/FITS`. This directory has a time-destroy limit of 1 day. Alternately you can set up an environment variable to direct `FITTP` to write the FITS file to any directory. For example, after setting the environment variable `MYAREA` with the UNIX command

```
% setenv MYAREA /DATA/ATLAS_1/miriad/rsault
```

you can direct `FITTP` to write into this directory, by setting the `OUTFILE` adverb appropriately. For example:

```
> OUTFILE = 'MYAREA:MYUV.FITS'
```

will direct `FITTP` to write a FITS file `MYUV.FITS` in `/DATA/ATLAS_1/miriad/rsault`.

A quick way to set an environment variable to the current working directory is with the `pwd` command:

```
% setenv MYAREA `pwd`
```

Note that the environment variable name *must* be in upper case, though the directory name itself need not be.

MIRIAD handles only the new binary table format (`donewtab=1` in `FITTP`). Typical inputs to AIPS `FITTP` are:

AIPS/FITTP	
doall=-1	One file at a time
outfile='myarea:0823.fits'	Specify area and file name or
outfile='0823.fits'	default to `/DATA/FITS` directory
dostokes=-1	No conversions
dotable=1	MIRIAD understands FITS tables
dotwo=-1	Single precision adequate
donewtab=1	Only the new tables can be handled
format=3	Floating point FITS. Values of 1 or 2 also OK

Using `format=1` (16-bit integers) for visibility datasets with many channels will result in an output MIRIAD dataset in compressed format. It also produces a much smaller FITS file – which may be important if disk space is low. Note that FITS data compression is different from MIRIAD and AIPS data compression. FITS compression uses one scale factor for the *entire* dataset, whereas *MIRIAD* and *AIPS* compression uses one scale factor per visibility record.

In most cases converting the FITS file to *MIRIAD* format is straightforward. Apart from the `op` parameter, which indicates the operation to be performed by `fits` (`op=uvin` is used to read in a visibility FITS file), you need only supply the names of an input FITS and output *MIRIAD* dataset. Note that *MIRIAD* follows the case conventions of the host operating system. Thus, under Unix, *MIRIAD* is case sensitive, and so you must give the FITS file name in the correct case. As it has most likely come from *AIPS*, the file name will usually be in upper case.

For spectral line observations, the velocity reference frame used in *MIRIAD* (and some other velocity information) must be set when the visibility dataset is created. The default is to extract the appropriate information from the FITS file. Alternatively, the `velocity` parameter can be used to override or alter the velocity information given in the FITS file. See Chapter 16 for more information.

FITS	
in=$MYAREA/0823.FITS	Re-use your environment variable
in=/DATA/FITS/0823.FITS	or give *AIPS* default directory; get case right
op=uvin	Read in visibility data
out=0823.uv	*MIRIAD* dataset name
velocity=lsr	Compute info for LSR velocities, for spectral line observations.

Leave everything else `unset`. Once `fits` has run to completion, you should see the new directory containing the 'items' describing your visibility data.

Information Missing or Incorrect in FITS Files

`fits` knows enough about the ATCA and a select group of other telescopes to provide some nominal values for parameters that FITS does not store – for example a system temperature of 50 K and a system gain of 13 Jy/K are stored with ATCA data. `fits` also attempts to estimate the integration time. Unfortunately given the information present in a FITS file, this integration time estimate is little better than an educated guess. Task `fits` gives a message informing you of the values it is using. If you know better values for system temperature, integration time, etc, than those guessed by `fits`, you may wish to correct the stored values. This is largely so that any noise estimates produced by later tasks are in more meaningful units. You can correct these with the task `puthd`. Task `puthd` writes an item into the visibility dataset, which overrides the value of a visibility variable with the same name. The following three parameter boxes show the inputs to `puthd` to correct the system temperature to 80 K, the system gain to 11 Jy/K and the integration time to 15 seconds.

PUTHD	
in=0823.uv/systemp	Parameter is dataset name followed by 'systemp'
value=80.0	Set to 80 K. Express it as a real number.

PUTHD	
in=0823.uv/jyperk	Parameter is dataset name followed by 'jyperk'
value=11.0	Set to 11 Jy/K. Express it as a real number.

PUTHD	
in=0823.uv/inttime	Parameter is dataset name followed by 'inttime'
value=15.0	Set to 15 sec. Express it as a real number.

AIPS FITS files from the ATCA sometimes contain other flaws which *MIRIAD* issues warnings about – these can usually be safely ignored. These flaws include visibilities with zero weight, incorrect apparent RA and DEC and circular polarizations from the ATCA.

AIPS Frequency Bugs

There are a number of bugs in *AIPS* which result in inconsistent or incorrect frequency information in the FITS file. These frequency bugs tend to become apparent when using two simultaneous IFs, particularly when the IFs are in different sidebands, or if one of the IFs is discarded at some stage. After reading with `fits`, it is *strongly encouraged* that you examine the frequencies that *MIRIAD* determines from the FITS file. Indeed it would be best if you checked these before writing a FITS file in *AIPS*, though this is more difficult because the frequency information is spread rather widely in *AIPS*. In *MIRIAD*, the easiest way to examine the frequency information is with task `uvindex` (see Section 10.4).

Errors in the FITS file frequency definition may affect the $u-v$ coordinates, and so may affect the RA/DEC scales on any images that might be produced from faulty visibility datasets.

One fairly innocuous *AIPS* frequency bug is that the bandwidth of 'channel 0' datasets can be wrong. Leaving this uncorrected will invalidate error estimates generated by *MIRIAD* tasks – which is not a major problem if you ignore these anyway!

Another, more serious, problem is that two locations in the FITS file give the "reference frequency" – the header and the antenna table. In principle these should agree. However there are instances when they will differ. Task `fits` will issue a warning when this happens, and will use the antenna table reference frequency as the true reference frequency. In some cases this will be the correct frequency. In others it will not.

There are two approaches to fixing the errors in the frequency definitions – either correct them in *MIRIAD* (using `puthd` and `uvputhd`) after reading the FITS file, or in *AIPS* (using `PUTHEAD` and `TABED`) before writing the FITS file. Which you will use will depend on your familiarity with the *AIPS* and *MIRIAD* visibility dataset structures and the complexity of your dataset (how many frequencies, etc), and where the bug occurred (are the $u-v$ coordinates correct?). It is best to seek help, both to fix your data, and to aid tracking down the *AIPS* bugs.

Time and FITS Files

The time stored in a *MIRIAD* visibility dataset is nominally UT1 (although the difference between UT1 and UTC is not significant for the purposes of *MIRIAD*). As *MIRIAD* relies on correct time in a number of calculations (e.g. computing the parallactic angle needed in polarization calibration and conversion, computing velocity information in `fits` and `uvredo` and recomputing $u-v$ coordinates done by `uvedit`), task `fits` does its best to convert the time into UT1. `fits` will give you messages about any time adjustments it makes.

When a FITS file contains multiple antenna tables, the times stored in the FITS file have usually been offset from the true time by the *AIPS* task `DBCON`. Task `fits` assumes that such an offset has been added, and corrects for it.

8.4 Writing Visibility FITS Files

Task `fits` can also be used to write a visibility FITS file. In addition to the keywords used when reading visibility datasets, `fits` uses keywords `select`, `stokes` and `options` to control extra processing performed on the visibilities.

Unfortunately `fits` is more sophisticated at reading FITS files than writing – note the following caveats carefully.

- `fits` can cope with only a single pointing of a single source, with a single frequency setup and single antenna array configuration (note that, unlike earlier versions, `fits` writes an antenna table). `fits` does quite limited checks to see if these conditions are met, and quite happily will use the relevant parameters of the first record.

- `fits` assumes that the data are a single IF band, with a constant frequency increment between channels. If presented with multiple spectral windows (multiple IFs), an average frequency and increment will be used. This is *very* undesirable. If you do this, `fits` will issue warnings about frequencies deviating from linearity. These warnings should be taken seriously. To avoid this, use the `select` or `line` parameters to force `fits` to write out a single spectral window.

- `fits` does not save *MIRIAD* calibration tables. However `fits` does give you the option of applying the calibration, as it writes out the data (see the `options` keyword).

- *MIRIAD* allows you flexibility in the choice of polarization correlations that you can write out. On-the-fly polarization conversion can be performed. However *AIPS* will refuse to read in a number of the possibilities that *MIRIAD* allows. In particular, *AIPS* `UVLOD` refuses to load visibility data where there are three different sorts of polarization correlations (rather than the more usual 1, 2 or 4 sorts).

Typical inputs follow:

FITS	
in=0823.uv	Name of input *MIRIAD* dataset.
op=uvout	Write out a visibility FITS file.
out=0823.FITS	Output FITS file – probably upper case for AIPS.
select	Select appropriate data.
stokes=i,q,u,v	Select appropriate polarizations.
options	Leave unset to apply calibration,
options=nopol,nocal,nopass	or turn off all calibration.

8.5 Reading and Writing FITS Images

Task `fits` can be used to read and write FITS images. The operation for `fits` (keyword `op`) is now `xyin` (read in a FITS image) or `xyout` (write out a FITS image). For images that are partially blanked, `fits` correctly converts between the *MIRIAD*-style blanking mask, and FITS-style magic value blanking. When reading a FITS image, CLEAN component (CC) tables are ignored.

8.6 Archiving and Transporting *MIRIAD* Datasets

As *MIRIAD* datasets are written in a machine-independent format (the bits stored on disk to represent a particular value are independent of the host system, whether it be a VAX, Sun or Cray, for example), no translation is needed when transporting datasets between unlike architectures.

Instead of relying on any special tasks to write tapes for archiving or transport, *MIRIAD* relies on the facilities of the host system. Probably the most convenient archiving format is `tar`. This is universally supported on UNIX machines, and is quite common on VAX/VMS machines.

The two most common commands used with `tar` are to copy to and from tape. To copy to tape, use

`% tar -cvf` *device datasets*

where `device` is a tape device name such as `/dev/nrst0` (the non-rewinding, raw interface, i.e. device names starting with `nr`), and *datasets* is a list of datasets and files to copy to tape (If a dataset, or directory, name is given, `tar` works recursively, copying all the contents of this directory). These can include wildcards. For example, to copy all datasets and files from the current directory to `/dev/nrst0`, use

```
tar -cvf /dev/nrst0 *
```

To retrieve data from tape, use the `tar` command

```
% tar -xvf device
```

Multiple `tar` files can be stored on the one tape so that you do not overwrite old data. Again the UNIX `mt` command can be used to manipulate the tape position.

See the `man` pages on the `tar` and `mt` commands for more information.

Chapter 9

Generating Dummy Visibility Data

9.1 Making a Fake Observation

The *MIRIAD* task `uvgen` can be used to simulate a variety of observing modes. Additionally it can simulate a number of systematic and random errors and non-ideal effects that a synthesis instrument will typically suffer.

Simulating a telescope approximately, let alone in some detail, requires a good many parameters. `uvgen` takes an appreciable number of parameters and input files to describe the telescope, correlator and source. Fortunately keyword and set-up files for some common situations exist (*e.g.* simulating the ATCA and VLA). Also the defaults are usually sensible.

Some inputs to `uvgen`, and typical values for the ATCA, are given below. *Note that these are not the default values.*

UVGEN	
source=	Source description file – see help file.
ant=	Antenna location file – see help file.
baseunit=-51.02	Basic antenna increment (15m) in nanosec, for coordinates in a topocentric system.
corr=0,1,0,104	Correlator model – see help file.
spectra=	Spectral line model – see help file.
ellim=12	Telescope elevation limit.
telescop=atca	Telescope name – see help file.
stokes=xx,yy,xy,yx	ATCA measures linear polarisations.
lat=-30	Latitude of Narrabri.
radec=	Source RA and declination, in hours and degrees.
systemp=50	Typical receivers have a T_{sys} of 50 K
pbfwhm=	Primary beam size? Varies with frequency
leakage=2	Typical polarisation leakage
jyperk=12.7	System gain is about 12.7 Jy/K
pnoise=	Rms antenna phase variation
gnoise=	Rms antenna amplitude gain variation

These typical values for the ATCA, and also the VLA, can be found in in 'keyword' files, found in the `$MIRCAT` directory.

Telescope Keyword Files	
`$MIRCAT/uvgen.def.atca`	Typical `uvgen` parameters for the ATCA
`$MIRCAT/uvgen.def.vla`	Typical `uvgen` parameters for the VLA

The parameters in these files can be loaded from within the `miriad` shell using the `source` command.

For example, to load typical parameters for the ATCA, use

```
miriad% source $MIRCAT/uvgen.def.atca
```

By leaving some of `uvgen`'s parameters unset (*e.g.* leakage, systemp, pbfwhm) you get an idealised telescope (no polarisation leakage, no receiver noise, no primary beam effects). Additionally there is no reason why you cannot request an ATCA-like telescope to produce Stokes I or circular polarisations!

The source and antenna description files are described in the help for `uvgen`, as are a number of other parameters needed in the simulation process. There are, however, a collection of description files for common situations. These are all kept in the `$MIRCAT` directory.

Source Description File	
File	Description
`no.source`	Nothing! No source at all
`point.source`	A 1 Jy unpolarised point source, at field centre
`poff.source`	A point source offset from the field centre
`qpoint.source`	A 1% linearly polarised point source
`spiral.source`	A spiral

Antenna Description File	
File	Description
`h75.ant`	ATCA configuration H75
`h168.ant`	ATCA configuration H168
`h214.ant`	ATCA configuration H214
`0.122.ant`	ATCA configuration 0.122A
`ew214.ant`	ATCA configuration EW214
`ew352.ant`	ATCA configuration EW352
`ew367.ant`	ATCA configuration EW367
`0.375.ant`	ATCA configuration 0.375
`0.75a.ant`	ATCA configuration 0.75A
`0.75b.ant`	ATCA configuration 0.75B
`0.75c.ant`	ATCA configuration 0.75C
`0.75d.ant`	ATCA configuration 0.75D
`1.5a.ant`	ATCA configuration 1.5A
`1.5b.ant`	ATCA configuration 1.5B
`1.5c.ant`	ATCA configuration 1.5C
`1.5d.ant`	ATCA configuration 1.5D
`3.0a.ant`	ATCA configuration 3.0A or 6.0A
`3.0b.ant`	ATCA configuration 3.0B or 6.0B
`3.0c.ant`	ATCA configuration 3.0C or 6.0C
`3.0d.ant`	ATCA configuration 3.0D or 6.0D
`vla_a.ant`	VLA configuration A
`vla_b.ant`	VLA configuration B
`vla_c.ant`	VLA configuration C
`vla_d.ant`	VLA configuration D

Given the various standard keyword and description files kept in the `$MIRCAT` directory, it is not particularly difficult to set up the inputs to `uvgen`. For example, to simulate the 'spiral' source using configuration `3.0A`, in a continuum mode, you would give the following commands to the `miriad` front-end:

```
miriad% source $MIRCAT/uvgen.def.atca
miriad% source = $MIRCAT/spiral.source
miriad% ant = $MIRCAT/3.0a.ant
miriad% inp uvgen
```

and then set or unset other parameters as desired.

If you wish to model a particular telescope, the task `telepar` contains a database of telescope characteristics.

For models which are more complex than those that can be readily simulated with `uvgen`, one approach is to generate the visibilities are the appropriate $u-v$ tracks, and then to use the task `uvmodel` to compute the visibilities, on those tracks, of a particular image model.

9.2 Generating Visibilities from a Model Image

If you just want to generate visibilities corresponding to a particular model image, and do not care whether they follow realistic tracks in the *uv* plane, then `im2uv` may suit you. Task `im2uv` simply Fourier transforms an image (whose units should be Jy/pixel) and writes this transform into a visibility dataset. So you end up with visibilities sampled on a rectangular grid. The task attempts to produce a visibility dataset which looks reasonably valid, but the output does have some odd characteristics (e.g. all output visibilities consists of a single baseline, measured at the same time!!).

IM2UV	
model=	Name of the input model image.
out=	Name of the output visibility dataset.
region=	Only pixels within the region are used to generate the model visibilities.
select=	Normal visibility selection. Only visibilities obeying the selection criteria are stored. Only `uvrange` selection is currently honoured.

Chapter 10

Flagging, Manipulating and Examining Visibility Data

10.1 Correcting Visibilites

10.1.1 Initial "fixes" to millimetre data

There are a number of "fixes" that are important for short wavelength ATCA data, especially at the 3mm band. Note that there is no record in the data that these steps have been performed. In particular the software will allow you to apply the fixes twice, which would be just as bad as not applying them at all. The task to do these fixes, `atfix`, performs the following functions:

- By default `atfix` applies the "standard" gain/elevation model for the antennas. You can disable this by using `options=nogainel`.
- By default `atfix` corrects for a "standard" instrumental phase model. You can disable this with `options=noinst`.
- As mentioned in Section 11.1.2, the ATCA uses a paddle to measure an above atmosphere system temperature. The nature of the hardware means the measurement is done only typically every 5 to 30 minutes. When observing, the ATCA on-line system can either apply a nominal system temperature value to the visibility data, or it can apply the actual measured system temperature values. When applying these measured values, the on-line system will apply the most recent measurement.

  Task `atfix` includes an algorithm to interpolate measured system temperature values, between measurements, and to correct the visibility data accordingly. In doing this it correctly handles whether a nominal temperature or the measured system temperature value was applied to the data on-line. The interpolation algorithm takes into account changes in elevation and meteorological conditions.

  The parameter `tsyscal` can be used to change this default behaviour. This parameter can take on values to revert the data back to the nominal system temperature value, to make no changes to the system temperature scaling, to use the most recent system temperature measurement, or to use an extrapolation (as distinct from interpolation) algorithm in estimating system temperatures.
- Often you will want to apply a correction to your data to better account for the exact locations of the antennas.

  On the night of an ATCA reconfiguration, a solution for the antenna positions is derived. The practise is to install this solution as the "standard solution". The on-line system will then use this solution in subsequent observations in that array configuration. The quality of this solution is important for the ultimate phase calibration of all data.

  Experience has shown that often the position solution derived at the time of the reconfiguration is not necessarily the best possible. For example the solution will depend critically on the atmospheric phase stability at the time of the solution, and this might be less than ideal on the night of the reconfiguration. Consequently, improved solutions will often be derived at a later time. If improved solutions become available, then high-frequency observers should correct their data accordingly.

Updated solutions are sometimes stored in parameter files in the *MIRIAD* system directory `$MIRCAT` (or otherwise can be obtained by request from ATCA observatory staff, or from http://www.narrabri.atnf.csiro.au/observing/antpos/). These solutions are distributed via the *MIRIAD* update mechanism, and have file names of the form "`dantpos.`*yymmdd*". Here where "*yymmdd*" is the date of the relevant array reconfiguration.

Task `atfix` corrects the data for an improved antenna location solution via the `dantpos` parameter. If an improved solution file is present, you can instruct `atfix` to read this directly using the indirect parameter input mechanism (see Section 2.5). For example, to use the antenna location solution appropriate for a hypothetical array reconfiguration that happened on 16 October 2002, use

```
dantpos=@$MIRCAT/dantpos.021016
```

- One of the flaws of the old ATCA RPFITS files is that antenna locations were not saved when antennas were not in use (RPFITS files written by the CABB correlator no longer suffer from this flaw). While this might not seem a serious flaw (who cares about the antennas if they are not in use?), the unused antennas can still cause shadowing – true antenna locations are needed for shadowing calculation. This was a particular issue with the interim 3-antenna 3-mm system when observing in compact arrays.

  By giving a value to the `array` parameter, `atfix` will fill in any antenna locations that are missing in the input visibility file. *NOTE*: This just fills in missing antenna locations, it does not perform any flagging of shadowed data.

  The value given to this parameter is either an array configuration name (e.g. EW352 or 750A) or a list of six station names ("W106" and "N3" are examples of station names). When giving the station names, these must be in the order of the antennas (i.e. CA01, CA02, CA03 etc), regardless of any possible array shuffles.

  NOTE: When antennas are in a shuffled order, or for arrays using the north spur, you should generally give the list of station names, as the standard array configuration names assume an antenna order which is probably not correct.

Typical inputs to `atfix` are:

ATFIX	
in=vela.uv	Input visibility dataset.
out=vela.fixed.uv	Output corrected visibility dataset.
dantpos=@$MIRCAT/dantpos.021016	Antenna location correction.

10.2 Flagging Visibilities

In *MIRIAD*, flagging a correlation means you set a bit in a mask. There are no explicit weights like in *AIPS* or ascii flagging tables. When you list visibilities with `uvlist` (see below), flagged correlations are indicated by an asterisk to the right of the phase.

There are four main flagging tasks in *MIRIAD*:

- `blflag` – a plot-based interactive flagger. This normally plots one baseline at a time, with a variety of possible axes (e.g. time against amplitude), and you click on bad data to flag it out. This is similar to tha *AIPS* `IBLED` task.

- `tvflag` – a "TV"-based, interactive flagger. This displays one baseline at a time on a "TV" display, with the axes of the display being channel number and time. You select regions of bad data in the display. This is similar to *AIPS* task `SPFLG`.

- `pgflag` – a "PGPLOT"-based, interactive flagger. This displays one baseline at a time on a "PG-PLOT" display, with the axes of the display being channel number and time. You select regions of bad data in the display. This is very similar to the `tvflag` task, except it can be run on any X display with PGPLOT installed, whereas `tvflag` can only be run on displays with 8-bit colour depth.
- `uvflag` – a non-interactive general flagger, where you specify the data to be flagged by the `select` keyword. This is like the *AIPS* task `UVFLG`.

There are a number of other flagging tasks which occupy some niche which are described in the next section.

Although there is some personal taste involved, a recommended scheme for flagging is as follows:

- For 20 and 13 cm observations *it is strongly recommended* that you start the flagging process with `pgflag`. Narrowband interference is common at these wavelengths, and `pgflag` is a good tool at finding and flagging this interference. `pgflag` is conveniently performed on a multi-source file, straight after the data are loaded into *MIRIAD*.
- Using `blflag` is generally quick compared to `pgflag`, and some forms of bad data are more apparent with `blflag`. However, bad data is not that apparent when attempting to use it with datasets containing multiple sources. We recommend that you split the data into single source datasets (using `uvsplit`) before using `blflag`. Using `blflag` is recommended even if you use have already used `pgflag` on your data, because `blflag` is relatively quick and painless.
- Normally `blflag` will apply any calibration tables to the data before generating its plot. Thus it is just as useful after calibration as before. It is useful to look at calibrators using `blflag` both before and after solving for calibration parameters. With the program source, you will probably only use `blflag` after you are happy with the calibration.

Interactive Flagging – BLFLAG

The task `blflag` plots visibility data (usually baseline at a time, with axes such as amplitude, phase, real part, imaginary part, time, etc), and allows the user to interactively flag that data.

The user positions the plot cursor, and uses the mouse buttons or single keyboard characters to perform an operation. The interactive commands are as follows:

Left-Button: Pressing the left mouse button flags the nearest visibility.

Right-Button: Pressing the right mouse button causes `blflag` to precede to the next baseline (or to finish up, if all baselines have been cycled through).

Carriage-return: This gives a brief help message.

`a:` Flag nearest visibility (same as left mouse button).

`c:` Clear the flagging for this baseline, and redraw plot.

`h:` Give help (same as carriage return).

`p:` Define a polygonal region, and flag visibilities within this region. You define the vertices of the polygon by moving the cursor and then hitting the left mouse button (or `a`). You finish defining the polygon by hitting the right mouse button (or `x`). You can delete vertices with the middle mouse button (or `d`).

`q:` Abort completely. This exits without apply the flagging.

`r:` Redraw plot.

`x:` Move to next baseline (same as right mouse button).

The inputs to `blflag` are straightforward.

- `vis:` The name of the visibility data-set to be flagged.
- `device:` The PGPLOT device to use for plotting. This needs to be an interactive device.
- `line:` This gives the normal linetype parameter (see Section 5.4). `blflag` averages together all the selected channels before plotting it. The default is to use all channels. If you flag a data point on a plot, you flag all selected channels.
- `stokes:` Polarization/Stokes selection – see Section 5.8. If several are given, *these are averaged together* before plotting. So normally you only select one. The default is '`ii`'. If you flag a point, you flag *all* polarizations (even if you did not select that polarization).
- `axis:` This gives two values, being the X and Y axes of the plot. Possible values are time, lst, uvdistance ($\sqrt{u^2+v^2}$), hangle (hour angle), amplitude, phase, real and imaginary. The default is time vs amplitude.
- `options:` The `options` keyword give extra processing options. These include:

  `nobase` Normally `blflag` plots each baseline separately, and cycles over all baselines. The `nobase` option allows all baselines to be displayed simultaneously. This is particularly useful for point sources or when the visibilities are purely noise-like.

  `noapply` This causes the flagging operations not to be actually applied to the dataset.

  `selgen` This option generates a text file, `blflag.select`. This file contains visibility data selection commands to select the flagged data.

  `nofqaver` Normally, `blflag` will plot only one point per spectrum, that being the average over frequency of the spectral data. For old-correlator data where the bandwidth is small, this is an adequate way to display the data, but for CABB data with large bandwidths, averaging the data often masks any problems. It is therefore recommended that for CABB data this option is used to make `blflag` display the unaveraged data.

The inputs are normally very simple:

BLFLAG	
vis=vela.line	Specify visibility dataset.
device=/xs	Specify an interactive PGPLOT device.
stokes	Defaults to '`ii`'.
line	Defaults to all channels.
axis	Defaults to time vs amplitude.
select	Defaults to selecting all data.

For a point source (e.g. calibrator), a particularly useful way of flagging, at least after initial calibration, is to plot the "scatter diagram" – real vs imaginary parts of the visibility, with all baselines simultaneously. This should show points scattered around the true flux of the source. An arc in such a plot indicates poor phase stability. Typical inputs in this mode are

BLFLAG	
vis=cal.uv	Specify visibility dataset.
device=/xs	Specify an interactive PGPLOT device.
stokes	Defaults to '`ii`'.
line	Defaults to all channels.
axis=real,imag	Plot real vs imaginary.
options=nobase,nofqaver	Plot all baselines simultaneously, and don't average over frequency

Interactive Channel/Baseline Flagging – TVFLAG

The task `tvflag` is akin to the *AIPS* `SPFLG`: it displays, on a TV device, one baseline at a time, the amplitude or phase (or amplitude or phase difference from a running average computed over some time). Task `tvflag` will cycle over the set of baselines present, displaying each one, and giving you the opportunity to flag each baseline.

Task `tvflag` is the last useful remnant of the "TV" suite of software (an old package of display software within *MIRIAD*), which uses two X-windows based tools, `xmtv` and `xpanel`, for displaying. You will need to start up these tools before using `tvflag`. To start these, use commands

```
% xmtv &
% xpanel &
```

in a terminal window of your local workstation (which need not be the machine that you are running `tvflag` on). To use these, you first have to ensure that you are correctly setup to use X-windows programs – see Chapter 3 for more information on using X-windows. Using `xmtv` and `xpanel` is simple enough when you can run them on your local workstation. If you have to run `xmtv` and `xpanel` from a compute server (e.g. the situation in Narrabri, when using kaputar via an NT workstation), then there can be problems arising from multiple users attempting to allocate a given TCP/IP port. See Appendix E for more information.

The input parameters to `tvflag` are pretty straightforward. We mention only the more useful ones:

- `vis:` The name of the visibility data-set to be flagged.
- `server:` The TV server parameter. This may seem a rather redundant parameter. It is a parameter of the form

  `xmtv@`*host*

  Here *host* is the host name of the workstation running `xmtv` and `xpanel` For example

  ```
  server=xmtv@tucana
  ```

  indicates that `xmtv` and `xpanel` are running on the machine `tucana`. There is no default to this parameter, however a value of `xmtv@localhost` will be adequate when you are running `tvflag`, `xmtv` and `xpanel` all on the one machine.
- `select:` The usual visibility data selection parameter. One of `tvflag`'s odd characteristics is in its handling of different polarizations. Task `tvflag` can deal with only a single polarization per run. If it finds that the input data-set contains several polarizations, it will average these together when forming its display. While unusual, this is moderately useful. Assuming the data-set contains all four linear polarizations,

  ```
  select=pol(xx,yy)
  ```

  will produce a Stokes-I display. On the other hand, no selection (i.e. allowing `tvflag` to average XX, YY, XY and YX) will produce a quantity which is not physically meaningful, but will still allow you to make a quick check for interference and outliers.
- `line:` The normal visibility linetype parameter. Task `tvflag` does not support spectral averaging.
- `mode:` This determines the quantity to be displayed. Possibilities are `amplitude`, `phase`, `real` and `imaginary`.
- `taver:` This gives a time interval used for the running mean calculation when displaying in "DIFF" mode (see below). This parameter takes two values, both in minutes. The first gives the maximum amount of time in an averaging interval, whereas the second gives the time gap between records which causes an averaging interval to be ended. The default is 5 minutes for both of these.

Typical inputs are:

TVFLAG	
vis=vela.line	Specify visibility dataset
server=xmtv@lupus	Specify TV server
mode=amp	Display amplitude
tvchan	Unset is channel 1 else specify
range	Unset for intensity auto-scale
tvcorn	Unset to centre the image
line	Unset for all channels
taver	Controls the computation of the running mean for the DIFF command.

After invoking `tvflag`, a control panel will pop up and a baseline of data will be displayed on the TV. The display shows channels along the x-direction and time along the y-direction. A gap in time in the data is shown by a dark gap. A wedge is displayed both above and to the right of the data. The top wedge is the data averaged over time, whereas the wedge to the right is the data averaged over the channels.

You now perform flagging operations by pressing "buttons" on the control panel. We review the meaning of some of the buttons:

- `SELECT, SLCT CHAN` and `SLCT TIME`: The first thing that you will normally do is to select some set of data, with one of the three "select" buttons. `SELECT` selects a rectangular region on the display (i.e. some range of times and channels), whereas `SLCT CHAN` selects all times for a particular range of channels, and `SLCT TIME` selects all channels for a particular range of times. After pressing the appropriate "select" button,
  - move the mouse so that the cursor is at one corner of the region to be selected,
  - depress the left mouse button,
  - drag the cursor to the other corner of the region to be selected,
  - release the mouse button.
- `FLAGOOD` and `FLAGBAD`: Having selected some set of data with the previous commands, this data can be either flagged as good or bad with the `FLAGOOD` and `FLAGBAD` buttons respectively.
- `UNDO:` The undo button causes the last flagging command to be undone and discarded.
- `RESCALE:` The rescale button changes the display scaling, so that the data occupies the full greyscale or colour range. If you have some bad data which has an extremely high amplitude, for example, you will want to flag this and then rescale the display range so that you can see the detail in the remaining data.
- `DIFF:` The "diff" button causes a running mean to be subtracted off the data.
- `EXIT, QUIT` and `ABORT:` The "exit" button causes the next baseline to be loaded. The "quit" button also causes the next baseline to be loaded, but discards any flagging commands performed on the current baseline. Normally `tvflag` will terminate when it has gone through all the baselines. It is only at this stage that it applies the flagging operations to the actual data-set. Alternately the "abort" button causes `tvflag` to terminate at any time, and not to apply any of the flagging operations to the data-set.
- `LIST:` This button lists the flagging operations that have been performed to the current baseline.
- `VALUE:` This button allows you to find the value of pixels on the display. It is somewhat cumbersome – after pressing the "value" button, you move the mouse cursor to the pixel of interest, and press the `[F3]` function key. You continue to move the cursor and press `[F3]` until you have exhausted the pixels of interest. Then press `[F6]` to return back to `tvflag`'s normal mode.

There are other buttons on the control panel – but we do not suggest that you use these. They reproduce (rather crudely) buttons on the TV display window. The useful buttons on the TV display are:

- `Panner`: Pressing this pops up a window that allows you to pan around the displayed image, and to fiddle the colour lookup table. The panner window that it pops up has one of two states – `pan` and `fiddle`.
- `Luts`: This allows you to select a colour lookup table from a small number of palettes.
- `+` and `-`: These allow you to zoom in and out. For mysterious reasons, these buttons are disabled when the `Panner` window is in the `fiddle` state.
- `Reset`: Reset the zooming, the lookup tables, etc.
- `Resize`: This allows the size of the display window to be toggled between full-screen and a more modest size.
- `Quit`: Hopefully its use is obvious enough!

One shortcoming of `tvflag` is that it averages the data in time so that it will fit on the display. Unfortunately full time resolution display cannot be recovered. To display the data at full time resolution, you may need to select the data in chunks (with `select=time()`).

Interactive Channel/Baseline Flagging – PGFLAG

The `pgflag` exists because `tvflag` can only reliably run on screens that have an 8-bit colour depth, which is a limitation few users are able to stomach. In almost all other respects, `pgflag` can do the same things that `tvflag` can do, and it adds further features as well.

The `pgflag` display is that of a waterfall plot, with channel across the x-axis and time along the y-axis, and it uses greyscale colour to represent amplitude, with black being low amplitude, and white being high amplitude.

The only requirement for running `pgflag` is that the machine you are running it from (either locally or remotely) has the PGPLOT libraries installed, and has an interactive PGPLOT device, such as `/XSERVE`.

The input parameters to `pgflag` are straightforward, and only the most useful ones are described here:

- `vis:` The name of the visibility data-set to be flagged.
- `device:` The PGPLOT device to use as the plot window. `pgflag` will complain and exit if the device supplied is not interactive.
- `select:` The usual visibility data selection parameter.
- `stokes:` The usual Stokes polarization selection parameter. Task `pgflag` can deal with only a single polarization per run, so if this polarization is not specified here it will average all available polarizations together before display. While unusual, this is moderately useful. Assuming the data-set contains all four linear polarizations,

  ```
  stokes=xx,yy
  ```

  will produce a Stokes-I display. On the other hand, no selection (i.e. allowing `pgflag` to average XX, YY, XY and YX) will produce a quantity which is not physically meaningful, but will still allow you to make a quick check for interference and outliers.
- `line:` The normal visibility linetype parameter. Unlike `tvflag`, `pgflag` does support spectral averaging.
- `mode:` This determines the quantity to be displayed. Possibilities are `amplitude` or `phase`.

Typical inputs are:

PGFLAG	
vis=vela.line	Specify visibility dataset
device=/xs	Specify PGPLOT device
mode=amp	Display amplitude
stokes=xx	Select first linear polarisation, or
stokes=yy	Select second linear polarisation
line	Unset for all channels height

After invoking `pgflag` a PGPLOT window will be created with the main screen. This screen has a title bar along the top with descriptions of some of the important keys, and an area that displays data information. In the centre is the main waterfall plot, that has channels along the x-direction and time along the y-direction. Gaps in time in the data are shown by dark gaps across this plot. A wedge is displayed both on the bottom of the plot and on the right edge, and these wedges show the data averaged over time (the bottom wedge) and over channels (the right wedge). A colour wedge is also displayed that shows the minimum and maximum values and how the colours represent the amplitudes.

You now work with `pgflag` by selecting data with the mouse or keyboard, and then performing actions on the selection. These actions are all initiated by key presses, and these will be described here.

Selection of data works as a two-corner model. The left mouse button controls the location of one corner of the selection box, and the right mouse button controls the location of the other corner. Once two corners have been defined, a green box will be shown that covers the data between those two corners. Further mouse clicks will move the appropriate corner and the box will be redrawn. The only other way to initially make a selection is to press `M` twice in succession; this will select an area centred on the sample with the highest amplitude.

Once a selection has been made, it can be manipulated using the keyboard. By pressing the `c` key, the selection box will be extended to encompass all the channels currently being displayed on the screen; hitting the `c` button again will extend the selection to encompass all channels present in the data - this is useful if you have zoomed in so not all channels are visible at once. Hitting the `t` key extends the selection box to encompass all the visible times, and pressing `t` again makes the box cover all available times. To clear the selection, press `C`.

Other keys act upon the display. The `z` key zooms the main display in so that only the visibilities within the current selection are displayed, while `Z` unzooms to display all available data. While zoomed in to a particular region, the visibilites shown can be changed by pressing `h` which moves the zoom window left (to a lower channel range) by half the range shown on the plot, while `H` moves the window left by the entire range. For example, if the x-axis is showing channels between 20 and 40, then pressing `h` will make the window show channels between 10 and 30, whereas pressing `H` will make the window show channels between 0 and 20. Similar functionality can be found by pressing the `l` and `L` keys to move the window right, the `j` and `J` keys to move the window down, and the `k` and `K` keys to move the window up. Old-school readers may recognise this as the cursor movement keys for the editor `vi`. While zoomed in, the display will indicate if there is further data in a particular direction by displaying a green arrow next to the main plot area for each possible scrolling direction. The plot can also be unzoomed in a specific direction: pressing `s` will unzoom to show all the available channels, and `S` will show all the available times.

The currently displayed baseline is changed with the `n` key – which displays the next available baseline – and the `p` key – which displays the previous available baseline. The selection and zoom region is preserved while changing baselines.

The display can also be changed to enable easier identification of bad data. Pressing the `x` key will subtract the average channel values (which are shown in the bottom wedge) from each visibility in the main plot, essentially correcting for any bandpass effects. While this subtracting is in effect, the bottom wedge will be outlined in red. To reset this behaviour press `x` again - the wedge will return to being outlined in white. The average time values (shown in the right wedge) can be subtracted by pressing the `d` key in the same manner.

The colour scaling can be changed in various ways. To select for yourself the range to use, press the `,` (comma) key, then press the left mouse button within the colour wedge to set the value to be assigned as white (the maximum value), or the right mouse button to set the value to be assigned as black (the minimum value). Regardless of whether the displayed is zoomed or not, the same colour scale will be used, and it is determined from all the available unflagged data on the currently displayed baseline. To use only the currently displayed data to set the colour scale, press the `[` key, or to use only the currently selected data, press the `]` key. Normally, changing baselines will cause the amplitude scale to be recomputed, however the current colour scaling can be locked by pressing the `;` key. To reset the colour scale and allow `pgflag` to determine it automatically, press the `.` (full stop) key.

There are many flexible ways of flagging the data with `pgflag`, the simplest of which is by pressing the `f` key to flag as bad the visibilities within the current selection box on the current baseline only. Pressing the `g` key will flag the selected visibilites as good. Almost any flagging operation can be reversed by pressing the `u` key; no flagging is applied to the data itself until the task `pgflag` ends. Be aware though that `pgflag` only allows the very last flagging action to be undone with the `u` key, so successive presses of `u` will in fact undo the undo.

Pressing the `F` key will mark the selected visibilities as bad on all the baselines in the file, while `G` will mark them as good on all baselines. The currently displayed baseline is shown at the top of the plot in the form `A-B` where A and B are both antenna numbers. To flag the selected visibilities as bad for antenna `A`, press the `1` key (the `!` key marks the selection as good), while to do it for antenna `B`, press the `2` key (or `@` to mark as good).

It is also possible to remove all flagging actions performed by this session of `pgflag` on the currently displayed baseline by pressing `rr`. It is for safety that the `r` key must be pressed twice successively, as this operation cannot be undone. Pressing `RR` will remove all flagging actions on all baselines. Press `?` to make `pgflag` print some information about how many flagging operations have been made during this session.

To exit `pgflag` and save all the flagging, press the `q` key. If you want to exit `pgflag` without applying any of the flagging, press the `a` key. You will be prompted to ensure that you don't exit the task accidentally.

General *MIRIAD* Flagging – UVFLAG

In `uvflag`, you use the standard selection keywords to select visibilities, or parts of visibilities for flagging. The main keywords are `select` and `line`. `uvflag` can also be used to unflag correlations, and to list the correlations that meet the selection criteria without actually flagging the data. There is a keyword, `edge`, that provides a way to select the same group of channels in spectral-windows (IFs). This cannot be done with the standard `line` and `select` keywords owing to the way spectral-windows are stored (for more information, issue the command `help windows`). Read the `uvflag` help file if you need this facility. It is unlikely that you will.

The following example flags some correlations between 1 and 2 hours on all days involving all baselines with antenna 6 in channel 25 for all polarizations and spectral-window (IF) 1.

UVFLAG	
vis=vela.line	Specify visibility dataset
select=ant(6),time(1:00,2:00),win(1)	Select correlations
line=channel,1,25,1	Select channel
edge	
flagval=flag	Flag selected correlations
options=brief	Summarise what happened
or	
options=noapply	Summarise what could happen
log	Unset to report information to screen

10.3 Other Flagging Tasks

AIPS Flagging and *MIRIAD*– FGFLAG

If you flag in *AIPS*, the flagging information will enter *MIRIAD*, via FITS files, in two possible forms:

- As flagging (FG) tables (generally only multi-source files).
- As negated visibility weights (both single- and multi-source files).

While *MIRIAD*'s `fits` task flags correlations which have negative weights, it *does not* apply any flagging tables. Instead it copies across the flagging tables, and includes them within the *MIRIAD* dataset. If you wish to use this flagging information, it is essential that you then apply the appropriate flagging tables. The task to do this job is `fgflag`. Its inputs are pretty simple – typical inputs are:

FGFLAG	
vis=vela.line	Specify visibility dataset
select	Extra selection – normally unset
fgflag	Table version. Default is highest version.

Note some caveats:

- The flagging tables written by `fits` cannot be copied to another *MIRIAD* dataset. Additionally they are ignored by all *MIRIAD* tasks, save `fgflag`.
- Task `fits` does not copy across information which distinguishes correlations by polarisation. If one polarisation is flagged in the FG table, all polarisations will be flagged by `fgflag`.

Flagging Near a Set-Up Change – QVACK

Another handy task is `qvack` (derivative of the *AIPS* task `quack` and so named for obscure reasons) for flagging a number of correlations after the observing set-up has changed (i.e., the source changed, the pointing centre changed, or the frequency changed). For example, it might be used if some evil-minded child turned on the family microwave oven for 30 seconds every time s/he saw the antennas start to move. If you were observing at 13 cm, here is how to get rid of such data. Getting rid of the child would be the best long-term solution.

QVACK	
vis=multi.uv	Specify visibility dataset
select=freq(2.699,2.829)	Select correlations in this range
interval=0.5	Flag data for 30 s after change
force	Unset
mode=source	Flag when source changes

`qvack` also has a keyword called `force`. This can be used if the observing setup has not changed, but you still wish to periodically flag some data. For example, you may have split out your observation and so you are left with a file with no observing change, but with a distinct "scan" based regularity. This keyword lets you flag some data after a certain time has elapsed.

Flagging Based on a Template – UVAFLAG

Often it is convenient to edit one visibility dataset ("the template"), and then to apply the flagging from this to another. For example, forming and flagging a "channel-0" dataset might be less painful than flagging a huge spectral line dataset. Or you might want to flag a calibrated dataset and then apply this to the raw data. The task that can do these sorts of operations if `uvaflag` – it matches visibilities in the template and the input, and flags those visibilities in the input that are flagged in the template.

The template dataset should be a subset of the dataset being flagged, and the order of the records in the two datasets should be the same. Normally correlations in the template and the input are matched by comparing time, polarisation, baseline and frequency. However options `nopol` and `nofreq` can turn off the matching of polarisation and frequency. Typical inputs might be:

UVAFLAG	
vis=multi.uv	Dataset to be flagged.
tvis=multi.chan0	Template dataset, say a channel-0 one.
options=nofreq	Do not match on frequency,
	i.e. ignore the frequency in multi.chan0

Flagging a Sector of Data – UVSECTOR

The task `uvsector` flags a set of visibilities based on their $u-v$ position angle. To understand the utility of this, consider the samples measured in the $u-v$ plane at a particular time instant. For an east-west array (i.e. the ATCA) the samples will lie on a spoke of an ellipse. If there is a glitch an a particular time, the bad data will cause a stripe pattern in the resultant image which is at right angles to the spoke. The position angle of the stripe can be used to determine the $u-v$ plane position angle of the glitch. From this you could determine the offending time and then flag the appropriate data. Task `uvsector` performs these operations for you.

Task `uvsector` will flag a sector of visibility data with a given $u-v$ position angle (or hour angle). While you can give the position angle of the sector to be flagged directly (via the `angle` parameter), it is more convenient to indicate the position angle indirectly via the stripe direction in an image. How do you specify a stripe direction? You do this by specifying a long, thin region in a image. Normally you will generate the region by using `cgcurs` (see Section 17.3 for more information on `cgcurs`). With `cgcurs`, you display a greyscale of the image, and then use a cursor to selection a long, thin, region containing the crest of a single stripe. The selected region is then written into a text file, `cgcurs.region`. While this may be a rather odd way to define a direction, it allows `uvsector` to use the interactive facilities of `cgcurs`.

Typical inputs to use `cgcurs` are

CGCURS	
in=vela.icln	Input image (probably CLEANed).
type=grey	Greyscale plot.
range=	Set the range to highlight the stripes.
device=/xs	PGPLOT device – Xwindows here.
options=region	Define a region of interest.

The width of the sector that `uvsector` flags is given as an angle (in degrees) via the `width` parameter. Remember that 1 degree is equivalent to 4 minutes of observing time, so do not set it to any value larger than you need. Normally you would set this to a few degrees, based on your confidence in the accuracy of the stripe direction. The default is 5 degrees.

As `uvsector` flags a sector, if your input visibility data-set contains multiple configurations, data within that sector for all configurations will be flagged by default. Generally this is *not* what you are likely to want. If you know which configuration contains the bad data, you can use the `select` keyword to ensure just that configuration is affected. Selecting by `time` is probably the easiest.

The breadth of a stripe also contains useful information – a glitch in just the short baselines will cause a broad stripe, whereas a glitch in just the long baselines will cause a narrow stripe separation. A glitch in just one baseline will give a two dimensional sinusoid. Given the separation between stripes, you could make an approximate calculation of the corresponding $u-v$ radius of the bad data. For a stripe separation (crest to crest) of x arcseconds, the corresponding $u-v$ radius is

$$r = \frac{206}{x} \text{kilowavelengths}$$

You may then use `select=uvrange` so that only data at the appropriate $u-v$ radius is flagged. For example if the crest-to-crest separation is 20 arcsec, we might flag data only with spacings of 10 ± 2 kilowavelengths.

UVSECTOR	
vis=vela.uv	Visibility data to be flagged.
angle	Leave unset if defining position angle by a stripe direction.
in=vela.icln	The image containing stripes, used only for its coordinate information.
region=@cgcurs.region	A long, thin, region along the crest of a stripe.
width=3	Flag data over 3 degrees (12 minutes of time)
select	Leave unset to flag all data in the sector,
or	
select=uvrange(8,12)	Flag data within the sector and between 8 to 12 kilolambda.

10.4 Listing Visibilities

1. The task `prthd` summarizes some basic information about your visibility dataset (it also summarizes images). It only reads the first few records of the data, so if you have multiple frequency setups or sources, it will only report the first setting. It will tell you the number of visibilities, the number of spectral-channels, the number of spectral-windows (IFs), the polarizations present and some information about frequencies. Output can be directed to a log file (the terminal is the default), and either a brief or full listing can be given (the default is the full listing). Several inputs can be given (wildcards are supported).

2. The task `uvindex` scans through the visibility dataset and reports changes in the source, pointing centre, number of spectral-channels, observing frequency, and polarization. Thus, it builds a description of your observation for general reference. Give the visibility dataset name with the keyword `vis`. The output can be directed to a text file for printing via the keyword `log`.

3. The task `uvlist` can be used to list the values of the correlations. The generic *MIRIAD* keyword `options` is used to control exactly what is listed (see the help file). The standard keywords `select` and `line` are used to select the desired visibility data and the desired part of the spectrum, respectively. Here is an example.

UVLIST	
vis=vela.uv	Specify visibility dataset
options=data,brief	Lists correlations (this is the default)
or	
options=spectra,brief	Lists info. about the spectral-windows
select=pol(xx),ant(1)	List XX correlation for all baselines involving antenna 1
line=channel,1,128,1,1	List data for channel 128 only
recnum=100	List 100 records
log=xx.txt	Write results to this text file or leave unset to write to standard output

10.5 Copying, Concatenating and Averaging Visibility Datasets

MIRIAD contains a single task, `uvaver`, which copies, concatenates and averages data (both in frequency and in time). It can also apply calibration corrections and perform Stokes conversion.

The inputs are all fairly self-explanatory – see the help file for more information. The default behaviour is to copy all the data, applying calibration corrections as it goes. If some averaging is being performed, vector averaging is normally used both for time and frequency. However an option (`options=ampscalar`) switches the behaviour so that amplitude-scalar averaging is used for time averaging. Vector averaging is always used when averaging in frequency.

A brief summary of the inputs are:

- `vis`: a list of the input visibility datasets. Several can be given. Wildcards are supported.
- `out`: The name of the output visibility dataset. The output will be the concatenation of all the selected input data. No default.
- `select`: The standard visibility data selection parameter. See Section 5.5. The default is to select all data.
- `stokes`: The standard parameter to determine Stokes conversion. The default is to copy the polarizations or Stokes parameters as is (no conversion). See Section 5.8.
- `interval`: Time averaging interval, in minutes. The default is to perform no time averaging.
- `line`: The standard `line` parameter, which dictates channel selection and frequency averaging in multi-channel datasets. The default is to copy all channels unaveraged. See Section 5.4.
- `options`: Extra processing options. The `nocal`, `nopass` and `nopol` options are used to disable correcting for antenna gains, bandpass functions and polarization leakage (see Section 5.3).

  There are also three options to determine the sort of averaging to be performed. Possible options are `vector`, `scalar` and `scavec`. Note these options only affect the averaging in *time*. Vector averaging is always used when averaging in *frequency*.

  - "Vector" averaging is just a straightforward average of the complex-valued visibility data. This is the optimum thing to do if the data are corrupted by just Gaussian noise. However it is common for the data to be affected by fast-varying phase fluctuations. While the amplitude of the data are good before averaging, a plain vector average will result in some decorrelation.
  - "Scalar" averaging avoids this by using the average amplitude for the amplitude of the output quantity (the phase of the output is still the phase determined by vector averaging). The problem with scalar averaging is that it results in a noise bias when the signal-to-noise ratio is small (less than a few) – you should not use scalar averaging in this regime.
  - For cases where phase stability is poor, and the signal-to-noise ratio on the parallel-hand correlations (XX and YY) is good, but the signal-to-noise ratio on the cross-hand correlations (XY and YX is poor, `uvaver` provides the `scavec` option. This performs scalar averaging on the parallel-hand correlations, and vector averaging on the cross-hand correlations.

For example, the inputs below show how to form a channel 0 dataset from a dataset with two spectral windows (IFs), each with 33 channels (selecting the middle 25 channels from each window). The two spectral windows are essentially treated as 66 contiguous channels. Thus, we must use the `line` keyword to select channels 4 to 28, and then channels 37 to 61. `uvaver` will correctly maintain the dual window status (use `prthd` to verify this).

UVAVER	
vis=vela.uv	Input dual window 33 channel dataset
out=vela.uv.0	Output channel 0 dataset
line=channel,2,4,25,33	Average channels 4 to 28 in each window

10.6 Plotting Visibilities

1. You may tire of all this listing of numbers, and would prefer to make some plots. The task `uvplt` provides a variety of choices when plotting visibilities, and ancillary information associated with the visibilities. It has rather a lot of inputs, so read the help file first for the details. We discuss some of the keywords below and then give a couple of examples.

   - `vis` specifies the visibility datasets to plot. It can take multiple dataset names, each separated by a comma, and wildcard expansion (via an asterisk) of file names is supported.
   - `line` and `select` are the standard visibility selection keywords. The default for `select` and `line` is all the data and all the channels.
   - Select the desired polarization or Stokes parameter with `stokes`. Conversions from linear (or circular) polarizations to Stokes parameters are supported. You can have any combination that you like on one plot, but they will all be plotted with the same symbol. This generally does not matter as any signal will make the distinctions clear and colours are used on devices that support it.
   - `axis` is used to tell `uvplt` what to plot on each axis. You can put any of the allowed choices (see the help file) on either of the axes. For example, you could set `axis=imag,real` to plot the imaginary part of the visibility on the x-axis, and the real part on the y-axis. Such a plot is very useful to examine calibrator data. The default is to plot visibility amplitude versus time.
   - `xrange` and `yrange` specify the plot ranges for the x- and y-axes. If they are unset, the plot(s) are auto-scaled. Generally, each of these keywords takes two values. However, if the corresponding axis is `time`, then you must specify eight values; a day, hour, minute and second for the start and stop times.
   - You can average the plotted quantities in time by setting `average` in minutes (but see the `options` keyword for exceptions to this unit). Baselines and polarizations are averaged separately (unless `options=avall`), but channels are not – they are all lumped in together, unless `options=nofqav` is specified.

     The complex visibilities are vector averaged by default, and the real quantities are scalar averaged by default. For example, if you ask for an averaged amplitude or phase, the real and imaginary parts of the visibilities are averaged, and then the amplitude or phase formed. If you ask for, say, u versus time, u is scalar averaged. You can override the vector averaging by setting `options=scalar`; a scalar averaged amplitude is useful if the phase is winding rapidly. A scalar averaged phase is pretty meaningless under most conditions.

     Averaged points are plotted with filled-in circles rather than dots, unless `options=dots`.
   - If you plot, say, amplitude or phase versus time, you may find the application of Hanning smoothing useful to knock off the rough edges, rather than applying box-car smoothing (which is what you get when you average in time). Set `hann` to an odd number.
   - If you do not want to see all the selected points, you can set `inc` to plot every `inc`th point that would normally have been selected. For example, it is useful for plotting u versus v. Be careful of a value of `inc` that divides exactly into the number of baselines in time-ordered data (you end up seeing the same baselines over and over).
   - `uvplt` has a vast complement of `options`. They are all described in the help file. We will mention a few of them here.

     Like most of the visibility tasks, it has `nocal, nopol` and `nopass` to turn off the calibration tables, which, by default, are applied if they exist. *MIRIAD* will tell you when it applies any calibration table.

     By default, `uvplt` puts each baseline on a separate sub-plot on the page (see `nxy` below). Setting `options=nobase` means that all baselines go on the same plot.

     `options=scalar` instructs `uvplt` to do scalar averaging in time.

     On occasion, you may not wish to time average baselines or polarizations separately. Setting `options=avall` instructs `uvplt` to average together everything that is to be plotted on a separate sub-plot.

On the other hand, you may not wish to have the channels averaged together at all. Setting `options=nofqav` will make each channel a separate point; this vastly increases the number of points that will be plotted (making `uvplt` much slower and memory hungry, especially with CABB data), but may make certain problems with the data far more apparent.

`options=rms` instructs `uvplt` to plot error bars on any time-averaged quantities. However, note that error bars are not yet implemented for vector averaging.

By default, if you plot many sub-plots (one for each baseline), all the sub-plots are displayed with the same axis ranges, which encompass the data on all the sub-plots. Setting `options=xind` or `options=yind` (or both) scales that axis on each sub-plot independently.

`uvplt` has an interactive mode invoked by setting `options=inter`. This gives you a chance to redraw the plot with different axis ranges, without having to read all the data in again. This option also prompts you for a new plot device, so that following display and range refinement on your workstation, you can then make a hard copy without re-running `uvplt`.

- Select the PGPLOT device to plot on with a command such as `device=/xs` (plot on local X-window) or `device=amp.plt/ps` (write plot to a PostScript file called 'amp.plt').
- Specify the number of sub-plots in the x- and y-directions with a command such as `nxy=4,3`.

The following example plots u versus v, and $-u$ versus $-v$, selecting channels at the start, middle and end of a band of 32 channels (so that the $u-v$ coverage benefit obtained across the band can be seen).

UVPLT	
vis=vela*.uv	Specify visibility dataset(s)
line=channel,3,2,1,15	Plot channels 2,17,32
select	Leave unset for all data
stokes=xx	$u-v$ coverage same for all polarizations
axis=uc,vc	Plot $u-v$ and conjugate $u-v$ plane
xrange	Auto-scale
yrange	Auto-scale
average	No averaging
hann	No Hanning
inc=1	Plot every point
options=nobase	All baselines on one plot
device=/xs	Plot on local X-window
nxy=1	One sub-plot only please
size	Default character sizes

The next example shows how to plot a scatter diagram (real vs imaginary) from a single-channel dataset for the Q, U and V. polarizations, with all baselines plotted on the one plot. This is a quite useful plot to make of calibrated calibrator data. Because you are asking for Stokes parameters, it is assumed that a calibration has been made (the calibration tables will be applied by `uvplt` and a reminder issued) or that you converted to Stokes parameters along the way (e.g. with the $\mathcal{AIPS}$ task `ATLOD` or with `uvaver`).

UVPLT	
vis=vela.uv	Specify visibility dataset
line	Plot the first channel only
select	Leave unset for all data
stokes=q,u,v	Select Stokes Q, U and V
axis=real,imag	Is the default
xrange	Auto-scale
yrange	Auto-scale
hann	No Hanning
inc	Plot all points
options=nobase	Scalar averaging
device=/xs	Plot on local X-window
nxy=1	1 plot per page
size	Default character sizes

2. Task `closure` is another useful plotting task. It is most useful for plotting the closure phases of point sources. The closure phase of an object is the sum of the baseline phases around a triangle. For example, for antennas 1, 2 and 3 we measure three baseline phases ϕ_{12}, ϕ_{23} and ϕ_{31}. It can be shown that the closure phase, which is the sum of these three baseline phases

   $$\phi_{\mathrm{closure}} = \phi_{12} + \phi_{23} + \phi_{31}$$

   will be independent of any antenna-based gain errors (atmosphere and instrumental). This quantity will not be affected by *MIRIAD*'s antenna calibration process. Additionally for a point source (or any source with 180 degree rotational symmetry) the closure phase should be zero. Plotting the closure phase of a calibrator is thus a way of checking the quality of the data. If the closure phase is large, the data are probably bad, or there is some calibration error that is not accounted for in *MIRIAD*'s antenna-based model.

   Task `closure` plots averages of closure phases (actually it averages together *triple products* and then takes the phase of this). The averages are taken over the the different correlator channels and polarizations, and optionally over time.

   The inputs to `closure` are simple enough. See the help file for more information. Typical inputs to `closure` are:

CLOSURE	
vis=vela*.uv	Specify visibility dataset(s)
line=	Set the channels of interest.
select	Leave unset for all data
stokes=xx	Closure phase for XX
yrange	Auto-scale
options=notrip	Plot all closure phases on one plot
device=/xs	Plot on local X-window
nxy=1	One sub-plot only please

3. If you are working with spectral data (line or continuum) you may want to look at some spectra. Use task `uvspec` for this. The inputs to `uvspec` are pretty much self-explanatory. Here is an example of how to plot the spectra (phase) of the XX and YY correlations averaged over 15 minutes for all baselines involving antenna 1 with all calibration turned off. `uvspec` makes a new sub-plot every time it begins a new averaging interval. Assuming that there are five antennas in this dataset, we have arranged for each baseline to occupy one column of the plot page with time increasing down the page.

UVSPEC	
vis=vela.line	Specify visibility dataset
select=ant(1)	Baselines involving antenna 1
line	Unset for all channels
stokes=xx,yy	XX and YY correlations
interval=15	15 minute averages
options=nocal,nopol,nopass	Turn off calibration,
axis=freq,phase	Plot phase versus frequency
yrange=-185,185	Specify y-range for plot
device=/xs	Plot on local X-window
nxy=5,6	6 time intervals in y directions

10.7 Examining Visibility Variables

There are a variety of variables associated with the dataset that you may like to examine. For example, you may like to plot the system temperature as a function of time; it is usually stored as a variable (the visibilities are variables too). A list of standard visibility variables is given in Appendix C. Some of these variables are specific to the BIMA interferometer (Hat Creek), and so may be absent, or have no relevance to ATCA data.

The task `varlist` will tell you the names of the variables in your dataset. It will also tell you their data types (ascii, double precision, single precision and integer) and lengths.

The following variables are particularly relevant for high-frequency data.

smonrms	ATCA seeing monitor rms (in microns)
refpnt	Reference pointing solution (in arcseconds).
axisrms	RMS tracking accuracy
axismax	Maximum tracking accuracy

After finding out the name of the variable of interest, you can plot it with `varplt`. The following example plots the on-line XY phase differences, stored in the variable `xyphase` (which will be present if you used `atlod` in *MIRIAD*). You can also use `varplt` to write the variable's values to a text file with the standard `log` keyword.

VARPLT	
vis=vela.line	Specify visibility dataset
device=/xs	Plot on local X-window
log=var.txt	Write values to text file
xaxis=time	Put variable `time` on x-axis
yaxis=xyphase	Put variable `xyphase` on y-axis
nxy=3,2	One sub-plot per antenna
xrange	Unset for auto-scale
yrange	Unset for auto-scale
options	Leave unset

10.8 Modifying Visibility Datasets

The task `uvedit` allows you to apply some specific sorts of corrections or modifications to your visibility dataset. For example, there may be a clock error that needs fixing, or the antenna coordinates were wrong so that u and v need to be recomputed, or the phase centre was wrong requiring a phase rotation and u and v recomputation. Note that any data not selected are copied to the output dataset unchanged.

The following example changes the RA phase centre by 10 seconds of time and also corrects a delay error of -3 ns on antenna 2 (by adding a phase gradient across the band). `uvedit` has the option to turn off the recomputation of u and v which might otherwise be done (`options=nouv`). This can be useful for say, entering a delay change (which does not need u and v to be recomputed) because *MIRIAD* does not compute u and v as accurately as the on-line ephemeris routines.

UVEDIT	
vis=multi.uv	Input visibility dataset
source=0823-500	Specify source; unset means all
antpos	No change to antenna coordinates
dantpos	
ra=10	Move RA phase centre
dec	No change to DEC
time	No clock offset
delay=0,-3,0,0,0,0	Delay error on antenna 2
out=multi_corr.uv	Output visibility dataset
options	Do the works

The task `uvredo` also 'edits' a visibility dataset. Unlike `uvedit`, however, it only modifies ancillary data in the dataset that are derived from other information. For example, it can recompute the sky feed angle ("chi") and velocity information.

Chapter 11

Calibration, the ATCA, and $\mathcal{MIRIAD}$

11.1 Some Theory

11.1.1 Polarisation

Much of the theory here has been borrowed from the memo 'AT Polarisation Calibration' (AT Memo 39.3/015). See that memo for more details. A more detailed description of polarimetric interferometry can be found in Hamaker, Bregman & Sault (1996) and Sault, Hamaker & Bregman (1996) (A&AS 117, 137 and A&AS 117, 149).

Recall from Chapter 5 that $\mathcal{MIRIAD}$ models a feed as having a composite gain of

$$g(t)g_P(\nu)\exp(i2\pi\tau(t)(\nu-\nu_0))$$

where $g(t)$ is a time variable complex number (often loosely called the antenna gain), $g_P(\nu)$ is the bandpass function, and τ is a time-varying delay term.

Additionally, we have to consider the response of the feeds to polarised emission. Whether a feed is linearly or circularly polarised, its instantaneous response to a signal is a linear combination of two of the four Stokes parameters that describe the wave. In the equatorial frame of the source, ideal linear feeds respond according to

$$\begin{aligned} XX &= I+Q \\ YY &= I-Q \\ XY &= U+iV \\ YX &= U-iV, \end{aligned}$$

where the X and Y feeds are at position angles 0 and 90°, respectively. Perfect circular feeds respond according to

$$\begin{aligned} RR &= I+V \\ LL &= I-V \\ RL &= Q+iU \\ LR &= Q-iU \end{aligned}$$

These equations show immediately why it is harder to calibrate an instrument with linear feeds, such as the ATCA, compared with an instrument with circular feeds, such as the VLA. In the latter case, one can make the excellent assumption that the calibrators, which are used to determine the antenna gains, are not circularly polarised. Thus, the RR and LL visibilities are a direct measure of I, for which we have a good model (*i.e.* a point source of known flux density). Consequently they can be used to calibrate the gains with time. On the other hand, it is not necessarily a good assumption that a calibrator is not linearly polarised, so that XX and YY correlations cannot always be used as a direct measure of I.

In addition, for 'alt-az' telescopes, the feeds rotate with respect to the equatorial frame. This causes the actual response of ideal linearly polarised feeds to vary with the angle between the "sky" and the feed – χ (which varies with time, although non-linearly), according to

$$\begin{aligned}
XX &= I + Q\cos(2\chi) + U\sin(2\chi) \\
YY &= I - Q\cos(2\chi) - U\sin(2\chi) \\
XY &= iV - Q\sin(2\chi) + U\cos(2\chi) \\
YX &= -iV - Q\sin(2\chi) + U\cos(2\chi)
\end{aligned}$$

So far we have assumed that the feeds are ideal. This is never the case, and their departure from the ideal can be characterised by leakage terms. D_{x} is the leakage of the y component of the electric field into the X feed, and D_{y} is the leakage of the x component of the electric field into the Y feed. Another way of thinking of them is the combination of the ellipticity and error in the position angle of the polarisation ellipses of each feed.

Incorporating these leakage terms results in fairly gory equations – see Hamaker, Bregman & Sault (1996) and Sault, Hamaker & Bregman (1996) (A&AS 117, 137 and A&AS 117, 149) for more details.

11.1.2 Atmospheric opacity and system temperature

To provide correct flux density units for the visibilities, V, from an interferometer, the measured correlation coefficient at the correlator (ρ) needs to be scaled by a system temperature T_{sys} and a telescope sensitivity factor (K). The telescope sensitivity factor is determined by the overall antenna size and system efficiency.

$$V = KT_{\mathrm{sys}}\rho$$

At millimetre wavelengths, the atmosphere can no longer be approximated as perfectly transparent. It degrades overall sensitivity in two ways: the atmosphere emits radiation, and so raises the system temperature, and the atmosphere attenuates the astronomical signal. For a zenith opacity τ, observing at an elevation of e, and if the atmosphere is approximated as having a uniform temperature T_0, then the atmospheric contribution to system temperature is

$$T_{\mathrm{sky}} = T_0\left(1 - \exp(-\tau/\sin e)\right)$$

and the atmospheric transmissivity is $\exp(-\tau/\sin e)$.

In converting correlation coefficients to visibility measurements, we can notionally include the effect of the atmospheric attenuation either in the effective system efficiency factor, or in an effective system temperature. The approach used will depend on the calibration scheme implemented at the telescope.

For the ATCA at 3-mm wavelength, it is most natural to include the effect of the opacity in an effective system temperature – the so-called "above atmosphere" system temperature. The above atmosphere system temperature is simply the physical system temperature divided by the atmospheric transmissivity. The widespread use of above atmosphere system temperatures in millimetre astronomy follows on because the above atmosphere system temperature can be measured directly! It can be deduced by periodically flipping an absorber at atmospheric temperature in front of the feed system. The technique to do this is the so-called "chopper wheel" method (e.g. Ulich 1980). The ATCA uses the chopper wheel method (also called paddle or vane calibration) in its 3-mm system to determine an above atmosphere system temperature.

This measurement will only be done occasionally. That is, unlike the centimetre, 12-mm and 7-mm bands, continuous measurement of system temperature is not available at 3-mm wavelengths. To partly remedy this, *MIRIAD*'s `atfix` includes an algorithm to "interpolate" the above-atmosphere system temperature measurements based on changes in telescope elevation and weather.

11.1.3 Antenna gain considerations and flux calibration

One characteristic of the ATCA dishes is that they are big by 3-mm standards. Indeed van Hoerner (1967) has considered the limits to the size of an antenna before various effects become significant. For construction styles like the ATCA dishes, he suggests that 11m is the maximum dish size for 3-mm observations before thermal effects become significant. As the ATCA dishes are twice this, it is no surprise that the ATCA antenna gain and primary beam response varies as a result of thermal distortions. Changes in temperature, the position of the sun, cloudiness and shadows will all affect the gain of the ATCA antennas. Measurements to date tend to bear this out, with changes in antenna gain of 25% noted that appear to be related to thermal effects. Distortion of the dishes by gravity is also significant (eg Subrahmanyan 2005). A gain/elevation correction will solve this - to first order at least. Because of the way the ATCA panels are set, the antenna gain peaks at about 60° elevation.

Note that gravitational and thermal distortions are coupled - a perfect correction for these effects is never possible.

To first order, the gain change will be calibrated out provided the phase calibrator is very close to the target source. Applying the standard gain/elevation curve to the data as the first step in the data reduction will also help. However these two steps will not account for the change in the beamshape. This might be an important effect for widefield imaging and mosaicing.

The best way to avoid thermally-induced gain variations is by observing in the pre-dawn hours or on a calm cloudy day! The best way to avoid large elevation changes in a synthesis is to observe in a hybrid array – hybrid arrays allow a synthesis to be performed without the need to track the source from horizon to horizon.

The gain variations must be considered when attempting to correct the flux density scale. The best approach to ensuring a good flux scale is to ensure that the phase and the flux density calibrators are observed nearly simultaneously and at the same elevation. Although this may at first sound difficult if not impossible, generally it is straightforward. Generally there should be a time when the phase and the flux density calibrators are at the same elevation. It may be that one is rising while the other is setting, and so they are at significantly different parts of the sky (ie significantly different azimuths). Unless the sky is cloudy, being at different azimuths is not important.

11.1.4 Instrumental phase

At the precision needed for 3-mm observing, the ATCA antennas are not identical in terms of their physical construction. In particular the offset between the elevation and azimuth axis will differ between antennas. VLBI astronomers, who have used heterogeneous arrays of antennas, have coped with this issue for years. For the ATCA at 3mm, what is presumed to be a difference in this offset introduces a instrumental phase error, that needs to be corrected.

11.1.5 Atmospheric phase and phase calibration considerations

With the old ATCA correlator, phase calibration at 3-mm wavelength is significantly more difficult than at the longer wavelengths. This is because the atmospheric phase errors are comparatively larger, the secondary calibrator grid is sparser and the system sensitivity is poorer. To use a tolerably nearby calibrator, you will need to use a calibrator that is weaker than you might wish - particularly if you wish to observe in a narrow bandwidth. Alternatively you might use a nearby strong SiO maser rather than continuum point source phase calibrator.

These lead to a different approach to calibration compared to that at longer wavelengths. At 3-mm wavelengths, the dual frequency band capability of the old ATCA correlator could be used to good effect in the calibration process. When sensitivity to calibrators was an issue, one of the two bands could be specifically set aside for calibration purposes. The correlator configuration could be selected so that one of the bands was either 128 MHz wide (and so as sensitive as possible), or suitable for observing an SiO maser. This calibration band could also be used for reference pointing during the course of the observation.

When using one of the bands for calibration purposes, the two bands need to be jointly calibrated.

SiO masers should be used with caution: they are often strongly polarised, and they may be structurally complex. Note, also, that the old ATCA correlator limited the dual observing bands to be set within 2.7 GHz of each other. If you use an SiO source as a calibrator, the part of the 3-mm band that is accessible to the observation is limited.

For data taken with the CABB correlator, the extra bandwidth provides a large increase in sensitivity. Also, since CABB provides continuum data at all times, regardless of whether spectral line data has been gathered as well, it is possible to do reference pointing and phase calibration in the same way for all observations at all wavelengths.

11.2 Calibration Methods

There are two distinct calibration situations with the ATCA, depending on whether the correlator configuration produced all four polarisation products (XX, YY, XY and YX) or just two (XX and YY). The former is done for all CABB data, and for the old-correlator continuum mode and some spectral line modes, whereas the latter is true of other old-correlator spectral line modes.

Calibration of Data with XX, YY, XY and YX

If all four polarisation products are measured, a full polarimetric calibration is possible. We generally solve for the leakage terms, the gains, a bandpass function and calibrator Q and U.

In the solution process, we must consider the time variability of the various parameters. The gains vary fairly rapidly with time, and we typically solve for them every hour or so, with the assumption that they are steady over a solution interval of a few minutes – the calibrator scan. On the other hand, the leakage terms are assumed to be steady over the entire length of the observation. Our experience so far appears to support this. The bandpass function and the calibrator Q and U can also be assumed to remain unchanged during an observation. It is not possible to solve for a calibrator's V (circularly polarised flux density) However, a calibrator's V is unlikely to be significant (< 0.1 %).

If the calibrator is weakly polarised ($< 5\%$), one quantity that it is difficult to solve for is the absolute XY phase, absolute feed alignment and absolute feed ellipticity. Generally these quantities on the reference antenna have to be either assumed, or measured, or derived from observations of a strongly polarised calibrator. Unfortunately most calibrators are only weakly polarised. Fortunately, however, the alignment and ellipticity errors are small (assuming Faraday rotation is negligible). Additionally, since November 1992, the ATCA makes a good measurement of the absolute XY phase (accurate to a few degrees). If we assume that the absolute alignment and ellipticity errors of the reference antenna are zero and that the XY phase measurements are accurate, a "polarisation position angle calibrator" is not needed.

The recommended strategy for calibrating ATCA data is thus to observe a flux density calibrator, normally 1934-638, as well as a phase calibrator close to the source. For polarimetric observations, as the polarisation of 1934-638 is known (in fact it is $< 0.2\%$), and the flux density is strong (at least in cm bands), it is possible to derive polarisation leakages from it given only a short observation (5-10 minutes). If you have good parallactic angle coverage of the secondary, it is possible to simultaneously determine the polarisation leakages as well as Q and U for the secondary ("good parallactic angle coverage" means at least 5 observations spread over a parallactic angle range of $100°$). This will be the normal case for a 12 hour observation of one source. Alternatively if there is poor parallactic angle coverage of the secondary (e.g. resulting from snapshot observations), the instrumental polarisation derived from 1934-638 can be used, and the Q and U of the secondary can still be determined.

The above strategy differs from that in the previous version of this guide. This is because prior to November 1992, the hardware to measure the XY phase produced unreliable results, particularly at 20 and 13 cm. Thus an observation of a strongly polarised calibrator was strongly advisable to determine XY phase. The strategy used for calibrating data prior to November 1992 will be somewhat different to that presented here.

Calibration of Data with XX and YY Only

Calibrating data containing XX and YY is somewhat simpler. You have insufficient information to solve for instrumental polarisation and the calibrator Q and U. You have no choice but to assume that these are zero. This means that we can skip a number of steps in the calibration process.

All these assumptions will result in some error in the calibration – this is fundamentally unavoidable if only XX and YY are measured. Provided the secondary calibrator is weakly polarised, the error is unlikely to be too substantial.

11.3 Determining Calibration Solutions

There are currently four main tasks which solve for calibration parameters – each with their own strengths and weaknesses. These are:

- `mfcal`: This determines antenna gains, band passes and delays given a point-source calibrator. Multi-frequency observations are supported. Although it can handle dual polarisation data, it assumes the calibrator is unpolarised, and that the polarisation leakages are 0. This assumption does not affect the bandpass or delay that it finds.
- `gpcal`: This determines antenna gains and leakages given a point-source calibrator. The polarisation properties of the calibrator can be determined as part of the solution process. Gains and leakages are assumed to be independent of frequency.
- `selfcal`: This determines a single set of antenna gains given a model of the visibility data. The antenna gains are assumed to be independent of all observing parameters other than time and antenna number. Consequently, its handling of multi-frequency and polarisation data is fairly simple. It can handle multiple pointings and sources (*e.g.* mosaiced observations).
- `gpscal`: This is useful when self-calibrating strongly polarised sources measured with a telescope with linear feeds (*e.g.* the ATCA). Given models of Stokes I, Q, U, and V, as well as the leakage parameters, this solves for the appropriate antenna gains.

We will now describe `mfcal` and `gpcal` in some detail. This is largely for reference – the following chapter will give a step-by-step approach to the calibration process. The self-calibration tasks, `selfcal` and `gpscal`, are discussed in Chapter 15.

11.4 Determining Gains and Bandpass Functions – MFCAL

As noted above, `mfcal` solves for antenna gains and bandpass function. It can also solve for the delay terms, but this is rarely useful. However it does assume an unpolarised calibrator and that the polarisation leakage terms are zero. If you have measured only the XX and YY correlations, you *have to* make these assumptions. However if you have measured all four polarisation products, then you can correct for these assumptions by running `gpcal` after `mfcal`.

We will now discuss some of the inputs to `mfcal`.

- `vis:` The input dataset – the dataset containing the primary calibrator in this case.

- `line:` The standard linetype parameter, used to select and average channels. See the discussion of this in Section 5.4. Assuming the old-correlator's 128 MHz / 33 channel continuum system, you will want to discard the first and last few channels, calibrating only the central 25 channels of so. Otherwise you will not normally want to perform any averaging.
- `edge:` This is an alternative way of indicating the channels to discard. When working with single IF datasets, it is probably easier to use the `line` parameter.
- `interval:` This gives one or two numbers, in minutes, being the maximum solution interval length, and maximum solution gap, in minutes. The latter can usually be allowed to default. The default maximum interval length is 5 minutes. During stable observing conditions (good phase stability) you should set the maximum solution length to no more than the calibrator scan time. Set it to a small value if phase stability is poor (e.g. 1 to 0.1 minutes). If you set it to a smaller value, however, the solution process becomes more sensitive to bad data – make sure you flag it well. Additionally you might run into program memory limitations, where it aborts due to "buffer overflows" or the like. However, it is generally advisable to set this as small as possible.
- `select:` Normal data-selection.
- `refant:` The reference antenna. Set this to the antenna which had the cleanest XY phase. The default is 3 (which may not be very appropriate). It should be noted also that the default antenna will not change from 3, even if antenna 3 is not present in the dataset.
- `flux:` This gives the flux density and spectral index of the calibrator.
- `options:` There are a few extra options, which are not of great use or interest.

  `delay` Note that old versions of `mfcal` attempted to solve for a delay by default. This is no longer the case. To solve for delays (which is strongly discouraged when calibrating a single IF band), you now have to explicitly use the `delay` option.

  `interpolate` The `interpolate` option may be useful if you cannot determine the bandpass for some completely flagged channels but still need an estimate of the bandpass function. With this option, the bandpass is spline fit (in real and imaginary) and evaluated for the flagged channels. Be careful if a substantial fraction of channels are bad – the algorithm is only implemented if less than 50% of the channels are flagged bad.

  `oldflux` If you are calibrated data that are to be combined or compared with ATCA data reduced before August 1994, you will generally want to use the '`oldflux`' option to select the pre-August 1994 ATCA flux scale. See Section 12.6 for more information.

11.5 Determining Gains and Polarimetric Properties – GPCAL

Task `gpcal` is the main workhorse of the *MIRIAD* calibration system, although it is only particularly useful if you have measured all four polarisation products (XX, YY, XY and YX). It has a plethora of options to turn on and off various solvers. Most of these will be irrelevant to normal use – particularly when calibrating a source, such as 1934-638, which is known to be unpolarised. However it does not determine a bandpass function. So, if you have not averaged your data into a channel-0 dataset, you should precede `gpcal` with `mfcal`, or if you are using a separate bandpass calibrator, ensure that its bandpass solution is copied across using `gpcopy` – `gpcal` will normally apply any bandpass function it finds with the dataset before it performs real work.

Do not be intimidated by the number of options – we give you advice on which ones to use in the following chapter. We now discuss the various inputs.

- `vis`: The input dataset.
- `line` can be used to select the range of channels, and the averaging to be performed on a multi-channel dataset. See Section 5.4 for more information. Generally you *should not* specify any averaging. Task `gpcal` performs its own averaging in a fashion which gives best results.

- `interval`: This `interval` parameter has the same meaning as that described in Section 11.4 above, and the advice for setting it is the same.

- `flux` gives the values of the four Stokes parameters for your calibrator source. If you choose to solve for Q and U, (see `options` below) then these are initial guesses only – it is not necessary to give them accurately, or at all, in this case. Make sure you get the signs of Q, U and V right; these are real numbers, not visibility amplitudes. Note that `gpcal` does recognise a small group of calibrators for which it knows the spectrum in I, and sometimes Q and U – see the help file for details. The flux calibrator 1934-638 is amongst these. If `gpcal` does not recognise the source, the default is an unpolarised source. This will be quite inappropriate if you have a polarised source and do not solve for Q and U. If no flux density is given, and the source is not known, then `gpcal` assumes that the rms gain amplitudes are 1 – and determines the calibrator flux density accordingly. This will be a good approximation if you equalised the gains at the start of the observation (which is the normal practise). However, at this stage, this approximation is largely a convenience – any error in the assumption will be corrected later.

- `tol`: The iterative procedure converges when the solutions are unchanged from the previous solutions by an amount `tol`. The default is 0.001. Make this smaller if you want more iterations.

- `xyphase`: This parameter is an artifact of history – it can be ignored when following the current recommended procedure for calibrating ATCA data. This parameter can be used as an alternative way of specifying the XY phase of the antennas. If the XY phase on the reference antenna is constant to good approximation, and provided you *have not* already applied any XY phase correction (i.e. neither in $\mathcal{AIPS}$ or $\mathcal{MIRIAD}$ `atlod` nor with `atxy`), then you can give the XY phases of the antennas here. Unless you explicitly turn off solving for XY phase in `gpcal`, then the only important XY phase value is that for the reference antenna. All the same you should give values for all antennas. One significant catch (for arcane reasons) is that the XY phase reported by $\mathcal{AIPS}$ is not the XY phase used by $\mathcal{MIRIAD}$. They are related by:

$$\phi_{miriad} = \begin{cases} \phi_{aips} - 180 & \text{if sideband indicator is 1} \\ -\phi_{aips} - 180 & \text{if sideband indicator is -1} \end{cases}$$

  The value of the sideband indicator is noted in the history generated by `ATLOD`. More simply, however, it will also be the sign of the frequency increment.

- `options` controls what `gpcal` solves for. Several options, separated by commas, can be given. It is important that you understand the different choices.

  `oldflux` If you are calibrated data that are to be combined or compared with ATCA data reduced before August 1994, you will generally want to use the '`oldflux`' option to select the pre-August 1994 ATCA flux scale. See Section 12.6 for more information.

  `qusolve` means that `gpcal` will solve for Q and U, taking the model you gave in `flux` as a starting point. You need many cuts of the calibrator with good parallactic angle coverage to do this successfully. You should not attempt to solve for Q and U of the flux calibrator, as you will invariably have too little data. Additionally 1934-638 is known to be unpolarised.

  `xyvary:` If the telescope settings are not altered, the XY phases (which are purely instrumental) appear to remain constant to better than a few degrees. By default, `gpcal` assumes the XY phases are constant. Because of the reliability of the XY phase measurement system, this assumption is now known to be inappropriate. It is better to let `gpcal` solve for the XY phases as a function of time – use option `xyvary` for this. Note that `gpcal` can do this for all antennas *except* the reference antenna – the reference antenna is assumed to be constant and (generally) zero.

  `xyref` means that `gpcal` will solve for the XY phase of the reference antenna. To do this, the source must be strongly polarised.

  `noxy` means that `gpcal` will not solve for any XY phases. By default it solves for all XY phases, except for the reference antenna. This is generally not an appropriate switch to use.

  `polref` means that `gpcal` solves for all the leakage parameters of the reference antenna. By default it does not attempt to solve for the misalignment and ellipticity of the X feed of the reference antenna. It is only possible to use this option if the source is strongly polarised (at least 5%).

If you specify Q and U, both these terms can be found. However, if you ask for the `qusolve` option, then the misalignment term cannot be determined.

`nopol` means that `gpcal` does not solve for the polarisation leakage terms. You *must* use this option if you are calibrating data without XY or YX correlations.

`noamphase` means that `gpcal` does not solve for the antenna gains. This option is rarely useful.

11.6 Copying Calibration Tables – GPCOPY

You will often wish to copy calibration tables from one dataset to another (*e.g.* between the calibrator dataset and the program source). This function is performed by `gpcopy`. Task `gpcopy` is quite simple, taking as inputs the input and output dataset names. The `options` keyword can be used to inhibit the copying of various tables using the `nocal`, `nopol` and `nopass` switches.

11.7 Displaying Calibration Tables – GPPLT

Calibration corrections can be examined with the task `gpplt`. This task can list and plot antenna gain, bandpass, delay and leakage terms. Most of the inputs are self-explanatory – see the help file for more information. Only a few parameters will be described here.

- The input dataset is given by the normal `vis` parameter.
- The `device` parameter specifies the PGPLOT device for the plot, whereas the `log` parameter gives the output logfile for the listing. Either `device` or `log` must be set.
- The `options` parameter determines what quantities to plot or list. Several quantities can be given, separated by commas. Possibilities are `gains` (*i.e.* the antenna gains), `polarisation` (the polarisation leakage terms), `delays` and `bandpass`. Additionally for dual polarisation systems, there are two quantities derived from the gains, namely `xygains` and `xbyygain`. `xygains` gives the X polarisation gain divided by the Y polarisation gain (or R divided by L for circular feeds). The phase of this quantity is the much reviled XY phase of the ATCA. xbyygain is the product of the X and Y polarisation gains – it is of more limited use.

  Two other options are sometimes useful. Option `dots` causes the plots to use small dots rather than larger blobs. This can reduce the size of plot files significantly and reduce the printing time. Option `dtime` causes the time axis to be plotted in decimal hours (rather than the normal hh:mm:ss). This can be useful for listings which are to be put into another package (*e.g.* a spreadsheet) for further analysis.
- The yaxis parameter controls whether the amplitude, phase, real or imaginary part of the gain is plotted. Several values chosen from `amp`, `phase`, `real` or `imag` can be given. When several values are given, then several plots will result from a given option for a given baseline.
- The `select` parameter behaves like the normal visibility selection parameter (see Section 5.5) – except that you are selecting gains rather than visibilities. Currently only a subset of the normal `select` commands are supported, and these are not entirely consistently handled. `Antenna` and `time` selection is supported for gains. `Time` selection is supported for delay and spectral correction. `Antenna` and `frequency` selection is supported for band passes.

11.8 Deleting Calibration Tables – DELHD

Occasionally (or perhaps not so occasionally) you might produce a bad set of calibration tables that you wish to delete. This might be particularly important for instances where a calibration program takes an old calibration table as a initial estimate of a new one yet to be derived.

As mentioned above, there can be three important calibration tables in a visibility dataset – the `gains`, `leakage` and `bandpass` items. Although there are subsidiary items associated with each, these are the main items, and by deleting any one of these you effectively prevent the calibration operations associated with that table being attempted.

The task to delete items is `delhd`. For example, to delete the `gains` item of the dataset `multi.uv`, use

DELHD	
in=multi.uv/gains	Delete the `gains` item of `multi.uv`

Chapter 12

Calibration Strategies

12.1 Introduction

The calibration process can be somewhat complicated, with several possible paths. We will describe only the main paths here. It is *strongly recommended* that you follow them. If you stray off them, there is usually a way ahead, but it may not be simple.

We will describe the calibration of data either with two (XX, YY) or all four (XX, YY, XY, YX) polarisation products. However when there are only two products, a number of the steps involved (usually involving `gpcal`) can be skipped.

This chapter discusses calibration with ATCA data loaded using *MIRIAD*. There are differences if the data were initially loaded with *AIPS* – see Appendix D for more details. *MIRIAD*'s software supports calibration of a fairly general interferometric model, including VLA and WSRT polarimetric and non-polarimetric data. Calibration of these is not described here, and is left as an exercise for the enthusiastic reader.

Figure 12.1 gives a flow chart of the steps typically involved.

12.2 Preparing your Data

Before you calibrate, you must prepare your data. This consists of loading, flagging, perhaps converting to "channel-0" datasets, and then splitting.

1. ATCA data will be initially in RPFITS format. *MIRIAD*'s task to read RPFITS files is described in Chapter 8. Alternatively, the data may be in FITS format (e.g. VLA data). Again see Chapter 8 for information on loading data in this format. Generally you will want to load all of an observation into a single *MIRIAD* file (i.e. use `atlod` to read multiple RPFITS files into a single output if necessary).

2. You should now flag your data, and possibly convert to channel-0 (using `uvaver`). Tasks for these operations are described in Chapter 10. For CABB data, you should not convert to channel-0.

   Things to look out for while flagging are:

   - Shadowing: when in a short array, it is possible for one antenna to be partially blocked by another antenna. To flag based on this, use `select=shadow(26)` in your `uvflag` command.
   - Seeing monitor measurements: the seeing monitor is an independent interferometer which constantly measures the atmospheric phase stability to a geostationary satellite. The baseline length is 220m, and the satellite elevation is 60°. The measurement is expressed in microns of excess path, and is contained in the dataset variable `smonrms`. For high frequency observations

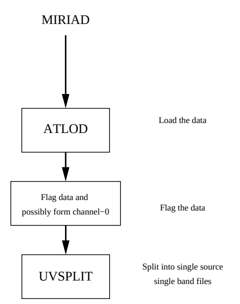

Figure 12.1: Flow chart to prepare for calibration in MIRIAD.

you may wish to discard data whose excess path length rms is greater than some threshold, and this can easily be achieved with the `seeing` parameter that the `select` keyword accepts.

- Tracking accuracy: the tracking accuracy of the ATCA antenna is usually adequate at low frequencies, but at higher frequencies (where the primary beam is smaller), tracking errors become more deleterious. These errors may occur during windy weather (gusts can blow the antennas off the directed tracking position). In addition, poor tuning of the antenna drive systems can sometimes cause poor tracking, particularly after a source change.

  Note that tracking errors should be distinguished from pointing errors. Tracking errors are the inability of the antenna drive system to follow the requested path. Pointing errors correspond to the difference between the position that the astronomer requests and the actual position. Tracking errors are caused by flaws in the drive servo system, whereas pointing errors are a sum of tracking errors and errors in the antenna pointing model.

  Since October 2003, tracking errors have been saved in the MIRIAD dataset. The uv variables corresponding to the rms and maximum tracking error in a cycle are `axisrms` and `axismax`.

  Both the maximum and rms give two tracking error values for each antenna, nominally corresponding to the tracking error in the azimuth and elevation axes. However the ATCA on-line system produces a single composite value, which is replicated for the two axes in the dataset. When the tracking error is less than $2'$, the ATCA on-line system believes that this error is sufficiently small, and thus considers the antenna to be tracking. However this tolerance is not generally good enough for 3-mm observations. To flag based on this criterion, use the `pointing` parameter that the `select` keyword accepts.

3. As MIRIAD datasets can contain only a single set of calibration tables, it is rather poor at handling the calibration of datasets containing multiple sources and multiple frequency bands. For calibration purposes, it is best to work on datasets containing a single source and single frequency band. So, it is best to break the multi-source, multi-band dataset into a collection of single-source, single-band datasets. The best task to do this is `uvsplit`. Task `uvsplit` generates the names of the output datasets itself, forming these from the source name and the central frequency (in MHz) of the data. It is rather unforgiving if you already have files with one of the names that it wants to use. Make sure your directory is free of files that may usurp `uvsplit`'s name choice. Task `uvsplit`

allows you to perform extra selection if you wish, which may be convenient if you only want to deal with part of your observation at a time.

UVSPLIT	
vis=multi.uvxy	The input dataset.
select	Extra selection.

For CABB data, or large spectral line data-sets, if disk space is low, it may be useful not to split off the program source. Rather you can split off the calibrators, determine the calibration tables from them, copy the calibration back to the multi-source file, and then image directly from the multi-source file. This way you avoid making a second copy of your program source data. For example, to avoid the source "`vela`" from being split off, use

UVSPLIT	
vis=multi.uv	The input dataset.
select=-source(vela)	Do not select `vela` data.

There are a few things to look out for when using `uvsplit`. If you don't care about the files already extant in the current directory, you may use `options=clobber` to make `uvsplit` overwrite any files it wants to create.

If the parent file contains more than one frequency band at the same frequency, then `uvsplit` will output two datasets in the normal way, but these will have `.1` and `.2` etc. appended to the end of their filenames, indicating which frequency band the data has come from.

Further to that, sometimes it can be handy to operate on only one IF as an individual dataset, and this can be achieved by using `options=nosource`. This creates a dataset for each IF in the parent file, but will not split out the individual sources. A similar thing can be done to keep all IFs in each source file (`options=nofreq`).

4. Handling CABB's large fractional bandwidths can sometimes cause problems, and it may be preferable to split the band up into smaller bandwidth chunks. This can be done using the `maxwidth` parameter. This parameter should be set to the largest bandwidth allowed in a single dataset, in GHz. For example, to split the 2 GHz CABB continuum bands up into 128 MHz chunks, use

UVSPLIT	
vis=multi.uv	The input dataset.
maxwidth=0.128	The maximum bandwidth for each output dataset, in GHz.

Splitting the data into smaller bandwidth chunks in order to more accurately calibrate it has been dubbed "divide and conquer".

12.3 Calibration

You are now ready to start the main calibration step. You should have a large number of *MIRIAD* datasets, each containing a single frequency band and source. The calibration process will determine bandpass functions (assuming you have not already averaged your data into a channel-0 dataset) and antenna gains as a function of time. If you have measured all four polarisation products, the process will also determine instrumental polarisation and the polarisation of the phase calibrator.

The calibration approach described here (and shown schematically in Figure 12.2 is quite general, and will apply, perhaps with slight modifications, in almost all situations at all wavelengths. However, for high frequency observations made with the old correlator, you may wish to follow the calibration approach described in Section 12.3.1.

You should keep in mind the following points when considering how to calibrate your observations:

- The two most commonly used flux calibrators at the ATCA are 1934-638 and the planet Uranus. For frequencies between 1 GHz and 10 GHz, 1934-638 is the preferred flux calibrator; its cm model is described in the memo "A revised flux scale for the AT Compact Array" (AT Memo 39.3/040). For frequencies above this, Uranus is the preferred flux calibrator, although until early 2010 the $\mathcal{MIRIAD}$ model did not correctly take into account how Uranus' brightness temperature changed with its orientiation to the Earth (a memo describing the new Uranus flux model is forthcoming). Other planets can be used as flux calibrators, but Uranus is the best choice in most situations since it is small enough to remain mostly unresolved on the short arrays that are used for most mm observations, and bright enough to provide an accurate flux in a short observation.

  Since May 2010, the $\mathcal{MIRIAD}$ model of 1934-638 covers frequencies all the way up to the top of the 7mm band (50 GHz), so it can be used a flux calibrator now for all bands other than 3mm. The flux model in the 12mm band is described in the memo by Bob Sault "ATCA flux density scale at 12mm" (2003). The flux model for frequencies above 25 GHz will be described in a forthcoming memo.

- It is possible to use other sources to flux calibrate your observations, but it is inadvisable to do so. *If at all possible, make an observation of 1934-638 or Uranus.*

  To flux calibrate your observation with another source, you must use the "bootstrapping" technique. In essence, this requires that you find an observation of the source you will be using as a flux calibrator within a few days of your observations. Most, if not all of the sources in the AT calibrator database will have fluxes that vary on timescales of days to weeks. If you can find such an observation, and that observation also observed one of the recognised flux calibrators, then you may be able to use the flux density determined from those observations to calibrate yours. However, the longer the time between your observations and the observation you are bootstrapping to, the less accurate this technique is likely to be.

- You should classify the sources in your observations into four categories: bandpass calibrator, flux calibrator, phase calibrator and program source.

  The bandpass calibrator should ideally be the brightest source you observed, or if you want to make a time-dependent bandpass solution, it should be the strongest source that you observed most frequently.

  The flux calibrator selection should be fairly obvious by now; if it isn't, reread the start of this section.

  The program source should also be obvious.

  The phase calibrator should be the source that was observed almost contemporaneously with the program source. There should be as least as many "cuts" of the phase calibrator as there is of the program source, unless you will be relying heavily on self-calibration later in the process. The phase calibrator must also be a point source, with as little confusion and defect as possible.

  The same source can be used for more than one of these roles. For example, 1934-638 can be used as a flux calibrator in all wavelengths between 20-cm and 7-mm, but is bright enough to be used as a bandpass calibrator as well at cm wavelengths.

1. We start by solving for the bandpass solution, using the bandpass calibrator,a ssuming that you have not averaged the data into a channel-0 dataset. To do this we use the task `mfcal`, which is described more completely in Section 11.4. After having solved for the bandpass function, `mfcal` writes a bandpass table in the visibility dataset – the data themselves are not modified. Although it is possible for `mfcal` to cope with datasets that contain multiple IF bands, and that frequency switch with time, we recommend the you *do not* use this feature; it is much easier to stick with datasets with a single frequency band.

   It is also possible for `mfcal` to determine a time-dependent bandpass solution, if you suspect that the instrumental response is changing significantly over the observation. For the CABB correlator's continuum IFs, the amount of bandpass variation observed is usually much less than the gain variations, making a time-dependent bandpass solution rather pointless for continuum observations where a moment map image will be created. However, for spectral line observations, using a time-dependent bandpass can halve the spectral rms noise. To make a time dependent bandpass solution,

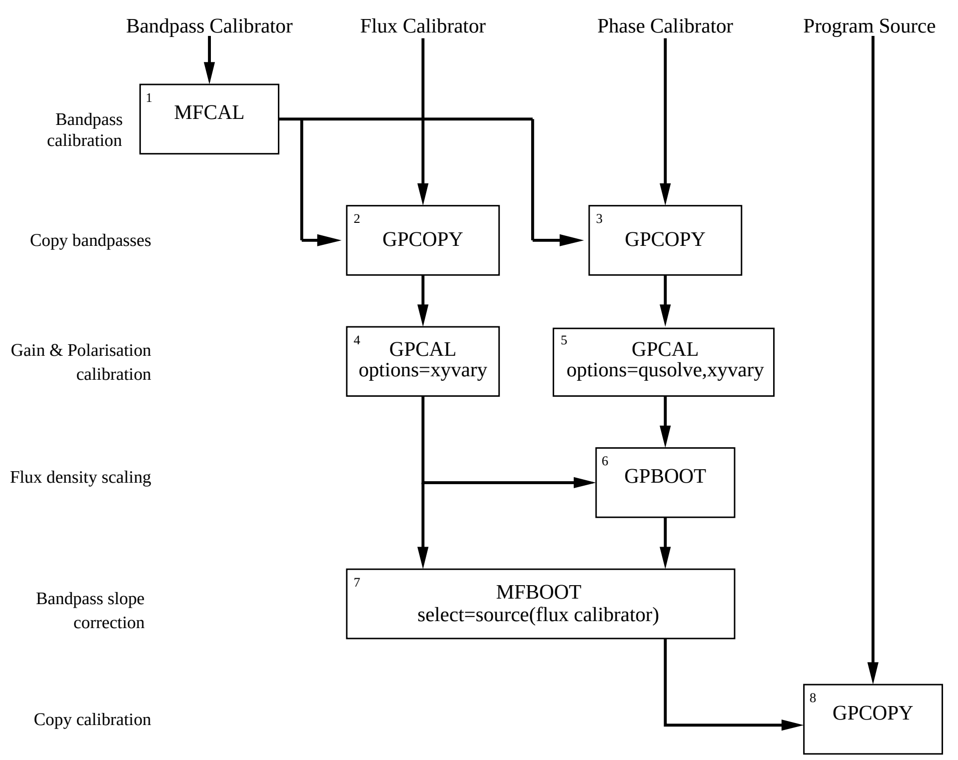

Figure 12.2: Flow chart for calibration in $\mathcal{MIRIAD}$ for most situations.

you will need to specify three numbers for the `interval` parameter in `mfcal`; for a bright source, `interval=1,1,1` should work well, and give a bandpass solution every minute. Of course, the bandpass calibrator would need to be visited often during your observations for this to be effective!

As well as determining a bandpass function, `mfcal` determines antenna gains. These gains are based upon the assumption that the source is unpolarised (which happens to be true for 1934-638) and that the instrumental polarisation is zero (which is not true). These assumptions do not affect the quality of the bandpass.

If your data are to be combined (or compared) with pre-August 1994 ATCA data, you will probably also use the `oldflux` option (see Section 12.6).

MFCAL	
vis=1934-638.5500	Specify bandpass calibrator.
flux	Leave unset.
line	Select good channels.
refant=2	Specify the reference antenna.
interval=1	Solution interval for gain solutions (minutes), or
interval=1,1,1	Solution interval for gains, max interval, bandpass (minutes)
options	Normally leave blank, but
options=oldflux	set to `oldflux` to get the pre-August 1994 scale.

2. The next step copies the bandpass solution you just made to the flux and phase calibrators, and for this you should use the task `gpcopy`. If your bandpass calibrator is the same as your flux or phase calibrator, then you won't need to copy the solution in that case.

   Although `gpcopy` can be instructed to copy only the bandpass solution, there is no harm in copying the gains and leakage tables too, as they will be overwritten when you run `gpcal` later.

GPCOPY	
vis=1934-638.5500	Specify bandpass calibrator.
out=0220-349.5500	Specify phase calibrator.
options	Normally leave blank, but
options=nocal,nopol	You may optionally copy only the bandpass table.

3. We now proceed to determine the antenna gains and instrumental polarisation from the flux calibrator. Task `gpcal` is used for this (see Section 11.5 for more information). Task `gpcal` will generally use the bandpass determined by `mfcal` to partially correct the data. If you are using a planet as your flux calibrator, you must skip this step.

   When your flux calibrator is 1934-638, the only options that you will normally turn on is for the XY phases to be solved for as a function of time; `options=xyvary`. If your data are to be combined (or compared) with pre-August 1994 ATCA data, you will probably also use the `oldflux` option (see Section 12.6). Typical parameters for calibrating 1934-638 are thus

GPCAL	
vis=1934-638.5500	Specify flux calibrator
flux	Leave unset
refant=2	Specify the reference antenna
interval=0.1	Solution interval for gain solutions (minutes)
options=xyvary	Solve for XY phase as a function of time, or
options=xyvary,oldflux	add `oldflux` to get pre-August 1994 flux scale.

Task `gpcal` will report the instrumental polarisation parameters (leakages) – two complex numbers per feed. Typically these are 1 to 2%, although they can be 4% under bad conditions. Typically these are quite constant with time, with similar values resulting from observations several months apart. However they are modestly frequency dependent.

This task will scale the average flux of the entire band to the known flux of 1934-638 at the centre frequency of the band. This is not guaranteed to produce a correct flux scaling at this point. It is very important to remember that `mfboot` will be required later in the procedure to ensure that the flux scaling is accurate.

Tasks appropriate to examine the effectiveness of the calibration are `uvplt` (to generate plots of the calibrated data – remember that many *MIRIAD* tasks apply any available calibration by default) and `gpplt` (to generate plots of the calibration tables – both antenna gains and bandpass functions). A particularly useful plot is the "scatter diagram" plot generated by `uvplt` with the following inputs

UVPLT	
vis=1934-638.5500	Specify flux calibrator
stokes=i,q,u,v	Plot all Stokes parameters.
axis=real,imag	Plot real vs imaginary
options=equal,nobase,nofqav	Equal X and Y axes. Plot all baselines, and don't average over the bandwidth.
device=/xs	Normal PGPLOT device – Xwindows for example

This scatter diagram should show four concentrations of points – one for I at the flux density of the calibrator, and the others for Q, U and V probably near zero. If there are outliers, you probably need to do some more flagging. If you see arcs, the phase calibration is probably bad – you might try decreasing the solution interval.

4. Now we do the same for the phase calibrator using `gpcal` again. The primary difference is that we now wish to solve for the Stokes Q and U values for the source, which are not known a-priori. To do this we add the `qusolve` option. Additionally, if your observations have measured only the XX and YY correlations, you must add the `nopol` option, as there is not enough information to solve for antenna leakages.

GPCAL	
vis=0220-349.5500	Specify phase calibrator.
flux	Leave unset
refant=2	Specify the reference antenna
interval=0.1	Solution interval for gain solutions (minutes)
options=xyvary,qusolve	Solve for Stokes Q and U, or
options=xyvary,qusolve,nopol	Do not attempt to solve for leakages.

In the unlikely event that your calibrator turns out to be more than about 5% polarised, you may wish to run `gpcal` again, but this time you should add options `xyref` and `polref`. This will solve for an XY phase offset on the reference antenna, as well as an instrumental polarisation characteristic that cannot be determined from a weakly polarised source.

If `gpcal` fails with an error stating that the "Solution for requested parameters is degenerate", then it is likely that you do not have enough parallactic angle coverage to solve for Stokes Q and U. In that case, you will have to omit the `qusolve` option, and suffer the subsequent degradation of polarisation calibration.

Good parallactic angle coverage enables you to disentangle the instrumental and secondary polarisations from each other. "Good coverage" generally means more than a few cuts. For sources near declination of -30°, the parallactic angle remains constant through much of the observation, except near transit where it goes through a rapid change, whereas for sources near the poles, parallactic angle changes linearly with time. You can plot your parallactic angle coverage using the task `varplt`. More strictly, it plots the angle between the sky and the feed (i.e. parallactic angle plus 45° for the ATCA).

VARPLT	
vis=0823-500.4800	Specify secondary calibrator
xaxis	Time is the default x axis
yaxis=chi	Plot parallactic angle + 45 degrees
device=/xs	PGPLOT device – xwindows in this case

5. At this point, both your flux and phase calibrators will have the same bandpass solution, but different flux scalings. The following table should make it clear what the calibration has achieved to this point.

	Bandpass Shape	**Bandpass Slope**	**Flux Scaling**	**Phase Solution**
Bandpass calibrator (not flux)	Correct	Incorrect	Incorrect	No
Bandpass calibrator (is flux)	Correct	Correct	Incorrect	No
Flux calibrator (not bandpass)	Correct	Incorrect	Incorrect	No
Flux calibrator (planet)	Correct	Incorrect	Incorrect	No
Phase calibrator (not flux)	Correct	Incorrect	Incorrect	Yes
Phase calibrator (is flux)	Correct	Incorrect	Incorrect	Yes

   The goal is to get the phase calibrator calibrated such that it has the correct bandpass shape and slope, the correct flux scaling, and a useful phase solution. If you are lucky enough to have been observing 1934-638 as a phase calibrator, then you have almost finished your calibration, and you can skip to step 8.

   Before we go any further, we should understand what exactly we mean by each calibration.

   - Bandpass Shape: The bandpass shape is simply the instrumental response to the incoming spectrum. Given a flat spectrum (i.e. equal flux density at all frequencies), the ideal instrument would produce a flat bandpass shape. However, any real instrument will respond to different frequencies in an unequal way, and this gives rise to the bandpass shape which is solved for by `mfcal`.
   - Bandpass Slope: As we said above, a flat spectrum should produce a flat bandpass shape, but what happens when the input spectrum is not flat, as will be the case for most real sources? Unless `mfcal` knows how the flux density of the source varies with frequency, it will have no way to determine whether a change in flux is due to instrumental response or a flux density change of the source. If you have used a source other than 1934-638 for your bandpass calibration, you should assume that `mfcal` has not properly accounted for the source's flux behaviour when it has solved for the bandpass. In these cases, the bandpass slope will be incorrect.
   - Flux Scaling: If the flux scaling is correct, then the flux density at the centre of the band is correct. When `gpcal` (or `mfcal`) solves for the amplitude of a source for which it does not have flux density information, it assumes that the rms gain amplitudes are 1, and determines the calibrator flux density accordingly. While this will be a decent assumption if during array calibration you set the amplitude calibration with 1934-638, it is not guaranteed to be correct. The amount of flagging present through the band can also affect the flux scale after execution of `gpcal`.
   - Phase Solution: The phases (or more accurately, the complex gains) of the cross-correlations will change with time due to atmospheric conditions, system temperature variations, elevation of the antenna etc. To properly account for these effects, it is necessary to observe an unchanging point source throughout your observations. For this reason, calibration of a source that was observed once (or infrequently) during the alotted time will not result in an adequate phase solution, but by bookending each "cut" of the program source with a phase calibrator, you can solve for time-dependent antenna gains.

6. You should examine the calibration solutions for reasonableness.

   Four tasks are recommended for this – `uvflux` (how well does the phase calibrator data fit a point source model?), `uvplt` (plot the calibrated phase calibrator data), `gpplt` (plot the gain solutions) and `gpnorm` (compare phase calibrator solutions with those determined from the flux/bandpass calibrator or elsewhere).

   The first, and simplest, check of the calibration process is to use `uvflux` to check how well the calibrated data fits a point source. Task `uvflux` assumes the data represents a point source, and determines the source flux density and the rms scatter about this point. It also prints out the theoretical scatter based on thermal noise arguments. It will do this for any of the Stokes parameters. Typical inputs are

UVFLUX	
vis=0220-349.5500	Specify phase calibrator
stokes=i,q,u,v	Check all four Stokes parameters.
options	Leave unset (apply calibration).

The first numeric column given for each Stokes parameter is the theoretical scatter. Note if you are using the old correlator's 33 channel / 128 MHz system, and if no channel averaging has been performed, or the `birdie` option has not been used with `atlod`, then the theoretical scatter printed by `uvflux` is a factor of $\sqrt{2}$ higher than the true theoretical value. This is as, for this correlator configuration (and not for others), individual channels are not independent – they overlap by a factor of exactly 2 (in noise bandwidth). Task `uvflux` fails to take account of this. The second and third numeric columns are the mean visibility amplitude and phase respectively. The phase should be near 0 for I, but could be either 0 or 180 degrees for Q and U (assuming there is signal in these!). V should be noise. The fourth column gives the actual scatter. This should be close to the theoretical value. Do not be concerned if it is a factor of 2 or so bigger. If it is more than a factor of a few greater than the theoretical, you probably still have bad data or a bad gain solution.

Using `uvplt` to plot the calibrated data is also recommended. As with the flux calibrator, a scatter diagram plot is quite useful and quick.

UVPLT	
vis=0220-349.5500	Specify phase calibrator.
stokes=i,q,u,v	Plot all Stokes parameters.
axis=real,imag	Plot real vs imaginary
options=equal,nobase	Equal X and Y axes. Plot all baselines together.
device=/xs	Normal PGPLOT device – Xwindows for example

Outliers in the scatter diagram probably indicate bad data – you might want to go back and flag some more, and redo some calibration steps. Note, however, that the ultimate objective of the calibration process is to get good calibration solutions – you are not attempting to produce perfect flux or phase calibrator data. If you do not believe that outliers affect the calibration solutions unduly, ignore them.

Another check is to plot the gain solutions and inspect them for consistency – use `gpplt` for this (see Section 11.7). Typical inputs for plotting the solved-for XY phases are given below. These XY phases are the difference between the actual XY phases and the XY phase correction applied by `atlod`. These should be constant (within the noise) and no more than a few degrees.

GPPLT	
vis=0220-349.5500	Specify phase calibrator
device=/xs	PGPLOT device – X windows in this case
yaxis=phase	Plot the phase of the ratio
options=xygains	of the X and Y gains

In addition to plotting these solved-for XY phases, it is probably worthwhile plotting antenna and bandpass gains and phases. If you see glitches in the solutions, check the data again. This might be due to bad data or interference. You might do some more flagging, and redo some of the calibration again.

A final check is to compare the instrumental polarisation solution (the leakage parameters). You can compare these with another independently derived set of these solutions. These solutions are moderately time independent (they are moderately consistent over months), although there is significant frequency variation. The independent set of solutions may have come from the flux calibrator (e.g. if you determined instrumental polarisation for the flux and phase calibrators independently – which was recommended if you had sufficient parallactic angle coverage and your flux calibrator was not a planet), or from a previous configuration, or possibly other phase or bandpass calibrators from the same observing run. The task to compare the different solutions is `gpnorm`. Apart from just taking some sort of difference between two sets of parameters, it adjusts certain parameters to minimise the difference. The parameters that it adjusts are the absolute feed orientation and ellipticity – two quantities that are not solved for in the preceding calibration process (unless you had a strongly polarised calibrator, and used `options=polref` in `gpcal`). Task `gpnorm` can also deduce an error in the absolute XY phase between the two observations (an error in the absolute XY phase leaves a signature in the instrumental polarisation parameters).

Task `gpnorm` reports three numbers – an XY phase offset, the offset in absolute orientation and ellipticity, and a residual rms error. As the preceding calibration process should have corrected for the XY phase offset, this should be no more than a few degrees. The offset in absolute orientation and ellipticity should be no more than 2-3%. The rms residual error should be no more than 0.005. You would expect the agreement to degrade with time difference in the observations used to derive the parameters. Agreement should be very good for parameters derived from observations on the same day, whereas agreement should be less good for observations several months apart.

Having solved for an offset in absolute XY phase, orientation and ellipticity, `gpnorm` can "correct" the calibration tables for these. This is generally not advisable.

Typical inputs to `gpnorm` are

GPNORM	
vis=0220-349.5500	The leakages to check.
cal=1934-638.5500	The 'good' set of leakages.
options	Leave unset.

7. What this step will accomplish will depend on whether your flux calibrator is a planet or not.

   If your flux calibrator is *not* a planet, then this step will correct the phase calibrator's flux scale. To do this we use the task `gpboot`. This task takes two inputs: `vis`, which is a dataset with a flux scale that needs correcting, and `cal`, which is a dataset with the reference – or in this case, correct – flux scale.

   In principle, you should use observations of the phase and flux calibrators that were taken at the same time and elevation, because atmospheric opacity affects the amplitude gain calibration. For obvious reasons this is not always possible, although the further apart the observations are in time and elevation, the less accurate the flux calibration will be, especially at high frequencies. However, you can use the `select` keyword to select the data from the phase calibrator that will give you the best flux calibration. In stable weather conditions, it is probably best to select the time range of the phase calibration where its elevation corresponds most closely to the elevation during the flux calibrator's observation. When the weather changes significantly during the observation, it is probably best to use a time range of the phase calibrator which is close to the flux calibrator's observation. Tasks `uvplt` and `varplt` can plot source elevation as a function of time.

GPBOOT	
vis=0220-349.5500	Specify phase calibrator.
cal=1934-638.5500	Specify flux calibrator.

   The output of `gpboot` will be a number indicating how large a scaling adjustment was made, with 1.000 being no adjustment at all.

   If your flux calibrator *is* a planet, then this step will transfer the gain scaling from the phase calibrator to the flux calibrator, again using `gpcal`. This time though, the reference flux scale will be that of the phase calibrator. This step is necessary because the task that adjusts the flux scales using a planet requires that all the input datasets (which will be the flux and phase calibrators) have the same flux scale to begin with. It is more than likely that at this point both the flux calibrator and phase calibrator will have the same (or very similar) flux scalings anyway, since the operation of `mfcal` on the bandpass calibrator and `gpcal` on the phase calibrator should be similar enough to produce consistent scalings. Therefore, you should expect that the scaling factor applied by `gpcal` should be very close to 1, otherwise there may be a problem with your data.

GPBOOT	
vis=uranus.43000	Specify flux calibrator.
cal=0220-349.43000	Specify phase calibrator.

The state of the calibration at this point is shown in the table below. Note that we are not attempting to further calibrate the bandpass calibrator. It is of course possible to calibrate the bandpass calibrator, in order to measure its flux (or something similar), but this is left as an exercise for the reader.

	Bandpass Shape	**Bandpass Slope**	**Flux Scaling**	**Phase Solution**
Flux calibrator (is bandpass)	Correct	Correct	Incorrect	No
Flux calibrator (not bandpass)	Correct	Incorrect	Incorrect	No
Flux calibrator (planet)	Correct	Incorrect	Incorrect	No
Phase calibrator (flux is a planet)	Correct	Incorrect	Incorrect	Yes
Phase calibrator (flux is not a planet)	Correct	Incorrect	Incorrect	Yes

8. At this point in the calibration, we need to correct both the bandpass slope of the phase calibrator, and the flux scaling. This step will accomplish both, using the task `mfboot`.

   For a known flux calibrator, be it a point source or a planet, `mfboot` constructs a model of what the flux should be on each baseline and for all frequencies across the band. It then determines by how much the gains must be scaled by to make the observed flux at the centre of the band match the model value. It then scales all the datasets it has been given by that factor.

   It also calculates the slope of the model between two points either side of the band centre, and then corrects the bandpass table to make the observed spectrum match that slope. Previous versions of `mfboot` did not perform this bandpass correction since spectral slopes were not as apparent across the narrow bands offered by the old correlator.

   The task operates a little differently to `gpboot`, which has separate inputs for the uncorrected dataset and the reference dataset. In contrast, `mfboot` has only one input type that takes a comma separated list of datasets, all of which are assumed to have the same bandpass and gain tables. The source to use as the flux calibrator is then selected using the standard `select` parameter.

MFBOOT	
vis=1934-368.5500,0220-349.5500	Specify all the input datasets, or
vis=uranus.43000,0220-349.43000	Specify all the input datasets.
select=source(1934-638)	Specify the flux calibration source, or
select=source(uranus)	Specify the flux calibration source
options	Usually leave this unset, but
options=noapply	Useful if you want to see what scaling will result, without applying it

   This task will output the flux scaling and bandpass slope correction factors it applied to each of the datasets.

   The state of the calibration at this point is shown in the table below.

	Bandpass Shape	**Bandpass Slope**	**Flux Scaling**	**Phase Solution**
Flux calibrator	Correct	Correct	Correct	No
Phase calibrator	Correct	Correct	Correct	Yes

9. All that remains is to transfer the calibration tables to the program source. Again we use `gpcopy` to do this. Typical inputs are

GPCOPY	
vis=0220-349.5500	Specify phase calibrator
out=vela.5500	Specify the program source dataset
options	Leave unset
mode	Leave unset

12.3.1 An alternative approach to calibration

This calibration approach is recommended when dealing with high frequency data from the old ATCA correlator, especially at 3mm wavelengths. The calibration at these wavelengths is more difficult because of weak calibrators, poorer system sensitivity and more challenging atmospheric conditions. The approach described here exploits the dual frequency bands that were available with the old correlator. The steps are as follows:

1. The first step is not a step at all! Unlike the conventional approach, the approach described here *does not* split the data into separate single source, single frequency files before calibration. Rather the current approach assumes that you have a single *MIRIAD* dataset containing bandpass calibrator, phase calibrator, flux density calibrator and program source. We focus on data having two simultaneous frequency bands here, but the approach works equally well for one or many simultaneous frequency bands within the dataset.

2. The first real step is akin to a bandpass calibration step. When dealing with two frequency bands, each with two polarisation products (XX and YY), it is generally appropriate to assume that the bandpasses of the bands and the phase offsets between bands and polarisations are constant with time.

   To determine the bandpasses and the phase offsets, you need to use `mfcal` on an observation of a strong continuum source. Typically this would be 1253-055, 1921-293 or 0537-441. Typical inputs are as follows:

MFCAL	
in=vela.fixed.uv	Input multi-source, dual-band dataset.
select=source(1921-293)	Select the strong continuum "bandpass" calibrator.

   The important output of this step is a bandpass calibration table which will contain the relevant phase offsets. The output also contains antenna gain calibration tables, but this will be overwritten below.

   In doing this step, you may well want to avoid using some edge channels in the bandpass calibration process. To achieve this, you can use the `edge` parameter of `mfcal`. The `edge` parameter sets the number of channels to drop for all bands equally. It does not allow you to set different number of channels to drop in the different bands. If the channels and bandwidth characteristics of the different bands differ significantly, the `edge` parameter probably does not give you sufficient flexibility. In this case, you could use `uvflag` to explicitly flag edge channels.

3. The next step is to determine the antenna gain calibration, again using `mfcal`. To do this, you will want to select those data for the phase calibrator which will give the best overall sensitivity appropriate to your observation.

   - If the calibrator is a continuum source, you need only select this source and use both bands. Task `mfcal` will weight the two bands according to their relative bandwidth. Consequently if one of the bands is quite narrow relative to the other, then it will contribute little to the overall gain solution.
   - If the calibrator is an SiO maser, you will want to select only those channels that contain strong SiO signal.
   - With either a continuum calibrator or an SiO maser, it is possible to either determine antenna gain calibrations for the two polarisation products independently, or to determine a single joint solution for the two polarisations. The latter will give a $\sqrt{2}$ improvement in sensitivity in the solution process. The latter is also the desirable approach if the calibrator is strongly polarised (some SiO masers are more than 50% polarised). Doing a joint solution is generally a reasonable approximation given that the antenna gains are dominated by changes common to both polarisations: in amplitude, it is from the physical antenna gain change, and in phase it is from the atmosphere and instrumental contributions common to both polarisations.

Note in this calibration scheme, it is implicitly assumed that the antenna gain changes are common to both frequency bands. This is a reasonable approximation given that the frequency separation between the bands is at most 2.7 GHz and potentially much less. Consequently the fractional frequency separation is at most a few percent, and so the calibration change between bands can be assumed to be minimal.

Typical inputs for `mfcal` follow. This assumes a continuum source as the phase calibrator and does a joint solution for the two polarisations:

MFCAL	
vis=vela.fixed.uv	Input multi-source, dual-band dataset.
select=source(1622-297)	Select phase calibrator.
options=nopassol	*Do not* solve for bandpass.
stokes=i	Do a joint solution for both polarisations.

To derive a separate solution for the two polarisation products, the `stokes` keyword would be left unset. Note the use of `options=nopassol`. This causes `mfcal` to not attempt to solve for the bandpass and phase offsets again, but rather to apply the previously determined bandpass and phase offset solutions.

Typical inputs when using an SiO maser would be as follows: Typical inputs to `mfcal` are as follows:

MFCAL	
vis=vela.fixed.uv	Input multi-source, dual-band dataset.
select=source(oceti)	Select SiO secondary calibrator.
options=nopassol	*Do not* solve for bandpass.
stokes=i	Do a joint solution for both polarisations.
line=chan,20,45	Select the range of channels where the SiO signal is strong.

At this point, you have a dataset that has been bandpass calibrated, and whose complex gains are determined as a function of time. All the remaining steps can be completed the same way as in the conventional approach: continue on from step 6 in Section 12.3.

12.3.2 Information on planets

Given the observing time and frequency, task `plplt` plots the expected visibility function for a planet. You will most likely want to check this visibility function before observing and discovering that the planet is resolved out! **NOTE** that `mfboot` and `plplt` do not include the effect of the ATCA's primary beam – the planet is implicitly (and silently) assumed to be small compared with the primary beam. This may not be the case at 3-mm. If there is a significant change in the visibility function between baseline lengths of 0 and 22 metres, then primary beam effects are important, and you probably do not want to use the planet (most likely it would be resolved out anyway).

Task `planets` can give you basic information about a planet, including rise and set times (but note these are quoted at the horizon, not the ATCA's elevation limit), the planet's angular size at a particular time, and its RA and DEC (note that the ephemeris used for RA and DEC is not sufficiently accurate to point or phase track the telescope).

Information on using the ATCA to observe a planet is available on-line at http://www.narrabri.atnf.csiro.au/observing/users$_{g}uide/html/new_{a}tug_{2}0.htmlScheduling-Strategy$

12.4 The Interpolation Process

You should now have a program source dataset containing the appropriate calibration solutions. The antenna gain solutions have been derived from observations of a phase calibrator, near the program source, taken typically for 2 minutes every fifteen minutes or so. The program source antenna gains are derived by interpolating and extrapolating these gain solutions. Although these can often be skipped, there are a few steps that you might consider doing to improve the interpolation process.

Interpolation/Extrapolation Tolerance

Clearly there is some limit to the interval of time over which you can sensibly interpolate or extrapolate gains. If the gap between the phase calibrator observations is too large, then the interpolation of the gain solutions may have no resemblance to the true gains at a given time. The software that interpolates or extrapolates between the gain solutions has a tolerance which limits the gaps in time that it will permit. If the time differences between a source observation and two gain solutions bounding it are both less than the tolerance, then the program source gain is the interpolation of the two gain solutions. If there is only one gain solution within the time tolerance, then this is used without any interpolation. If there are no gains within the time tolerance, the corresponding program source data are marked as flagged.

This interpolation tolerance is stored as the `interval` item the visibility dataset. It is given as a double precision number with units of days. Task `gpcal` sets its value as half a day (`interval=0.5`). Although this is excessively generous, this will not be a problem unless there are large gaps between calibrator scans (perhaps because some calibrator scans had to be discarded). You can set the set the interpolation tolerance using `puthd`. For example

PUTHD	
in=vela.5500/interval	Set the interpolation tolerance of dataset vela.5500.
value=0.1	Set the tolerance to 2.4 hours (0.1 days).

Averaging Antenna Gains with Time – GPAVER

If the phase stability is bad during the observation (the atmospheric conditions are poor), the phase of the antenna gains can vary rapidly. A change of tens of degrees or more during the calibrator scan is not uncommon. However the software that determines the antenna gains assumes that the gains are constant during a solution interval. So during periods of poor phase stability, it is often desirable to make the solution interval of the calibration software quite short. While the resultant gains probably track the phase during the calibrator scan, what we are really interested in are the antenna gains for the program source. If you solved for a number of time intervals during the calibrator scan, the best guess at the antenna gains for the program source is derived by interpolating between some average of the calibrator scan gains. Thus after determining the gains at a fine time step in the calibrator scan, you should average these gains together to get some representative gain for the whole calibrator scan. This is probably the best guess you can make (at least as far as correcting the program source is concerned), although in times of truly awful phase stability, its a pretty poor guess (self-calibration will be needed in this case).

There are two ways to average your gains, either the "vector" or "scalar" averages. Which is the most appropriate will depend on whether you want good estimates of the gains or good estimates of the resultant images.

- Scalar averaging consists of averaging the amplitude of the gains separately (the 'average' phase is still determined by a traditional (vector) average of the real and imaginary parts of the gains). Assuming the variation in gain is purely due to poor phase stability, a scalar average will give you a good estimate of the amplitude of the gain. If you are going to self-calibrate later, then scalar averages are probably the most appropriate. This means that when you come to self-calibrate, you only have to solve for the phase (at least initially), because you already have a good amplitude estimate.
- On the other hand, if you are not going to self-calibrate, you are more interested in getting a good estimate of your image at this stage, and less concerned about partially correct gains (the two do not necessarily go hand in hand). If we were to use vector averaged antenna gains (averaging the real and imaginary parts), the poor phase stability will cause partial decorrelation in both the

program source and calibrator. This will result in apparent reduced flux densities of both of them. Assuming that the decorrelation is approximately the same for both, then we could scale up the program source by the decorrelation that we note in the calibrator. This can be achieved by vector averaging the antenna gains.

To summarise the above discussion, when phase stability is poor, it is best to use a very short solution interval when solving for the antenna gains of the calibrator scan. The gains during a scan should then be averaged before applying them to the program source. Scalar averaging is appropriate if self-calibration is to be used later. Otherwise vector averaging should be used.

The task `gpaver` can be used to perform averaging of antenna gains. Its pretty straightforward.Typical inputs are

GPAVER	
vis=vela.5500	Average the gain table for the program source
interval=2	Averaging interval (calibrator scan length)
options=scalar	Scalar average if self-calibrating later,
or	
options=vector	vector average otherwise

Preventing Gain Interpolation – GPBREAK

Up until now we have assumed that the antenna gains vary smoothly with time, and so it is appropriate to interpolate between them. However this is not always the case. If there is a major instrumental glitch between calibrator scans, then it may be more appropriate to attribute all the antenna gain difference to this glitch. In this case, interpolating the gain solutions will probably give poorer results than assuming constant gains before and after the glitch.

The task `gpbreak` can be used to insert a 'break-point' into the antenna gains table, which indicates to the software which applies the calibration, that interpolation across this point should not be performed. Rather the gain immediately before and after the break-point should be the gain from the previous or next calibrator observation.

Inserting a break-point modifies the antenna gains table, and so it is only meaningfully performed after the antenna gains table has been determined. To insert a break-point, you will need to know the exact time of the glitch. You may well know this from the observing log, or you may be able to deduce it from the program source data if you are observing a strong source.

Task `gpbreak` takes as its parameters the name of the visibility dataset (`vis` parameter), the times at which to insert break-points (`break` parameter), and the antennas and feeds for which to insert the break-points (parameters `ants` and `feeds` – the default is to insert break-points for all antennas and feeds). The `break` parameter gives the time in normal $\mathcal{MIRIAD}$ format – either *hh*:*mm*:*ss* or *yymmmdd*:*hh*:*mm*:*ss*. The second form (giving year, month and day) will generally not be required for a single day's observing. Several times can be given, separated by commas. For example:

GPBREAK	
vis=vela.5500	Insert break points into the program source.
break=20:30:10,21:15:45	Give the times of the glitches.
ants=1,2	Only antennas 1 and 2 were affected.
feeds=x	Only the X feeds were affected.

Unfortunately break-points do not survive recalibration. If you re-determine the antenna gains (*e.g.* re-run `gpcal` or the like), then you will overwrite the old antenna gains table, and will need to re-insert the break-points.

12.5 High precision polarimetric calibration

The ATCA potentially can produce polarimetric data to very high precision, with polarimetric measurements down to the level of 1 part in 100000 of Stokes I. The extra steps to attain this are beyond the scope of this manual. David Rayner's memo "Circular polarization user's guide" (2000) contains much useful information on this topic.

12.6 Combining Pre- and Post-August 1994 Data

In August 1994, the nominal flux density of the ATCA's primary flux calibrator, 1934-638, was revised to give better agreement between the ATCA's flux density scale and the flux scales used in the northern hemisphere (the VLA in particular). For more discussion, see John Reynolds' memo "A revised flux scale for the AT Compact Array" (AT Memo 39.3/040).

This introduces a problem when mixing data (either combining or just comparing) calibrated before and after this change. By default *MIRIAD* tasks use the new flux scale. However if you use the option 'oldflux' in `gpcal` and `mfcal` when calibrating 1934-638, the old flux scale will be propagated to the calibrated data. In this way, new data can be calibrated to have the old flux scale, and so can be combined/compared directly with data calibrated before the revision.

Chapter 13

Imaging

13.1 Introduction

The task `invert` is used to produce an image (or image cube) given a collection of visibility datasets.

Even if you are only interested in continuum imaging, your data may contain several frequencies. Unless you have averaged over frequency channels earlier, your data still contains the 2048 channels produced by the CABB correlator, or 33 frequency channels produced by the old ATCA correlator. Additionally as the ATCA can observe two frequency windows simultaneously, and if these two windows are relatively close (within about 25% of each other), then you can get extra sensitivity and improved $u-v$ coverage by combining the two sets of data during imaging. Indeed you can enhance your $u-v$ coverage further (but not sensitivity) by time-switching the frequency setup used, and so you might have observations at several frequencies, perhaps from different configurations.

The net result is that you possibly have data observed at multiple frequencies, and you are faced with the question of how to make an image from these data. Generally, unless perhaps your observation consists of a single frequency band at 3 cm, now is *not* the time to do any more averaging. It is better to let the imaging task, `invert`, do its job with the UN-averaged data. By giving `invert` data that has not been averaged in frequency, it can grid the individual channels at their correct location in the $u-v$ plane (rather than some average location). In this way bandwidth smearing is reduced and the better $u-v$ coverage results in a better beam. The advantages of doing this decrease with the fractional bandwidth that your data occupies. With CABB data, a 2 GHz band represents a large fractional bandwidth even at 20 GHz, so frequency averaging is not recommended for CABB data at all. In the past, averaging was useful as a computational time and disk space saver, but with modern processing speeds, it is hardly necessary even with CABB data, and certainly not for data from the old ATCA correlator.

The technique of forming a single continuum image from a variety of frequencies is called "multi-frequency synthesis" (MFS). It produces an image with a frequency corresponding to some average of the frequencies of the input data. There is nothing particularly magical about combining different frequencies to produce a single continuum image. This is done all the time (interferometers are never monochromatic). The advantage of MFS is that it can take a greater fractional spread in frequencies before artifacts become a problem.

Eventually MFS will break down because of the variation in the source's flux density with frequency. If the fractional spread in frequencies is small ($< 10\%$) or if high dynamic range imaging is not required (dynamic ranges less than a few hundred or so), then spectral index effects are unimportant. You can make an MFS image, and deconvolve it using the conventional deconvolution tasks (if deconvolution is required). Otherwise, spectral index effects are likely to be important, and you should use a special task (`mfclean`) for the deconvolution. Note that the decision to use MFS in the imaging *does not* require you to use `mfclean` in the deconvolution. While most ATCA continuum observations will benefit from using MFS in the imaging, not as many will benefit from accounting for spectral index effects in the deconvolution. However, at low frequencies, it is likely that a large fraction of sources will display some variation of flux density over the large CABB bandwidth, making `mfclean` an important step.

13.2 Spectral Line Imaging Considerations

Although you do not have to decide whether to use MFS or not for spectral line observations (clearly you cannot), there are a few other issues you must consider. These are discussed in Chapter 16 in more depth – we only list some pointers here. Some questions to ask yourself are:

- Do I need to remove the continuum from the visibility data?
- Is the velocity information in the dataset correct? What velocity system convention am I interested in?
- Do I need to account for the Doppler effect caused by the Earth's motion?

13.3 Weighting

Each visibility sample is given a weight in the imaging step. This weighting can be used to account for differences in the density of sampling in different parts of the $u-v$ plane, or to account for different noise variances in different samples, or to improve sensitivity to extended objects, etc. Here we briefly review some weighting schemes:

Natural weighting This gives constant weights to all visibilities (or, more strictly, inversely proportional to the noise variance of a visibility). This weighting gives optimum point-source sensitivity in an image. However the synthesised beam-shape and sidelobe levels are usually poor.

Uniform weighting This gives a weight inversely proportional to the sampling density function. This form of weighting minimises the sidelobe level. However the noise level can be a factor of 2 worse than natural weighting.

Super- and sub-uniform weighting Uniform weighting computes the sampling density function on a grid that is the same size as the gridded $u-v$ plane. This results in the synthesised beam sidelobes being minimised over the same field-of-view as the region being imaged. Surprisingly, making the field-of-view very large (bigger than the primary beam size) or very small (comparable to the synthesised beam) both cause uniform weighting to reduce to natural weighting. Super- and sub-uniform weighting decouple the weighting from the field size being imaged. Instead, the sidelobes in the synthesised image are minimised over some arbitrary field size, with this field being either smaller or larger than the field being imaged (for super- or sub-uniform weights respectively).

Robust weighting Uniform weighting (including super- and sub-uniform weighting) minimises sidelobes, whereas natural weighting minimises the noise level. Robust weighting provides a compromise between the two, doing so in an optimal sense (similar to Wiener optimisation). See Dan Briggs' thesis for more information.

Tapering In signal processing theory, the optimum way to detect a signal of known form, which is buried in noise, is to convolve that signal with a "matched filter". This filter has an impulse response which is just the reverse of the form of the signal that is being detected. Applying this principle to detecting sources in radio interferometry, the optimum weighting for detecting a Gaussian source is to weight the visibility data by a Gaussian. This is often called 'tapering'. Using a Gaussian weight will significantly increase the detectability of an extended source. However it also degrades the resolution. Gaussian weighting can be combined with any of the above weighting schemes to achieve some form of balance between sidelobes and sensitivity.

MIRIAD gives good (excessive?) control over the visibility weighting schemes, via three parameters and one option.

- `fwhm` controls tapering of the data. Unlike *AIPS*, this taper is specified in the image domain, in arcseconds. If you are interested in features of a particular angular size, then the signal-to-noise ratio in the resultant dirty image is optimised for these features if the taper FWHM is the same as the source FWHM (or source scale size).

- `sup` is used to control super- and sub-uniform weighting. The `sup` parameter indicates the region of the *dirty beam* (centred on the beam centre) where sidelobes are to be suppressed or minimised. Like the `fwhm` parameter, the `sup` parameter is given in arcseconds. The weights that `invert` calculates are optimal, or near optimal, in a least-squares sense to minimising the sidelobes in the specified region of *the beam*. Although the sidelobe suppression region is not a direct control of resolution and signal-to-noise ratio, it does have an effect on these characteristics.

  Suppressing sidelobes over the entire field of the dirty beam corresponds to uniform weighting – that is, we get uniform weighting if `sup` is set to the field size of the dirty beam; this is the default. Alternately making no attempt to suppress sidelobes (`sup=0`) corresponds to natural weighting.

  Increasing `sup` from 0 to the field size results initially in an improvement in resolution until the value of `sup` is approximately equal to the best resolution. Increasing `sup` beyond this results in a slow degradation in resolution. The noise level varies in a less regular way with `sup`. Apart from saying that `sup=0` (natural weighting) gives the optimum signal-to-noise ratio, it is not possible to generalise. However the noise level will be no worse than a factor of a few from the optimum.

  The default value for `sup` is the field size (*i.e.* uniform weighting).

- `robust`: In $\mathcal{MIRIAD}$, robust weighting is parameterised by a "robustness" parameter. Values less than about -2 correspond essentially to minimising sidelobe levels only, whereas values greater than about +2 just minimising noise. A value of about 0.5 gives nearly the same sensitivity as natural weighting, but with a significantly better beam.

- `options=systemp`: The basic weight of a visibility (the weight used for natural weighting, or the weight used for a point in determining the local sampling density function) is ideally $1/\sigma^2$, where σ^2 is the noise variance of a visibility. Because historically the value for this in a $\mathcal{MIRIAD}$ data-set has been of dubious reliability, `invert` normally assumes the noise variance is inversely proportional to the integration time of a visibility. However ATCA data loaded with $\mathcal{MIRIAD}$ `atlod` will contain good estimates of σ^2: using the true σ^2 will result in some improvement in sensitivity in the final image. In this case, the `systemp` option can instruct `invert` to compute the basic weights using σ^2.

13.4 Computation

Task `invert` is a fairly conventional imaging program, which produces a dirty image from a visibility dataset. It normally does this using a grid-and-FFT approach, although there are switches to use a direct Fourier transform and a median algorithm. Task `invert` does not require the data to be sorted in any way. Normally any calibration tables are applied by `invert` on-the-fly (although this can be turned off with the `nocal`, `nopol` and `nopass` options). Both continuum or spectral line observations are handled.

We describe the inputs to `invert`. For MFS imaging, note `options=mfs` (and `options=sdb`) options.

- `vis` gives the name of the input visibility datasets. Several datasets can be given, as may be convenient when a source is observed with multiple configurations or multiple frequencies. The selected data are assumed to be of the same object, with the same pointing centre. Additionally, when making spectral line cubes, the number of channels derived from each dataset must be the same (this restriction does not apply for MFS images). Dataset names can include wildcards.

- `map` is the name of the output image dataset(s). When several Stokes parameters are being imaged, you need to give one name for each Stokes type. There is no default.

- `beam` is the name of the output beam dataset. The default is not to create an output beam. If you wish to deconvolve, then you must create an output beam. Note that `invert` produces a single beam which corresponds to all image planes and Stokes parameters.

- `cell` gives the image pixel size in arcseconds. Either one or two values can be given. If only one value is given, the pixels are square. The default is to choose a pixel size which samples the synthesised beam by about a factor of 3 (i.e. the recommended sampling for deconvolution).

- `imsize` gives the size of the images in pixels. It need not be a power of two. Generally the beam is also this size, but see `options=double` below.
- `line` is the normal data linetype selection. The default linetype is the first channel when performing normal imaging and all channels when doing multi-frequency synthesis. Generally you should set this parameter if you have more than one channel.
- `select` is the normal visibility data selection. The default is to select all data.
- `stokes` gives the Stokes or polarisation types to be imaged. Several values can be given, separated by commas. For example

  ```
  stokes=i,q,u,v
  ```

  will cause images of Stokes I, Q, U and V to be formed. Note that there needs to be a corresponding output file (see `map` above) for each Stokes parameter being imaged. The default Stokes parameter to image is 'ii', which images Stokes-I, and assumes that the source is otherwise unpolarised.
- `offset`: This gives an offset, in arcseconds on the sky, to shift the image centre away from phase centre. Positive values result in the image centre being to the north and east of the observing centre. If one value is given, both RA and DEC are shifted by this amount. If two values are given, then they are the RA and DEC shifts. The default is no shift.
- `options` gives extra processing options. Several options can be given (abbreviated to uniqueness), separated by commas. Possible values are:

  `mfs:` This invokes multi-frequency synthesis imaging, as described above. All selected channels and frequencies will be gridded simultaneously to form a single output image.

  `sdb:` This is used in MFS imaging to cause `invert` to produce a spectral dirty beam (this is stored as a second plane of the beam data-set). The spectral dirty beam is used by the deconvolution task `mfclean` in determining the spectral index of the image. Generally if you think you *may* use `mfclean` to deconvolve your image, then you should set `options=sdb`. See the discussion of multi-frequency synthesis in Section 14.4 for more information.

  `double:` Normally `invert` makes a beam which is the same size as the image. However *MIRIAD*'s deconvolution tasks generally require a beam which is four times the area than the region being deconvolved. The `double` option causes `invert` to produce a beam which is twice the linear extent (four times the area) of the image. In this way, you can request `invert` to map just the region containing real emission, plus a guard band (at least 5 pixels on each edge).

  `systemp:` See the description in the weighting section above.

  `nocal:` By default, `invert` applies any gain calibration to the data before imaging. The `nocal` option turns off this step.

  `nopass:` By default, `invert` applies any bandpass calibration to the data before imaging. The `nopass` option turns off this step.

  `nopol:` By default, `invert` applies any polarisation leakage correction to the data before imaging. The `nopol` option turns off this step.

  `mosaic:` This invokes `invert`'s mosaic mode. See Chapter 21 for more information.

  `imaginary:` This makes the 'imaginary' image. This is useful for non-Hermitian data (holography) or for investigating certain instrumental errors.

  `amplitude:` Set the phase of each visibility to zero before imaging.

  `phase:` Set the amplitude of each visibility to unity before imaging.
- `mode:` This determines the imaging algorithm to be used. Possible values are `fft` (a conventional grid and FFT imaging algorithm), `dft` (a direct Fourier transform approach) and `median` (a median Fourier transform). The `dft` approach produces fewer artifacts in the resultant images, but at substantial computational expense. The `median` approach is more robust to bad data and sidelobes, but at an even more substantial computational cost. The default mode is `fft` – this should be used on all but the smallest datasets and images.

- `slop`: The slop factor. This is used in spectral-line imaging. See Section 16.10 for more information.

Typical inputs to `invert` for a continuum experiment are given below.

INVERT	
vis=vela.uv.1,vela.uv.2	Name of input visibility datasets
map=vela.imap,vela.vmap	Name of the output images – one per Stokes
beam=vela.beam	Name of the output beam
cell=1	Pixel size is 1 arcsec square
imsize=256,512	Image size is 256 × 512 pixels
stokes=i,v	Image Stokes I and V
sup	Leave unset to get uniform weighting,
or	
sup=0	Set `sup` to zero for natural weighting
fwhm	Specify desired resolution – unset gives the max resolution
options=mfs	Use the MFS option for multi-frequency synthesis, or
options=mfs,sdb	use the sdb option as well if using `mfclean`.
line	Controls channel selection and averaging. Default is all channels when `options=mfs`.

For a spectral line observation, typical inputs would be

INVERT	
vis=vela.uv.1,vela.uv.2	Name of input visibility datasets
map=vela.imap	Name of the output image
beam=vela.beam	Name of the output beam
cell=1	Pixel size is 1 arcsec square
imsize=256,512	Image size is 256 by 512 pixels
stokes=i	Image Stokes I only
sup	Leave unset to get uniform weighting,
or	
sup=0	Set `sup` to zero for natural weighting
fwhm	Specify desired resolution – unset gives the max resolution
line=velocity,10,1.5,1.0,3.0	Map channels to velocities. Image 10 velocities centred at 1.5, 4.5, 7.5 etc km s^{-1}

In its computation `invert` determines the theoretical rms noise. This noise is calculated assuming that the only source of error is the system temperature of the front-end receiver. No account is made of calibration errors, sidelobes or any other 'instrumental' effects. The calculation correctly accounts for the weighting scheme used in the imaging process. This theoretical noise is the level you can expect in a detection experiment (assuming no interference or confusion), and it is the best one can hope for in high dynamic range work (usually instrumental effects will limit you before the noise in these sorts of experiments).

The noise calculation of `invert` (and all other *MIRIAD* tasks that compute the variance of a correlation) is based on values of system temperature, system gain, integration time and bandwidth stored in a dataset. Unfortunately data loaded into *MIRIAD* using `fits` will have only nominal system temperatures and system gains, and an educated guess is made for the integration time. For data loaded using *MIRIAD* `atlod`, the system temperatures are those measured on-line, and the integration time will be correct. See Chapter 8 for a discussion of setting your dataset up so that noise estimates are correct. The system gain, however, is still a nominal figure. If system temperature, system gain or integration time are incorrect by some factor, then the theoretical rms noise will also be wrong by the same factor.

In the old correlator's continuum mode (i.e. the 33 channel/128 MHz mode), when all 33 channels are retained after `atlod` (i.e. `options=birdie` was not used) there is another effect which will cause `invert`'s noise estimate to be a factor of $\sqrt{2}$ too pessimistic (i.e. the actual noise is a factor of $\sqrt{2}$ lower than the printed value). With the old ATCA correlator in this mode, the channel bandwidth is twice as large as the separation between channels (*i.e.* the channels are oversampled). Unfortunately `invert` assumes that the bandwidth is the same as the separation. This will only affect you if you are imaging individual correlator channels (i.e. no frequency averaging) and if you keep all channels. It will not affect "channel 0" or multi-frequency synthesis imaging.

Chapter 14

Image Deconvolution

14.1 Introduction

Because synthesis arrays sample the $u-v$ plane at discrete locations, there is incomplete knowledge about the Fourier transform of the source intensity distribution. The measured visibility data can be thought of as the true distribution, $V(u, v)$, in the $u - v$ plane multiplied by some sampling function, $S(u, v)$. The convolution theorem states that the Fourier transform of the sampled distribution (the dirty image, I_D) is equal to the convolution of the Fourier transform of the true source visibility distribution (the true image, I) and the Fourier transform of the sampling function (the dirty beam, B_0):

$$I_D(\ell, m) = I(\ell, m) * B_0(\ell, m) \rightleftharpoons V(u, v) \times S(u, v)$$

where $*$ indicates convolution, and $\rightleftharpoons$ indicates the Fourier transform. Deconvolution algorithms attempt to account for the UN-sampled regions of the $u - v$ plane. If it was fully sampled, there would be no sidelobes, since the sampling function would be a constant, and the Fourier transform of a constant is a delta function; a perfect beam. Thus, deconvolution tries to remove the sidelobes of the dirty beam that are present in the image. It is important to realize that in doing so, the algorithm is guessing at what the visibilities are in the UN-sampled part of the $u - v$ plane. The solution to the convolution equation is not unique, and the problem of image reconstruction is reduced to that of choosing a plausible image from the set of possible solutions.

You should be extremely cautious when deconvolving images formed from a small number of snapshots. In these cases, there will be large areas of the $u - v$ plane that are unsampled because of the poor instantaneous $u-v$ coverage of the ATCA. If the source is complicated, the deconvolution algorithm may go badly wrong in its guess of what the source really looks like in the gaps. The best way to make a decent image of an object is to observe it, not to allow a deconvolution algorithm to guess what it looks like.

If you are using multi-frequency synthesis (MFS) in a situation where spectral index effects are significant (i.e. when the fractional spread in frequencies is $> 10\%$ and/or images with dynamic ranges better than a few hundred are required), then the simple convolution relationship no longer applies – see Section 14.4.

There are two techniques used commonly in radio astronomy; CLEAN and maximum entropy (MEM). CLEAN is rarely used outside of radio astronomy, but MEM is more far-reaching. For detailed discussion on the 'pros' and 'cons' of these algorithms, see the NRAO imaging workshops and references therein. Much blood has been spilt over their relative merits in the last decade or so.

It is probably fair to say that in general, CLEAN is easier to drive than MEM, although using MEM can result in reduced processing times for large problems. All dirty images produced by `invert` (continuum, line, MFS, any Stokes parameters) can be deconvolved with either `clean` (CLEAN) or `maxen` (MEM). However to deconvolve an MFS image where spectral index effects are important, a special version of CLEAN is used, `mfclean`.

After the deconvolution is finished, you have produced a the model of the source. CLEAN produces a CLEAN component image (a collection of delta functions), whereas MEM produces some more smooth model. The output images of both CLEAN and MEM are in units of Jy/pixel, and are super-resolved (i.e. they contain spatial frequencies beyond those measured). Looking at the CLEAN component image is a sobering experience. While the MEM output is more visually appealing, it will generally contain a positive bias. To improve the qualitative appearance of the models produced by the deconvolution tasks, and to suppress what is essentially unmeasured high spatial frequency structure, it is common (essentially universal with CLEAN) to follow the deconvolution by a 'restore' step. This step involves convolving the model sky with a Gaussian. This convolved image is then added to the residual image. The Gaussian is chosen to match the main lobe of the dirty beam, and it is generally called the CLEAN beam (regardless of whether CLEAN or MEM was used) or the restoring beam.

14.2 Deconvolution with CLEAN

The CLEAN algorithm (Högbom 1974) (A&AS 15, 417) represents the image as a number of point sources in an otherwise empty field of view. A simple iterative procedure is used to find the locations and strengths of these point sources.

The Högbom CLEAN algorithm looks for the brightest (in an absolute sense) pixel (called a CLEAN component) in a specified region in the image (called the CLEAN window which must include all the real emission in the image). For Stokes I images (but not Q, U or V), ideally only positive components would be encountered. However noise, errors in previous CLEAN iterations and calibration errors will mean that eventually negative components will also be found. Often it is quite valid to CLEAN these negative components. Having found this peak pixel, it subtracts a fraction (called the loop gain, and generally about 0.1 or less) of the dirty beam from the dirty image at the location of that brightest pixel. This subtracted image is called the residual image. The search and subtraction loop is repeated until the sidelobes in the image are reduced to below the noise level. It is easy to see how CLEAN works if you just think about a single point source. For extended sources, one thinks of the emission as a collection of point sources. Except for point sources, the flux density units of the dirty image are not very useful. However, convolved CLEAN components do have meaningful units of Jy per CLEAN beam area, which can be converted to Jy per unit area, because the equivalent area of the CLEAN beam can be calculated.

The Clark CLEAN

The Clark CLEAN algorithm (Clark 1980) (A&A 26, 89) is a variant of the Högbom CLEAN, and aims at improved speed for large images. The original Högbom algorithm spends a lot of time searching residuals which are negligible, and shifting and scaling the dirty beam. The Clark algorithm avoids these by searching a list of only the largest residuals, working with only a sub-region of the beam for much of the time, and by using FFTs to perform convolutions when it has to resort to using the full beam. This leads to an algorithm which is more efficient for large images (although a good deal more complicated). For small images though, the Högbom CLEAN may still be faster.

The Clark CLEAN plays a few tricks which you should know about. It has what are termed major and minor cycles. In the minor cycle, CLEAN components are searched for, and subtracted from the image, in a fashion very similar to that of the Högbom CLEAN. The difference is that only the central portion of the dirty beam is used for the subtraction, and a list of the largest residuals are searched (not the full region being CLEANed). The rationale is that for dirty beams that have a fairly good sidelobe pattern, this will be good enough to find the CLEAN components. At some designated time, this minor cycle is terminated and a major cycle computed. This means that the list of CLEAN components from the current cycle are 'FFTed', and subtracted from the FFT of the residual image that resulted from the *previous* major cycle. In this way, errors that might have accumulated in the subtraction phase of the minor cycle with only a part of the beam are largely set to rights. It is also at this stage that the list of the largest residuals is re-determined.

This use of a small beam patch in the Clark CLEAN is a potential danger if the beam is poor. Images made from ATCA snapshot data, even if there are a few cuts, often have a beam in which the first sidelobe response is quite strong and outside of the beam patch that the Clark CLEAN typically uses.

The SDI CLEAN

CLEAN sometimes runs into an instability when working on extended sources of fairly uniform surface brightness. It produces obviously wrong 'CLEAN stripes' in your image. The Fourier transform of these stripes tends to be found in UN-sampled parts of the $u-v$ plane. This is a complicated way of saying that the visibility data do not constrain CLEAN's interpolation between the measured visibilities sufficiently well to prevent it putting in badly wrong values.

Another variant of the CLEAN algorithm which helps suppress CLEAN stripes, is the SDI CLEAN (Steer, Dewdney, and Ito 1984) (A&A 137, 159). In this variant any point in the residual image greater than some factor times the maximum residual point is taken as a CLEAN component. The factor is less than one. Thus, when the residual image has become very smooth, this avoids introducing ripples into it (which occur when subtracting the beam from just one location) which lead to the stripes. This algorithm can be quite successful.

Region-of-Interest Considerations

Note that the Högbom, Clark, and SDI versions of CLEAN can only properly deconvolve a region a quarter the area of your beam. If the image and beam are the same size – which is the default behaviour of `invert` – you can deconvolve only a quarter of the image. For example, consider images that are 256 pixels in size (one dimension will do), with a point source located at pixel 192. A beam image centred on that point source will extend only to pixel 64, so that the full image cannot be CLEANed.

However there is a switch in `invert`, `options=double`, to make the beam four times the area (twice the linear extent) of your image. In this way, you can CLEAN to the edge of your image (you should make the image at least 5 pixels larger in each direction than the real emission).

In practise, you can sometimes violate the above rule that the beam must be four times the area of the CLEAN region, especially if the field is very sparse (e.g. a central source, and some simple confusing sources at a modest distance away). Generally the Högbom CLEAN will be more forgiving if you attempt to use a CLEAN region larger than a quarter the area of the beam. However there is a switch (`options=pad`) in `clean` to improve the stability for Clark and SDI CLEANs as well.

Apart from restricting the area CLEANed to a quarter of the area of the beam, CLEAN works much better if the region being CLEANed snugly encompasses the real emission in the image. At first, you may not know where this is. However, after some quick and dirty (sic) CLEANing, you could redo it properly with a better idea of the CLEAN region. Essentially, you are giving CLEAN *a priori* information about the source, which better constrains the deconvolution problem.

The Cotton-Schwab CLEAN

Yet another CLEAN variant, similar to a Clark CLEAN, is the MX algorithm (Schwab 1984). During the major cycle, the Fourier transformed CLEAN components are subtracted from the *ungridded* residual visibility data, instead of the residual gridded visibility data (*i.e.* the FFT of the residual image). This means that the subtraction is more accurate. $\mathcal{MIRIAD}$ does not have a task which implements the MX algorithm.

CLEAN Stripes

CLEAN stripes are, in part, introduced by the approximations made during the minor cycles of a Clark CLEAN – they are less common when using a Högbom algorithm. Thus striping problems can be alleviated by taming the minor cycle approximations. There are two simple ways to tame the approximations used in the minor cycle of a Clark CLEAN – use a larger beam patch, and relax the criteria to perform a major cycle (*i.e.* perform fewer minor cycles per major cycle).

Cornwell (1983) (A&A 121, 281) suggested that the CLEAN stripe instability can be suppressed by the addition of a delta function to the dirty beam. For obvious reasons, this is popularly termed the 'Prussian helmet' CLEAN. The spike is usually set at about 0.3. Together with this, one usually makes the loop gain smaller, to perhaps 0.05 or even 0.01. However, recent wisdom suggests that in general, the Prussian helmet is probably a red herring.

Computation

MIRIAD's CLEAN task – surprisingly called `clean` – implements the Högbom, Clark and SDI algorithms. Given an input dirty image and beam, it produces an output CLEAN component image which has flux density units of Jy/pixel. This CLEAN component image is CLEAN's best guess at what the source really looks like. Invariably its extrapolation at high spatial frequencies is very poor (and probably a reason why *AIPS* does not make it easy to view it). To reduce the undesirable effects of this extrapolation, and to add in any emission remaining in the residuals, you will need to use the task `restor`. This gives you what is normally thought of as the 'CLEAN' or restored image.

The various input parameters to `clean` are:

- `map, beam`: These keywords give the names of the input image and beam datasets – as generally produced by `invert`.
- `out`: This gives the name of the output CLEAN component image.
- `mode`: *MIRIAD* `clean` can perform either Högbom, Clark or SDI CLEANs. The `mode` parameter can be set to indicate the particular algorithm to use. Possible values are 'hogbom', 'clark', 'steer' or 'any' (the default). With 'any', `clean` determines what it believes is the best algorithm for your particular image. In this case, `clean` can switch between different algorithms, as the nature of the residuals change. This is particularly useful when CLEANing a large image which contains some strong point sources and much lower brightness extended emission. In this case, `clean` may well switch from a Clark or Högbom algorithm to the SDI algorithm when it finds that the residuals are becoming very smooth.

  Generally you can allow `clean` to choose the algorithm. Task `clean` will override your choice if you attempt to use the Högbom algorithm on an image that is too big.
- `region`: As noted above, it is important in `clean` to limit the CLEAN region to an area one quarter that of the beam, and to make it fit snugly around the real emission. The `region` parameter can be used to describe quite complex CLEANing regions. See Section 6.3 on how to specify this. Alternatively, you can use task `cgcurs` to describe the region interactively from a display of an image on a PGPLOT device. Task `cgcurs` (see Chapter 17.3) can produce a text file, `cgcurs.region`, describing the region selected, which you can then input to `clean` (see Section 2.5). For example:

  ```
  region=@cgcurs.region
  ```

  The default CLEAN region is the largest region (centred on the dirty image centre) that can be safely deconvolved.
- `gain`: A good number for the loop gain is 0.1 (the default). If CLEAN stripes occur, then try lowering the loop gain by a factor of about 10.
- `cutoff, niters, options=negstop`: You can tell `clean` when to stop in three ways. You can tell it to quit once the brightest pixel (in an absolute sense) in the residual image reaches some cut-off. This is typically some multiple of the rms noise level (3–σ or so). Specify this level with the parameter `cutoff`. Alternatively, you can tell `clean` to stop when it encounters the first negative component, by using the `options=negstop` switch. Otherwise, CLEANing will proceed until `niters` CLEAN iterations have been performed. When CLEANing a cube, `niters` is the

number of iterations per plane. For small and simple sources, a few hundred iterations are usually sufficient. For complicated and large sources, you can CLEAN forever.

You can set all three of `cutoff, options=negstop` and `niters`. Task `clean` will stop when any one of these stopping criteria are satisfied.

- `phat`: This gives Cornwell's Prussian helmet parameter, which is the value of the spike to add to the central lobe of the beam. Using this parameter may give some success at eliminating CLEAN stripes. When using a Prussian helmet, typical values are 0.3 or so. Note that the loop gain should also be reduced when using Prussian helmets.

- `model`: If you CLEAN an image, and find that you need to CLEAN some more, `clean` can be restarted from where you left off. Do this by setting the `model` parameter to the name of the old output CLEAN component image, and setting the `out` parameter to a new name. Actually the `model` image need not have been produced by `clean` – it can be any image with units of Jy/pixel. Of course, it should be a representation of your source.

- `speed`: The 'speedup factor' is a knob for the Clark algorithm which controls how often major cycles are performed. The more major cycles, the closer a Clark CLEAN is to a Högbom CLEAN. For point sources, this can be about 1. For extended sources, or if you are having trouble with CLEAN stripes, set `speed` to some small negative number – typically -1 is good. The default is 0.

- `minpatch`: For Clark CLEANs, the parameter `minpatch` is the minimum full width (*not* half-width, as it is in $\mathcal{AIPS}$) of the beam patch used in the minor cycle. The default is 51 pixels. However, for ATCA work, where the beam can often have large distant sidelobes, this value may be too small. Setting `minpatch` to larger values will slow the algorithm, but may avoid CLEAN striping problems. The maximum value that `clean` will accept is 257 – larger values will be trimmed back to 257.

- `clip`: For SDI CLEANs, this parameter gives the 'clip level'. During an SDI iteration, all pixels greater than `clip` times the peak flux density are considered to represent true structure, and so are taken as components for that iteration. Typically the clip level is 0.9. The default that `clean` computes is image dependent, and will be a function of how many pixels there are across the beam. The default value is usually adequate.

- `options`, as usual, gives several processing options. Possible values are

  `negstop` Stop CLEANing when the first negative component is encountered.

  `positive` Apply a positivity constraint. This constrains the component image to be non-negative. A side-effect of this is that CLEAN will stop iterating if it cannot continue to ensure this. This does not have any effect with SDI iterations. This option does not seem to produce as much improvement as one might hope.

  `asym` Normally `clean` assumes that the beam has a 180 degree rotational symmetry, which is the norm in radio interferometry. Making such an assumption allows some optimisations. You should instruct `clean` if this is not the case, by using the `asym` switch.

  `pad` Double the beam size by padding it with zeros. This will give better stability with Clark and Steer modes if you are daring enough to CLEAN an area which is more than a quarter of the beam area.

Typical inputs are given below:

CLEAN	
map=vela.imap	Dirty image
beam=vela.ibeam	Dirty beam
out=vela.icmp	Output CLEAN component image
mode	Algorithm used – let CLEAN decide
region	Defaults to max area safely CLEANed.
gain=0.1	Loop gain
phat	Unset means no Prussian helmet
cutoff=0	Terminate CLEAN at this residual level or
niters=500	Specify total number of CLEAN components
speed	Speedup factor; -1 for extended sources, +1 for point sources
minpatch=127	Minimum beam size for minor cycles
clip	SDI clip level

The total CLEANed flux density (*i.e.* the cumulative sum of the CLEAN components) should eventually settle down to a roughly constant number. This indicates that you are just picking up noise, and that there are no sidelobes left to remove. If the total CLEANed flux density starts to decrease again, this usually indicates that you have been a bit heavy handed, and CLEANed too much. You might also look at the result and see if you can see any sidelobes left over.

Having completed `clean`, you will almost certainly want to "restore" your image – see Section 14.6.

14.3 Deconvolution with maximum entropy algorithms

This discussion was lifted from Tim Cornwell's article in the NRAO imaging workshop (1988).

CLEAN approaches the deconvolution problem by using a *procedure* which selects a plausible image from the set of feasible images. This makes a noise analysis of CLEAN very difficult. The Maximum Entropy Method (MEM) is not procedural. The image selected is that which fits the data, to within the noise level, and also has maximum entropy. This has *nothing* to do with physical entropy, it is just something that when maximised, produces a positive image with a compressed range of pixel values. The latter aspect forces the MEM image to be smooth, and the positivity forces super-resolution on bright, isolated objects.

One general-purpose definition of entropy (championed by Gull and Skilling) is

$$H = -\sum_k I_k \log\left(\frac{I_k}{M_k}\right)$$

where I_k is the brightness of the kth pixel, and M_k is some default image. An example might be a low-resolution image of the object. This allows *a priori* information to be incorporated into the problem. An alternate form, suggested by Cornwell (sometimes call the 'maximum emptiness' criterion) is

$$H = -\sum_k \log(\cosh(\frac{I_k}{M_k})).$$

This second form does not enforce positivity, and so can be used for Stokes parameters other than I.

The requirement that each visibility be fitted exactly by the model usually invalidates the positivity constraint. Therefore, data are incorporated with the constraint that the fit, χ^2, of the predicted visibility to that observed, be close to the expected value:

$$\chi^2 = \sum_r \frac{|V(u_r,v_r) - \hat{V}(u_r,v_r)|^2}{\sigma^2_{V(u_r,v_r)}}.$$

Maximising H subject to the constraint that χ^2 be equal to its expected value leads to an image which fits the long spacings too well, and the zero and short spacings poorly. To remedy this, an added constraint can be added to the problem. Typically this is a flux constraint, which ensures that the flux density in the maximum entropy image is correct. That is, the flux density of the maximum entropy image,

$$F = \sum_k I_k,$$

is constrained to equal the known flux density for the source. This is quite a useful constraint. Although the maximum entropy task, described below, does not force you to give a flux constraint, you should attempt to, if at all possible.

Computation

Maximum entropy algorithms tend to be less robust and harder to drive than CLEAN algorithms. The quality of the maximum entropy solution can depend very strongly on the `rms` and `flux` parameters – these parameters should be set with some care, or some experimentation may be necessary.

Various parameters to the task `maxen` are described below:

- `map, beam`: As with CLEAN, you give task `maxen` a dirty image and a dirty beam.
- `default`: You also have the option of supplying a default image. The default image (M_k in the equations above) is some estimate of what the algorithm should tend towards. It could be some *a priori* model or low-resolution image of the source. The default default (sic) image is a uniform image. The units of the `default` image are not too important, though it should be positive valued if the Gull and Skilling entropy measure is used.
- `model`: The `model` is some initial estimate of the deconvolved image (units of Jy/pixel). In principle, the `model` image is simply a way to kick start `maxen` towards the correct solution. In principle it should not affect convergence to the final solution or its quality – it should just speed up the process. In practice, a good initial `model` can sometimes help.
- `region`: As with CLEAN, this gives the region in the dirty image which contains all the source emission, and much the same can be restated here about setting the parameter. In particular, you can deconvolve an area no more than a quarter the area of the beam (but see `options=pad` if you fail to heed this advice).
- `measure`: This determines the entropy measure used; specify either `gull` or `cornwell` (these are the $p \log(p)$ and $\log(\cosh[p])$ forms, respectively). The default is to use the Gull form, but if deconvolving Q, U or V images, the Cornwell form may be useful.
- `out`: This gives the name of the output image, which has units of Jy/pixel. This is the equivalent image to `clean`'s CLEAN component image. However, unlike CLEAN, maximum entropy techniques tend to be more conservative in their extrapolation of high spatial frequencies, and so this output is more commonly viewed and used as a final image than the CLEAN component image would be. However it is just as valid an operation to pass this output from `maxen` through `restor`, and so produce either a restored or a residual image. Those with some courage may choose to look at the residual image. Unlike CLEAN, the residuals tend to be strongly correlated with source structure.
- `niters`: Task `maxen` uses an iterative algorithm to arrive at its solution, and terminates when it believes it has converged. The parameter `niters` gives the maximum number of iterations that `maxen` will attempt before giving up if it does not converge. For low dynamic-range images, 10 iterations are usually sufficient. Higher dynamic-ranges (greater than 1000) can require 30 iterations to converge. The default maximum number of iterations is 20.
- `rms`: This is a crucial parameter. It gives the rms noise in Jy/beam. The rms noise value printed by `invert` should be some guide to setting this parameter, but see the caveats about this value under `invert`. An alternative way is to measure the rms in a blank portion of the sky. If the beam has few sidelobes you will probably be able to measure this directly from the dirty image. Otherwise you would really need to CLEAN first! If `rms` is too large, the output image will be too smooth. If

it is too small, convergence will be prevented. A useful trick is to underestimate `rms` and then stop after a few iterations, and then reset `rms` to the level achieved up to that point. Do not leave `rms` unset.

- `flux`: The `flux` specifies how the zero-spacing flux density is to be estimated. There are three modes of use. First, you specify a known value which must be fitted to within 5%. Second, if you have no idea what the zero-spacing flux density is, then leave **`flux`** unset; `maxen` will attempt to estimate it. Third, if you have a rough idea (within a factor of 2, say) then set `flux` to the negative of your guess. If at all possible, you should give some estimate of the total flux density.
- `q`: This gives an estimate of the volume of the main lobe of the dirty beam in units of pixels. Typically it is 10 to 30, and the algorithm does not depend critically on it. The default is determined from the data, and generally will be adequate.
- `options`, as usual, gives several processing options. Possible values are

  `quiet, verbose:` This controls the verbosity of `maxen`'s messages.

  `asym:` Normally `clean` assumes that the beam has a 180 degree rotational symmetry, which is the norm in radio interferometry. Making such an assumption allows some optimisations. You should instruct `clean` if this is not the case, by using the `asym` switch.

  `pad:` Double the beam size by padding it with zeros.

MAXEN	
map=vela.imap	Dirty image
beam=vela.ibeam	Dirty beam
model	An initial solution solution – generally unset
default	The image that the solution should tend towards – generally unset
out=vela.ivm	Output dataset
niters=20	Maximum number of iterations
region=	Region to be deconvolved
measure	Leaving unset gives you the Gull measure
flux=	Total source flux – give some value
rms=	Rms noise in the dirty image – *IMPORTANT*!
q=	AN estimate of the value of the beams main lobe.

14.4 Multi-frequency deconvolution – MFCLEAN

If you have used multi-frequency synthesis in the imaging stage, you will have to have considered a fly in the ointment – the flux density of the source will vary with frequency (as well as spatially). For detection and low dynamic-range work, mild spectral variation can generally be ignored. For high dynamic range imaging, the variation with frequency will cause a distortion in the resulting image.

The *MIRIAD* deconvolution task `mfclean` can be used to avoid this distortion. Rather than just determining a flux density estimate, `mfclean` simultaneously solves for both a flux density and spectral variation for each source. To understand how this is possible, consider a noiseless observation of a single point source at the phase centre. Assume that it has a linear spectral variation. While this is obviously an approximation, it is an adequate one for dynamic ranges up to several thousand to one (assuming typical spectral variation and fractional spread in frequencies of no more than 30%). The visibility function for this source can be described by two parameters: the source brightness at some reference frequency, and the brightness derivative.

$$V(\nu) = I(\nu_0) + \frac{\partial I}{\partial \nu}\Delta\nu$$

If we temporarily assume that the derivative is zero, then the image resulting from a multi-frequency synthesis observation of this source will just be the normal point-spread function (beam pattern) scaled by $I(\nu_0)$. If we now assume some non-zero derivative, we will get a different image. Because of the linearity of the system, the output image will be a sum of the two terms. The first term is just the normal point spread function scaled by $I(\nu_0)$ as before, while the second term is caused by the derivative. The importance of the second term will be related to the size of the derivative and the range of frequencies sampled. This second term has been called the "spectral dirty beam" – it is a point-spread function resulting from a linear frequency variation. While it may seem that the most logical definition of the spectral dirty beam is the additional response resulting from a source with a flux density derivative of 1, there is a more convenient scaling. If we assume that the flux density variation is due to a spectral index, α, where

$$I(\nu) = I(\nu_0)(\frac{\nu}{\nu_0})^\alpha$$

then

$$I(\nu_0)\alpha = \nu_0 \left.\frac{\partial I}{\partial \nu}\right|_{\nu_0}$$

Conventionally the spectral dirty beam is scaled so that

$$\begin{aligned} I_D(\ell,m) &= I(\nu_0)B_0(\ell,m) + \nu_0 \left.\frac{\partial I}{\partial \nu}\right|_{\nu_0} B_1(\ell,m) \\ &= I(\nu_0)B_0(\ell,m) + (I(\nu_0)\alpha)B_1(\ell,m) \end{aligned}$$

For a general source, a sum of the two beams is replaced by a convolution relation:

$$I_D = I * B_0 + (I\alpha) * B_1$$

(in the above equation, I is evaluated at ν_0). The problem of multi-frequency deconvolution is to estimate I and $I\alpha$ images simultaneously. A more detailed discussion of the algorithm `mfclean` uses is given in Sault & Wieringa (1994) (A&AS 108, 585).

But how important are the effects that spectral variation causes? Generally you will want to ignore them if you can, because then the deconvolution procedure has to solve only the standard convolution equation,

$$I_D \approx I * B_0,$$

and the other deconvolution tasks – `clean` and `maxen` – can be used.

Surprisingly spectral effects are not very important in a wide variety of problems. For fractional bandwidths of 10%, spectral effects need to be considered for dynamic ranges of better than 1000. For fractional bandwidths of 30%, the effects become important for dynamic ranges of a few hundred. This assumes typical spectral variations equivalent to spectral indices of order 1. When imaging beyond the half-power points of the primary beam, the frequency dependence of the primary beam will be quite substantial, and so spectral variation will need to be considered for lower dynamic range images.

Generally you will need to consider at the imaging step (task `invert`) whether you *may* need to account for spectral variation in the deconvolution step. If you *may*, then you need to form the spectral dirty beam, B_1, as well as the normal dirty beam, B_0. If so, you must have set the `sdb` option in `invert`. This produces the spectral dirty beam as a second plane of the beam dataset. If you did produce this spectral dirty beam plane, and then use `clean` and `maxen`, they will issue warnings (which can be safely ignored) about the existence of the second plane.

The inputs to `mfclean` are fairly similar to `clean`. Task `mfclean` differs in that it does not have a Prussian helmet or SDI CLEAN capability. Task `mfclean` does not support a number of the `options` given by CLEAN. More fundamental, though, is that, whereas `clean` can only CLEAN an area a quarter of the beam area, `mfclean` can strictly speaking CLEAN an area only one ninth of the beam area. To explain the reason for this, we must consider some of the details of `mfclean`. To aid it in finding peaks in the flux density and spectral index domains, it correlates the residuals with the beam and the spectral dirty beam. More strictly it correlates the residuals with a portion of the beam and spectral dirty beam – the sub-beam. Strictly the sub-beam size should be the same as the region being CLEANed. However in practise, provided it contains most of the main sidelobes, a smaller portion is adequate. Task `mfclean` will tell you the sub-beam size, and indeed may complain about it being small. However provided it is either at least as big as the region being CLEANed, or is 50-100 pixels, you are probably OK. All the same, however, you should attempt to produce images with larger guard bands if `mfclean` is to be used.

As with `clean`, you will almost certainly want to restore your resultant image – see Section 14.6. The output from `mfclean` is a CLEAN component image with a difference – it contains two planes. The first plane is the normal CLEAN component image (like the one determined by `clean`), while the second plane contains the estimate of $\alpha(\ell, m)I(\ell, m)$. Both `restor` (Section 14.6) and a number of other tasks use this second plane.

14.5 Other deconvolution tasks

The following two tasks are primarily written for multiple-pointing (mosaiced) observations (see Chapter 21). However they also work equally well for a conventional single pointing dataset (in $\mathcal{MIRIAD}$, a single pointing observation is treated as a degenerate mosaic). They both provide functionality not available with the other single-pointing deconvolution tasks.

- `mosmem`: This is a maximum entropy deconvolution task which can handle a joint deconvolution with an interferometric and single dish observation. A joint deconvolution of a single dish and single pointing interferometric observation is not common: usually if the source is large enough to warrant a single dish observation, then it is likely that the interferometric observation would be a mosaic.
- `pmosmem`: This is another maximum entropy deconvolution task. It differs in that it can perform a joint deconvolution of the four Stokes parameters simultaneously. With polarised emission being potentially positive and negative valued, the entropy measure for this is different from a simple total intensity deconvolution. See Sault et al. (1999) (A&A, 139, 387) for more background on joint polarimetric deconvolution.

  Often joint polarimetric deconvolution does not do a better job than doing the different polarisations individually (e.g. by using `clean`, or by using `maxen` and the maximum emptiness entropy measure).

14.6 Restoring an image

Having produced a model of the sky with either `clean`, `maxen` or `mfclean`, you will probably want to use `restor` to convolve this with a Gaussian CLEAN beam (restoring beam) and add the remaining residuals. For models produced by `mfclean`, `restor` convolves only the intensity (I) plane with the CLEAN beam, although it does use the $I\alpha$ plane when determining the residuals.

Having the restoration as a separate step does have some flexibility advantage, but the user then has to run another task to produce the final image. Task `restor` optionally fits the dirty beam with a Gaussian CLEAN beam, and then either produces a restored image or, alternatively, a residual image. There are a few other possibilities, but these are less common. The inputs to `restor` are:

- `model`: This is normally the output CLEAN component image, but can be any image that is a good representation of the source. It should have units of Jy/pixel.
- `map, beam`: These are the input dirty image and beam.
- `mode`: This parameter determines the operation to be performed by `restor`. The most common operations are to produce a restored image (`mode=clean`) or a residual image (`mode=residual`).
- `out`: The output restored or residual image.
- `fwhm, pa`: Parameters of the CLEAN beam; `fwhm` gives the size, in arcseconds, of the Gaussian beam to be used in the restoration. This will normally be two numbers, giving the full-width at half maximum of the major and minor axes of the Gaussian. If only one value is given, then the Gaussian is assumed to be circularly symmetric. Parameter `pa` gives the position angle, in degrees, of the restoring beam, measured east from north. If no values are given for `fwhm`, `restor` first checks the beam dataset to see if it contains Gaussian fit parameters. If not, `restor` performs a fit to the beam data. Note that the fitting procedure will probably give different values to the $\mathcal{AIPS}$

MX and APCLN tasks. Generally the value computed by restor is to be preferred to the APCLN and MX values.

Typical inputs are given below:

RESTOR	
map=vela.imap	Dirty image
beam=vela.ibeam	Dirty beam
model=vela.icmp	Component image produced by CLEAN
out=vela.icln	Output restored image (or residuals)
mode	Leave unset to get restored image
fwhm	Beam size – Leave unset to let restor fit it
pa	Leave unset to let restor fit it

Chapter 15

Self-Calibration

Because the basic interpolated calibrator gains do not determine the antenna gains perfectly at each time stamp, the quality of the resultant image suffers. Often the technique of self-calibration is used to make additional corrections to the antenna gains as a function of time. It is notionally very similar to the basic calibration. The main difference is than the model of the source is generally more complex than just assuming it's a point source at the phase centre. The self-calibration technique finds antenna gains, g_i, which minimise the difference between the measured visibilities, V_{ij}, and the model visibilities, $\hat{V}_{ij}$, viz

$$\epsilon^2 = \sum |V_{ij} - g_i g_j^* \hat{V}_{ij}|^2.$$

Like basic calibration, these gains are normally described as 'antenna gains'. This, however, is somewhat of a misnomer when a telescope measures dual polarisations. In this case, it would be better to describe the gains as 'feed gains', as the gain factors will generally differ between the two antenna feeds. The above equation should include subscripts to indicate the polarisation type of the correlation (*e.g.* V_{XX}) and the gain (*e.g.* g_X).

The model visibilities are generally derived from a model image (which should have units of Jy/pixel). For a complex source, the model image will usually be determined by the deconvolution tasks (the outputs of `clean`, `mfclean` or `maxen` *without* the restore step). If the source is a point source, then a point-source model could be used, although the restriction that it be located at the phase centre is lifted.

The self-calibration procedure is often performed iteratively, each time with a better model, until finally the sequence converges and no more improvement in the image quality can be made.

Self-calibration is not a technique that should be applied blindly. This is especially true for ATCA data because, compared with the VLA, the problem is only slightly over-determined. This is because the ATCA has, at most, 15 baselines to determine the complex gains for 6 antennas, whereas the VLA has, at most, 351 baselines to determine the complex gains for 27 antennas. Because we can set the phase of one antenna to zero, the problem reduces to finding 11 real numbers from 30 real equations. With the 5-antenna compact array, we must find 9 real numbers from 20 real equations. This problem is exacerbated when one antenna is absent from the data for a period of time, or you have flagged it out because of poor data quality.

There are other problems to do with the east-west nature of the ATCA – see Bob Sault's technical memo 'Some simulations of self-calibration for the AT' (AT Memo 39.3/058). One important point to keep in mind is that self-calibration with the ATCA depends crucially on the initial model that you start with – much more so than with the VLA where you can start with quite a poor model but arrive at the correct result after just a few iterations.

Self-calibration, like basic calibration, requires that the signal-to-noise ratio on each baseline be of the order of at least 5 or so. For weak sources, this may require a long solution interval, within which the gains are assumed to be constant. If, in reality, the gains are changing on a time-scale significantly shorter than your solution interval, and these changes are degrading your image quality, then you will be unable to improve the image quality. The time-scale on which you will often find it necessary to correct the gains is approximately one minute. Thus, weak sources often cannot be self-calibrated. For a continuum observation (100 MHz bandwidth) with the old ATCA correlation, you will, typically, need a signal strength of 100 mJy before self-calibration is possible. For the CABB correlator, with its 2 GHz of bandwidth, you will typically be successful with sources with flux density of around 10-20 mJy or more.

Note that it is not the receivers that cause the gains to be unsteady with time, but the atmosphere. In a similar way to the degradation of an optical image by the atmosphere, a radio image is defocused by the phase (and amplitude) noise that atmospheric cells induce into the wavefront. Self-calibration can be thought of as an off-line mimicry of adaptive optics in optical astronomy.

15.1 Computation

MIRIAD contains two self-calibration tasks – `selfcal` and `gpscal`. After some general comments, we describe the way in which these tasks handle polarisation and multi-channel data. Finally, the general inputs to the tasks are described.

Figure 15.1 gives a typical self-calibration flow chart.

Apply Your Original Calibration Before Self-Calibrating

Both `selfcal` and `gpscal` produce antenna gain tables in the same format as the calibration tasks previously described. Consequently, after running these tasks, the gains they derive are applied to the data 'on-the-fly'. However, the new gain tables derived by the self-calibration tasks will overwrite any old gain tables. Provided the self-calibration produces an improvement, this is fine. However, if the self-calibration process degrades the quality of the image (possibly because you gave it bad inputs or a poor model, or because you were attempting to calibrate a weak source), then you may have lost your good set of gains. An additional shortcoming is that the self-calibration tasks do not apply bandpass corrections – they assume bandpass corrections have been previously applied. Finally, if the primary gains have appreciable XY phase (for any antenna for `selfcal`, but for the reference antenna only for `gpscal`), or significant amplitude gain mismatches (for `selfcal` only) then these corrections should be applied to the data before self-calibration. Consequently, *it is usually wise, before starting to perform self-calibration iterations*, to form a calibrated dataset (apply bandpass corrections and primary antenna gains). Additionally if you are going to use `selfcal`, as discussed below, you should convert to Stokes parameters. The task used to apply calibration and convert to Stokes parameters is `uvaver`.

Thus when using `selfcal`, use inputs to `uvaver` such as;

UVAVER	
vis=vela.uv	Raw visibility dataset, with primary calibration tables.
out=vela.uv.cal	Output with primary calibration applied.
stokes=i,q,u,v	Perform Stokes conversion.

If using `gpscal`, do not perform Stokes conversion.

UVAVER	
vis=vela.uv	Raw visibility dataset, with primary calibration tables.
out=vela.uv.cal	Output with primary calibration applied.
stokes	Unset for no Stokes conversion.

Once you have started self-calibrating, however, you probably do not need to form a calibrated dataset after each iteration. Provided you keep your best model of the image, if you do overwrite good gains with bad ones, then they are simple enough to regenerate using self-calibration and the best model.

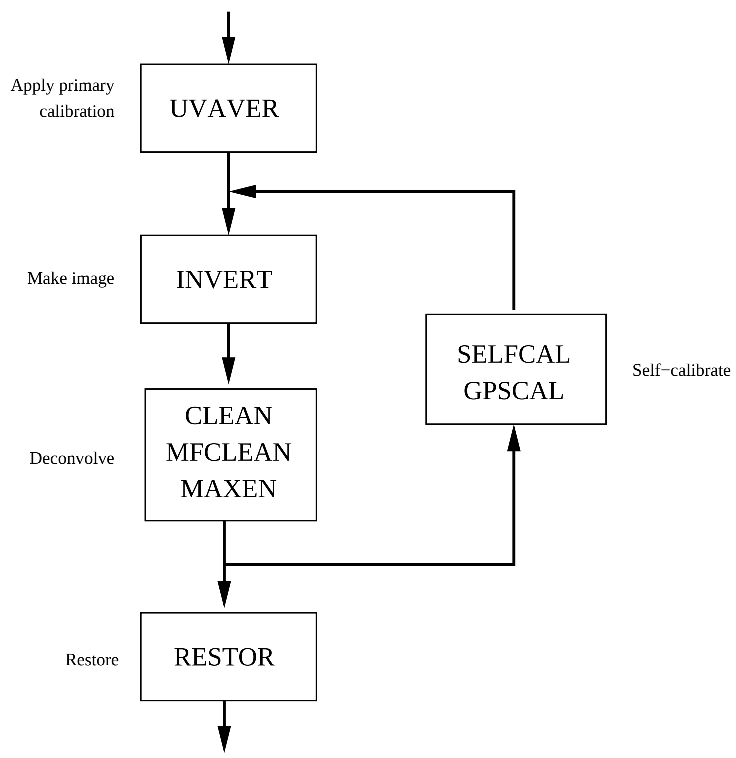

Figure 15.1: Typical self-calibration path

Self-Calibrating with Multiple Models

Both `selfcal` and `gpscal` can be given multiple input model images. The input models can be cubes for multi-channel data. These different models represent some different aspect of the data. For example, the data could contain multiple pointing centres from a mosaiced observation or multiple spectral lines. The self-calibration tasks assume that the gains are not dependent on these different aspects of the data (*e.g.* the gains are independent of frequency and pointing centre). All the information will be used in a simultaneous solution of the gains. The self-calibration tasks will generally determine the appropriate correspondence between data and model (*e.g.* it will associate the data from a particular pointing with a model with the same pointing). There are, however, some caveats described below.

Polarisation Handling

The biggest difference between `selfcal` and `gpscal` is in the handling of polarisation and dual feeds.

Though `selfcal` can handle multiple pointings and frequency ranges, it assumes that the antenna gain is independent of feed (i.e. the X and Y gains are the same). It also only handles Stokes-I models (and ignores visibilities that are unrelated to determining Stokes-I).

Task `gpscal` is intended for telescopes with dual feeds (either linear or circular), such as the ATCA. Indeed, it will fail for other sorts of data.

- `gpscal` solves for the two feeds either totally independently, or with the constraint that the XY (or RL) phase of an antenna must be constant with time.
- `gpscal` converts between Stokes parameters and raw polarisations. It can take multiple input models for different Stokes parameters. However, if a model of a particular Stokes parameter is not given, it is assumed to be zero. For example, if you give it models of I, Q and U, but not V, then the source is assumed to have no circular polarisation.
- `gpscal` uses the antenna polarisation leakage table if this information is present. If it is not present, leakages are assumed to be zero.
- Unlike `selfcal`, `gpscal` cannot take models from multiple pointings or from different frequency ranges. All models must correspond to the same pointing and same frequency range.

Frequency Handling

Both `selfcal` and `gpscal` assume that the gains are independent of frequency. This allows them to use data from all channels in forming gain solutions. However, as mentioned above, they do not apply bandpass corrections. Consequently, if you have multi-channel data and models, you will probably want to form a bandpass-corrected visibility dataset before doing self-calibration.

The assumption that the gains are independent of frequency should be treated with some caution. If all the visibilities for a given baseline go through one signal path, then the instrumental phase for the different visibilities should be equal (assuming the signal path is well equalised). However, atmospheric phase should vary linearly with frequency. Provided the fractional bandwidth is small, it is still a reasonable approximation that the gains are independent of frequency. Indeed this is the normal approximation used in deriving calibration solutions from a 'channel 0' dataset.

If the visibilities for a given baseline go through multiple signal paths (*e.g.* the 2 IF system for the ATCA), then the instrumental phases for the different paths could be significantly different. Tentative results for the 2 IF system of the ATCA show that, apart from an offset, the phase of the two signal paths track each other as the ratio of the frequencies (*i.e.* the path length difference is constant). The offset between them can be eliminated either by adjusting the phases of the two paths to zero at the start of the observation, or by the primary calibration process. Provided the offset has been eliminated, and provided the frequencies are not too different, it may be useful to approximate the gains as independent of frequency.

If the phases vary significantly between the frequencies, then the different frequencies need to be self-calibrated individually. Ideally there would be a self-calibration task which knows that the phase errors at different frequencies tracked as the ratio of the frequencies.

When self-calibrating multi-channel visibility datasets, there are two distinct modes of calculating the variation of the model with frequency, depending on whether the model contains multiple planes, or whether multi-frequency techniques were used. We discuss these two possibilities in turn.

Multi-Plane Models

When the model contains multiple planes, it should be noted that the self-calibration tasks *do not* perform any interpolation of models along the spectral dimension. The self-calibration tasks assume that there is a one-to-one correspondence between a model plane and some average of the channels. The correspondence is determined in a rather imperfect fashion – the self-calibration tasks use the standard `line` parameter (see Section 5.4). This parameter determines the channels selected from the visibility dataset, and the averaging performed on them. Unfortunately, the self-calibration tasks only perform rudimentary checks on whether the `line` parameter makes sense (*i.e.* whether the selected channels have the same frequencies as the model planes). Despite this, in many instances, the self-calibration tasks will provide appropriate defaults for the `line` parameter. Below we outline those instances where you can or should allow the `line` parameter to default, and those instances where you should not.

- *MIRIAD* tasks normally maintain a record of the `line` parameter used to form an image. The imaging task, `invert`, records the `line` parameter it uses, and the deconvolution tasks propagate it to their outputs. The default `line` parameter used by the self-calibration tasks is that which was used to form the model. This is generally quite appropriate if the model was formed from the data being self-calibrated. However if the model was formed by some other means, then the default will be quite inappropriate, and you should set the `line` parameter explicitly.

- `selfcal` can cope with the case of several models being derived from different channel ranges from one visibility dataset. This might occur, for example, if multiple lines were being observed simultaneously (as often happens at millimeter wavelengths). In this case, the different models were formed with different `line` parameters. `selfcal` *must* rely on the record of the `line` parameter stored in the model.

- If a point-source model is used (*i.e.* no input model dataset), then you should give the `line` parameter explicitly.

- Similarly, if using the `mfs` option, you should also set the `line` parameter explicitly – see the next subsection.

Multi-Frequency Models

When using multi-frequency techniques for continuum work (*i.e.* specifying `options=mfs` to `invert`), the model will generally consist of a single plane (if derived from `clean` or `maxen`) or an intensity plane and flux scaled derivative plane (`mfclean`). In these cases, the self-calibration tasks can generate the model for all needed frequencies. However, you *must* tell the self-calibration tasks that the model is a multi-frequency one by specifying `options=mfs`.

In the case of the models produced by `clean` and `maxen` for multi-frequency images, as these deconvolution tasks do not determine the derivative plane, the self-calibration tasks assume that the spectral index is 0.

The `line` parameter can still be used when performing multi-frequency self-calibration. In this case, it selects out those channels in the visibility dataset that are appropriate to the model. Indeed, the default `line` parameter can be rather inadequate, and you should give the parameter explicitly.

Input Parameters to the Self-Calibration Tasks

We now describe the input parameters to the self-calibration tasks. Given the similarity of the two tasks, we will describe them together, noting any differences as we go.

- `vis`: This gives the name of the input dataset to be self-calibrated.
- `model`: This gives the names of the input models. Several models can be given. Task `selfcal` can take models from different pointings and frequency ranges. Task `gpscal` can take models from different polarisations only.
- `interval`: This gives the self-calibration solution interval time, in minutes. This time must be long enough to integrate an appreciable signal-to-noise ratio (5 or more) on visibilities. On the other hand, it should be short enough so that the gains can be approximated as being constant during this interval. The default is 5 minutes.
- `select`: This selects the $u-v$ data to be used in the self-calibration process. For `selfcal`, you need not perform `source`, `dra`, or `ddec` selection if you use the `selradec` option (see below).
- `minants`: The minimum number of antenna that must be present before the tasks will attempt to find a solution. The default is 3 for phase self-calibration, or 4 for amplitude self-calibration.
- `refant`: The reference antenna (*i.e.* the antenna which has zero phase) to be used in the solution process. If this is not specified, `selfcal` chooses a default for each solution. `gpscal`, however, uses antenna 3 as the default reference antenna.
- `line`: The standard `line` parameter, as described in Section 5.4. This parameter is described in some detail above.
- `clip`: This parameter allows you to set the limit of the faintest believable flux density in the model. For Stokes I or parallel-hand correlations, this is a maximum limit – flux densities below this limit are assumed to be noise or error. For Stokes Q, U and V, this is an absolute limit – flux densities less than |`clip`| are assumed to be noise or error.
- `offset`: The self-calibration tasks can be run assuming a point-source model. A point-source model will be assumed if no `model` parameter is given. In this case, `offset` gives the offset, in arcsec, from the observation centre, of the point source. The default is to assume that the point source is at the observation centre. Note that you should generally also give the `line` parameter when self-calibrating multi-channel data with a point-source model.
- `flux`: For `selfcal`, when using a point-source model, this gives the Stokes I flux density. The default is 1. For `gpscal`, when using a point-source model, this parameter gives the flux density of I, Q, U and V. The default is 1,0,0,0. Also see the `noscale` option below.
- `options`: The `options` parameter, as usual, gives a number of extra processing options. Options common to `selfcal` and `gpscal` are:

  `amplitude` and `phase`: This determines whether the self-calibration tasks solve for just phase (`options=phase`, which is the default) or amplitude and phase solution (`options=amplitude`).

  `mfs`: This option is described in the section on frequency handling (see above).

  `noscale`: Normally the self-calibration tasks scale the gains so that their rms amplitude is 1. This preserves your primary flux density calibration. The `noscale` option prevents this scaling, thus allowing the absolute flux density scale to be changed by the self-calibration process. You *will want* to allow the flux density scale to change if you are self-calibrating several datasets of the same source (i.e. using the same model) separately. For example, you may have several datasets containing different configurations or frequencies of the one source, and yet all these datasets are used to produce the one output image. Using `noscale` ensures that the flux density scales resulting from the different self-calibrations are the same.

  Options specific to `selfcal` are:

`relax`: This relaxes the convergence criteria of gain solutions somewhat, and may be helpful if there are a large number of solutions which fail to converge. This option will be useful when the model is particularly poor. This will be so in the early self-calibration iterations when the primary calibration was quite poor.

`mosaic`: This causes `selfcal` to select only those visibilities whose observing centre is within plus or minus three pixels of the model reference pixel. This is needed if the input visibility dataset contains multiple pointings or sources. By default, no observing centre selection is performed.

Task `gpscal` takes a few options to dictate how it handles XY phases:

`noxy`: Do not attempt to solve for XY phase. By default `gpscal` solves for a constant XY phase on all antennas except the reference antenna.

`xyref`: Solve for the XY phase of the reference antenna (as well as the XY phase of the other antennas). This cannot be used together with the `noxy` option. To use this option, the source should be strongly polarised (at least 5%), and Stokes Q and/or Stokes U models should be given.

`xyvary`: Solve for a variable XY phase. It is recommended that you use this option, although this option does not work for phase-only self-calibration. If used with `options=xyref` then `gpscal` additionally solves for a variable XY phase on the reference antenna.

Summary

We finish with two examples, the first being for `selfcal`.

SELFCAL	
vis=vela.uv.cal	Visibility data-set to be corrected,
	Generally I,Q,U,V visibilities.
model=vela.icmp	CLEAN component image
select	Data to select.
interval=5	Self-cal solution interval.
options=phase	Phase self-calibration, or
options=amp	Amplitude self-calibration, or
options=phase,mfs	Phase-selfcal, MFS model, or
options=amp,mfs	Amplitude self-cal, MFS model
line	You need to specify this for MFS selfcal,
	can generally default otherwise.

The parameters for `gpscal` are very similar.

GPSCAL	
vis=vela.uv.cal	Visibility data-set to be corrected,
	Should be XX,YY,XY,YX visibilities.
model=vela.icmp,vela.qcmp,vela.ucmp	CLEAN component images
select	Data to select.
interval=5	Self-cal solution interval.
options=phase	Phase self-calibration, or
options=amp,xyvary	Amplitude self-calibration, or
options=phase,mfs	Phase-selfcal, MFS model, or
options=amp,mfs,xyvary	Amplitude self-cal, MFS model
line	You need to specify this for MFS selfcal,
	can generally default otherwise.

Chapter 16

Spectral Line Data Reduction

16.1 Velocities in $\mathcal{MIRIAD}$

Because of the Doppler effect, the observed frequency of a spectral line can be related to the radial velocity between the observer and the source. The relativistic expression relating frequency to radial velocity is

$$v = c\frac{\nu_0^2 - \nu^2}{\nu_0^2 + \nu^2},$$

where v is the radial velocity, ν the observed frequency, ν_0 the rest frequency, and c is the speed of light. For various reasons, astronomers usually approximate this formula. There are two common approximations – the "radio definition",

$$v_{\mathrm{radio}} = c(1 - \frac{\nu}{\nu_0}),$$

and the "optical definition",

$$v_{\mathrm{optical}} = c(\frac{\nu_0}{\nu} - 1).$$

The radio definition has the advantage that points sampled at equal increments in frequency translate to equal increments in velocity. However the radio definition is now deprecated by the IAU.

In defining a velocity, a rest frame must also be given. Because of the Earth's diurnal and annual motion (spin on its axis, and rotation around the Sun), the Earth's surface is not a good rest frame. The diurnal and annual motion have maximum velocity components of approximately 0.5 km s^{-1} and 30 km s^{-1} respectively. Two commonly used rest frames are the solar system barycentre (the dynamical centre of the solar system) and the local standard of rest (LSR) (which accounts for motion of the solar system relative to a collection of local stars). The local standard of rest is generally used as the rest frame for Galactic astronomy, whereas the barycentric frame (often misnamed heliocentric frame – which is slightly different) is generally used for extragalactic work. $\mathcal{MIRIAD}$ uses the "definition" of the LSR that the solar system barycentre is moving at 20 km s^{-1} in the direction of (RA,DEC) = 18 hours,+30 degrees (B1900) (this is a so-called "kinematic" definition of the LSR).

Because of the Earth's motion, the frequency that corresponds to a particular velocity of an astronomical source will change with time. When observing spectral lines, many observatories continuously change the observing frequency to account for the effect of the Earth motion, and thus make a particular source velocity correspond to a single channel. This is known as Doppler tracking.

Note, however, that the ATCA does not Doppler track.

This means that the channel that a particular source line corresponds to varies with time. If the velocity resolution is coarse compared to the diurnal velocity component of the Earth (<0.5 km s^{-1}), and only a single configuration of data is being used, this effect can be ignored. However when high velocity resolution is required or multiple configurations are being combined, account must be made of the way the line wanders between channels. That is, account must be taken of the changing radial velocity of the observatory.

For a dataset from an observatory that does Doppler track (i.e. *not* the ATCA), the frequency descriptors in a dataset *should* vary with time. This will be the case with some datasets, but not others – the time varying Doppler frequency component has been lost somewhere. Although this loss will affect the calculation of $u - v$ coordinates, the error that it introduces is generally negligible (see the discussion in the spectral-line processing chapter of Neil Killeen's "Analysis of Australia Telescope Compact Array Data").

Regardless of whether or not the observatory Doppler tracks, *MIRIAD* stores the radial velocity of the observatory (with respect to the rest frame) to account for the Earth's motion when determining source velocities. This is stored in the visibility variable `veldop`.

16.2 ATCA Spectral Line Correlator Configurations

Many of the ATCA correlator configurations are specifically intended for the spectral line observer. One aspect of the ATCA correlator that you should note is that the spectral resolution is often moderately different to the channel increment. For a full description of this, see the memo by Neil Killeen (AT Memo 39.3/057).

16.3 Spectral Line Processing and RPFITS files

When using `atlod` to read in spectral line data, there are a few extra parameters that should be considered

`ifsel`: The ATCA allows you to observe two frequency setups simultaneously`ifsel` allows you to load just one of them by setting `ifsel=1` or `ifsel=2`. The default is to load both of them.

`restfreq`: The RPFITS file does not contain the rest frequency of the spectral line observed, so you need to give this at some stage during data reduction – the earlier the better. The rest frequency can be entered (via `restfreq`, in GHz) in `atlod`.

There are a few `options` in *MIRIAD*'s `atlod` of particular interest to spectral line observers:

`bary`: By default, task `atlod` uses the LSR as the rest frame when computing velocity information. However you can change this to the barycentre by using `options=bary`. Note that LSR velocities are the standard for Galactic astronomy, whereas barycentric are more commonly used by extragalactic astronomers. If you load with the wrong velocity reference frame, you can change this later with `uvredo` (for visibility datasets) or `velsw` (for images).

`birdie`: The ATCA suffers self-interference at frequencies which are multiples of 128 MHz (e.g. 1408 MHz). For spectral line modes, the `birdie` option flags any channels that are affected by the self-interference. This is strongly recommended.

`hanning`: This option performs Hanning smoothing of the spectra, and discards every second channel.

`compress`: This causes the data to be written in a scaled 16-bit format, thus reducing disk usage by a factor of two.

`noif`: If you are loading the ATCA's two frequency setups simultaneously, `atlod` normally tries to stack the two together as "spectral windows" (IFs in *AIPS* terminology). However this will not be possible if the two setups sample different sets of polarisation parameters. In this case, you will need to use `options=noif` to get `atlod` to store the different frequency setups as different visibilities.

16.4 Spectral Line Processing and FITS files

When `fits` reads a visibility FITS file, it can use one of two schemes for computing the observatory's radial velocity. Possible schemes are:

1. `fits` assumes that a velocity of a given channel remains constant. This means that one of the following three conditions is satisfied:

   - Correcting shifts have been performed on the spectra by offline software to account for the Earth's motion (e.g. by *AIPS* task `CVEL` or various tasks in *MIRIAD*).
   - The velocity resolution is sufficiently coarse that Doppler tracking is not required.
   - Doppler tracking has been performed during the observation (not ATCA data).

   If any one of these is satisfied, then given the velocity of a particular channel, `fits` can determine the corresponding observatory radial velocity that this would imply.

   You can either allow `fits` to retrieve the needed velocity information from the FITS file, or you can give this information explicitly using the `velocity` keyword. The default behaviour (if you do not set the `velocity` keyword at all) is to use the information in the FITS file. You will need to explicitly enter the velocity information to `fits` if the information in the FITS file is incorrect.

   The `velocity` keyword gives three values: the velocity definition and rest frame, the velocity of a particular channel, and the channel velocity in km s^{-1}. Possible values for the velocity definition and rest frame are `lsr` (LSR velocity, radio definition), `bary` (barycentric velocity, radio definition), `optlsr` (LSR velocity, optical definition), `optbary` (barycentric velocity, optical definition) and `obs` (velocity relative to the observatory – not of much use). For example, to indicate that channel 257 has an LSR velocity (radio definition) of 4310 km s^{-1}, use

   ```
   velocity=lsr,4310,257
   ```

   Specifying the velocity of a channel for a multi-source FITS file is not a particular meaningful thing to do (unless you are interested in only one source).

2. `fits` assumes that the velocity of a particular channel varies as a result of the Earth's motion. This will be the case for ATCA observations and so should generally be used (unless `CVEL` has been used in *AIPS*, or similar corrections performed in *MIRIAD*). In this case, `fits` will compute the Earth's radial velocity as a function of time. The model of the Earth's motion used by `fits` is accurate to better than 0.005 km s^{-1}– which is adequate for most radio-astronomical applications. `fits` will compute the radial velocity from its Earth model if a velocity frame and definition is given in the `velocity` keyword (either `lsr, bary, optlsr, optbary` or `obs`, as before), but the velocity and reference channel are omitted. For example, `fits` will use its Earth model to determine the observatory's radial velocity with respect to the LSR if given:

   ```
   velocity=lsr
   ```

It is recommended that the second approach is used for ATCA data that has not already been corrected for Earth motion. This is especially so if the velocity information in the FITS file is incorrect.

16.5 Listing and Plotting Velocity Information

The task `uvlist` can be used to list information about the frequency and velocity setup, with the `options=spectral` listing. This prints out the velocities for a visibility record. It is suggested that you do this before proceeding much further, to ensure that the velocity and rest frequency information is what you expect.

The task `varplt` can be used to plot or list the observatory's radial velocity as a function of time. Use the parameter setting `yaxis=veldop`.

16.6 Recomputing Velocity Information

Although `atlod` or `fits` is the best place to set the velocity information, this can also be handled by the task `uvredo`. This task recomputes a number of the 'derived' quantities (such as observatory radial velocity) which are stored in a visibility dataset. You have to select `uvredo`'s `velocity` option, and set the keyword `velocity` to indicate the rest frame to be used. Typical inputs are:

UVREDO	
vis=line.uv	Input dataset name
out=line2.uv	Output dataset name
options=velocity	Recompute velocity information
velocity=lsr	Use the LSR as the rest frame

16.7 Setting the Rest Frequency

The rest frequency of a line may sometimes need to be set or corrected (e.g. you forgot to enter it in `atlod`, or the FITS file failed to contain this information). This can be done with `puthd`.

PUTHD	
in=line.uv/restfreq	Set the parameter `restfreq` of dataset `line.uv`
value=1.420405752d0	Rest frequency, in GHz. Express it as a double precision number.

16.8 Velocity Linetype

As mentioned in Section 5.4, many *MIRIAD* tasks allow a range of channels to be given by their velocities (rather than their channel numbers). Because these channels often do not correspond to any channels that the correlator might produce, they will be call "velocity channels". Velocity channels are specified using the `line` parameter. The velocities are given in km s^{-1}, in the radio or optical definition, with respect to the rest frame of the dataset. The main reason for 'velocity channels' is that they are corrected for the Earth's motion – a particular source velocity component will remain in the same velocity channel regardless of whether the observatory Doppler tracks or not.

The velocity specification is given in the form

`line=velocity,`*nchan*`,`*start*`,`*width*`,`*step*

or

`line=felocity,`*nchan*`,`*start*`,`*width*`,`*step*

where *nchan* is the number of velocity channels to select, *start* is the centre velocity of the first channel selected, *width* is the width of each channel, and *step* is the velocity step between channels. With the `velocity` form, the radio velocity definition is assumed, whereas the `felocity` form uses the optical definition. Note, however, both forms produce channels at equal increments in *radio* velocity (*even the* `felocity` *linetype*). See Section 5.4 for more information.

For example

```
line=velocity,10,1.5,1.0,3.0
```

will specify 10 channels, having velocities (radio definition) centred at 1.5, 4.5, 7.5, etc, km s^{-1}, with respect to the rest frame. Each channel has a width of 1 km s^{-1}.

If the *start*, *width* and *step* values cause the velocity channels not to map directly to the underlying correlator channels, a weighted sum of correlator channels is used to determine a velocity channel. If the magnitude of the velocity channel width (the *width* value in km s^{-1}) corresponds to correlator channel increment, i.e. if

$$|width| = |c\frac{\Delta\nu}{\nu_0}|$$

(where $\Delta\nu$ is the correlator channel increment) then the weighted sum is equivalent to linear interpolation.

For telescopes that do not Doppler track, such as the ATCA, it will rarely be the case that velocity channels map directly to correlator channels because of the continual change in the observatory's radial velocity.

There are some caveats when using velocity channels:

- The rms noise in each velocity channel will vary if there is not a one-to-one mapping between these and correlator channels. Also there may be some noise correlation between consecutive channels. The theoretical noise given by *MIRIAD* will generally be an overestimate. This is worst for a long synthesis when the channel resolution is 0.5 km s^{-1}or finer. In this case, the true theoretical noise will be, on average, a factor of $\sqrt{3/2}$ less than that suggested by *MIRIAD*.
- The "weighted sum" method of determining a velocity channel is inferior to more sophisticated interpolation schemes (e.g. the FFT based interpolation in *AIPS* `CVEL`).

16.9 Continuum Subtraction

Spectral line observations will always be "corrupted" by some continuum emission. If this is stronger or comparable with the spectral line, then the final spectral images may be dominated by the continuum and its artifacts (sidelobes, deconvolution errors, calibration errors, etc). If the final dynamic range of your image is dominated by continuum errors, then weak spectral features will be completely obscured. It is possible to remove the continuum so that the final dynamic range of the spectral image is dependent on thermal noise and artifacts resulting from the spectral line (and *not* artifacts from the continuum). Consequently it is usually best to subtract the continuum from the data. A good description of the algorithms to do this is given by Cornwell, Uson and Haddad (1992). (A&A 258, 583). Neil Killeen's "Analysis of Australia Telescope Compact Array Data" also gives a good summary as well as describing the *AIPS* possibilities. The paper Sault (1994) (A&AS 107, 55) describes one of the algorithms in detail. Yet another summary is given below.

Although three approaches to continuum subtraction are possible in *MIRIAD*, we recommend only one of these – the `UVLIN` approach. The other two are briefly mentioned for completeness:

`UVSUB`: This is the most time consuming of the techniques, which involves forming a continuum image from the line-free channels, deconvolving this and subtracting it from the visibility data-set. In addition to the normal imaging and deconvolution tasks, `uvmodel` will be used to perform the actual subtraction of the continuum model from the visibility data. The `options=mfs` switch will be useful in this context. No more details to this approach will be given here.

`IMLIN`: This is an image-based technique, where a polynomial fit of the continuum is done along the spectral axis at each pixel in the dirty image. Although *MIRIAD* tasks `contsub` and `avmaths` can be used to perform this, no more details will be given here.

`UVLIN`: This is the recommended approach in *MIRIAD* to perform continuum subtraction. It is implemented with the task called (surprisingly) `uvlin`.

In this scheme, each visibility spectrum is fitted by a polynomial. Only the line-free channels of the spectrum are used to determine the fit The polynomial can be taken to represent the continuum emission, and so can be subtracted from the spectrum. Because bandwidths in spectral line experiments are often narrow, the continuum emission is usually well modelled by a fairly low order polynomial (zeroth, first or second order). The fit is performed to the real and imaginary parts of the data separately. In general, this is better than fitting to the amplitude and phase, as the whole process remains a linear one. A non-linear fit to amplitude and phase couples together the errors on all sources, and produces an amplitude bias at low signal-to-noise ratios.

Fit Order	Maximum Residual Continuum Error
0	$S\frac{\pi^2}{6}\eta$
1	$S\frac{\pi^2}{9}\eta^2$
2	$S\frac{\pi^4}{150}\eta^3$
3	$S\frac{\pi^4}{525}\eta^4$

Table 16.1: Approximate residual error for visibility-based continuum subtraction.

The main advantage of `UVLIN` over the other two approaches is that it is generally the most robust to a large variety of systematic errors (such as antenna gain errors). `IMLIN` is also moderately robust to these errors, whereas `UVSUB` is not particularly robust at all. `UVLIN`, along with `IMLIN`, is substantially less expensive computationally than `UVSUB`, and they generally requires less care and intervention on the part of the user.

`UVLIN`, however, is not perfect. There will be some residual continuum left behind, and the noise level in the output will be amplified in those channels that were excluded from the fitting process. These drawbacks are not unique to `UVLIN` – they will be a characteristic of any continuum subtraction technique. There are three parameters to control `UVLIN` – the set of channels to be used in the fitting process (i.e. the set of channels that are assumed to be line-free), the order of the polynomial to fit and an offset to be applied to the data before the fit is performed. The combination of these three parameters will determine the residual continuum. They also determine the factor by which the noise level increases in the data. We clearly want to keep this noise amplification to an acceptable level.

Usually we have little direct control over the channels that are line-free. Obviously, though, the more line-free channels we have, the better the fitting process will be. *Do not discard any line-free channels until after continuum subtraction.*

If there are line-free channels on only one side of the spectrum (i.e. you have to extrapolate the continuum fit), then generally it is unwise to use anything but a 0th order fit. This is because the residual continuum is not greatly affected by the fit order, whereas noise amplification can become extreme.

On the other hand, if there are line-free channels on both sides of the spectrum, and provided the fraction of channels involved in the fitting process is appreciable (at least 25%), then higher order fits are quite useful. The residual continuum depends on the continuum source flux density and the parameter η. For a source at location (ℓ_0, m_0) (with respect to the phase centre),

$$\eta = \frac{1}{2}\frac{\Delta\nu}{\nu}\frac{\sqrt{\ell_0^2 + m_0^2}}{\theta}.$$

Here $\Delta\nu$ is the total bandwidth and ν is the observing frequency. For a point source θ is the resultant image resolution. For resolved sources, θ is the angular extent of typical image features. Unfortunately you cannot really determine η without having formed a continuum image. The maximum residual continuum is approximately as given in Table 16.1. Because of the narrow bandwidth of most spectral line experiments, $\eta < 1$, and so the residual continuum decreases rapidly as the fit order is increased. A zeroth or first order fit will be quite adequate for many spectral line experiments. Given a continuum image, the task `conterr` can determine images which should be indicative of the residual. There should *not* be viewed as strict error images. Rather they indicate the magnitude and location of the residual errors.

As the fit order is increased, noise amplification also increases. Generally, provided we have a significant number of line-free channels (and you are not extrapolating) the noise amplification is modest. You should be cautious of noise amplification for fourth order fits if fewer than 50% of the channels are line-free, or for third order fits when fewer than 25% of the channels are line-free. $\mathcal{MIRIAD}$ task `contsen` can be used to determine the noise amplification given the fitting order and the channels included in the fitting process.

The fact that the residual continuum is a function of the distance of the point source from the phase centre should not be a surprise. As $u - v$ coordinate is proportional to frequency, the channels in a visibility spectrum are sampling slightly different locations in the $u-v$ plane. For a point source (which we assume to have a spectral index of zero, for simplicity) will have a visibility function whose real and imaginary parts are sinusoids. The fraction of a period of this sinusoid contained within the $u - v$ coordinates spanned by a visibility spectrum will be directly proportional to the distance of the point source from the phase centre. The further the point source is from the phase centre, the greater proportion of a sinusoid period is present in the visibility spectrum, and the poorer the approximation of the visibility spectrum by a low order polynomial.

This leads us to another way of reducing the residual continuum – we can shift the phase centre to the location of the continuum source. The amount of benefit derived from such a shift will depend strongly on the skewness of the distribution of the continuum, and its distance from the phase centre. The task `conterr` can suggest shifts which will minimise the error (see `options=shift`).

Computation

For a relatively simple program to drive, `uvlin` has a rather large number of input parameters. Task `uvlin` behaves somewhat like a copying program, taking an input dataset (selecting, calibrating and performing Stokes conversion, as appropriate), performing the continuum subtraction, and writing out a resultant dataset. Input parameters are:

- `vis`: The name of the input dataset.
- `select`: The subset of data to select.
- `line`: The normal line parameter.
- `chans`: This gives the set of channels to be used in the fitting process. It consists of a number of pairs giving the start and end range of the channels to include in the fitting process. Note that there channel numbers are *relative* to the channels selected with the `line` parameter.
- `out`: The output dataset.
- `order`: The fit order. The default is 1.
- `offset`: An offset to be applied to the data before the fit, and "deapplied" after the fit. The net result is that the phase centre of the data is unchanged. However the fitting process is better behaved if the phase centre during the fitting process is located at (or near) the dominant continuum emission. The default is to do no shifting.
- `mode`: This determines what form of data are written to the output dataset. Normally you will want to write the `line` data (the data after continuum subtraction). However you can also write out the fitted continuum (`mode=continuum`) or a fitted channel-0 dataset (`mode=chan0`). Task `uvlin` also has a option where it can solve for line shape (see `options=lpropc` below). In this case, you can also write out the fitted line (`mode=fit`).
- `options`: The `options` parameter gives the normal ability to turn off calibration. Note that you will generally want calibration to be applied, as a copy of the dataset is being formed. It is particularly important to apply bandpass correction. There are a few extra switches in addition to the calibration switches:

  `sun`: With this option, the `offset` keyword is ignored, and a shift appropriate for the Sun is determined instead. Thus, if your continuum is dominated by the Sun, this will help eliminate it. You will normally want to use the "`twofit`" option with this (see below). Also see Sault & Noordam (1995) (A&AS 109, 593) for information and an example.

  `twofit`: If an "`offset`" is given, or if the "`sun`" option is used, `uvlin` normally fits for continuum just at the offset or solar position. By using the "`twofit`" option, `uvlin` will fit for continuum at both the phase centre and the offset/solar position.

`relax`: Normally `uvlin` attempts to avoid overfitting the data if it finds a significant number of flagged continuum channels (more than 40% bad). If it does find a large number of flagged channels, it starts reducing its fit order. The `relax` option tells `uvlin` to fit the order you ask for regardless of large numbers of flagged channels.

`lpropc`: In recombination line experiments, it might be a good approximation that the line strength is proportional to the continuum. Using this extra constraint, `uvlin` can simultaneously fit for both a continuum spectrum (varies from visibility to visibility) and a line spectrum (which is a function of the continuum strength). This results in an iterative algorithm, which can take an appreciable time, with unfortunately only modest improvement in results. You probably do not want to use this option.

Typical inputs to `uvlin` are:

UVLIN	
in=line.uv	Input dataset before continuum subtraction.
select	Leave unset to select all data.
line	Leave unset to select all channels.
chans=1,100,300,512	Channels 1 to 100, and 300 to 512 inclusive contain only continuum emission.
out=line.sub	The output continuum subtracted dataset.
order=1	Use first order fit (which is the default).
offset	The offset to apply. Probably leave unset.
mode	Leave unset to get line data after continuum subtraction.
options	Probably leave it unset

16.10 Imaging and Image Velocity Definitions – INVERT and VELSW

A few comments are in order when using the imaging task, `invert`, to create spectral cubes (more information on `invert` can be found in Chapter 13):

- To conserve space, `invert` produces a single beam dataset for all planes of a cube. For it to do this, the cell size of each plane is scaled with frequency. For the small fractional bandwidths typical in spectral-line experiments, this affect is small. Additionally the coordinate-handling software accounts for this scaling when converting between pixel and celestial coordinates.

  For one beam to be valid for all planes, `invert` usually insists that the $u-v$ coverage for all planes is the same. That is, `invert` usually insists that no channels of a spectrum are flagged before that spectrum can be used for imaging. This is required to ensure each image plane has the same point-spread function.

  This rule can be too restrictive for some uses. For example if deconvolution is not envisaged or if one channel (the birdie channel?) is always bad. `invert`'s `slop` keyword allows the rule to be relaxed. The `slop` keyword takes two values. The first is the fraction of channels that `invert` will tolerate as being bad before a spectrum is rejected. The default is 0 (i.e. the entire spectrum is rejected if even a single channel is bad). A value of 1 indicates that `invert` will accept a spectrum provided at least one channel is good. The second value dictates the 'replacement' method, i.e. what value to assign for the flagged channel, when imaging with flagged channels. Possible values are '`interpolate`' and '`zero`'.

  Interpolation replacement would be appropriate if there are a small number of channels that are always bad, and that you wish to replace them with something for, essentially, aesthetic reasons. Note that the interpolation is simple-minded linear interpolation.

  Surprisingly zeroing the flagged correlation also has its advantages. When zeroing, the true beam (point-spread function) will be channel dependent and will differ from the computed beam. Consequently deconvolution should not be attempted. `invert` attempts to scale each plane so that the true beam has a peak value of 1. This scaling is correct only for natural weighting when no tapering is used (i.e. `sup=0` and `fwhm` unset). If this is the case, the channel images can be astrophysically useful. For other weightings and taperings, the peal value of the true beam will not be exactly 1 – the flux density scale will no longer be what is conventionally understood by Jy/(dirty beam). Additionally the beam peak value can vary from plane to plane, and so the flux density scale may not be comparable between planes. Be warned.

- `invert` checks that the velocity of visibilities gridded onto a particular plane remains constant. The velocity will not remain constant if the observatory does not Doppler track and velocity linetype is not being used. If the velocity of visibilities gridded onto a plane varies by more than 10% of the plane separation, `invert` gives a warning.

- Regardless of the `line` parameter, `invert` labels the velocity axis using the radio definition. However it is possible to switch the labelling of this axis to be either frequency or velocity using the optical definition, or indeed back to velocity using the radio convention. Note that for the optical velocity definition, the velocity separation between planes is not constant. The velocity increment stored in the header is that which corresponds to the reference frequency. The task `velsw` performs this task. Its inputs are fairly simple:

VELSW	
in=line.map	Input image.
axis=optical	Label as velocity using optical definition,
axis=radio	or label as velocity using radio definition,
axis=frequency	or label as frequency.

Chapter 17

Displaying Images

17.1 Introduction

As discussed in Chapter 3, there are two basic suites of display software for *MIRIAD* data. One uses the ATNF Visualisation suite, and the other is PGPLOT-based.

The ATNF visualisation software, which lies outside the domain of *MIRIAD*, uses an X-windows based user interface, with pop-up menus, etc. It is largely interactive, and tends to be less useful when quantitative information is needed. The PGPLOT-based software uses the normal *MIRIAD* user interface, tends to be better suited to hardcopy output, quantitative work, and for more specialised astronomical applications (e.g. plotting polarisation vectors).

17.2 The ATNF Visualisation Software

The ATNF Visualisation software is a suite of tools (outside *MIRIAD*) designed to display images and data cubes (including *MIRIAD* images and cubes). See Chapter 3 for more information.

The main visualisation tool of interest in image display is `kvis`. This tool allows you to inspect one or two 2- or 3-dimensional datasets in several ways. The simplest thing to use it for is displaying a dataset. `kvis` supports flexible zooming, special colour maps for velocity fields etc. For 3-dimensional datasets one can play the channels – as well as the RA-VEL and DEC-VEL slices – of the data cube as a movie, in order to get an idea of the 3D structure of the emission in the data cube.

`kvis` can load two datasets, which can then be viewed simultaneously in several ways, eg. blinking, or by displaying one data set as contour levels on top of the other. The advantage of `kvis` is that one has interactive control over zooming, contour levels etc., so it gives more flexibility than e.g. `cgdisp`. This mode of `kvis` can be used for overlaying a radio-continuum map on an optical image, or the channels of an HI data cube on the continuum or an optical image and inspecting this interactively. Note that for the contouring, the two datasets do not have to be on the same grid, although they should have a proper coordinate system defined. Also note that the visualisation software can read data in `FITS` format so one can load images from e.g. `skyview` directly and overlay *MIRIAD* datasets.

`kvis` can also produce full colour postscript output of whatever you display.

17.3 PGPLOT Device Tasks

There are a number of tasks that display on a PGPLOT device. These devices include, of course, postscript disk files that you can spool to a laser printer (see Chapter 3 for details of how to use PGPLOT). For image display, there is a suite of tasks called `cginit`, `cgdisp`, `cgcurs`, `cgslice`, and `cgspec`. We will discuss the first four in this chapter, and `cgspec` in Chapter 19.

The 'CG' tasks attempt to label the plots in the correct non-linear axes. Provided the field is not too large, the labelling is good (use `options=grid` to see the non-linearities more clearly). However for for very large fields or where the non-linearity is particularly severe (e.g. south celestial pole), the 'CG' tasks will fail to label adequately.

Common Keywords

This suite has many common keywords and much common functionality. We describe briefly now some of the common keywords, although not all the tasks in the suite use them in all possible ways. Refer to the individual help files to see what each task offers. Specific uses by `cgspec` will be described in Chapter 19. Note that minimum match is usually accepted for the value of keywords. Each keyword usually defaults to the same value in all tasks, but you should again refer to the help files for details.

- The keyword `in` takes a list of input images as its arguments. These images are displayed or used in the display in some way indicated by the keyword `type`.
- The keyword `type` indicates how the images given in the `in` keyword will be displayed. These two keywords are in one to one correspondence with each other. Possible values are `pixel` (display image as a pixel map representation, formerly called a grey scale, but now full colour is available), `contour` (display image as a contour map), `box` (display image as little boxes, where the size of the box is proportional to the value of the pixel, and it is hollow for positive and solid for negative values), `amplitude` and `angle` (display these two images as line vectors). In addition, `type` can take on the value `mask`. In this case, the relevant image given in the keyword `in` is not displayed, but its pixel blanking mask is applied to all the other images which are being displayed.
- The standard *MIRIAD* keyword `region` is used to select the region of the images for display. It interacts with the keyword `chan` (see below) which allows you to group planes (or channels) together onto separate subplots. The region selected applies to all the input images and they must all have the same size on the first 2 axes. However, you can input a mix of 3-D and 2-D images. If you input more than one 3-D image, they must all have the same size on the third axis. Any 3-D region selection indicated with the `region` keyword is applied only to the 3-D images that you input. An example of this usage would be overlaying a continuum image on each channel of a spectral line cube.
- The keyword `xybin` allows you to spatially bin up (average) pixels, or to pick out pixels at regular increments. This can be very handy for displaying large images, especially if you are making a hard copy on a postscript printer where the printer resolution does not merit retaining the individual pixels. Using this keyword can make the postscript file much smaller and print much faster.

  You can bin up the two spatial axes of an image independently, and you can enter up to 4 values, 2 for each axis specifying the increment and binning size. As an example, `xybin=4,4,3,1` would bin up the image by 4 pixels in the x direction and pick out every third pixel in the y direction. If the binning size is not unity, it must equal the increment.
- The keyword `chan` is used to allow you to group the planes (or channels) that you selected with the keyword `region` onto separate subplots of the plotting page. The first value you give is the channel increment to step through the image in. The second value is the number of channels to average, for each sub-plot. For example, `chan=5,3` would average groups of 3 planes together, starting 5 channels apart such as 1-3, 6-8, 11-13 and so on.

  The channels available are those designated by the `region` keyword. A new group of channels (sub-plot) is started if there is a discontinuity in the `region` selected channels (such as `region=image(10,20),image(22,30)`. The combination of the `region` and `chan` determines how many sub-plots there will be.
- The keywords `slev` and `levs` (or `levs1, levs2, levs3` in `cgdisp` – it can draw up to 3 contour maps at once) indicate at what values the contours should be drawn. The first value of `slev` can be `p` or `a`, indicating percentage or absolute levels. The second value of `slev` is a scale factor to multiply the contour levels given in `levs` by. For example, `slev=p,1` and `levs=-1,1,2,4,8,16`

would draw contours at -1%, 1%, 2%, 4%, 8% and 16% of the peak value in the image. On the other hand, `slev=a,0.001`, `levs=3,5,10` would draw contours at values of 0.003, 0.005, 0.01.

- The keyword `range` is used to control the way in which pixel intensities are mapped onto the colour lookup table of the device for pixel map representations (`type=pixel`). Its use varies somewhat from task to task, but the basic functionality is the same; see the individual help files for details. See also `options=fiddle` which provides an interactive way to alter the lookup tables.

  The first two values indicate the range of pixel values (intensities) to map onto the lookup table. Pixels with values outside this range will be represented with the colour of the nearest extremum. Thus, if the image was being displayed with a simple linear black and white transfer function on an interactive device, `range=`-0.2, 2.0 would cause all pixels with values below -0.2 to come out black, all pixels with values greater than 2.0 to come out white, and all pixels in between that range to have shades of grey from black to white.

  The third argument of `range` allows you to specify a transfer function so that the pixel values can be mapped onto the lookup table in some way other than linearly. Allowed values are `lin,sqr,log,heq` for linear, square root, logarithmic and histogram equalisation transfer functions respectively. Histogram equalisation can be very handy for images which have a large dynamic range. What this does is use the device colour levels for pixel values which occur the most often.

  The fourth argument of `range` is an integer between 1 and 9 specifying the type of lookup table. The available tables are 1 (b&w), 2 (spectrum colours), 3 (linear pseudo colour), 4 (floating zero colour contours), 5 (fixed zero colour contours), 6 (rgb), 7 (background), 8 (heat) and 9 (absolute b&w) . If you enter a negative integer, then the reversed lookup table is displayed.

  Fixed zero colour contours fix a colour boundary (blue-green) at 0 intensity, with 4 colour pairs ([light blue, light green], [dark blue, dark green], [purple,yellow], [black,orange]) distributed positive and negative of 0. There are then two more colours (red and white) for the remaining positive intensity values. Once you have arranged the colour pairs so that they define the noise level, the red and white colours quickly show you the true signal. You need to use `options=fiddle` to get the scaling to the noise level right.

  Note that in `cgdisp`, you can enter a group of 4 values for each subplot that is drawn. This is useful for hardcopy output, as you can have an individual scaling and lookup table for each subplot. For example, you may have made a "cube" with unlike quantities in different planes (total intensity, polarised intensity, fractional polarisation, rotation measure etc) and it would be impossible to display them all with just one set of 4 values for the `range` keyword.

  The following figure shows the possible colour table types for a simple image. The colour bars or wedges show the differences most clearly.

- The keyword `device` indicates the PGPLOT device upon which the plot will be drawn.

- The keyword `nxy` takes two values, which are the number of subplots in the x and y directions on the page (dictated by the `chan` keyword).

- The keyword `labtyp` specifies what units the axes will be labelled in. It can take two values and they can be different.

  Possible values are:

  - `hms` for a label in H M S.S (e.g. for RA)
  - `dms` for a label in D M S.S (e.g. for DEC)
  - `arcsec` for a label in arcsecond offsets
  - `arcmin` for a label in arcminute offsets
  - `absdeg` for a label in degrees
  - `reldeg` for a label in degree offsets
  - `abspix` for a label in pixels

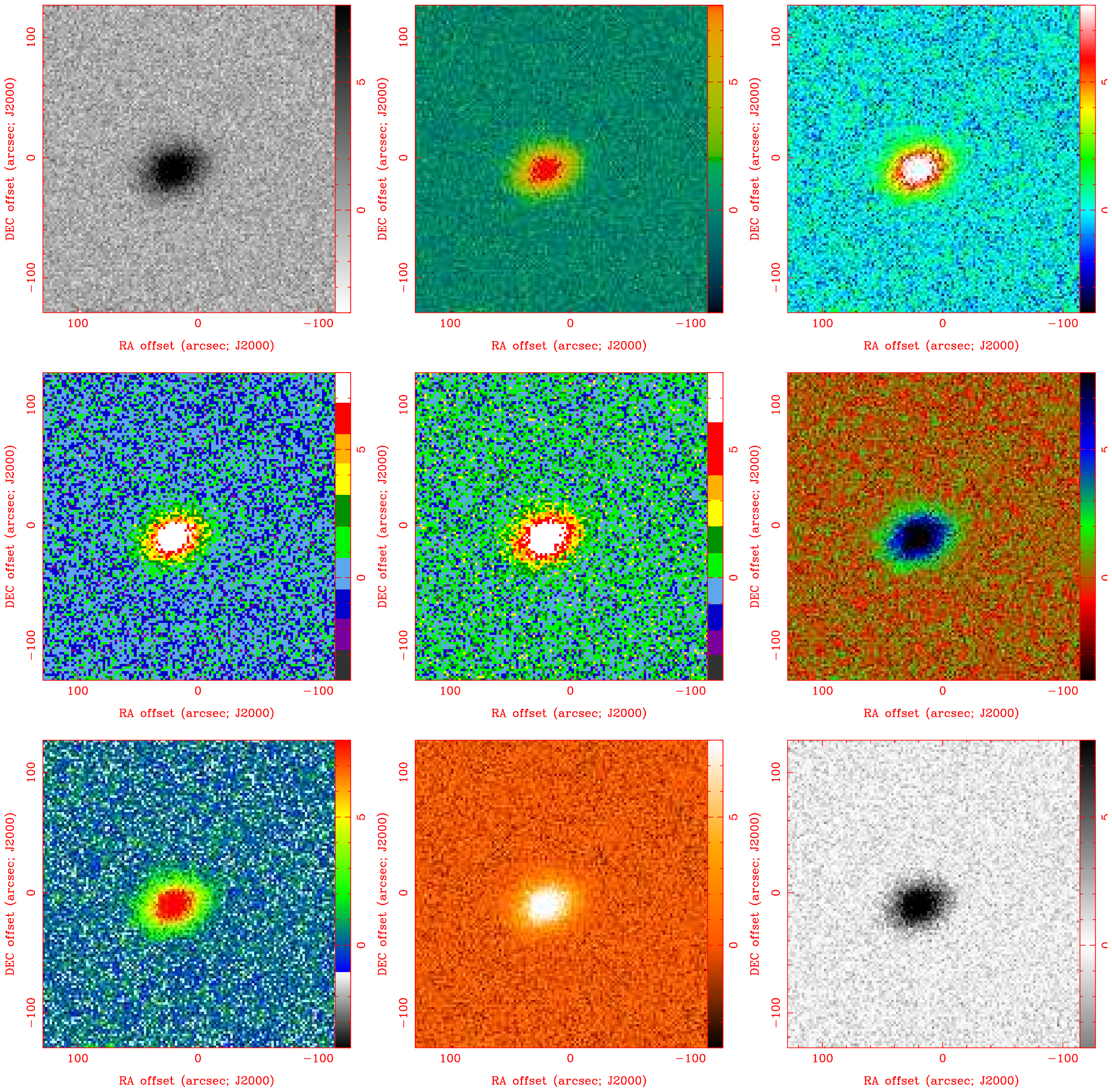

Figure 17.1: Available Colour Lookup Tables

  - `relpix` for a label in pixel offsets
  - `abskms` for a label in km s^{-1}
  - `relkms` for a label in km s^{-1} offsets
  - `absghz` for a label in GHz
  - `relghz` for a label in GHz offsets
  - `absnat` for a label in natural units as given by the axis label.
  - `relnat` for a label in offset natural coordinates
  - `none` for no label and no numbers or ticks on the axis

  All the offsets are with respect to the reference pixel. Note that you are expected to match your values with the order of the axes. For example, if you asked for `labtyp=abskms,dms`, it is expected that the first two axes of the image are velocity and declination. You will get rude messages if this is not the case.

- The keyword `csize` gives you control over the size of characters on the plot such as axis labels, or annotation text. These are given as factors of the default PGPLOT character size `csize=1.0`, which gives a character about 1/40 of the maximum dimension of the view surface. See the help files for details on what you can control in which task.
- The keyword `options` offers a variety of enrichment options for each task. There are a few that are common to most of the tasks in this suite.
  - Option `wedge` causes a labelled colour bar or wedge to be drawn, showing the mapping between pixel intensity and colour.
  - Option `fiddle` invokes the interactive lookup table fiddler. This is mouse (or keyboard) driven, and instructions are given the in the window from which you invoked the task. Basically, it is tree structured, with one branch to select different colour lookup tables, and one branch to modify the transfer function. You can swap between these branches.

    This option allows you to cycle through all of the same colour tables offered by keyword `range` (see above), and reverse them if desired. You can cycle through all of the same pre-defined transfer functions that are available with keyword `range` (see above).

    In addition, there is an interactive linear transfer function fiddle mode, where you can change the slope and offset of the linear transfer function. When you invoke the different transfer function modes, you get a little plot in the bottom right corner showing you what the transfer function is doing.

    Note that if you are plotting on a hard-copy device such as a postscript file, this option is then activated with keyboard inputs. You have the usual ability to cycle through lookup tables and transfer functions, but you do not get the interactive linear transfer function fiddler for obvious reasons. This gives you the same functionality as you could have obtained with the keyword `range`.
  - Option `grid` extends the coordinate ticks into a full grid. Minor ticks are excluded with this option.
  - The option `unequal` instructs the software to fill the plotting page maximally. By default, is tries to make the scales of the x and y axes the same. It is important that you use this option when you are displaying unlike quantities such as a velocity-DEC diagram, as the efforts to make equal scales will go wrong here.
  - Option `relax` instructs these tasks not to be concerned if the axis descriptors for the input images are different. The axis sizes still have to be the same though. Use this option with great care, and it's only useful if you are labelling the axes with `abspix` or `relpix` types.
  - When you are displaying multiple subplots, with each subplot being a different channel or plane from a 3-D image, the options `3value` and `3pixel` instruct the software to write the coordinate value and pixel value of the displayed subplot in its top left hand corner. You can use either or both of these.
  - The option `full` will cause a full set of plot annotation to be written at the bottom of each page of the plot. This includes things like coordinate information, contour levels, pixel map representation intensity ranges and so on.

Examples

Let us continue with some examples of usage of these tasks.

- `cginit` initialises a PGPLOT device. Its only input is the keyword `device`. Most interactive PGPLOT devices are ephemeral, so that `cginit` is not useful for them. However, the PGPLOT `/xs` X window is persistent, so `cginit` is useful for it.
- `cgdisp` can display (simultaneously) contours, pixel map representations (i.e. images or "grey scales"), vectors and boxes (pixels are displayed as filled [positive values] or hollow [negative values] boxes with size proportional to the pixel value) on a PGPLOT device. Up to three contour plots, one pixel map, one vector plot and one box plot may be displayed in multi-panel plots of multi-channel images. It also offers a "mask" image type, which is not displayed, but its blanking mask is applied to all the images that are. `cgdisp` can also display labelled overlay locations (plotted as boxes, stars, circles, ellipses, lines or see-through) which may be specified from an ASCII text file.

  We will just give a few examples of some common things that you might like to do with it.

  1. In the first example, `cgdisp` bins an image up by 2 pixels in the x and y directions, and then displays the image as a pixel map representation on a persistent X window. The transfer function wedge is drawn to the right and then you enter the interactive cursor driven fiddle loop. `cgdisp` then draws some overlay crosses on the plot (the overlay file format is described in the help file) and draws the beam size in the bottom right corner of the plot. The plot is annotated with useful information at the bottom and all four axes are labelled in arcsecond offsets from the reference pixel.

CGDISP	
in=cena.icln	Input image
type=pixel	Display as a pixel map
region=quarter	Display central quarter of image
xybin=2	Bin up image spatially
chan	Unset; no channels to bin
slev	Unset; no contours
levs1	Unset
levs2	Unset
levs3	Unset
range=#,#	Intensity range or unset for full range
vecfac	Unset
boxfac	Unset
device=/xs	PGPLOT device; persistent X window here
nxy=1	One plot per page
labtyp=arcsec,arcsec	Label with offset arcseconds
options=full,beambr,wedge,fiddle,trlab	Full plot annotation,
	draw beam in BRC
	Draw wedge, interactive fiddle
	label top and right axes too
lines=1,2	Linewidths for labels and overlays
break	Unset; no contours
csize=1,1,0.8	Set overlay character size to 0.8
scale	Unset to fill page
olay=stars.txt	Text file containing overlays

  2. The second example contours channels in the velocity range -100 to 100 km s^{-1} from two 3-D images (cubes); each channel is put on a separate sub-plot (but both cubes go on the same sub-plot) on the page and annotated with its velocity. Only every third pixel from the central 40 by 60 arcsec is displayed. The two cubes must have the same axis descriptors; *i.e.* the same axis dimensions and increments etc. The selected channels are binned up such that five channels are averaged together. However, the increment between the start group of each set of binned channels is 10. Thus, the displayed channel width will be one half of the displayed

channel increment (they could of course be equal if you wish). The first cube is contoured via percentage levels, the second via absolute levels. The second cube's contours are drawn with a thicker linewidth. The two cubes could be from different spectral-lines, for example.

Finally, a histogram equalised pixel map representation of a continuum image is overlaid on each channel sub-plot. As the output file is a disk file (a colour post-script file) we have used the `range` keyword to specify the desired lookup table (rainbow colours; fourth argument of keyword) and transfer function (histogram equalisation; third argument of keyword). Alternatively, we could still have used the interactive fiddle to do these things as the dialogue is keyboard driven for non-interactive devices. In fact, if we had set a transfer function via the `range` keyword and then invoked the interactive fiddle, we could have applied another transfer function to the already histogram equalised image !

The `options` keyword is used to ask for an intensity wedge so that the map of colour to intensity can be seen. It is also used to ask for a full coordinate grid on the plot instead of just ticks as well as to annotate the plot with information about the images. Finally it is also used to ask for the value of the third axis (usually velocity or frequency) to be written in the corner of each subplot.

When you ask `cgdisp` to display a mix of 2-D and 3-D images, the region (keyword `region`) that you specify applies equally to the first two dimensions of all images. However, any third axis region information that you give (*e.g.* `region=image(10,120)`) applies only to the 3-D images, and is ignored for the 2-D image. Finally, if you ask for more sub-plots than can be fitted on the page, then the task will advance to a new page (you will be prompted to hit the carriage return key if you are plotting on an interactive device) when needed.

CGDISP	
in=oh.cube,hcn.cube,cont.icln	Input images; two cubes and continuum
type=c,c,p	Two contours and a pixel map
region=arcsec,kms,box(-20,-30,20,30)(-100,100)	Display central 40 by 60 arcsec and -100 to 100 km s^{-1}
xybin=3,1,3,1	Display every third pixel
chan=10,5	Increment is 10 channels, width is 5 channels
slev=p,1,a,0.05	Percentage and absolute levels for cubes
levs1=-5,5,10,30,50,80	Percentage levels for first cube
levs2=-3,3,10,20,30	Absolute levels (by 0.05 for second cube)
levs3	Unset
range=0,0,heq,2	Ask for histogram equalised display of image with rainbow lookup table between min and max.
vecfac	Unset; no vectors
boxfac	Unset
device=plot/cps	PGPLOT device; disk postscript file here
nxy=4,3	4 and 3 sub-plots in x and y directions
labtyp=hms,dms	RA and DEC labelling
options=wedge,grid,3val,full	Wedge, grid, velocity labels & full page annotation
lines=1,2	Linewidths for cubes 1 and 2
break	Unset
csize=1,0.7	Character sizes for axis and velocity labels
scale	Unset to fill page
olay	Unset for no overlays

3. This third example shows how to display a position-velocity image. You could produce such an image by reordering a cube with `reorder` into velocity-x-y order. You could then look at one or several planes of this cube. Normally, `cgdisp` tries to display with equal x and y scales. This equality is worked out using the linear axis descriptors. Equal scales are meaningless when displaying unlike axes, so you should set `options=unequal`.

CGDISP	
in=neon.vxy	Input vxy cube
type=p	Display as a pixel map
region=image(1,16)	Select first 16 planes, say
xybin	Unset for all spatial pixels
chan	Unset for all spectral pixels
slev	Unset
levs1	Unset
levs2	Unset
levs3	Unset
range=#,#	Image intensity range or unset for full range
vecfac	Unset
boxfac	Unset
device=/xs	X window
nxy=4,4	16 sub-plots on the page
labtyp=relkms,arcsec	Relative velocity and arcsecond labelling
options=unequal	Unequal scales in x and y
lines	Unset
break	Unset
csize	Unset
scale	Unset
olay	Unset

4. This fourth example is a polarimetry display. It is common practice to display these images by way of a pixel map representation and/or contours for the total intensity, and vectors for the (fractional) polarised intensity at the specified position angles. The length of the vectors are proportional to the (fractional) polarised intensity. Here is an example of how to do this sort of polarimetric plot, where in this case the total intensity is displayed as both contours and a grey scale. The contours are displayed as multiples of the rms of the noise (you input the rms) in this case, although you can also use percentage-of-the-peak contours (this is controlled by the `slev` keyword – see the help file). You may need to fiddle with the keyword `vecfac` which allows you to scale the lengths of the vectors, and also specifies whether you draw a vector for each pixel or leave some of them out. The default is to draw a vector every second pixel and to scale the vectors so that the longest one takes up 1/20 of the plot. You can display any combination of the images (*e.g.* vectors only), but you must give both the vector position angle and amplitude images.

CGDISP	
in=0823.fp,0823.pa0,0823.i,0823.i	Input images, order unimportant
type=amp,ang,c ,p	Image types
region=arcsec,box(-30,-30,30,30)	Display central 60 by 60 arcsec region
xybin	Unset
chan	Unset
slev=a,#	Contour levels are to be multiplied by the specified number; use the noise rms
levs1=-5,-3,3,5,10,30,50,100	Contour levels
levs2	Unset
levs3	Unset
range=#,#	Image intensity range or unset for full range
vecfac=#	Defaults may be OK, or may require fiddling
boxfac	Unset
device=plot/ps	Postscript file
nxy=1	One plot per page
labtyp=hms,dms	Label with RA and DEC, say
options=full,beambl	Full plot annotation, draw beam in BLC
lines=1,2	Linewidth for contours and vectors
break	Unset
csize	Unset or fiddle with
scale	Unset to fill page
olay	Unset for no overlays

5. The final example is also polarimetry display. Displaying rotation measure images as a pixel map is sometimes unsatisfying, as the sign changes, if any, can be hard to see. Similarly, contours can be pretty useless as the RM distribution is often very unsmooth. `cgdisp` offers the image type "box" whereupon each pixel is displayed as a small box. Positive pixel values are shown as a solid box, and negative pixel values as a frame only. The size of the box is proportional to the values of the pixel, so that this method is not so useful if you have RMs close to zero. By default, the boxes are scaled so that there is a little gap between adjacent boxes. You can control the boz sizes with the keyword `boxfac` which gives an extra scale factor to multiply the box widths. This keyword also gives the x and y increments (in pixels) across the image at which to plot the boxes.

   In this example, we also overlay contours of total intensity. In addition, a global blanking mask is applied to both images.

CGDISP	
in=0823.i,0823.rm,0823.msk	Input images, order unimportant
type=con,box,mask	Image types
region=quarter	Display central quarter
xybin	Unset
chan	Unset
slev=a,#	Contour levels are to be multiplied by the specified number; use the noise rms
levs1=-5,-3,3,5,10,30,50,100	Contour levels
levs2	Unset
levs3	Unset
range	Unset
vecfac=#	Defaults may be OK, or may require fiddling
boxfac=1,3,3	Self scale box widths, and plot boxes every 3 pixels in x and y
device=plot/ps	Postscript file
nxy=1	One plot per page
labtyp=hms,dms	Label with RA and DEC, say
options=full,beambl	Full plot annotation, draw beam in BLC
lines=1,2	Linewidth for contours and vectors
break	Unset
csize	Unset or fiddle with
scale	Unset to fill page
olay	Unset for no overlays

- The task `cgcurs` is similar to `cgdisp` except that it is less flexible in terms of what it can display, but it does offer an interactive capability with the cursor. You can read image pixel locations and values, compute statistics over a region defined by the cursor, and define polygonal spatial regions that can be output into a text file for use in other tasks (using the `region=@`*file* facility). In addition, `cgcurs` can be used to generate text files in a format suitable (almost) for input to `cgdisp` and `cgspec` as overlay files.

  `cgcurs` only displays pixel maps or contour plots, and only one image at a time. You can invoke all of `cgcurs`' cursor options in the one run if you wish. You can also display many channels in the same way as `cgdisp`; in this case, the cursor options are invoked after each sub-plot (channel) is drawn. The following example shows how to display all pixels in the x direction, but only every third pixel in the y direction (who knows why you might do this), activate the interactive lookup table fiddler, read some image values with the cursor, mark their locations on the plot, and output them into a text file (called 'cgcurs.cur') for use as an overlay file in `cgdisp`. In addition, the option to evaluate some statistics in a region defined by the cursor is also activated.

CGCURS	
in=neon.mom0	Input image
type=p	Display as a pixel map
region=arcsec,box(-10,-10,10,10)	Display central 20 by 20 arcsec
xybin=1,1,3,1	Pick out every third y pixel
chan	Unset
slev	Unset
levs	Unset
range=#,#	Image intensity range or unset for full range
device=/xs	PGPLOT device; must be interactive
nxy=1	Just one plot
labtyp=arcsec,arcsec	Offset arcsecond labelling
options=fiddle,curs,stats,mark,log,cgdisp	Fiddle, cursor and statistics options
csize	Unset

- The task `cgslice` allows you to generate 1-D cuts (slices) through a 2-D image and then fit Gaussians to these slices. You can define many slices with the cursor or input their positions via a text file. In addition, you can display many channels from an image, and generate slices differently from each channel. Like `cgcurs`, `cgslice` only displays one image (which could be 3-D of course) at a time via a pixel map or a contour plot. You can save the slice locations, values and model fits in text files.

In the example, we make slices from 9 channels of a cube, one channel at a time. We then fit Gaussians plus a baseline to them and output the fits into a text file.

CGSLICE	
in=1333.icln	Input image
type=p	Display as a pixel map
region=quarter(1,9)	Display central quarter of first 9 channels
xybin	Unset
chan	Unset
slev	Unset
levs	Unset
range=#,#	Image intensity range or unset for full range
xrange	Auto x-range scaling
yrange	Auto y-range scaling
device=/xs	PGDISP server
nxy=3,3	9 plots on the page
labtyp=hms,dms	RA and DEC labelling
options=fit,base	Fit gaussian plus baseline
csize	Unset
posin	Define slices with cursor
posout	Do not save slice locations
valout	Do not save slice values
modout=1333.model	Save Gaussian fits in this file

Chapter 18

Image Analysis

18.1 Image Statistics and Histograms

- Task `histo` is a fairly commonly used task, giving a number of image statistics and a simple histogram. Apart from the input image (keyword `in`), it can take a region of interest (`region`), a range to determine the histogram over (`range`), and the number of bins in the histogram (`nbins`).

HISTO	
in=vela.imap	Input image
region	Select region of interest
range=-0.1,1	Range for histogram. Default is image min and max.
nbins=#	Number of bins in the histogram Unset for default of 20

- If you do not like `histo`, or you would like a PGPLOT plot of the histogram, you could use the task `imhist` instead. This task will also optionally draw, on the histogram, a Gaussian with the same mean, rms and integral as the actual histogram. This task, and its friend `imstat` use the `options` keyword atypically to input numeric (rather than just logical) information – see the help file and the example below.

  Here we plot the histogram with 20 bins with a connected line style. We also do not allow any values below zero to contribute to this histogram.

IMHIST	
in=vela.imap	Input image
region=#	Select region of interest
options=nbin,20,style,connect	Specify desired options
cutoff=0.0	Discard negative pixels
device=/xs	Plot on X window

- Complementing `imhist` is `imstat`. This task gives you statistics about your image in the selected region. It computes the statistic either over a row or a plane of data.

  1. For example, let us look at the rms value of a cube computed over each RA–DEC plane. Note that it does not matter what order the cube is in. For example, it might be in vxy order. `imstat` will average over the xy planes regardless.

IMSTAT	
in=zeeman.vcub	Input image cube
region=image(50,450)	Select region of interest
plot=rms	Plot rms
options	Unset
axes=ra,dec	Compute statistic over RA-DEC plane
cutoff	Unset
device=/xs	Plot on X window
log=numbers.log	Write numbers to log file

2. By setting the `axes` keyword to a single axis, `imstat` will compute the statistic over a single row. Be warned that this can produce a lot of output. For example, if you wanted to see the sum along the velocity axis plotted as a function of RA and DEC position in your cube you could do

IMSTAT	
in=zeeman.vcub	Input image cube
plot=sum	Compute the sum.
axes=velocity	Compute statistic over the velocity axis.
device=/xs	

- You may wish to integrate your image in concentric ellipses, thus producing an azimuthally averaged profile. You can do this with `ellint`. Each plane of a cube is integrated separately. There is no graphical output, just some tables of numbers which you can optionally write to a log file and plot with your favourite plotting program. In this example, we integrate 10 circular rings from 0 to 100 arcsec centred on a pixel offset from the reference pixel by -5 and 10.3 arcsec in RA and DEC. We also make a primary beam correction and write the results to a log file.

ELLINT	
in=ngc1313.mom0	Input image
region	Full region
center=-5,10.3	Offset from reference pixel
pa	Unset for face-on
incline	circular rings
radius=0,100,10	Radii of rings
options=pbcorr	Correct for primary beam
log=rings.1313	Write the results to log file

18.2 Listing Image Values

You might like to list some of the pixel values from a section of an image. You can do this with the task `imlist`. Because it is always very difficult to format numbers in a way which suits every application, `imlist` gives you this job. You do this via the `format` keyword, into which you put a FORTRAN format specifier such as '1pe11.4', or 'f7.2' etc.

IMLIST	
in=ic4296.icln	Input image
options=data	Display pixel values
region=relpix,box(-10,-10,10,10)	Select region
format=f8.3	Select display format

Any pixel which is blanked will be listed as "..." (ellipsis).

18.3 Source Fitting and Positions

- *MIRIAD* has a couple of tasks available for finding positions of sources (see also `cgcurs` in Chapter 17). The simplest of these is `maxfit`. It fits a 2-D parabola to the image intensities in a

3×3 pixel box centred on the brightest pixel in a specified region and reports the fitted location and intensity.

MAXFIT	
in=ic4296.icln	Input image
region=relpix,box(-20,-20,20,20)(1,10)	Search region
log=ic4296.maxfit	Write results to log file

- If you would like to do something slightly more sophisticated than just find a good value for the peak and location of your source, you might like to fit a Gaussian to it with `imfit`. This task can also fit levels and disks.

  In this example, we fit a Gaussian to a region of an image by inputting the region in a file. The file was generated interactively by `cgcurs` (see Chapter 17); you delineate a polygonal region around the source of interest with a cursor. The quality of your fit can improve significantly by selecting a snug region, and/or using the `clip` keyword to keep pixel values with little signal (e.g. sidelobes and noise) out of the fitting process.

IMFIT	
in=ic4296.icln	Input image
region=@cgcurs.region	Fit region
clip=0.1	Only use pixels greater than 0.1 in the fitting process
object=gaussian	Fit a single gaussian
spar	Unset to let task set initial guess
fix	Unset to let everything float
out=ic4296.model	Write residuals of fit out to a dataset
options=residual	Specify residuals

  Task `imfit` can fit multiple components. In this case, you will need to set good initial estimates of the source parameters with the `spar` keyword, and good values for `region` and `clip` become more important.

- There is a task called `gaufit` for fitting Gaussians to profiles from cubes. You are on your own here.

- Tasks often report positional information in absolute image pixel coordinates, but you may often find you would like to convert to some other units. For example, a task has told you that there is something of interest at a certain pixel in your image, but you would like to know its RA, DEC, and velocity. You can use `impos` to convert for you. This task enables you to convert coordinates from one system to another. The keyword `type` specifies the input units of the specified coordinate, given by the keyword `coord`. The coordinate is then converted to all useful types (e.g. pixel to absolute and offset coordinate).

  Note that if you specify a coordinate in km s^{-1} then the keyword `stype` indicates what convention that coordinate is in (radio or optical). You can specify the spectral-axis coordinate in any of frequency and the two velocity conventions, regardless of what the header says the spectral axis is; all spectral axis conversions are done as needed. In addition, any spectral-axis coordinate is converted to each of frequency, and optical and radio convention velocities (if possible).

  See the help file for all the choices.

IMPOS	
in=ic4296.icln	Input image
coord=-20.3,234.5,2400.123	Input coordinate
type=arcsec,abspix,abskms	Input coordinate types
stype=radio	Velocity is in radio conventions

18.4 Copying, Reordering and Regridding Images

- The task `imsub` allows you to extract out some region of one image and copy it to a new image. For example, let us copy every other pixel from the central 20 by 20 arcsec from the first 20 planes of a cube into a new file. Naturally, `imsub` can also be used to make an identical copy of your entire image.

IMSUB	
in=vela.imap	Input image
region=arcsec,box(-10,-10,10,10)(1,20)	Select region of interest
incr=2,2,1	Pixel increments
out=vela-sub.imap	Output image

- The task `imcat` allows you to concatenate images along the third dimension. Thus you can build cubes from 2-D images, and bigger cubes from smaller cubes. One use, for example, is that you might like to put dissimilar images (e.g. 2-D images from different spectral lines) into the one cube and make a Post-Script plot of all of them together on one plot (each plane in its own sub-plot) with `cgdisp`.

  When concatenating dissimilar images, you will probably want to set `options=relax`. This tells `imcat` not to be concerned if axis descriptors differ. However, the image dimensions must still be the same.

IMCAT	
in=*.imap	Input images
out=big.icub	Output image
options=relax	Ignore header discrepancies

- The task `reorder` enables you to reorder the axes of an image. For example, to reorder a cube in xyv order to vxy order and flip the direction of the velocity axis you would do the following:

REORDER	
in=ngc253.xyv	Input cube
out=ngc253.vxy	Output cube
mode=-312	Reordering instructions

- Similar to `reorder` is `imframe`. It too can reorder the axes , but generally we recommend you use `reorder` for that application as its inputs are much simpler than those of `imframe`. `imframe` is useful if you want embed an image inside a larger one and pad the border with zeros.

  In the example, we take an image of some size and pad it out to be 512 by 512 pixels. Note that the `frame` keyword which specifies the output image size in relative pixels is in the non-standard order of `xmin,xmax,ymin,ymax` compared to the `region` keyword which expects `xmin,ymin,xmax,ymax`.

IMFRAME	
in=cena.imem	Input image
out=cena-pad.imem	Output image
region	Unset
box	Unset
frame=-256,256,-255,255	Output size in relative pixels
goal	Unset

- When comparing images, it is often necessary to have them on the same grid so that tasks such as `cgdisp` can overlay them graphically, or so that `maths` (see below) can evaluate some expression involving the two images.

  The task `regrid` enables you to take an image, and regrid it via a cubic interpolation scheme on a new grid. This task can regrid any combination of the the first three axes of an image. The new grid can be specified in one of two ways. The first way (and generally easiest) is to give a template image and make the image of interest look like the template image. The alternative way requires you to specify the output header axis descriptors explicitly with the keyword `desc`. You have to input the reference value, reference pixel, pixel increment and number of pixels for each axis you want to regrid – pretty tedious stuff.

Note that `regrid` has comparatively recently been upgraded to handle non-linear axes correctly (older versions used a linear approximation

In this simple example, we regrid the first two axes of an image via a template image.

REGRID	
in=0336-vla.icln	Input image
tin=0336-atca.icln	Template image
out=0336-vla-atca.icln	Regridded image
desc	Unset for template
axes=1,2	Regrid first two axes only please

Task `regrid` is smart enough to correctly handle different projection geometries, conversion between equatorial and galactic coordinates, equinox conversion between B1950 and J2000, as well as conversions between different velocity systems (radio vs optical, LSR vs barycentric).

If the template does not have quite the geometric grid that you want, then `regrid` provides a number of options to fiddle what it treats as the template coordinate system before it does the regridding. Option `galeqsw` causes the template to be switched from galactic to equatorial, or visa versa before the regridding operation, whereas the option `equisw` switches the equinox from B1950 to J2000 (or visa versa). The `project` keyword can be used to reset the projection geometry of the template. The `noscale` option causes the template coordinate system's cell size *not* to scale with frequency (the images made by `invert` are such that the cell size in inverse proportional to frequency).

Normally `regrid` makes the output exactly overlay the template or the axis descriptors. However, if you really only want them in the same coordinate system, with potentially different image sizes and offset perhaps by an integral number of pixels, then use `options=offset`.

Task `regrid` can be used to resample an image on its own. You would use this to set a different projection geometry, to switch between galactic and equatorial coordinates, or to change equinoxes. If both the template and axis descriptor are unset, `regrid` uses the main input as the template. To do this use `options=offset` along with the other options or parameters to change the template coordinate system.

There is a potential trap when regridding ATCA images to equatorial coordinates: ATCA images produced by `invert` are in the so-called "NCP" projection. As this projection has a singularity at the equator, it is generally disastrous to regrid an ATCA image which is near or on the galactic equator without also changing the projection. The "SIN" (sine) projection is a reasonable choice for this.

18.5 Image Arithmetic – MATHS

The task `maths` evaluates a mathematical expression at each pixel of a set of input images. The expression (given by the `exp` keyword) is given in a FORTRAN-like syntax, using all the normal FORTRAN operators, real-valued functions (using the FORTRAN generic, rather than specific, names), brackets and real constants. Logical operators are also handled with a number zero or less being `.FALSE.`, and a positive number being interpreted as `.TRUE.` (the results of a logical expression are always zero or one). Dataset names take the place of FORTRAN variables, and the expression is each pixel of the input images. For example, `maths` can evaluate expressions such as

```
exp=sqrt(vela.qmap**2+vela.umap**2)
```

This will cause `maths` to take the squares of the pixels in `vela.qmap` and `vela.umap`, add them, and then take the square root.

Note that when there are multiple input datasets, `maths` insists that they are *exactly* the same size. However it does not check whether they align, or indeed whether they use the same axis system. Task `maths` does not check coordinate types.

There are some restrictions on dataset names, however. To start with, `maths` treats the names `x`, `y` and `z` as special (see below). Also there are some inherent ambiguities. For example, `maths` will think `1934-638` is a subtraction of two integers (not a dataset named after a source). Avoid dataset names which start with numerics, or those containing special characters (e.g. plus and minus signs). Also avoid extensions like `.not` (there are situations where this can introduce ambiguities). If you follow the normal FORTRAN or C rules for variable names, ambiguities will not arise. If you insist on using names which could be ambiguous, you can bracket them inside angular braces. For example, `maths` will interpret `<1934-638>` as a dataset `1934-638` (not a subtraction operation).

Another useful (or indeed needed) keyword is `mask`. This gives a second expression (generally a logical expression) indicating where the expression given by `exp` is to be evaluated. For example,

```
exp=sqrt(vela.imap)
mask=vela.imap.gt.1e-3
```

indicates to take the square root of an image *only* when the image value is greater than 10^{-3}. Note that `maths` believes it is your responsibility to protect against illegal mathematical operations (square roots or logs of negative numbers, divide by zero, etc). You must protect any potentially dangerous operations by appropriate masking.

As mentioned above, `maths` treats the names `x`, `y` and `z` in a special manner. They are taken to represent variables that vary linearly across the image, in the x, y and z directions respectively. The values that they take are set with the keywords `xrange`, `yrange` and `zrange`. Expressions can potentially be constructed with no input datasets. In this case, a image size needs to be given via the `imsize` keyword. For example, to generate a two dimensional Gaussian in the range $[-2, 2]$ on both the x and y axes, use

```
exp=exp(-(x**2+y**2))
xrange=-2,2
yrange=-2,2
imsize=128,128
```

The only remaining keywords are `out` (the output dataset) and `region` (region-of-interest in the input datasets).

Here is a simple example of how to use `maths` to make a spectral-index image from two total intensity images. For example, consider two images (`i1` and `i2`) at frequencies 4.8 and 8.4 GHz. You can compute the spectral-index image ($S_\nu \propto \nu^\alpha$) from

MATHS	
exp=log(i1/i2)/log(4.8/8.4)	Divide log ratios
mask=(i1.gt.1e-4).and.(i2.gt.1e-4)	Blank below these values
region	Leave unset to do full image
out=4.8-8.4.spin	Output spectral index image
imsize	Leave unset
xrange	Leave unset
yrange	Leave unset
zrange	Leave unset

In this example, we have computed the output image only when both the input image pixel intensities were above 0.1 mJy/beam (this might be 5-σ or the like).

Alternatively, if you were very keen, you could also get `maths` to create an error image by inserting the appropriate expression (from propagation of errors). Then you could make the spectral index image and blank it according to the value of the error in the spectral-index image via the `mask` keyword.

18.6 Smoothing Images

MIRIAD offers tasks to convolve images by Gaussians (`convol` and `smooth`) and also to convolve images by other images (`convol`). It can also bin up (or block) an image (`imbin`).

- The fastest way to convolve an image by a Gaussian is with an FFT-based algorithm. This is implemented in task `convol`. In the example below we convolve an image by an elliptical Gaussian.

  Note that FFT based algorithms cannot correctly deal with blanked pixels. If your image has a blanked pixels, `convol` pretends that they are zero when it does the convolution. However, the corresponding pixels are blanked in the output image.

CONVOL	
map=1331-09.icln	Input image
beam	Unset
fwhm=10,20	FWHM of Gaussian
pa=45	Position angle of Gaussian
out=1331-09.icln2	Convolved image

  Task generally `convol` does its best to scale the output pixels so that the output image units are Jy/beam. If you are convolving an image which is already in Jy/beam by Gaussian, then `convol` needs the beam parameters (`bmaj, bmin, bpa`) to be in the dataset header to correctly determine the scale factor, and the output effective beam parameters. It will give you messages about what it thinks its doing as far as scaling factors go. If you know better than `convol`, you can give your own scale factor, via the `scale` keyword.

  Often you know the output resolution that you want, rather than the resolution of the Gaussian that you want to convolve with. In this case using `options=final` causes `convol` to treat the parameters given by the `fwhm` and `pa` keywords as the desired final resolution, and to work backwards to determine the Gaussian that it needs to convolve with to achieve this result. Again, `convol` needs to know the beam parameters (`bmaj, bmin, bpa`), and does its best to maintain the intensity units in Jy/beam.

- If your image does in fact contain a blanking mask and it is important that they do not take part in the convolution, then you could convolve your image by a Gaussian (or a boxcar if you so desired) with task `smooth`. This task will be significantly slower than `convol` for any but the smallest images because it operates entirely in the image domain.

  As with `convol`, the scale of the output image is adjusted so that it is in units of Jy/beam – and you can override this by setting `scale` yourself.

SMOOTH	
in=1331-09.icln	Input image
out=1331-09.icln2	Convolved image
fwhm=10,20	FWHM of Gaussian
pa=45	Position angle of Gaussian
options	Unset
scale	Unset for auto scaling

- If you wish to convolve one image by another, you should use task `convol`. Again scaling is as with the above two descriptions.

CONVOL	
map=1331-09.icln	Input image
beam=1331-09.icln2	Convolving image
region	Unset for full image
out=1331-09.convol	Output image
sigma	Unset

- Finally, there is `imbin`; it doesn't smooth an image, rather, it bins up (averages) an image (like the keyword `xybin` in the `cg` suite of programs. You can bin up pixels, and/or pick out every Nth pixel along any of the first three axes; this is controlled by the keyword `bin`.

  In the first example, we bin up the first three dimensions of an image by a factor of 2 along the x axis, a factor of 4 along the y axis and a factor of 3 along the z axis.

IMBIN	
in=gc.icln	Input image
region	Unset for full image
bin=2,2,4,4,3,3	Bin and increments for each axis
out=gc.icln-rebin	Output image

  In the second example, we pick out every 4th pixel along the z axis.

IMBIN	
in=gc.xyv	Input image
region=quarter	Only write out inner quarter
bin=1,1,1,1,1,4	Pick out every 4th pixel along third axis
out=gc-2.xyv	Output image

18.7 Modifying Images by Models

The task `imgen` can be used to create an image with a source of some simple model disposition. It can generate noise, point sources, Gaussians, elliptical disks, Bessel functions and DC terms. It can also be used to modify an already existing image by a model. In the following example, we subtract an offset (by -1.5 and 3.5 arcsec in x and y from the reference pixel) circular Gaussian model from an image. The Gaussian parameters are given in the keyword `spar` (height 0.87 Jy, FWHM of 1.7 arcsec). Note that `imgen` cannot create elliptical Gaussians unless they align with the x and y axes.

IMGEN	
in=ic4296.icln	Input image
out=ic4296-sub.icln	Output image
factor=1	Multiply input by 1
imsize	Unset
object=gaussian	Gaussian model
spar=-0.87,1.7,1.7	Gaussian parameters
xy=-1.5,3.5	Offset from reference pixel
cell	Unset

If you wish to create an entirely new image with `imgen`, you should leave `in` unset, and specify `cell` and `imsize`.

18.8 Polarimetric Analysis Tasks

In this section we see how to make total polarised intensity and position angle images (task `impol`), fractional polarisation images, and rotation measure images (task `imrm`).

- Task `impol` is used to generate images such as the total linearly polarised intensity and position angle (in degrees North through East; see also `options=radians`) from the primary Stokes Q and U images that you produced with `invert`. It can also be used to make the fractional polarisation image if you input the Stokes I image as well. It is convenient to make this image with `impol` rather than, say, `maths`, because `impol` can be used to generate error images as well.

It is generally necessary to deconvolve the Q and U images in the same way that you do the Stokes I image before combining them. Note however, that negative values in these images are as valid as positive values; it just depends upon the position angle of the polarised emission. `impol` can optionally (and by default) debias the total polarised intensity image with a first order algorithm (see help file). A polarised intensity image is positive definite ($\sqrt{(Q^2+U2)}$), even with no signal and the intensities of its pixels follow the Ricean distribution. At high signal-to-noise ratios, this is the normal distribution. At low signal-to-noise ratios, it is skewed biased distribution.

The ability of `impol` to produce error images is very important to assist you in assessing what is real and what is not. These are computed via a simple propagation of errors, and you must specify the rms level of the input image (σ_{QU} is also needed for debiasing the polarised intensity). Tasks `histo` (see above) or `cgcurs` (see Chapter 17) can give you estimates of the relevant statistical information of an image.

`impol` offers the opportunity to blank the output images based upon the signal-to-noise ratio of the polarised intensity image (keyword `sncut`), and also on the absolute value of the position angle error images (keyword `pacut`).

Interpreting polarised intensity images below about 2-σ is pretty deadly because the debiasing becomes inaccurate. In general, it is recommended that you blank the output images in the ways provided. You have the choice of whether you blank the polarised intensity images (not the position-angle image) with zeros or via a bit in a mask. It is important that you use zero if you are planning to evaluate statistics over a region that encompasses the blanked pixels. This is because zero is generally a better estimate of the polarised intensity of a blanked pixel (which has a small signal) than just not contributing to the sums with that pixel.

A case where debiasing and blanking is not appropriate is a detection polarisation experiment. You should ignore `impol` when it implores you to blank and debias in this case.

`impol` can also make a plot showing the effect of bias in the polarised intensity; see the help file for details. You can access this plot even if you do not input any images. The plot is made via a Monte Carlo simulation. It may take a while to run so be patient; just fill in the `device` to activate this option. For extra discussion of debiasing and blanking see the VLA scientific memorandum no 161. by Patrick Leahy.

The following example takes CLEANed Q, U and I images and their rms noise value (assumed the same for Q and U), and computes images of the linearly polarised position angle, the debiased total polarised intensity, the fractional polarisation and their associated errors. These images are *all* blanked when the error in the position angle image is greater than 10 degrees *or* when the signal-to-noise ratio in the polarised intensity image is less than three. Note that the error image for the polarised intensity is constant (and equal the rms of the Q and U images that you input) so that there is no output polarised intensity error blanking option. The choice of the blanking levels depends upon your particular data; do not treat the ones given below as being necessarily appropriate.

IMPOL	
in=qcln,ucln,icln	Input Q, U and I images respectively
poli=p,ep	Polarised intensity and error images
polm=m,em	Fractional polarisation and error images
pa=pa,epa	Position-angle and error images
sigma=0.003,0.004	Set standard deviation of noise in Q or U and Iin Jy/beam
sncut=3	Blank below 3-σ
pacut=10	Blank when p.a. error greater than 10°
options=zero	If evaluating area statistics
rm	Leave unset
device	No bias plot

- The position-angle image you generated with `impol` is the orientation of the electric vectors following the traversal of the photons through the interstellar medium. The position angle is defined to be positive North through East. You can be sure that some Faraday rotation of the electric vectors will have occurred in that magneto-ionic medium. We are ultimately interested in the orientation of the magnetic field before Faraday rotation. `imrm` can be used to generate rotation measure (RM)

and zero-wavelength position-angle (χ_0) images by fitting the equation $\chi = \chi_0 + RM\lambda^2$ to a series of position-angle images (in degrees) at different frequencies. The more frequencies that you can give it the better.

By default, `imrm` tries to remove $N\pi$ ambiguities with the assumption that there are none between the angles of the *first two* given images (note this has changed from an earlier algorithm). The details of the algorithm are given in the help file. Basically, it estimates the rotation measure from these two images, and then estimates the position angle expected at the other frequencies. It then adds or subtracts integral multiples of π radians to make the measured values as close to the predicted values as possible. Then a least squares fit is done to solve for the rotation measure.

You can set `options=ambiguous` to turn this ambiguity removing off, but you should read the help file carefully, and examine the results carefully (you should do this anyway) if you do this. In principle, if there are no ambiguities, the ambiguity removing algorithm should not find any.

The units of the rotation measure, RM, and position-angle images are rad m^{-2} and degrees, respectively.

The keyword `rmi` is very useful. It enables you to specify an initial estimate of the rotation measure. For example, you may know the Galactic contribution in the direction of your source. Angle appropriate to this amount is subtracted from the data before any attempt to compute the rotation measure is made. If it is substantial, it will help quite substantially to reduce ambiguity problems.

`imrm` offers three blanking options for the output images (all of the blanking criteria are applied equally to the rotation measure and position angle images). First, you can blank the output images if any of the input error images exceeds a certain value. Second, you can blank the images if the output errors exceed a certain value, and third you can blank the images if the goodness of fit is less than a certain value. You should think about what these values should be. Remember that you are now generating a tertiary image, and that the errors compound rapidly. Unfortunately, it makes no sense to blank rotation measure and position-angle images based upon a measure of the output signal-to-noise ratio; a rotation measure or position angle of zero does not necessarily imply little signal. Probably blanking with the input position-angle errors (keyword `errcut`) is the best way to blank the output.

You can only compute the goodness of fit if you have position angle images at more than two frequencies, *and* if you input their error images. If it cannot be computed, the goodness of fit is assumed to be unity so that if you have more than 2 frequencies, but do not input the error images, the output errors are computed with this assumption (that is, you cannot get independent estimates of the the output errors and the goodness of fit).

If the goodness of fit is above about 0.1, then the fit is believable. If it is as low as 0.001, the fit may be believable if you have non-normally distributed errors in the position angle images. If it is below 0.001, the fit should probably be rejected; this is the default value for the keyword `qcut`. Note that the distribution of the noise in the position angle images is indeed non-normal at low signal-to-noise ratios. Below $P/\sigma \approx 2$ they are seriously non-normal (but pixels such as these should already have been blanked in `impol`). Above $P/\sigma \approx 5$, the noise is approximated by a normal distribution. The grey area is between 2- and 5-σ.

If you set the keyword `device`, then `imrm` will draw some plots showing the fits to the data. The plots show the data that the least squares fit was done on (i.e. with turns removed as needed), but with the initial estimate `rmi` added back in. Set `options=accumulate` to put all the plots on one panel, else use `nxy` to specify the number of sub-plots per page, with each sub-plot containing the data fit for one spatial pixel.

IMRM	
in=pa.f1,pa.f2,pa.f3,pa.f4	Input p.a. images at different frequencies
inerr=epa.f1,epa.f2,epa.f3,epa.f4	Error images associated with the p.a. images
rmi=35	A priori estimate of RM
rm=rm	Output RM image
pa0=pa0	Output $\lambda = 0$ position-angle image
qcut=#	Blank if goodness of fit below this value
errcut=#	Blank if any input errors greater than this
rmcut=#	Blank if RM error greater than this
pacut=#	Blank if position angle error greater than this
device=/xs	Draw plots please
nxy=10,10	100 subplots per page

18.9 Miscellaneous Analysis Tasks

- The task `imdiff` can be used to compare two images. It finds optimum parameters (in a maximum likelihood sense) for making one image approximate another image. The parameters are an amplitude scale factor, dc offset, shifts in the x and y directions and an expansion factor. Any of these parameters can be fixed at a given value, and the others allowed to vary.

  Note that this task is suited only to small shifts, expansions and scale factors. You can use the self-explanatory task `shifty` for large integral pixel shifts. The keyword `in2` gives the image which must be adjusted to look like the image given by keyword `in1`. Task `imdiff` can also output a residual image – see the help file for the definition of the residual, but it is basically a combination of all the residuals of all the adjustable parameters.

  To hold any particular parameter fixed, you simply specify the appropriate `options` string (see help file). Otherwise, `imdiff` will solve for that parameter. Initial guesses can be entered through the keyword for each parameter.

IMDIFF	
in1=vela.atca	Template image
in2=vela.vla	Adjust this image
adjust=vela.vla-atca	Adjusted image
resid=vela.resid	The residual image
region	Unset for entire image
guard	Unset for default OK
xshift	Unset for initial guess of 0
yshift	Unset for initial guess of 0
expand	Unset for initial guess of 1
amp	Unset for initial guess of 1
offset	Unset for initial guess of 0
options=expand	Unset to vary everything and x and y expansions separately

- A very useful tool for diagnosing errors in images is to Fourier Transform them. The error may, under some conditions, be easier to recognise in the Fourier plane. *MIRIAD* offers the task `fft` for this purpose. It can Fourier Transform real or complex images (you input there real and imaginary parts with separate keywords). Note that blanked pixels in the input are treated as if their value was zero.

  In the following example, we FFT a CLEANed image and form the amplitude and phase images of the Fourier Transform.

FFT	
rin=ic4296.icln	Input real image
iin	Unset
sign=+1	Forwards transform
center	Use ref. pix. for centre of transform
rout	Unset
iout	Unset
mag=fftp.amp	Amplitude image
phase=fft.phase	Phase image

Task `fft` produces gridded images, so you can use all the usual display tools to look at them (see Chapter 17).

- For those of you wrestling with optical data in $\mathcal{MIRIAD}$, the task `impoly` may be of benefit. You may like to make a polynomial sky subtraction. `impoly` allows you to define a region over which a 2-D polynomial is fit. The fit is the subtracted from the the full image, not just the designated fit region.

IMPOLY	
in=cluster.deep	Input image
out=cluster-sub.deep	Output image
order=#	Order of polynomial
coeffs	Unset not to see coefficients of fit
region=@cgcurs.region	Fit over this region

- As is discussed in Chapter 17, there are a variety of transfer functions that you can apply to your image with the display tasks. One that is particularly useful is histogram equalisation. Essentially, this method assesses, via a histogram, which image intensities have the most pixels. It then optimally expends the dynamic range of the device in displaying these pixels, rather than the pixels which have intensities that occur seldom.

  It may be that you would like the image to be written out with histogram equalisation applied, rather than it just being computed but not saved by the display task. The task `imheq` will do this for you. You can specify the intensity range in which pixels will be included when computing the histogram. It is useful to use this if you have outliers significantly from the bulk of the distribution. `imheq` will equalised each plane of a cube, and by default, use the image minimum and maximum of that plane. If you set `options=global` it will take the minimum and maximum from the whole cube instead.

  If you set the `device` keyword, `imheq` will draw a plot of the image histogram and discretized cumulative histogram (see help file). The latter is essentially the transfer function that is applied. This plot is drawn after every plane of a cube.

IMHEQ	
in=cluster.deep	Input image
out=cluster-heq.deep	Output image
nbins	Default for 128 histogram bins
range	Default is image min to max
options	Unset for plane by plane equalisation
device=/xs	PGPLOT device for histogram

- It is sometimes useful to replace the value of all image pixels which are blanked by some number (often zero). Task `imblr` offers this facility; it unblanks all blanked pixels and replaces their value with the one specified.

IMBLR	
in=test.m1	Input image
out=test-2.m1	Output image
value=0.0	Replacement value

Chapter 19

Displaying Spectral Cubes

19.1 Introduction

For sophisticated visualisation software, you are recommended to use the ATNF visualisation suite (briefly described below). On the other hand, *MIRIAD* contains tasks to display simple spectra (`imspec` and `imspect`) or spectra overlayed on an image (`cgspec`).

19.2 The ATNF Visualisation Software

The ATNF visualisation software is a suite of tools (outside *MIRIAD*) specifically aimed at displaying spectral-line data cubes. See Chapter 3 for more information. The tools available are:

- `kvis`: This tool allows you to inspect one or two 2- or 3-dimensional datasets in several ways. This tool is partially described in Chapter 17.
- `kslice_3d`: This tool allows you to interactively inspect data cubes. The RA-DEC, RA-VEL and DEC-VEL slices of the data cube are shown simultaneously, and by moving the cursor one moves through the data cube.
- `xray`: This tool does volume rendering of data cubes. This is very useful for getting an overview of the emission in data cubes, as well as for finding small errors in continuum subtraction or the cleaning of the cube.
- `kpvslice`: This tool allows you to interactively display Position-Velocity slices through a datacube. It provides a window in which you can load an image (say an optical image or a moment map) and draw a slice across the sky. In the other window the corresponding slice through the datacube is displayed. This slice may be exported to *MIRIAD* for further analysis.
- `krenzo`: This tool will draw a contour map for every channel in a datacube over an image, using a different colour for each channel. This is a much better way of showing the velocity structure than using 1st moment maps, since a 1st moment map cannot show multiple peaks in a velocity profile.
- `kshell`: This tool computes and displays Velocity-Radius images from a datacube, allowing you to interactively browse for expanding shells. An expanding shell in a datacube appears as an ellipse in a Velocity-Radius image. This is an easy way of distinguishing between spherical shells and those which have distorted shapes (i.e. a gas bubble which has blown out through a galactic disc).
- `koords`: This tool allows you to easily add a coordinate system to an image (say a CCD image), by using a reference image (perhaps from the DSS). You select matching objects (stars) in the two images, and then press a button to compute a coordinate system for your target image.

19.3 Displaying spectra only

The tasks we discuss here allow you to make a line plot plot of a single spectrum averaged over some spatial region from a cube. There are two rather similar tasks with rather similar names; `imspect` and `imspec`. The latter offers more options, the former is easier to drive if you want something basic. We will give examples of both.

- We begin with `imspect`. The input cube can be in xyv or vxy order. This example plots a 3-point Hanning smoothed spectrum averaged over a small spatial region centred on the reference pixel.

IMSPECT	
in=neon.xyv	Input cube
region=relpix,box(-2,-3,2,3)	Spatial region to average over
xaxis	Default to label x-axis in natural spectral units (km s^{-1} or GHz)
yaxis=average	Average over spatial region
yrange	Auto-scale
hann=3	Hanning smooth
options	Unset
device=plot/ps	PGPLOT device
log	Unset; use to write spectrum to text file
comment	Unset

Note that `imspect` can also be used to plot the derivative of a spectrum (`options=1der` or `options=2der`) which may be useful for Zeeman enthusiasts.

Now for `imspec`, which offers wider flexibility than `imspect`. It also plots spectra from a cube, but the cube can be in any order. It offers options to average or sum the data in RA-DEC, RA-VEL and DEC-VEL planes (keyword `axes` – see the help file for details) so that you get profiles along the velocity, DEC, and RA planes respectively. You can also convert the units from Jy to brightness temperature and write the spectrum to a text file. `imspec` offers Hanning and boxcar spectral smoothing as well as the derivative options available in `imspect`. You can also specify a cut-off level below which data do not contribute to the spectrum and convert from Jy/beam to Jy.

Note that the region you specify with `imspec` is limited to rectangular boxes only. The following example takes a cube in xyv order, computes the spectrum from channels 1 to 256 averaged over the central 4 by 6 arcsec, and then 3-point Hanning smooths it before displaying it on the local X-window.

IMSPEC	
in=neon.xyv	Input cube
region=relpix,box(-2,-3,2,3)(1,256)	Region of interest
plot=average	Plot average
options=hann,3	Hanning smooth
cutoff	Unset
beam	Unset
axes=ra,dec	Average over RA and DEC axes
device=/xs	PGPLOT device
log	Unset

19.4 Overlaying spectra on images

The final task in the `cg` suite is `cgspec`. It displays both images and spectra, but its primary aim is spectral. You are referred to Section 17.3 for extensive discussion of the many common keywords in the `cg` suite of software.

`cgspec` allows you to display an image, and then overlay small plots of spectra at different locations on the image. You can input up to five different 3-D images (cubes) from which the spectra are extracted (*e.g.* each one representing a different spectral-line), as well as up to three contour images and one pixel map representation image. On colour devices, the spectra from different cubes are plotted in different colours. The spectral cubes can be of any size and be in any order. Neither do the pixel increments have to be the same between different cubes. However, the 2-D contour and pixel map images do have to have the same axis descriptors. The locations at which the spectra are drawn can be either on a regular grid or irregularly spaced at locations specified in a text file. In addition, you can bin the spectra up over some spatial region centred on the given locations.

`cgspec` uses the value `spectrum` for the keyword `type` to indicate that spectra are to be extracted from that image listed in the keyword `in`. For Zeeman enthusiasts, you can also plot the derivative of a spectrum image (with keyword `type=dspectrum`). See the help file for details.

The following example shows how to overlay spectra at irregular locations from two cubes (which happened to have different axis orders) on a grey scale. Because spectra from different lines rarely have a similar range of intensities, you can use the keyword `iscale` to scale the spectra from each cube by some number to bring them into the same range. Alternatively, you can normalise all spectra to a peak of 1.0 (`options=normalize`). You may need to specify the size of the spectra as a fraction of the plot window taken up by the spatial image (*i.e.* the grey scale and contour plots), as the defaults of 0.1 may not be suitable.

The only keyword amongst all the scaling keywords that there is no default for is `stick`. This specifies the major tick mark increments on the velocity and intensity axes for the spectra; it is pretty much impossible to guess at a default.

There are a variety of choices about to how you draw the spectra. You can draw labelled borders around them or not (`options=frame`), you can mark their spatial location with a cross or not (`options=mark`; it can be hard to know sometime exactly which pixel the spectrum refers to), you can draw and label the $x = 0$ and $y = 0$ axes or not (`options=noaxes`), you can choose any of these axis options but not put any numeric labels on them (`options=naked`) as things can get pretty cluttered, and you can write a spectrum 'number' in the corner of each spectrum (`options=number`) which reflects the numeric location of the spectrum in the text file or the automatically generated grid. There are a few more options too, so see the help file for more details.

CGSPEC	
in=oh.xyv,6cm.icln,hcn.vxy	Input images
type=s,g,s	Images are spectrum, grey, spectrum
region=quarter	Display central quarter of grey scale
xybin	Unset for no spatial binning
slev	Unset; no contours
levs1	Unset
levs2	Unset
levs3	Unset
grange=#	Grey scale range or unset for auto-scale
vrange=#	Velocity range or unset for auto-scale
irange=#	Spectrum intensity range or unset for auto-scale
iscale=#	Specify a scale factor for each cube or leave unset
spsize=#	Specify size of each spectrum. Default may be OK
stick=100,0.5	Ticks at 100 km s^{-1} and 0.5 in intensity
device=/xd	PGDISP server
labtyp=arcsec,arcsec	Offset arcsecond labelling for images
options=mark,norm,num	Mark location, normalize and number spectra
clines	Unset
slines	Unset; set linewidths for spectra on non-colour devices
blines	Unset; set linewidths for axes on non-colour devices
break	Unset
csize	Unset; character sizes of axes may need fiddling
olay=locations.spc	Specify locations of spectra in this file; see help

Chapter 20

Analysing Spectral Cubes

20.1 Introduction

$\mathcal{MIRIAD}$'s collection of tasks to analyse spectral cubes is of mixed quality. This chapter describes what $\mathcal{MIRIAD}$ has to offer. For some purposes, a better suite of software is available in the ATNF visualisation suite. This suite was briefly outlined in Section 19.2.

20.2 Moment Analysis

The task `moment` can be used to generate moment images from a spectral-line cube. The moment axis should be either the first or third in the cube (e.g. vxy or xyv order). You should use `reorder` if you want the moment of the second axis for some reason. The moments (evaluated for each spatial pixel and along the velocity axis) are defined as

$$M_n = \int I(v) v^n dv$$

where $I(v)$ is the intensity at a given velocity v. Thus, the zeroth moment corresponds to the integrated intensity over velocity, the first moment corresponds to the intensity weighted velocity, and the second moment corresponds to the intensity weighted velocity dispersion squared.

Note that `moment` is actually a little inconsistent in applying this equation depending on which moment it is working out. Here is what it finds:

$$M_0 = \int I(v) dv$$

in units of Jy km s^{-1}.

$$M_1 = \frac{\int I(v) v dv}{\int I(v) dv}$$

in km s^{-1}.

$$M_2 = \sqrt{\frac{\int I(v)(v - M_1)^2 dv}{\int I(v) dv}}$$

in km s^{-1}.

In the example we compute the second moment of a cube in *xyv* order excluding all pixels below 3-σ which happens to be 2 mJy it seems; there is no point to adding noise to our sums. We select only the inner quarter of each spatial plane and we select only planes 40 to 480 for this analysis.

MOMENT	
in=ngc253.icln	Input cube
region=quarter(40,480)	Select region
out=ngc253.m2	Output image
mom=2	Second moment
axis=3	Cube in xyv order
clip=0.002	Clip below this value

20.3 Smoothing the Velocity Axis

The task `hanning` allows you to Hanning smooth the velocity of a cube (regardless of which axis it is) and write out the smoothed data set. Any blanked pixels are treated as zeros.

HANNING	
in=ngc253.icln	Input cube
region	Unset for full cube
out=ngc253.icln-hann	Output image
width=3	Standard Hanning smooth

Having produced the smoothed cube, you could then drop every other channel (as they are no longer independent) with `imsub`. In the example, we assume the velocity axis is the third axis.

IMSUB	
in=ngc253.icln-hann	Input cube
out=ngc253.icln-hann-ind	Output image
region	Unset for full cube
incr=1,1,2	Pick out every other pixel on third axis

20.4 Velplot

`velplot` is a many faceted task for display and analysis of velocity fields – read the *MIRIAD* help file for `velplot`. `velplot` differs in its mode of operation from most other *MIRIAD* tasks in that a small portion only of the inputs are given in the usual *MIRIAD* form. The main body of the inputs are given by the user interactively after selection from an initial menu.

VELPLOT	
in=data.cube	Input data cube to be examined.
device=/xw	Screen is the initial plot device
region=box(32,32,96,96)(5,60)	Examine the centre of planes 5 to 60
log=data.log	Text output where appropriate will go here

Now run `velplot` and the following main menu will appear:

```
        MAIN MENU
   Comment   Write comment into log
   List      List header and velocity information
   Integral  Integrated flux and statistics
   Options   Select plot parameters
```

```
  Pos-Vel   Plot versus position & velocity
  Spectra   Plot spectra
  Vel-map   Plot integrated velocity maps
  File      Write spectra & position-velocity file
  Read      Read spectra & position-velocity file
  Write     Write Miriad Image to disk file
  Exit      Exit from program

 Select option (type 1st character) :
```

This selection is not case sensitive, and only the first character is used. This is what the options are :

Comment A comment will be written into the ASCII file. It goes like this:

```
Select option (type 1st character) : c
Enter comment > It's time I got to work
      comment > I had cereal for breakfast
      comment >
```

You need to get used to the `velplot` input style. You entered 2 lines of comment in response to the `>` prompt. A carriage return on the third line returned you to the `velplot` main menu. If you exit now and examine the file data.log you will find it contains header information from `data.cube` plus those two comment lines.

Also be aware that for many `velplot` inputs gives a carriage return/line feed after the question and you are faced with entering answers on a blank line. This can be quite disconcerting if you do not anticipate it.

List List header and velocity information. The information is written into `data.log` a 4 line summary about `data.cube`

Integral Integrated flux and statistics. This lists statistics for each data plane specified in the `region` originally selected. The headings are: plane (number), Velocity, Total Flux, Maximum, Minimum, Average and rms. Unfortunately this list is not written in to the log file.

Options Select plot parameters. This is what happens:

```
Selection (type 1st character) :o

   OPTIONS MENU
Select alternative options [1/2] for the following

Units for display [J/K]..... J
Negative contours [Y/N]..... Y
Plot header [Y/N]........... Y
Write map to file [Y/N]..... N
Absolute coordinates [Y/N].. N
Integer plot [Y/N].......... N
Spectra positions [Y/N]..... Y
Fit Gaussians [Y/N]......... N
Overlay Gauss Fits [Y/N].... Y
Gray Scale [Y/N]............ N
Exit from options menu
Contour levels:  percent
 level( 1) =    15.000
 level( 2) =    30.000
 level( 3) =    45.000
 level( 4) =    60.000
 level( 5) =    75.000
```

```
  level( 6) =    90.000

Select option (type 1st character,<cr> for options) :u
-plot units are now Janskys

Select option (type 1st character, <cr> for options) :e
```

Now the main menu will reappear. It is easy to get confused here between the main menu and this options menu that selects plot parameters.

The contour setting option is of particular interest in making contour plots. This is the equivalent of the `levs` input to *MIRIAD* `cgdisp`.

Pos-Vel Plots a position velocity map. This is best described by walking through an example. We want a position velocity map made along the major axis of the galactic disk in data.cube. The position angle of the major axis is 40 degrees (measured north to east).

Run `velplot` and the main menu appears. Editorial comments below are in italics

```
Select option (type 1st character) :p
Plot intensity versus position & velocity along selected cuts
>Type H for help, <cr> to continue :<cr>
--- no current selection of cuts ---

>List of cuts, RA-vel or DEC-vel ? [L]/X/Y):<cr>
```

Here, `velplot` in its usual cryptic style is giving us a choice of three options with the default option – which we chose by just hitting carriage return, is in square brackets.

```
--- Enter list of pos-vel cuts to plot ---
angles measured from north to east (0 to 180).
>Type -n to delete cut n
      -99 to delete all
      L to list         (ie list the already chosen cuts)
      <cr> to use the current list.
>Enter cut number, position and angle (n,x,y,pa):1,0,-60,40
```

We think the galactic centre is at 0,-60 arcsec with respect to the plot centre.

```
>Enter cut number, position and angle (n,x,y,pa):<cr> No more cuts wanted
>Enter convolving beam (major("),Minor("),pa(deg):<cr>No convolution requested
Gaussian falls to 6% at edge of array
 size of convolution array:  3 pixels,  90.000 arsecs
Contour or Intensity plots ? [C]/I:<cr>
```

Here is where `velplot` can get very confusing. `C` and `I` are options for quite different functions, not like grey scale or contour options for the same plot. The function we want corresponds to the C option - hence the carriage return.

```
>Enter number of windows in x and y: [1,1]:
<cr>
```

Notice a couple of things here. `velplot` inserted the carriage return after the prompting message and left us dangling there(slap on wrist to the programmer) but it is waiting for input so we enter return to choose the `[1,1]` option, i.e. single plot to the page (like `nxy` option in much of the rest of *MIRIAD*) so something will happen. Something does happen, the pgplot window with the position-velocity map appears on the screen. After dismissing the pgplot window we see some text about the plot on the screen and velplot continues as follows:

```
Another PGPLOT device ? (Y/[N])Y       The last Y is our response
>Enter new PGPLOT device:pvplot.ps/ps Usual convention for a postscript plot
>Enter line width [1]:<cr>            Single width lines

>Enter number of windows in x and y: [1,1]:<cr>
```

Now some `velplot` output about the plot appears on the screen. If you want several position-velocity cuts plotted on the one page you keep answering the "Enter cut number etc" question giving each successive cut a higher number and `velplot` will make a sensible choice of page layout.

```
Another PGPLOT device? (Y/[N])<cr> We do not want any more
```

Now we are finished and the main menu appears again. On exit we find the postscript file `pvplot.ps` in our directory.

Still in the position-velocity option, let us go back to the question

```
Contour or Intensity plots ?[C]/I:
```

and answer `I`. Instead of the standard position-velocity plot we are now requesting plots of intensity versus position at position angles *in certain* (x, y) *image planes that will be requested*. This is a very different function from the standard position-velocity plot that comes from a 3-dimensional slice through the cube. So suppose we have entered 4 cuts:

```
1,0,0,40
2,100,100,30
3,-100,-100,80
4,500,-500,70
```

(to nominate an arbitrary set), the dialogue with `velplot` would be

```
Contour or Intensity plots ? [C]/I: I
How many channels ? (5 max) :3       Data from 3 channels to be plotted
Enter the channel numbers : 25,30,35 Channel numbers 25,30,35 only
Fix the scales ? Y/[N] : <cr>        As before
>Enter number of windows in x and y: [2,2]: <cr>
```

We asked for 4 cuts but the cuts are now in the x,y planes of the specified channels. There will be 4 subplots and each subplot will have 3 lines plotted, one for each specified channel number. And now here is where it gets confusing. The PGPLOT window appears but only one of the 4 subplots.The PGPLOT window is hiding a question being asked by `velplot` of the form:

```
>Enter name for ASCII file(<cr> to continue):
```

So give a carriage return and keep answering with carriage returns to the text window and watch successive subplots appear on the screen. `velplot` will offer the chance to make a postscript file as before and you will have to go through answering the question about the ASCII file for each subplot. Cumbersome but that is the way it is. The ASCII file would be a text list of the values in the plots , if you wanted such.

Spectra Plot spectra. A spectrum is a plot of intensity vs position along a line parallel to the z (velocity) axis. This is an extremely useful function of `velplot` but quite complicated. We will explain via an example:

```
Select option (type 1st character) :s

Plot spectra at selected positions.
(x,y) positions are in (HA,DEC) directions.
>Type H for Help, E to Exit, <cr> to continue :<cr>
--- no current selection of spectra ---

>Enter G to enter a grid of positions,
      <cr> to edit or enter a list of positions :g
```

So we have chosen to define a grid, the most complicated option. Had we given just a carriage return we would have been offered the opportunity to make a list of positions similar to what is done for Position-Velocity plots.

```
>Enter tlc (x,y), and PA of grid:
100,200,0
```

So we are saying that the top left corner of the rectangular grid of positions at which we want spectra is at HA=100arcsec and DEC=200arcsec wrt to the centre of the plot. Our choice of PA=0 here means the grid has axes parallel to the x,y axes. If we had chosen PA=45 the whole grid would have been rotated through 45 degrees.

```
        (Grid PA is measured from N through E)
>Enter interval(arcsec), nrows(along PA) and ncols:
100,2,2
```

So we are asking for a 2 X 2 grid with 100arcsec spacing between elements.

```
--- current selection of spectra ---
 spectra(1) (x,y) arcsecs: (  100.000,  200.000)
 spectra(2) (x,y) arcsecs: (  200.000,  200.000)
 spectra(3) (x,y) arcsecs: (  100.000,  100.000)
 spectra(4) (x,y) arcsecs: (  200.000,  100.000)
```

`selection OK? ([Y]/N):<cr>` *Accept default- that is our list*
`GAUSSIAN FITS TO SPECTRA`
`>Enter convolving beam etc etc :<cr>` *We do not want a convolving beam*
Another 5 lines of text are output
`>Smooth spectra (H=Hanning, B=Boxcar, [None]) :<cr>` *No smoothing thanks*
`>Omit plots and Gaussian fits ? (Y/[N]) :<cr>` *We want the plots*
`>Fix scale for spectra ? (Y/[N] :<cr>`

Had we said `Y` here we would have been offered the opportunity to set the vertical scale in Jy - like the `range` in `cgdisp`

```
>Enter number of windows in x and y: [2,2]:
<cr>
```

So `velplot` sensibly offers a 2 by 2 format for the 4 spectra we have specified. What follows next can be very confusing. The PGPLOT window appears and the first spectrum is plotted in the top left corner as expected but nothing more. Depending on your screen setup, the PGPLOT window is probably obscuring your text output and you do not see that `velplot` has asked

```
>Continue with gaussian fit ([Y]/N) :n
```

and we say No because we do not want a gaussian fit. We told it that before but still we have to go through this to get all the spectral plots on the screen. The same question is asked repeatedly by `velplot` and we answer No each time as we watch successive subplots appear on the PGPLOT screen. Finally, after all the subplots have appeared the same question is asked yet again and when we say No the PGPLOT screen disappears and velplot continues as follows:

`Another PGPLOT device? (Y/[N])Y` *Yes, lets get a postscript file*
`>Enter filename for spectra (<cr> to continue):`
`spectra.ps/ps` *Call the file spectra.ps*

So now we are finished this spectrum plotting exercise. You may get the main menu again or you may be returned to *MIRIAD* and have to restart `velplot` if you want. `spectrum.ps` can now be plotted.

Another example of plotting spectra: Suppose you have a galactic disk data cube and you want to plot spectra at positions along the major axis. Then the grid needs to be collapsed into a line and rotated to the PA of the disk major axis (say 40 degrees). You think the position x,y=-1200,1000 is on the major axis toward the top-left corner of the x,y plane. You want 20 plots at 100arcsec intervals along the major axis. So the inputs to `velplot/spectra` would be:

```
choose grid- g
>Enter tlc (x,y), and PA of grid :
-1200,1000,40
>Enter interval(arcsec), nrows(along PA) and ncols:
100,20,1
```

`velplot` would offer a 5 by 5 plot format but use only the top four rows (for the 20 plots actually 5 by 4)

Vel-map Make plots of various moments of velocity averaged over specified velocity ranges. This works like the position-velocity option above. Type H for additional documentation. The most useful feature here is that the cursor can be used to change the plot parameters, or to plot the positions for spectra, or for position-velocity maps. These CURSOR OPTIONS are only available if a single velocity interval is selected in Vel-map. Here is what happens. The cursor appears in the PGPLOT window, and the CURSOR OPTIONS must be entered by typing a single character in the PGPLOT window. The results, or additional questions appear in the main window. It is easy to get confused here.

```
>CURSOR OPTIONS: Type L to list options :
```

so we type L in the PGPLOT window, and in the main window we see:

```
 Cursor options:
 A - absolute coordinates
 C - change contour interval
 D - clear spec/pos-vel stacks
 E - exit from plot
 X - exit from plot
 G - change grayscale
 H - header label
 J - Jy contours
 K - Kelvin contours
 N - negative contours
 P - define pos-vel cut
 Q - Plot spec/pos-vel positions
 I - integral and rms in box
 B - blc for integral
 T - trc for integral
 R - replot maps
 S - Spectra position
 V - value and cursor position
 W - write out this map
(x,y) positions are in (HA,DEC) directions.
(Position angle is measured from N through E)
>CURSOR OPTIONS: Type L to list options :
```

We can enter a spectra position with the cursor by typing S, and position-velocity cuts by typing P at any two positions along the desired cut. The lists of spectral positions and position-velocity cuts are stored and can be plotted after you return to the main menu.

Typing Cursor option B and T to define a box, followed by I, prints out the Total Flux, Maximum, Minimum, Average, and rms in the box in both Jy and Kelvin units.

Type E or X to exit from the cursor options.

Write Write *MIRIAD* image to disk file. It does just this after prompting for a new name ie it gives you a copy of the region (or all) the input data.

File Write file of spectra and position-velocity cuts. Suppose you have been doing pos-vel and or spectra then there will exist lists of cuts. This option enables you to save the lists to a named file so you could refer to them on a subsequent running of `velplot`.

Read Read file of spectra and position-velocity cuts You saved the lists in a previous WA and now you can retrieve them.

The File and Read options allow you to have several lists of cuts and access them alternately without leaving `velplot`.

20.5 Modelling galactic disks

Fitting a rotation curve

`velfit` is a task to fit a rotation curve to an isovelocity image of a rotating disk. This is classical "moment analysis".

VELFIT	
in=inten,veloc	
centre=0,-60	Rotation centre is 60 arcsec south of map centre.
pa=40	Position angle of major axis.
incline=65	The believed inclination of the disk.
vsys=440	Systemic velocity of the galaxy.
log=rot.log	File to hold results - defaults to terminal.

`inten` is the (x, y) intensity distribution of `data.cube` integrated over the z (velocity) axis. It is obtained using `moment` as follows:

MOMENT	
in=data.cube	
out=iten	
mon=-1	Get average intensity.
axis=3	Squash along the third axis
clip=-2.0,0.0	Or whatever.

`veloc` is an (x, y) model for the velocity values, a mean velocity image. It would be obtained as follows:

MOMENT	
in=data.cube	Input dataset.
out=veloc	The output dataset.
mon=1	First order moment.
axis=3	Squash along the third axis
clip=-2.0,0.0	Or whatever.

Fitting a tilted ring model to a warped galactic disk

Here we illustrate typical use of the *MIRIAD* tasks `velmodel` and `velimage`. To make the tilted ring model one must utilise also `imgen` and `maths` (see Sections 18.7 and 18.5). The warp is simulated by varying the inclination and position angles of successive annuli (usually suggested from various plots from `data.cube` such as closure of isovelocity contours along the major axis and twists in channel maps showing the minor axis). The velocity rotation profile in the plane of the disk may be suggested by inspection of position-velocity maps made with velplot. For the purpose of this exercise we will assume a flat rotation velocity profile in the plane of the disk of 150 km s^{-1}. So we make the 3-dimensional data cube for each tilted ring with `velimage` and add them together with `maths`.

VELIMAGE	
in=aaa,vvv	
sigma=10	Velocity dispersion,mandatory if no 3rd image above
nchan=64	Number of velocity channels in the output cube
start=230	Velocity at channel 1
step=6.6	Velocity increment between channels
out=model.cube	The output cube
options=relax	Usually necessary in this type of scheme.

Obviously, prior to running `velimage`, we must have generated the input files `aaa` and `vvv` which contain our assumptions about the variation in intensity and velocity in our model.

`aaa` is the (x, y) intensity distribution of what will be `model.cube` integrated over the z-axis. In this example it is an elliptical annulus of uniform intensity made by subtracting an inner disk from an outer disk using `maths`, the disks having been made using `imgen` as follows :

IMGEN	
out=aao	Output dataset.
object=disk	Disk model.
spar=0.01,0,-60,1000,500,30	Source parameters, 10 mJy intensity annulus, centred 60 arcsec south of the image centre, major axis=1000 arcsec,minor axis=500 arcsec major axis pa=30 degrees
imsize=128,128	Image size.
cell=30,30	Cell size in arcsec.

IMGEN	
out=aai	The outer dataset.
object=disk	Disk model.
spar=0.01,0,-60,1000,500,30	Source parameters, 10 mJy intensity annulus, centred 60 arcsec south of the image centre, major axis=1000 arcsec,minor axis=500 arcsec major axis pa=30 degrees
imsize=128,128	Image size.
cell=30,30	Cell size in arcsec.

IMGEN	
out=aai	The inner disk.
object=disk	Disk model again.
spar=0.01,0,-60,800,400,30	Similar parameters, but smaller radii.
imsize=128,128	Same image size and
cell=30,30	resolution.

MATHS	
exp=(aao-aai)	Generate a annulus.
out=aaa	Output dataset name.

`vvv` is a model for the velocities (z-axis) made using `velmodel`.

VELMODEL	
in=data.cube pa=30 incline=65 radius=30 vrot=150 vsys=440 out=vvv	Template input.

Note that the input (`data.cube` in the above) a template only but if you use your actual data cube you will finally end up with a model with much of the header information the same as the data, e.g. it will appear to be in the same position in the sky.

One makes a number of such `model.cubes` and adds them together with `maths` to obtain the final 3-dimensional tilted ring model.

20.6 Zeeman Analysis

There is a suite of tasks to help analyse Zeeman data. These tasks implement the techniques discussed in gruesome detail by Sault, Killeen, Zmuidzinas & Loushin (1990) (ApJS 74, 437) and by Killeen, Lo & Crutcher (1992) (ApJ 385, 585).

- `zeestat`: In the small-splitting Zeeman analysis, our goal is generally to look for the characteristic "S" Zeeman signature in the Stokes V spectrum. It can be shown that this spectral signature is more accurately expected to follow the equation

  $$V(\nu) = kB\cos\theta \frac{dI(\nu)}{d\nu}$$

  where k is a constant depending upon the particular spectral line, $B\cos\theta$ is the line-of-sight component of the magnetic field, and $I(\nu)$ is the total intensity spectrum. The procedure is then to fit the derivative of the I spectrum to the V spectrum thus giving $B\cos\theta$.

  Zeeman work can be very complicated, especially if the signal is weak (and the signal-to-noise ratio parameter η – see Sault et al. – is close to the noise value), and a description of all the pros and cons of the fitting techniques is beyond this guide. We suggest you read the above papers, and if you are still awake at the end, then run the software.

  Generally, the fit is done over some spatial and spectral region in which it is assumed the splitting is constant. In the spatial averaging technique, all the spectra in the spatial region are averaged. However, if the spectra are not all similar in shape, this will average down the signal as well as the noise. Thus, the spatial summing technique is also offered. Here, all the spectra in the spatial area are fit simultaneously so that the signal is not averaged down. However, one has to be concerned with the spatial correlation of the data when predicting the errors in this case. This is a very big concern indeed (see `Zeesim` below).

  The fitting procedure is implemented in the task `zeestat`. This task expects the I (keyword `iin`) and V (keyword `vin`) input cubes to be ordered such that the first axis is the spectral axis (frequency or velocity [radio or optical definition]) and the second and third axes are the spatial axes (RA and DEC). Use the `reorder` if they are not. The beam image (keyword `beam`) is used to determine the spatial correlation when using the spatial summing technique in order to get a good estimate of the error in the result (it does not affect the result itself much). However, generally, this does not work well and accurate errors for spatially summed data should be obtained with a monte carlo technique implemented in the task `zeesim` described below. The beam is not needed when spatially averaging.

  If the spectra are spectrally correlated, this information can be transmitted to `zeestat` via the spectral correlation coefficient assigned to the keyword `rho`. For Hanning smoothed spectra in which every other channel has been dropped, this should be 1/6. You must specify with `freq`, to the nearest MHz, the rest frequency of the spectral line. This is used to determine the conversion from splitting in channels to a magnetic field. `zeestat` only knows about a few particular transitions. If it does not know about yours, come and tell us, and we will add it in for you.

You can read endlessly in the above papers about the pros and cons of least squares and maximum likelihood fitting techniques. All the fitting techniques are biased, in the sense that they give a splitting biased towards zero in cases of poor signal-to-noise ratios. The maximum likelihood technique is the least biased, so we suggest you use that (`mode=m`). It is also common to fit for a leakage term allowing for a fraction of the I spectrum in the V spectrum (`mode=l`). Because, in poor signal-to-noise ratio cases, there can be spurious minima, we suggest you set `mode=x` to perform extra checks in finding the global minimum. Generally, sharp edges in spectra increase your chances of detection (because this gives a large derivative). If you set `mode=2`, `zeestat` will work out a two-sided derivative of the I spectrum. Otherwise it uses a one-sided derivative. The former gives a lower noise derivative but performs less well for sharp spectral features. In summary, we suggest you set `mode=mxl` or maybe `mode=mxl2`.

Set keyword `aveop=s` for spatial summing and and `aveop=a` for spatial averaging. `zeestat` is rather an elderly task coded at the dawn of time. It does not know about the `region` keyword. Thus you input the spectral range to fit with the `chan` keyword, and the spatial area via the `blc` (bottom left corner) and `trc` (top right corner) keywords. These are all in absolute pixels. You can write the results to an ascii file with the `out` keyword.

Here is a simple example of a spatially summed fit of spectrally uncorrelated OH main line spectra over channels 200 to 300 (only include channels that have strong I signal) and over some small spatial region.

ZEESTAT	
iin=sgra.Ivxy	Input I cube
vin=sgra.Vvxy	Input V cube
beam	Unset as technique does not work
rho	Unset
freq=1.667	Rest frequency in GHz
mode=mxl	Select mode
aveop=s	Sum in spatial region
chan=200,300	Channel range to fit
blc=240,255	Spatial BLC to fit
trc=260,278	Spatial TRC to fit
log	Unset
out=log.out	Write results to this file

- The task `zeesim` is used to get reliable estimates of the error in the fitted magnetic field derived by `zeestat`. See Killeen, Lo & Crutcher (1992) (ApJ, 385, 585) for gruesome details. It is used when you have invoked spatial averaging, or when the signal-to-noise ratio is very poor for spatial averaging. The technique involves taking the actual spectra and fitting them, thus producing an estimate of the noise free true spectra. Then `zeesim` adds noise (correlated by the beam if simulating spatial summing) back into the spectra and refits them to get the splitting. The noise addition and refitting is performed as many times as is necessary to get a good histogram of the splitting. The width of that histogram gives a good error estimate.

  `zeesim` can work out the fiddle factor to apply to the error estimates of `zeestat` for may spatial regions at once. It is much more efficient for spatial summing to use `zeesim` like this than to do one run per spatial region. This is because the step of producing noise correlated by the beam is very expensive. Multiple spatial regions can be input via a text file assigned to the keyword `infile`. A single spatial region can be entered through the usual `blc` and `trc` keywords.

  In this example, we run the simulation appropriate to the example of `zeestat` above. You should do as many simulations as is necessary to get a decent histogram or until your computer tires.

  It is important to use the `nran` keyword if you restart the simulations and want to use the results in conjunction with a previous run. Computers only generate semi-random numbers. If you give the generator the same seed you get the same random numbers. `nran` allows you to throw some random numbers before beginning the simulation procedure so you can get different noise to last time you ran the program.

ZEESIM	
iin=sgra.Ivxy	Input I cube
vin=sgra.Vvxy	Input V cube
beam=sgra.ivbem	Give the beam
mode=mxl	Select fitting mode
aveop=s	Sum in spatial region
freq=1.667	Rest frequency in GHz
chan=200,300	Channel range to fit
blc=240,255	Spatial BLC to fit
trc=260,278	Spatial TRC to fit
bin	Unset
split	Unset
nruns=500	Number of simulations
	How big is your computer
infile	Unset
log=res,summ	Log files for results
nran=#	Throw away random numbers before simulating

`zeesim` gives you what it calls a "fiddle factor". You should multiply the error estimate from `zeestat` by this factor to get the true error estimate.

Chapter 21

Primary Beams and Mosaicing

21.1 Primary Beams and Primary Beam Correction

In interferometry, the image formed normally by the imaging and deconvolution steps is a representation of the sky *multiplied by the primary beam response* of the antennas. The primary beam is typically similar to a Gaussian function, although it also has sidelobes.

MIRIAD tasks which require knowledge of the primary beam response of a telescope use built-in models of the responses of various telescopes (e.g. ATCA, VLA, Hat Creek, WSRT) – the primary beam model used is determined by the '`pbtype`' or '`telescop`' item or variable (`pbtype`, if present, takes precedence over `telescop`). Currently these models assume the primary beam is circularly symmetric and time independent. The task `pbplot` produces some information and can make a plot of the primary beam models. The model of the ATCA primary beam is described in Mark Wieringa & Mike Kesteven's memo (AT Memo 39.3/024).

If you wish to override *MIRIAD*'s model, or if *MIRIAD* does not have a model of the telescope of interest, you may set the primary beam associated with an image or visibility dataset to be a particular type or Gaussian of a given size. This is done by setting the `pbtype` item using `puthd`. `pbplot` (without any additional parameter) produces a listing of the primary beam types known to *MIRIAD*.

Setting a Gaussian primary beam type differs from setting other primary beams in that you also must give an additional parameter enclosed in brackets) giving the FWHM of the primary beam, in arc seconds, at the reference frequency (the 'reference frequency' for a visibility dataset is the frequency of the first channel imaged!!). For example, to set the primary beam to be a Gaussian of the image `lmc.map`, use:

PUTHD	
in=lmc.map/pbtype	Set `pbfwhm` of `lmc.map`.
value=gaus(1200)	Set the primary beam FWHM to 20 arcmin (1200 arcsec).

Although there are a few exceptions (e.g. `mfspin` and `ellint`), *MIRIAD*'s analysis tasks do not correct for primary beam attenuation automatically. The task to correct an image for the primary beam is `linmos`. To use `linmos` for this function, you need only set the input and output dataset names. For example

LINMOS	
in=lmc.map	Image to primary beam correct.
out=lmc.pbcorr	Output, corrected, image.
options	Leave unset.

Table 21.1: Mosaic grid spacing for ATCA dishes

Frequency ν (GHz)	Pointing Spacing $\theta_{\rm hex}$ (arcmin)
1.384	19.6
2.496	10.9
4.800	5.6
8.640	3.1

21.2 Mosaicing

The primary beam limits the size of an object that we can observe with a conventional experiment. To circumvent this, a large object can be observed using multiple pointings – this is the practise known as mosaicing. In interferometry, mosaicing is not simply the practise of pasting together multiple tiles of the sky. In interferometry, the adjacent pointings are not independent, and so we can get fundamentally better images by processing the different pointings together. This is particularly so for extended emission or when the signal-to-noise ratio is low. A description of the theory behind mosaicing can be found in the NRAO Synthesis Imaging Summer School (Lecture 15 – Wide Field Imaging III: Mosaicing – by Tim Cornwell). Other notable references are Cornwell (1988) (A&A, 143, 77) and Cornwell, Holdaway & Uson (1993) (A&A, 271, 697). Sault, Staveley-Smith and Brouw (1996) (A&AS, 120, 376) describe some of the algorithms used by $\mathcal{MIRIAD}$.

21.3 Mosaicing Observing Strategies

The job of planning a mosaic experiment requires extra thought over a simple conventional observation. Issues that you must decide in the planning of an experiment include:

- Pointing grid pattern: In a mosaic experiment, you observe a number of pointings – possibly a few to several hundred, depending on the size of the source of interest. To consider how dense the sampling grid needs to be, consider the primary beam of an antenna. In the $u-v$ plane, the Fourier transform of the primary beam pattern is just the cross-correlation between two antenna illumination patterns. For the 22-m ATCA dishes, the Fourier transform of the primary beam pattern will be of finite and circular extent, having a diameter of 44-m. Because it is of finite extent, Nyquist's sampling theorem indicates that, provided we do not sample in the sky domain coarser than some limit (i.e. provided the pointing grid pattern is sufficiently fine), all information can be retrieved. Assuming a standard, rectangular grid, the sky plane Nyquist sampling limit is

$$\theta_{\rm rect} = \frac{\lambda}{2D}$$

  (λ is the wavelength, and D is the dish diameter). For a well-illuminated dish, this spacing corresponds roughly to half-power point spacing between field centres. Because the extent of the transform is circular, we can do somewhat better than this, by using a so-called hexagonal grid. This grid places pointing centres at the vertices of equilateral triangles – packing six triangles together gives a hexagon. An extension of Nyquist's theorem indicates that

$$\theta_{\rm hex} = \frac{2}{\sqrt{3}}\frac{\lambda}{2D}.$$

  So a hexagonal grid allows a given area of the sky to be covered in a smaller number of pointings (it does also require slightly longer drive times between pointings – see below – which may occasionally be a consideration). Table 21.1 gives this grid spacing for ATCA dishes.

- Dwell time: Most mosaiced experiments will continually switch between the different pointing centres (or a subset of them, if there are too many pointing centres to visit in a single observation).

Normally they will be visited in a raster scan fashion. Switching to a new pointing centre typically results in 0.5 to 4 seconds of 'lost' time while the antennas are slewing to the new pointing. This time can be a significant consideration in some experiments – e.g. if the integration time was 10 seconds, and the pointing centre was switched every integration, up to about 40% of the observing time could be lost. To avoid this, you will want to dwell on a given pointing centre for as long as reasonable. This must, however, be traded against loss of tangential $u - v$ coverage that occurs when each pointing is not visited sufficiently frequently. To determine the balance, recall that a correlation does not measure the value of a single point in the $u-v$ plane, but a region corresponding to twice the diameter of the dishes. At transit (when the projected baselines are changing fastest), the time taken for a baseline to rotate to a completely independent visibility point is

$$\tau = \frac{86400}{2\pi}\frac{2D}{L} \text{ seconds}$$

Here L is the maximum baseline length of interest when imaging and D is the dish diameter. Ideally you will want to sample twice as frequently as this, i.e. for N pointings, a dwell time of $\tau/2N$ would be best. You may, however, decide to suffer tangential holes in the $u - v$ coverage.

- Field Naming Convention: When preparing the observe files for an ATCA mosaic experiment, you will create a 'mosaic file'. This gives a field offset, integration time and field name for each pointing centre. To simplify a step in the reduction process (the splitting step only), it is recommended that field names be composed of two parts, separated by an underscore character. This recommendation is purely to simplify some steps in the data reduction in $\mathcal{MIRIAD}$. The first part should be common to all fields. Typically this will be the name of the object being mosaiced. The second part is unique to each field, typically being a field number. For example, the field name for pointing 123 for a Large Magellanic Cloud mosaic would be called `lmc_123`.

21.4 Visibility Processing

The flagging, splitting and calibration of a mosaic experiment are rather similar to a conventional experiment. The following concentrates on the differences only. This section will assume that, as is usual, only a single phase calibrator is used for all pointings.

- *Flagging:* Flagging differs only in that, when the instrument is continually changing to a new pointing centre, it can be more difficult to spot the outliers which indicate bad data.

  In its processing, `tvflag` usually produces a gap in its display every time the source being observed is changed. When mosaicing, this can very quickly mount up to taking the entire display (and `tvflag` will complain about not being able to determine an appropriate averaging interval). In this case, `options=nosrc` can be used to suppress the gap and allow `tvflag` to average across source changes.

- *Splitting:* As with conventional reduction, for calibration and imaging purposes, it is easiest in $\mathcal{MIRIAD}$ to split the data into datasets containing a 'single source' and single frequency band. For a mosaic experiment, all the pointings from the object of interest should be considered a 'single source'. As the ATCA on-line system labels each pointing with a different field name, `uvsplit` would normally break the dataset into one file per pointing. To avoid this, use `uvsplit`'s mosaic option. This causes `uvsplit` to use the source name up to any underscore character. Thus, assuming you have used the field naming convention of Section 21.3, `uvsplit` will not split apart all the separate pointings.

  If you failed to follow the naming convention, the `select` keyword of `uvaver` and `uvsplit` can be used to split the dataset up. This is left as an exercise for the reader.

  For example, assuming a dataset containing calibrators and an observation of the LMC (with appropriately named fields), the following inputs will generate datasets for the calibrators and a single dataset for all the LMC data.

UVSPLIT	
vis=mosaic.uv	The input dataset.
options=mosaic	Assume mosaic experiment, so create multi-pointing output where necessary.

- *Data Selection:* With mosaic visibility datasets, the `select=ra` and `select=dec` selection subcommands can be useful to extract or manipulate a subset of the visibility data.
- *Calibration:* Determining the calibration solutions does not differ from that described in Chapter 12. You will then use `gpcopy` to copy the calibration tables to the multi-pointing dataset.

21.5 Summary of Imaging Strategies

There are two quite distinct methods for reducing a mosaic experiment – the so-called "joint" and "individual" approaches. Although hybrid approaches are also conceivable, they will not be discussed.

The joint approach, which is the simplest, takes advantage of $\mathcal{MIRIAD}$'s mosaicing software. In this case, all pointings are handled simultaneously by the imaging and deconvolution software. With the individual approach, a mosaic experiment is treated like a large number of conventional observations, where each pointing is imaged and deconvolved separately. In this case you, the user, are responsible for keeping track of all the pointings. Only as a final step are all the pointings pieced together.

The advantage of the joint approach is speed and simplicity. Also because the deconvolution of all the pointings is done together, it can produce fundamentally better deconvolutions. This is particularly so for low signal-to-noise ratio mosaics and for extended emission (emission comparable in extent to the primary beam – see Cornwell's papers for the argument). It is the approach normally used. However there are disadvantages – the joint approach depends more critically on the model of the primary beam. Errors in the model of the primary beam will tend to be amplified by this approach, particularly when the $u - v$ coverage is poor. Generally the joint approach will be limited to dynamic ranges of several hundred or so.

A more practical difference between the two approaches is that the joint approach generally uses significantly less disk space (this can be an order of magnitude or more for spectral line experiments). However, because the joint approach does all pointings simultaneously, it does use significantly more computer memory in its reduction steps. With current computers, the joint approach is not possible for full resolution ATCA images (6 km array) if you have more that a few pointings.

21.6 The Joint Approach

For the joint approach, the reduction proceeds in a fashion which appears very similar to conventional observations. First a dirty image is formed (with associated point-spread function), and then a deconvolution algorithm is used to 'clean' this dirty image. Finally the 'restore' step is performed. There are, however, substantial differences – although these are largely hidden from the user.

The task to form the dirty image is still `invert`. The dirty image is formed by imaging (using a conventional algorithm) each of the pointings separately. These individual pointing images are them combined in a linear mosaicing process. This linear mosaicing simply consists of a weighted average of the pixels in the individual pointings, with the weights determined by the primary beam response and the expected noise level. The resultant output dirty image is thus an image of the entire region mosaiced.

The weights are computed to minimise the noise in the resultant image as well as to correct for primary beam attenuation. The output image, $I(\ell, m)$, is given by

$$I(\ell, m) = g(\ell, m)\frac{\sum_i P(\ell - \ell_i, m - m_i)I_i(\ell, m)/\sigma_i^2}{\sum_i P^2(\ell - \ell_i, m - m_i)/\sigma_i^2}.$$

Here the summation, i, is over the set of pointing centres, (ℓ_i, m_i). $I_i(\ell, m)$ is the image formed from the

i'th pointing, and $P(\ell, m)$ is the primary beam pattern. The expected noise variance in the i'th field is σ_i^2.

Primary beam attenuation is only corrected for within limits. Because there are large variations in sensitivity across a mosaiced image (the edges of a mosaiced region will have low sensitivity), the imaging software does not always attempt to fully correct for primary beam attenuation. Instead, it constrains the weights so that the noise level does not exceed a certain limit (this limit is based on the noise in individual pointings). This results in some residual primary beam attenuation at the edges of a mosaic (or in holes in the pointing grid). This is done by the function $g(\ell, m)$. This function normally has a value of 1, but its value drops towards 0 at edges or holes. In this way, the noise level across a mosaiced image is crudely uniform.

Task `invert` also applies geometric corrections to account for the fact that the sky is not a plane. For an east-west array, such as the ATCA, these corrections are exact, meaning that the coordinate geometry of the resultant images is also (nominally) exact. For other array types, the corrections are optimal in the sense that they are the best approximation that still results in a convolution relationship (in the sense that such arrays obey a convolution relationship!).

Because the $u-v$ coverage of the different pointings will not be identical, the synthesised beam patterns will differ between pointings. This, and the weighted average process, means that the point-spread function of the resultant dirty image is position-dependent. As most deconvolution algorithms assume a position-independent point-spread function, a conventional algorithm cannot be used. However the point-spread function from the linear mosaicing process is still reasonably compactly described and readily computed. The beam dataset that `invert` produces is not a normal one; it is a cube of beam patterns, one for each pointing. Given this, and some information stored in an auxiliary mosaic information table, the deconvolution tasks can compute the true point-spread function at any position in the dirty image. Being able to compute a point-spread function (or rather, being able to compute a dirty image, given a trial deconvolved image) is the difficult part of writing a deconvolution task. A maximum entropy-based deconvolution algorithm is readily implemented.

The practicalities of this processing are now described.

Imaging – INVERT

Most of the inputs to `invert` are the same as with conventional imaging. Only mosaic-specific considerations will be mentioned here. See Chapter 13 for more information. Note that `invert` supports multi-frequency synthesis, Stokes and spectral imaging.

- `options=mosaic`: The most important thing to remember is to invoke `invert`'s mosaic mode! This causes `invert` to expect multiple pointings in the input visibility data, and to perform the linear mosaicing steps and geometric corrections.
- `options=double`: If you intend to deconvolve, `options=double` should always be used. This is because the full field of the each individual pointing is potentially filled with emission.
- `vis`: When mosaicing, `invert` handles input datasets which contain multiple pointing centres.
- `imsize`: In mosaic mode, this is interpreted as the image size, in pixels, of each subfield. There are two constraints that are important if you wish to deconvolve. These can be relaxed, with corresponding degradation in deconvolution.
  - Ideally `imsize` should be large enough to contain all emission in the main lobe of the primary beam (as $\mathcal{MIRIAD}$ only models the main lobe, making it larger has no beneficial effect). Table 21.2 gives these sizes as a function of frequency for the ATCA.
  - The image size *should not* be a power of 2, or a number within the range (approximately) $[0.9 \times 2^n, 2^n]$ (note that just 1 pixel more than a power of 2 is fine – and indeed good). This restriction is to help reduce the effects of the aliasing caused by a "grid-and-FFT" imaging algorithm which `invert` uses.

Table 21.2: Size of Main Lobe of the Primary Beam for ATCA dishes

Frequency ν (GHz)	Primary Beam Main Lobe Size θ (arcmin)
1.384	70.0
2.496	42.4
4.800	20.6
8.640	12.2

- `sup`: Recall that `sup` gives the sidelobe suppression region, and is a way of setting uniform, super-uniform and sub-uniform weights. In mosaic mode, `invert`'s default is to suppress the sidelobes in a region the size of each sub-image. However suppressing sidelobes in an individual sub-images is not the same as suppressing them in the linear mosaic of the sub-images. This is because the linear mosaicing process weights down sidelobes some distance from the pointing centre. In mosaicing, minimum sidelobes will be achieved by setting `sup` equal to about twice the pointing separation (which should be about the side of the primary beam FWHM).

  As always, best point-source sensitivity results by setting `sup` equal to 0.

- `offset`: This has a different meaning in mosaic mode. It gives the position on the sky (the so-called tangent point), in RA and DEC, which is used for geometry calculations. The value is given in the form *hh:mm:ss,dd:mm:ss*, or as decimal hours and degrees. Normally you can allow this to default, and `invert` will choose a central pointing centre as the tangent point.

Typical inputs to `invert` would be:

INVERT	
vis=lmc.uv	The input multi-pointing dataset.
options=mosaic,double	Use mosaic mode and make large beam.
offset	Usually can leave blank.
map=lmc.map	Output image name.
beam=lmc.beam	Output beam name.
cell=	Set cell size.
imsize=	Set output image size.
sup=2000	Set to approximately twice the pointing separation for minimum sidelobes, or
sup=0	set to zero for best point-source sensitivity.

Deconvolution and Restoration

MIRIAD contains two tasks to deconvolve the mosaiced dirty images produced by `invert`. In terms of theory, practical use and indeed internal implementation, these tasks are quite similar to the deconvolution tasks described in Chapter 14. The major difference is that the 'convolution' operation (which turns a prospective model into a dirty image) is somewhat more involved. Also account must be made of the changing noise level across the dirty image.

The two mosaic deconvolution tasks are `mosmem`, which implements a maximum-entropy-based deconvolution algorithm, and task `mossdi`, which uses a Steer, Dewdney & Ito (SDI) CLEAN algorithm. Generally `mosmem` is superior, although `mossdi` can be better for images containing point sources. Note that, although you can make mosaiced, multi-frequency synthesis images with `invert` (and, indeed, produce a mosaiced, spectral dirty beam), there is no mosaic equivalent to `mfclean`. In deconvolving a mosaiced, multi-frequency image you will have to tactically assume that the spectral index is 0. This should not be a problem – primary beam model errors are probably more significant than spectral errors in these deconvolutions.

If you are deconvolving, note the recommendations for `invert`'s `imsize` parameter, and the use of `options=double`.

If you are familiar with the inputs to the conventional deconvolvers, the inputs to `mosmem` and `mossdi` should be fairly straightforward. In the case of the inputs to `mosmem` and `maxen`, apart from differences in the `options`, the meaning of the `flux` keyword and the default `region`, the only significant difference is in specifying the expected RMS noise level in the dirty image. Because the noise level varies across the dirty image, `mosmem` uses the theoretically expected noise level (which it computes) times a user-specified *fudge factor*, `rmsfac`. That is, if `rmsfac` is set at 1 (the default), then `mosmem` uses the theoretical noise level when calculating its χ^2 statistic.

Typical inputs to `mosmem` are:

MOSMEM	
map=lmc.map	Dirty image produced by `invert`.
beam=lmc.beam	Beam dataset.
model	An initial model estimate – generally unset.
default	The image that the solution should tend towards – generally unset.
out=lmc.model	The output dataset.
niters=30	Maximum number of iterations – default is 30.
region=	Region to deconvolve. The default is the entire image.
measure	Leave unset gives you the Gull measure.
flux=	Estimate of the total flux – its best to give a value.
rmsfac=1	RMS noise fudge factor. Default is 1.
q	An initial estimate of the beams volume. Generally you can leave this unset.
options	Generally leave unset, or
options=doflux	use `doflux` to enforce the flux constraint.

The inputs and use of `mossdi` should be equally simple for someone familiar with `clean`.

Having produced a model, we generally want to convolve this with a Gaussian CLEAN beam and add in the deconvolution residuals. This is done by `restor`. The inputs and use of `restor` is identical to a conventional observation (`restor` is the only general task which is smart enough to recognise a mosaiced experiment directly). Task `restor` uses a constant CLEAN beam – it is not a function of position. The only caveat is that, when determining a default CLEAN beam, `restor` fits a Gaussian to the synthesised beam which corresponds to the first pointing. Provided the first pointing is a fairly typical pointing, this will probably be adequate. Otherwise you may wish to use task `mospsf` (see Section 21.6 below) to generate an actual point-spread function (at some position) and then use `imfit` to determine Gaussian parameters for it.

Typical inputs to `restor` are:

RESTOR	
map=lmc.map	Dirty image produced by `invert`.
beam=lmc.beam	Beam dataset.
model=lmc.model	Model produced by `mosmem`.
mode	Leave unset to get restored image.
fwhm	Beam size – leave unset to let `restor` fit it, but to the first pointing!
pa	Again leave unset to let `restor` fit it.

Self-Calibration

The software to perform self-calibration is workable and reasonable flexible, although it is rather inelegant. In part, this software has not been upgraded from the time before the 'joint approach' suite was developed in *MIRIAD*.

The self-calibration process is performed in two main steps (there is a minor third step). First the task `demos` ("de-mosaic") is used to break the model produced by `mosmem` into models for individual pointings. That is, `demos` produces many models each one of which corresponds to the nominally true sky multiplied by the primary beam pattern at a pointing. The second step is performed by task `selfcal` (`gpscal` cannot cope with mosaiced observations). Task `selfcal` takes all the models simultaneously and then, for each visibility in the input visibility dataset, it computes a model visibility using the model with the same pointing centre. The observed and model visibilities are then processed by a conventional antenna-gain solver, to produce a table of antenna gains vs time.

In reality, antenna gains will be a function of both time and pointing centre. However `selfcal` assumes that the gains are purely a function of time – not pointing. In practice this should not be a great problem, as time and pointing change together, and integrations that are close in time will also be close on the sky. Note that, short of setting a self-calibration solution interval to be smaller than the integration time, you cannot be sure that a solution interval will contain data from a single pointing.

In the above process, only a subset of all pointings need be used in the self-calibration process. If, for example, you have a strong source in one pointing and all the other pointings have only weak emission, it may well be appropriate to assume that the antenna gains are completely independent of pointing. In this case, the gains can be determined from the one strong field.

We now address the steps in more detail:

1. The `demos` step: This step consists of producing a number of models, one for each pointing. The inputs are described in turn.

   - `map`: This gives the name of the input model image (produced by `mosmem`) to be de-mosaiced.
   - `vis` This will usually be the visibility dataset to be self-calibrated. This dataset is used to determine the pointings present and the primary beam to be used.
   - `select`: This provides normal visibility selection. If only a subset of pointings are being processed, it is convenient to select them here. In this way, models are not generated for pointings that are not of interest. Typically, if you wish to self-calibrate with only a subset of pointings, you would use the `ra`, `dec` and/or `source` subcommands to select the appropriate ones.
   - `out`: This gives a template for the names of the output de-mosaiced models. Task `demos` will generate an output name by appending a number to the template name. For example, the output template `lmc.dmos.` would produce names such as `lmc.dmos.1`, `lmc.dmos.2`, etc.
   - `imsize`: This gives the maximum size of the output models. Task `demos` may make the outputs smaller where needed. The default used by `demos` is derived from the primary beam size and the input model, and should be adequate (although if disk-space is tight, you might set a smaller number than that chosen by `demos`).
   - `pbtype`: This gives the primary beam type to use in the de-mosaicing process. The default, which is determined from the `vis` dataset, should be adequate.
   - `options`: You *must* invoke the option `detaper`. This causes `demos` to account for any residual primary beam attenuation that `mosmem` has left.

   Typical inputs to `demos` are:

DEMOS	
map=lmc.model	Model produced by `mosmem`.
vis=lmc.uv	The visibility dataset to be self-calibrated.
select	Leave unset if self-calibrating with all pointings, or
select=source(lmc_123,lmc_124)	select just the fields to be used in the self-calibration process.
out=lmc.dmos.	Output name template.
options=detaper	Account for any residual primary beam attenuation.
pbtype	Generally leave unset.
imsize	Generally leave unset.

2. The `selfcal` step: In general, the inputs to `selfcal` are fairly conventional – see Chapter 15 for more information. There are, however, multiple input models (produced by `demos`) corresponding

to each of the pointings to be used in the self-calibration. Note that wildcards will generally make this easy. The other difference *which you must remember* is to use `options=mosaic` to invoke the mosaicing machinery. Note also that `selfcal` will not use a visibility of a particular pointing if there is no model for this pointing. Thus, if you are self-calibrating using only a few of the stronger pointings, you do not have to explicitly select the data for these pointings.

Typical inputs are:

SELFCAL	
model=lmc.dmos.*	Models produced by `demos`.
vis=lmc.uv	The vis dataset to be self-calibrated.
select	Set as with normal self-calibration.
options=mosaic,phase	Use mosaic mode and phase self-calibration, or
options=mosaic,amp	amplitude/phase self-calibration.

3. Fixing the interpolation tolerance: As noted in Section 12.4, a *MIRIAD* gain table has an associated interpolation tolerance (the time interval over which you can interpolate or extrapolate a gain). Task `selfcal` will set this to the solution interval. If you are self-calibrating with only a few pointing centres, you will want the gains to apply to the entire cycle through the mosaic grid. In this case, you may well want to increase the interpolation tolerance from the default. See Section 12.4 for the details. In summary, you use `puthd` with inputs like:

PUTHD	
in=lmc.uv/interval	Set the 'interval' item of a vis dataset.
value=0.1	Set the tolerance to 2.4 hours (0.1 days).

Some Additional Tools

We briefly describe some other useful tools.

- Mosaic Coverage – MOSLST: In doing a mosaic observation, you will want to good Fourier sampling (coverage) as well as even sampling of the different pointing centres. Task `moslst` produces a simple plot of LST vs point number for mosaiced visibility files. In doing this, it required that you follow the *field naming convention* mentioned earlier (see Section 21.3).
- Listing Mosaic Tables – IMLIST: Task `invert` stores information concerning its linear mosaic operation in the item `mostable` (stored in both the `map` and `beam` datasets). The table can be printed by `imlist`, using `options=mosaic`.
- Mosaic Point-Spread Function – MOSPSF: It is occasionally instructive to look at the point-spread function at a particular position in a mosaic experiment. Task `mospsf` can compute this. Apart from the input beam data-set, the user must specify a position and frequency of interest (the point-spread function is also frequency dependent).
- Mosaic Sensitivity and Gain Images – MOSSEN: Just as the point-spread function varies, so does the expected noise level in an output mosaiced image. Additionally, as mentioned above, `invert` does not attempt to completely correct for the primary beam attenuation where there is too little data (i.e. some primary beam attenuation is in the output image). The task `mossen` can produce images of the expected rms noise and the remaining primary beam attenuation given a mosaiced image.
- MOSTESS: `mostess` implements a mosaicing algorithm identical to the *AIPS* `VTESS` program. It is more cumbersome to use, and produces results which are no better than `mosmem`. Its use is not recommended.
- PMOSMEM: `pmosmem` can be used to do a joint deconvolution of a polarimetric mosaic. The "joint" in this case applies to both jointly processing the multiple Stokes parameters as well as processing the different pointings. With polarised emission being potentially positive and negative valued, the entropy measure for this is different from a simple total intensity deconvolution. See Sault et al. (1999) (A&A 139, 387) for more background on joint polarimetric deconvolution.

21.7 The Individual Approach

The individual approach images, deconvolves, self-calibrates and restores each pointing separately. It is only when you are happy with the individual images that you would combine them – using a linear mosaicing algorithm.

As noted above, the individual approach is less automated and can produce fundamentally poorer deconvolutions. However in some high-dynamic range applications, it may be preferable. The reason for this is that the deconvolution process does not depend on the primary beam model (nor do some errors such as a constant pointing error affect the deconvolution). With the 'individual approach', it is possible to deconvolve sources (and thus eliminate their sidelobes – which is the important issue) beyond the limit where *MIRIAD*'s primary beam model gives up. To do this, however, you must make images larger than the main primary beam lobe (see Table 21.2 above).

In the following, we assume that the 'joint approach' section has been read and understood.

Splitting and Imaging

Although this is largely a matter of taste, it may be convenient (particularly if self-calibration is to be used) to split the multi-pointing visibility dataset into single pointing ones. Task `uvsplit` (with no options, and only the multi-pointing visibility dataset as input) will do this function. It will also copy across any calibration tables associated with the input dataset.

In the individual approach, you will run `invert` many times, once for each pointing (you may have split the multi-pointing visibility dataset into single pointing ones, or you could use selection by source name to select out the appropriate subset of data). Apart from possibly the names of the input and output datasets, the parameters to `invert` should not be changed between runs.

Even though you are imaging just a single pointing, you will still want to use `invert`'s mosaic mode (`options=mosaic`). This causes `invert` to perform its geometry corrections and to create the the images of the different pointings on the same pixel grid. In this way, no interpolation will be needed when the images from the different pointing are finally combined. Consequently the artifacts and problems associated with interpolation can be avoided.

To compute the geometry, however, *you must provide* a reference position on the sky – the tangent point. The default tangent point is the pointing centre of the data being imaged – *this is not appropriate* as it will vary from pointing to pointing. You will want a tangent point which is the same for all the pointings. Although it can be any arbitrary point, it is best to make it near the centre of the source being imaged. If there is a point source which dominates the image, you might choose its position as the tangent point to help reduce deconvolution problems. The tangent point is given through the `offset` keyword, in the format *hh:mm:ss,dd:mm:ss* (or as decimal hours and degrees).

As an example, consider an LMC observation, where we wish to image field 123 (which has field name `lmc_123`). Assuming we have a multi-pointing dataset, and wish to use position (RA,DEC)=(4:30,-71:00) as the tangent point. Typical inputs to `invert` would be:

INVERT	
vis=lmc.uv	The input dataset.
select=source(lmc_123)	Select a single pointing.
options=mosaic	Use mosaic mode.
offset=4:30,-71:00	Set position for geometry computation.
map=lmc_123.map	Output image name.
beam=lmc_123.beam	Output beam name.
cell=	Set cell size.
imsize=	Set output image size.

Deconvolution, Restoration and Self-Calibration

Deconvolution is no different to conventional observing. When restoring, you may wish to use the same restoring beam size for all fields. Otherwise treat each pointing as a separate observation.

Much of the discussion in the self-calibration section of the 'joint approach' applies equally well here. The major difference is, of course, that because the deconvolution step has been performed on the individual fields, the `demos` step is not required. If you have split the visibility data into separate single-pointing datasets, you will not need to use `options=mosaic` (although it will not hurt). Also for single-pointing datasets, if you wish to use just one or a few of the stronger fields for self-calibration, you will need to copy the resultant antenna gain tables across to the other datasets, using `gpcopy`.

Image Combination

When you are satisfied with the deconvolution, restoration and self-calibration of all the individual images, task `linmos` can be used to combine them in a linear mosaic. Usually you will just combine the restored images (if you are going to do quantitative analysis on the composite image, it may be best to do a deep CLEAN and use the same restoring beam for all pointings). Although `linmos` can interpolate input images to align them, its algorithm, particularly for geometric correction, is very poor, and so this is *strongly* discouraged. You should use `invert` to make all the input images on the same grid, by setting a common tangent point (`offset` keyword).

Task `linmos` uses the same weighted sum of the input pointings as the 'joint approach' software (see Section 21.6). Normally the expected rms noise in the image is determined from the images themselves (image item `rms`). However if this item is missing, or if you wish to override it to get a different weighting, you may enter the expected rms noise via keyword `rms`. Also note that `linmos`, by default, *fully corrects for the primary beam attenuation* even when this excessively amplifies the noise. The `taper` option can be used to reduce the correction at the edge of the field, and thus avoid excessive noise amplification.

Typical inputs to `linmos` are:

LINMOS	
in=lmc_*.cln	Use wildcards to select all images.
out=lmc.mos	The output linearly mosaiced image.
rms	Generally left unset.
options	Leave blank to fully correct primary beam,
options=taper	or set to taper at the edge of the mosaic.

Miscellaneous

In a similar way to `mossen`, `linmos` can also produce an image giving the expected rms noise as a function of position, and a gain image – see the help file on using the `options` keyword for these.

21.8 Combining Mosaics and Single Dish Data

For a good discussion of the theory and practice of combining single dish and mosaic data, see the paper by Snezana Stanimirovic (ASP Conf. Series, vol 278, p 375).

One of the benefits of mosaicing is that it partially recovers spacings shorter than the shortest projected interferometer spacing. In principle, the shortest spacing present in a mosaic can be the projected interferometer spacing minus the antenna diameter, D (see the references mentioned earlier for the argument). When antennas are as closely packed as is physically possible, the minimum physical spacing will be D, and so in principle a mosaic can reduce the effective minimum spacing almost down to the zero spacing. In practise, mosaicing tends to reduce the effective minimum spacing by about $D/2$, rather than the theoretical D. For the ATCA observations of sources appreciable far south, the effective minimum spacing in a mosaic is about 20 m (the minimum physical interferometer spacing is 31 m).

So whereas mosaicing helps recover short spacings, there are invariably some short spacing missing. To fill these spacings, interferometer data must be augmented with single-dish data. Just as there are two approaches to mosaicing (the joint and individual approach), *MIRIAD* provides two approaches to single-dish combination – the linear and a non-linear methods, using tasks `immerge` and `mosmem` respectively. Which method produces best results is quite problem specific, and indeed it is perhaps best to try both if possible. However,

- The non-linear combination method works on the dirty mosaic produced by `invert`, whereas the linear method works from deconvolved images. Thus the non-linear method cannot be used with the "individual" mosaicing approach.
- The linear method can be more robust to the single-dish data failing to satisfy some of the assumptions of the non-linear method.
- The non-linear technique can perform better when there is emission right to the edge of the sampled region of the sky.
- The linear method assumes that the single-dish point-spread function is a gaussian. For the non-linear method, you give a "beam" dataset, which can be an actual measurement of the single-dish point-spread function (which can be asymmetric).

Given that combining mosaic and single-dish data can be a bit of an art, its worth doing a few checks of the result:

- The simplest check is to see that the total flux agrees with what you expect. Task `histo` is the simplest way of measuring the total flux in a continuum image (or plane of a spectral cube). Otherwise `imstat` can be used.
- The non-linear methods produce *model* outputs, that need to go through a restore step. Task `restor` will also generate residual images. You probably want to look at the residuals for both the mosaic and single-dish images, which is readily done with `restor` using `mode=residual`.

In general, to combine single dish and interferometer data, you need to image the complete region containing emission. In practise this means that sidelobes, from emission outside the region of interest, are not significant. This applies both to the single-dish and interferometric observations. Both observations should map the complete region of the emission.

Some Theory and Preparing the Single-Dish Data

Both the linear and non-linear methods require a mosaic and a single-dish image as inputs. *MIRIAD* has not tasks to form images from single-dish data. Thus you must use another package to generate the single-dish image, which you will then import into *MIRIAD* (presumably via FITS – see Section 8.2). Further massaging might then be needed within *MIRIAD* to prepare the single-dish image for the combination process. The steps needed are as follows:

1. Coordinate grids: The coordinate systems of the single-dish and mosaic image *must be identical*. That is the image size, pixel increment, equinox, projection geometry, etc, must be the same. Often, the `regrid` task can achieve this reasonably painlessly. Given an input image and a template image, `regrid` resamples (by interpolation) the input onto the grid of the template. In doing so, it correctly handles different projection geometries, equatorial/galactic coordinate conversion, equinox conversion and different velocity definitions.

   Note you should not regrid the mosaiced image. In regridding, the information that describes the mosaicing process is lost, which will make life difficult for you.

   Typical inputs to `regrid` would be

REGRID	
in=lmc.sd	Input single dish image.
tin=lmc.mosaic	The mosaic giving the coordinate system that we want to regrid to.
out=lmc.sd_regrid	The output, regridded, single-dish image.

For interpolation to produce a faithful output, the grid of the input cannot be too coarse – the number of pixels across the single-dish beam should be at least 3. Although this is usually readily satisfied on the spatial axes, spectral line users should be more wary: the number of channels across the spectral axis response function is usually very small. In addition, whereas the spatial resolution of the single dish will be poorer than the mosaic, this is not necessarily true for the spectral axis. So, in addition to regridding, some spectral smearing might need to be performed to reduce the spectral resolution of the single-dish image to that of the mosaic. Unfortunately, $\mathcal{MIRIAD}$ is poorly equipped to solve these problems; these issues are left as a difficult exercise for the user.

2. Flux scale calibration: The single-dish and mosaic images both should have flux units of Jy/beam. Although appropriate calibration during the observations is obviously the best method to ensuring that the flux scales of the mosaic and single-dish images are the same, this is not always achieved. The flux scales of two images, that are nominally in the same units, can differ by a modest, but appreciable, amount.

   A common approach to estimating flux calibration factor is to compare the single-dish and mosaic images in that annulus of spatial frequencies where both are sensitive. For example, a Parkes observation will sample spatial frequencies from 0 to near 64 meters, and a ATCA mosaic (assuming the shortest ATCA spacing is observed) will be sensitive from about 20 meters upwards. As the reliability of the data near the extremes is suspect, spatial frequencies from 25 to 40 meters should be modestly reliably represented in both an ATCA mosaic and a Parkes image. Comparing in the overlap annulus works very well when there is a simple source which is well represented in this region (e.g. a dominant point source).

   The tasks to perform the linear and non-linear combination both have parameters for setting *and* deducing the flux calibration factor.

3. Point-spread function: In principle, knowing the point-spread function (the "beam") of the single-dish image is just as important as it is for the mosaic. It is best to 'map' this at the time of the single-dish observations and to account for any change in the beam that is caused by the single-dish imaging process.

   While it is possible to imagine algorithms that could deduce the beamwidth based on the spatial frequency overlap between the single-dish and mosaic data, in practise this is not possible (the overlap region is not wide nor is the data reliable enough). In estimating beamwidth parameters, the beamwidth and flux calibration factor are highly coupled; an error in the beamwidth has much the same effect in the overlap region as an error in the flux scale. Thus you cannot determine the beamwidth and flux scale simultaneously.

   The linear combination method assumes the single-dish point-spread function is a gaussian form, whereas the non-linear method takes a image dataset as the point-spread function (the point-spread function need not be symmetric). If a point-spread function is not readily available for the non-linear method, a gaussian dataset can be produced.

The Linear Method – IMMERGE

The linear method, sometimes known as feathering, can be thought of in terms of operations in the Fourier (spatial frequency) domain of the images. If the single-dish image is a good representation of the object at low spatial frequencies, and if the mosaic image is a good representation at mid to high spatial frequencies, then the two sorts of data can be merged in the Fourier plane to form an image accurate up to the resolution of the mosaic image. This is exactly what `immerge` does.

The way that `immerge` normally combines the data is to give unit weight to the single-dish data at all spatial frequencies, and to taper the low spatial frequencies of the mosaic data. This tapering is such that the sum of single dish and tapered mosaic data produces a gaussian beam equal to the mosaic gaussian

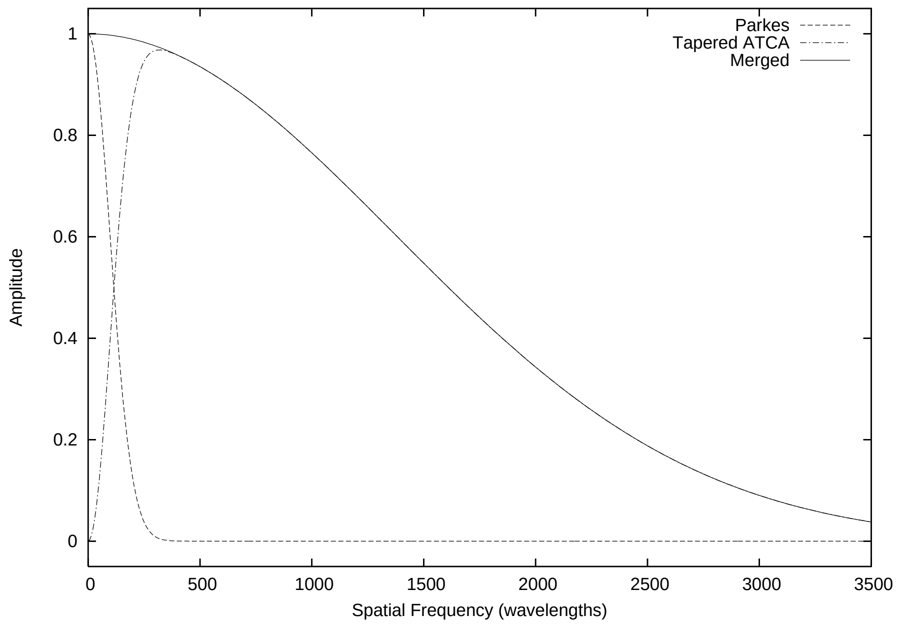

Figure 21.1: Spatial frequencies sampled by Parkes and the ATCA

beam. Figure 21.1 shows this process for a point source. The Parkes data plus the tapered ATCA data neatly add to give an overall response of a gaussian form.

The inputs to `immerge` are moderately straightforward: the `in` parameter gives the name of the input mosaic and single-dish image (in that order), whereas the `out` parameter gives the name of the output dataset. Because `immerge` assumes that the beams of both the single-dish and mosaic images are gaussian forms, you will want to use images which approximate this. Whereas single-dish images usually approximate this reasonably, a dirty mosaic does not. Instead, *you will want to use a deconvolved/restored mosaic* in the `immerge` process.

Task `immerge` allows you to set a parameter giving the flux calibration factor, `factor`: `immerge` multiplies the single-dish image by this factor before further processing. If you believe your calibration, you should set this parameter to 1 (or whatever number you believe the calibration factor to be). Note that if the parameter is unset, `immerge` attempts to deduce this factor (i.e. the default is not simply to use a value of 1). It does this by comparing data in the overlap annulus in the spatial frequency coverage of the mosaic and single-dish images. In doing this `immerge` first convolves the mosaic image to the same resolution as the single dish data (including scaling fiddles to account for the inputs being in Jy/beam in differing resolutions), and it then performs a robust (L1) fit between the two sets of pixel data. If you use `immerge` to determine the flux calibration factor, you will want to consider setting the following parameters:

- `uvrange` giving the range in the Fourier domain where there is overlap. This parameter can take two or three values, the first two being the numbers giving the inner and outer radius of the overlap region, and the third parameter giving the units of the first two values. The most useful units are `meters` and `klambda` (kilowavelengths – which is the default). For example, typically the overlap between a Parkes and ATCA observation (assuming the shortest ATCA spacing is included in the observation) would be

  ```
  uvrange=25,40,meters
  ```

  The default is to use all spatial frequencies.
- `region` parameter allows you to select those planes used in determining the flux calibration factor. Unlike the normal behaviour of `region` parameter, *only plane selection* is allowed, with the `image`

sub-command being the only really useful command. Generally you would choose planes with appreciable structure which is well represented in both single-dish and mosaic data. The default is to include all planes.

- `device` sets the PGPLOT plotting device where plots are make of a comparison between the data in the overlap region. Two plots are produced. For the first plot, the axes gives the value of a pixel (both real or imaginary values are included) in the mosaic (x axis) and single-dish data (y axis). These pixel values are after tapering, scale factor fiddles, and after applying the calibration scale factor that has just be deduced. The data should follow a "$y = x$" form (which is also plotted in red). The second plot gives the residual difference between the mosaic and single-dish data (i.e. the difference from $y = x$) as a function of spatial frequency. A trend in these residuals might suggest that the single-dish beam parameters are grossly wrong. Although the default is not to create a plot, there seems little reason why you would not want to inspect it.

The gaussian parameters of the mosaic and single-dish beams are important parameters in the processing that `immerge` performs. Task `immerge` tries to determine these by first looking for the relevant gaussian beam parameters within the image datasets. If it cannot find these, it will attempt to estimate the beams, based on its knowledge of the telescope and the observing frequency. These estimates, however, are far from perfect – *it is far better to ensure that the beam parameters are within the dataset before using* `immerge`. If you are unsure whether the beam parameters have been set for a dataset, `prthd` includes these (if found) in amongst the information it prints out. For the mosaic, normally `restor` will have saved the beam parameters in the dataset. However it is likely that you will need to add the single-dish beam parameters to the single-dish dataset. The items you want to set are `bmaj`, `bmin` and possible `bpa` (beam major, minor axis size and position angle respectively), which are set using `puthd`.

PUTHD	
in=lmc.sd/bmaj	Set beam major axis.
value=18.8,arcmin	Width is 18.8 arcminutes.

PUTHD	
in=lmc.sd/bmin	Set beam minor axis.
value=18.8,arcmin	Width is 18.8 arcminutes.

When emission continues to the edge of the image (or near the edge), algorithms such as that used by `immerge` can suffer edge artifacts. If these are problematic, `immerge` has a some parameters which you might like to try to reduce the problem. Task `immerge` pads images with a guard band before doing a Fourier transform. Edge effects may be lessened by increasing this from the default value by setting the `guard` parameter. Task `immerge` also tries to avoid edge discontinuities by padding the guard band with a mirror image of some of the data. If the image is mostly zero, then zero padding (rather than than a reflection) might be better. In this case use `options=zero`.

Typical inputs for `immerge` are

IMMERGE	
in=lmc.mosaic,lmc.sd	Give the mosaic and single-dish inputs
out=lmc.combo	Give output combination dataset.
factor=1	Set to the correct flux scales, or
factor	leave blank and
uvrange=25,40,meters	specify the spatial frequency overlap annulus,
region=image(10,20)	and use planes 10 to 20 inclusive,
device=/xs	and let it plot the comparison.

The Non-Linear Method – MOSMEM

The non-linear path to combination is conceptually quite different to the linear one, even though its success ultimately rests on the same data. The non-linear combination uses the task `mosmem` (and the maximum entropy principle) to perform a joint deconvolution of the mosaic and single-dish images. It finds the model which maximises the entropy, subject to two separate χ^2 constraints (one each for the mosaic and single-dish images). Usage of `mosmem` is similar to that in Section 21.6, expect that there are now two parameter values for the `map`, `beam`, `rmsfac` and `q` keywords, with these corresponding to the mosaic and single-dish values (in that order).

Unlike `immerge`, the non-linear approach works with dirty images.

Note that you have to give a single-dish `beam` dataset. Unlike all the other deconvolution tasks, the image size of this dataset can be much smaller than the image being deconvolved (`mosmem` will zero pad to make it the size that it requires), and it can be asymmetric (normally beams are assumed to have "even" symmetry). If you do not have a beam dataset, then probably the best thing you can do (apart from actually going out and determining it through an observation) is to generate a gaussian image, with the appropriate resolution, with `imgen`. Task `imgen` takes a template image as its input, multiplies the template by a scale factor, and then adds an "object". A convenient approach would be to use the single-dish image as the template (or a single plane of this in spectral line experiments), multiply this by 0 and then add a gaussian. Typical inputs would be:

IMGEN	
in=lmc.sd	Give the single-dish image as template.
factor=0	Multiply template by 0.
object=gaussian	Add a gaussian.
spar=1,0,0,1128,1128,0	Gaussian is 18.8 × 18.8 arcmin (1128 arcsec) at image centre and with amplitude 1.
out=lmc.sd_beam	Output is the beam dataset.

Task `mosmem` needs to know the rms noise in the two input images. It determines this by multiplying the theoretical rms noise (the theoretical noise is required to be present in the dataset) with the `rmsfac` input parameter. While `invert` will have saved the required noise information in the mosaic dataset, you will most likely have to add some estimate of the theoretical noise level to the single-dish dataset. You could estimate this by using `cgcurs` (using `options=stat`) to determine the rms level in some blank part of the sky (see Section 17.3). Task `puthd` can then be used to set the appropriate item (named, surprisingly, `rms`) in the single-dish image. For example:

PUTHD	
in=lmc.sd/rms	Set the rms item in the single-dish dataset.
value=5.0	Set the theoretical noise as 5.0 Jy/beam.

As you have probably set the `rms` item in the dataset to the noise level that you believe is true, the `rmsfac` parameter for the single-dish image that you input to `mosmem` can usually be left unset (it will default to 1).

As with `immerge`, parameter `factor` allows you to set the flux calibration factor (the factor by which the single-dish is scaled) as one of the inputs to `mosmem`. However, unlike `immerge`, if you do not set the parameter, `mosmem` assumes the flux calibration factor is 1 – that is, `mosmem` does not determine this parameter by default. However `mosmem` still has the ability to solve for the calibration factor – use `options=dofactor` to turn this on. In this case, the `factor` parameter is used as an initial estimate (it should be a reasonably good estimate). Unlike `immerge`, you do not need to tell `mosmem` what the overlap annulus is – this information is implicit in the beam datasets.

It might be instructive to compare the factor estimated by `mosmem` and `immerge` – they can differ significantly, and yet both seem to be consistent with the data.

If you use `options=dofactor` with a spectral cube, `mosmem` will work out a separate flux calibration factor for each plane. This is not what you probably want (particularly for planes that are predominantly noise!). In this case, its better to deduce the calibration factor beforehand, and not to use `options=dofactor` in `mosmem`. Some approaches to deduce the calibration factor beforehand include:

- use `immerge` (but recall that `immerge` requires a deconvolved mosaic), or

- use `mosmem` on an integrated intensity (zeroth-order moment) image or on a plane with substantial emission.

Note that `options=doflux` cannot be used when doing a joint mosaic/single-dish deconvolution – an integrated flux constraint is implicit in the single-dish data.

Typical inputs to `mosmem` would be

MOSMEM	
map=lmc.mosaic,lmc.sd	Inputs are the dirty mosaic and single-dish datasets.
beam=lmc.beam,lmc.sd_beam	Mosaic and single-dish beams.
niters=30	Maximum number of iterations.
region=	Region to deconvolve.
rmsfac=1,1	RMS noise fudge factors.
out=lmc.model	The output model.
factor=	Set this to the calibration factor, or it defaults to 1.
options options=dofactor	Leave unset to fix the calibration factor, or get MOSMEM to solve for this.

Having performed a joint deconvolved, you will want to use `restor` with the model produced by `mosmem` along with the mosaic dirty image and beam.

Some Alternative Non-Linear Methods

Here we describe two ad hoc methods of combining single-dish and mosaic images. To some extent, these are of historical interest only – the linear and non-linear methods mentioned before are to be preferred.

These alternative methods are not implemented as tasks within *MIRIAD*, so the steps to implement them are more hands-on. For example, if needed, you will want to correct the flux calibration factor manually before using either of these (e.g. `immerge` can do this for you). Some examples of using these approaches are given in the paper by Snezana Stanimirovic (ASP Conf. Series, vol. 278, p. 375).

- An alternative combination technique using `mosmem` is to use the single-dish image as the default image (`default` keyword) rather than as a separate χ^2 constraint. That is, you use just the mosaic image and beam with the `map` and `beam` parameters, and set the single-dish image as the `default` image. Strictly the default image should be in units of Jy/pixel. However `mosmem` is smart enough to divide the pixel values by the beam volume (where necessary) to perform some sort of crude conversion from Jy/beam to Jy/pixel. You can constrain the total flux of the resultant model to agree with the single-dish total flux, by using `options=doflux`. Task `mosmem` is smart enough to use the default image as an flux estimate when the `doflux` option is used and when the `flux` is left unset.

- You could form an image which is a simple weighted average of the mosaic and single-dish dirty images (i.e. before any deconvolution), and then pass this to `mosmem` as if it was a normal mosaic. In doing this, you are trying to fool `mosmem` to some extent, so there are a number of fudges that you should do to make the data better approximate what `mosmem` expects.

  Before adding the mosaic and single-dish images, you should apply the residual primary beam attenuation present in the mosaic to the single-dish data. Task `mossen` will generate an image of the residual primary beam attenuation (set the `gain` parameter). Having generated a the residual primary beam attenuation, you can then apply this to the single-dish image using `maths`.

  The weights used to add the mosaic and single-dish images should sum to 1, but the actual way of determining the relative weights is not well defined. One good choice, advocated by Stanimirovic et al., is to weight the images in inverse proportional to the beam volumes. When the difference in resolution between the mosaic and single-dish observations is appreciable (i.e. the normal situation), this weighting reduces to nearly the same as that performed by `immerge`. Normalisation the weights to add to 1 gives

$$
\begin{aligned}
w_{\rm mos} &= \frac{\Omega_{\rm SD}}{\Omega_{\rm mos}+\Omega_{\rm SD}} \\
w_{\rm SD} &= \frac{\Omega_{\rm mos}}{\Omega_{\rm mos}+\Omega_{\rm SD}}
\end{aligned}
$$

(where $\Omega_{\rm mos}$ and $\Omega_{\rm SD}$ are the beam volumes of the synthesised mosaic beam and the single-dish beam respectively).

To generate the effective beam dataset, you will also want to add a component to the mosaic beam. Because of the process that `mosmem` believes was used to generate a mosaic, the component you add is not simply the single-dish beam – the mosaicing process narrows down the width of the beams from that of individual pointings. If both the interferometer primary beam and the single-dish beam are gaussian forms with widths $\theta_{\rm int}$ and $\theta_{\rm SD}$ respectively, then you will want to add a gaussian to the mosaic beam dataset which has a width

$$\theta = \sqrt{\frac{2\theta_{\rm int}^2\theta_{\rm SD}^2}{2\theta_{\rm int}^2-\theta_{\rm SD}^2}}.$$

This gaussian will be somewhat broader than the single-dish resolution (for an ATCA and Parkes combination, it will be about 7% broader than the Parkes beam). You will want to add this gaussian and the mosaic beam with the same weights that were used for the single-dish and mosaic images. Note that the mosaic beam dataset consists of one image per pointing, and that you will want to add a gaussian to each plane. Task `imgen` can be used to do this, although it warns you that its handling of cubes is crude! With `imgen`, you can set the weighting by appropriately setting the scaling factor (`factor` parameter) and the amplitude of the gaussian (`spar` parameter).

Note that this technique ignores some edge effects. Also it uses the theoretical noise level derived for the mosaic observation. This will be close to what you want if the noise levels for all pointings are about the same.

Chapter 22

Reducing 12-mm ATCA observations

22.1 Loading data into *MIRIAD*: ATLOD

ATCA millimetre observations are loaded into *MIRIAD* in the same fashion as centimetre data: task `atlod` is used.

At 12mm, the opacity of the atmosphere can be significant: in mediocre conditions, the signal loss is can be a 20% at low elevations. With 12-mm data, an opacity correction should be applied to the data during the `atlod` step. To apply this correction, `atlod` uses a model of the atmosphere (using meteorological data contained within the dataset) to compute a model opacity. Note that this is just that – a model – and so has its limitation. To apply atmospheric opacity correction, use `options=opcorr`.

ATLOD	
options=opcorr,birdie,reweight,xycorr options=opcorr,hanning,compress	Options for typical 12-mm continuum observation, or possible options for spectral line observations.

Please note the following:

- ATCA observations made before October 2003 failed to contain the meteorological data, and so `options=opcorr` cannot be used.
- At 12mm, no self-generated RFI at multiples of 128 MHz is seen. Task `atlod`'s `birdie` option for continuum data is used to discard edge channels and every second channel (as noted earlier, successive channels are not independent, and so there is no sensitivity penalty in discarding every second channel).

22.2 Calibration

For 12-mm data, calibration is essentially identical to the centimetre bands. Primary flux calibration can be done using `gpboot` and observations of 1934-638, in much the same fashion as at the lower frequencies. 1934-638 is about 1 Jy at 12-mm wavelengths: the *MIRIAD* calibration software contains a built-in model of 1934-638's change of flux density with frequency. For information about this flux scale, see the memo by Bob Sault "ATCA flux density scale at 12mm" (2003).

As 1934-638 is comparatively week at 12-mm wavelength, it is not an good bandpass calibrator. If you require a bandpass calibration, you should take observations of a strong source such as 1921-293 or 1253-055.

Because atmospheric opacity affects the amplitude gain calibration step, it is important to consider it when high precision flux calibration is required. In the flux bootstrapping step, `gpboot` determines the boot factor to be applied to the gains of the secondary calibrator to make them the same as the primary. As the opacity correction done by `atlod` is only correct to first order, ideally the comparison between primary and secondary gains would be made *at the same elevation and under the same weather conditions*. Selecting the range of times that the gains should be compared is a more important issue at 12mm than at lower frequencies. To select the time range of the secondary to compare, use `time` selection in the `select` keyword.

Some judgement will need to be made about the set of amplitude gains of the secondary to use in the comparison. In stable weather conditions, it is probably best to select the time range of the secondary where its elevation corresponds most closely to the elevation during the primary's observation. When the weather changes significantly during the observation, it is probably best to use a time range of the secondary which is close to the primary's observation. Tasks `uvplt` and `varplt` can plot source elevation as a function of time.

Typical inputs to `gpboot` are:

GPBOOT	
vis=vela.uv	Input visibility dataset.
select=time(1:00,2:00)	Select a time range in the "vela.uv" dataset where the amplitude gains should be comparable to those in the primary calibrator.
cal=1934-638.18001	Primary flux calibrator.

Chapter 23

Reducing 3-mm ATCA observations

By many measures, the ATCA 3-mm observing band is a completely different domain to the other ATCA bands. The science is different, the weather is much more important, and the mode of observing is quite different. Not surprisingly many aspects of the data reduction are also different. This chapter reviews the theory behind some of the differences, and then considers a practical approach to calibration.

23.1 Some theory

23.1.1 Atmospheric opacity and system temperature

To provide correct flux density units for the visibilities, V, from an interferometer, the measured correlation coefficient at the correlator (ρ) needs to be scaled by a system temperature $T_{\rm sys}$ and a telescope sensitivity factor (K). The telescope sensitivity factor is determined by the overall antenna size and system efficiency.

$$V = KT_{\rm sys}\rho$$

At millimetre wavelengths, the atmosphere can no longer be approximated as perfectly transparent. It degrades overall sensitivity in two ways: the atmosphere emits radiation, and so raises the system temperature, and the atmosphere attenuates the astronomical signal. For a zenith opacity τ, observing at an elevation of e, and if the atmosphere is approximated as having a uniform temperature T_0, then the atmospheric contribution to system temperature is

$$T_{\rm sky} = T_0\,(1 - \exp(-\tau/\sin e))$$

and the atmospheric transmissivity is $\exp(-\tau/\sin e)$.

In converting correlation coefficients to visibility measurements, we can notionally include the effect of the atmospheric attenuation either in the effective system efficiency factor, or in an effective system temperature. The approach used will depend on the calibration scheme implemented at the telescope.

For the ATCA at 3-mm wavelength, it is most natural to include the effect of the opacity in an effective system temperature – the so-called "above atmosphere" system temperature. The above atmosphere system temperature is simply the physical system temperature divided by the atmospheric transmissivity. The widespread use of above atmosphere system temperatures in millimetre astronomy follows on because the above atmosphere system temperature can be measured directly! It can be deduced by periodically flipping an absorber at atmospheric temperature in front of the feed system. The technique to do this, the so-called "chopper wheel" method (e.g. Ulich 1980). The ATCA uses the chopper wheel method (also call paddle or vane calibration) in its 3-mm system to determine an above atmosphere system temperature.

This measurement will only be done occasionally. That is, unlike the centimetre and 12-mm bands, continuous measurement of system temperature is not available at 3-mm wavelength. To partly remedy this, *MIRIAD*'s `atfix` includes an algorithm to "interpolate" the above-atmosphere system temperature measurements based on changes in telescope elevation and weather.

23.1.2 Antenna gain considerations and flux calibration

One characteristic of the ATCA dishes is that they are big by 3-mm standards. Indeed van Hoerner (1967) has considered the limits to the size of an antenna before various effects become significant. For construction styles like the ATCA dishes, he suggests that 11m is the maximum dish size for 3-mm observations before thermal effects become significant. As the ATCA dishes are twice this, it is not surprise that the ATCA antenna gain and primary beam response varies as a result of thermal distortions. Changes in temperature, the position of the sun, cloudiness and shadows will all affect the gain of the ATCA antennas. Measurements to date tend to bear this out, with changes in antenna gain of 25% noted that appear to be related to thermal effects. Distortion of the dishes by gravity is also significant (eg Subrahmanyan 2005). A gain/elevation correction will solve this - to first order at least. Because of the way the ATCA panels are set, the antenna gain peaks at about 60° elevation.

Note that gravitational and thermal distortions coupled - a perfect correction for these effects is never possible.

To first order, the gain change will be calibrated out provided the secondary calibrator is very close to the target source. Applying the standard gain/elevation curve to the data as the first step in the data reduction will also help. However these two steps will not account for the change and the beamshape. This might be an important effect for widefield imaging and mosaicing.

The best way to avoid thermally-induced gain variations is by observing in the pre-dawn hours or on a calm cloudy day! The best way to avoid large elevation changes in a synthesis is to observe in a hybrid array – hybrid arrays allow a synthesis to be performed without the need to track the source from horizon to horizon.

The gain variations must be considered when bootstrapping the flux density scale. The best approach to ensuring a good flux scale is to ensure that the secondary and the flux density calibrator are observed nearly simultaneously and at the same elevation. Although this may at first glance sound difficult if not impossible, generally it is straightforward. Generally there should be a time when the secondary and the flux density calibrator are at the same elevation. It may be that one is rising while the other is setting, and so they are at significantly different parts of the sky (ie significantly different azimuths). Unless the sky is cloudy, being at different azimuths is not important.

23.1.3 Instrumental phase

At the precision needed for 3-mm observing, the ATCA antennas are not identical in terms of their physical construction. In particular the offset between the elevation and azimuth axis will differ between antennas. VLBI astronomers, who have used heterogeneous arrays of antennas, have coped with this issue for years. For the ATCA at 3mm, what is presumed to be a difference in this offset introduces a instrumental phase error, that needs to be corrected.

23.1.4 Atmospheric phase and phase calibration considerations

Phase calibration at 3-mm wavelength is significantly more difficult than at the longer wavelengths. This is because the atmospheric phase errors are comparatively larger, the secondary calibrator grid is sparser and the system sensitivity is poorer. To use a tolerably nearby calibrator, you will need to use a calibrator that is weaker than you might wish - particularly if you wish to observe in a narrow bandwidth. Alternatively you might use a nearby strong SiO maser rather than continuum point source phase calibrator.

These lead to a different approach to calibration compared to that at longer wavelengths. At 3-mm wavelengths, the dual frequency band capability of the ATCA can be used to good effect in the calibration process. When sensitivity to calibrators is an issue, one of the two bands can be specifically set aside for calibration purposes. The correlator configuration can be selected so that one of the bands is either 128 MHz wide (and so as sensitive as possible), or suitable for observing an SiO maser. This calibration band can also be used for reference pointing during the course of the observation.

When using one of the bands for calibration purposes, the two bands need to be jointly calibrated.

In a typical 3-mm observations, in addition to the target source, your will observe a bandpass calibration, a secondary calibrator and a flux density calibrator.

SiO masers should be used with caution: they are often strongly polarised, and they may be structurally complex. Note, also, that the ATCA limits the dual observing bands to be set within 2.7 GHz of each other. If you use an SiO source as a calibrator, the part of the 3-mm band that is accessible to the observation is limited.

23.1.5 Flux density calibrators

At wavelengths shorter than a few centimetres, extra-galactic sources generally prove to be too variable to be useful as flux density calibrators. So at these wavelengths, the blackbody emission from the planets are often used as flux standards. The two most commonly used planets for the ATCA at 3mm are Mars and Uranus. Note Jupiter is too large for use as an ATCA flux density calibrator. Neptune would also be a possible flux density calibrator. However Neptune is quite close to and half the strength of Uranus, and so does not offer anything over Uranus.

Uranus is the preferred flux density calibrator for the ATCA. Mars is less suitable for a number of reasons: some models of Mars' average brightness temperature suggest variations in the temperature of up to 10% as a result of Mars' diurnal rotation, its distance from the Sun and the viewing geometry of Earth-based observers. Definitive measurements and confirmation of these models are poorly studied in the literature. Depending on its distance and the array configuration, structure on Mars is also readily detected with the ATCA: Mars has hot equatorial and cold polar regions.

Although Uranus or Mars (preferably Uranus) are the best flux density calibrators, frequently they will not be accessible to an observation. To account for this, the Narrabri Observatory regularly monitors some additional sources (usually 1253-055 and 1921-293) to allow them to be used as flux density calibrators. Be warned though that these sources can vary substantially on month timescales. If Uranus and Mars are not accessible to your observation, you should enquire about the next best source to be used for flux density bootstrapping.

23.2 Practical loading and calibration

When observing at two bands with the ATCA, it is generally sensible to set the two bands so that they do not overlap. In terms of sensitivity, there is nothing to be gained by setting the bands to the same frequency as the noise will be common to the the bands. However having the bands overlap complicates the data reduction process in $\mathcal{MIRIAD}$. The best approach is to avoid the problem, by setting the bands so that they do not overlap at all, or overlap by a comparatively small amount.

23.2.1 Loading data into $\mathcal{MIRIAD}$: ATLOD

ATCA millimetre observations are loaded into $\mathcal{MIRIAD}$ in much the same fashion as centimetre data: task `atlod` is used.

One difference is that it is *generally desirable* to load both observing bands together, to allow joint calibration. A shortcoming of $\mathcal{MIRIAD}$ is that it is poor at handling multiple observing bands *when the polarisation products differ between the bands*. However the standard ATCA correlator configurations that combine narrowband and wideband observations generally do measure different polarisation products in the two bands. As polarimetry with the ATCA 3-mm system is not possible anyway, the simplest approach to ensuring the polarisation products of the two bands are the same is by discarding polarisation products that are not needed. The `atlod` option `nopol` causes `atlod` to discard any XY and YX products.

Typical inputs to `atlod` are:

ATLOD	
in=2006-02-18_1234.C123	Input RPFITS file.
out=vela.uv	Output *MIRIAD* dataset.
options=nopol,birdie,reweight	Typical options used.

The `birdie` option for continuum data is used to discard edge channels and every second channel: for the continuum ATCA correlator configuration, successive channels are not independent and so there is no sensitivity penalty in discarding every second channel.

23.2.2 Initial "fixes" to millimetre data

The next step after loading is to apply a number of "fixes" that are important for 3-mm data. Note that there is no record in the data that these steps have been performed. In particular the software will allow you to apply the fixes twice, which would be just as bad as not applying them at all. The task to do these, `atfix`, performs the following functions:

- By default `atfix` applies the "standard" gain/elevation model for the antennas. You can disable this by using `options=nogainel`.
- By default `atfix` corrects for a "standard" instrumental phase model. You can disable this with `options=noinst`.
- As mentioned above, the ATCA uses the chopper wheel method to measure an above atmosphere system temperature. The nature of the hardware means the measurement is done only typically every 5 to 30 minutes. When observing, the ATCA on-line system can either apply a nominal system temperature value to the visibility data, or it can apply the actual measured system temperature values. In applying these measured values, on-line system will apply the most recent measurement.

  Task `atfix` includes an algorithm to interpolate measured system temperature values, between measurements, and to correct the visibility data accordingly. In doing this it correctly handles whether a nominal or the measured system temperature value applied to the data on-line. The interpolation algorithm takes into account changes in elevation and meteorological conditions.

  The parameter `tsyscal` can be used to change this default behaviour. This parameter can take on values to revert the data back to the nominal system temperature value, to make no changes to the system temperature scaling, to use the most recent system temperature measurement, or to use an extrapolation (as distinct from interpolation) algorithm in estimating system temperatures.
- Often you will want to apply a correction to your data to better account for the exact locations of the antennas.

  On the night of an ATCA reconfiguration, a solution for the antenna positions is derived. The practise is to install this solution as the "standard solution". The on-line system will then use this solution in subsequent observations in that array configuration. The quality of this solution is important for the ultimate phase calibration of 3mm data.

  Experience has shown that often the position solution derived at the time of the reconfiguration is not necessarily the best possible. For example the solution will depend critically on the atmospheric phase stability at the time of the solution, and this might be less than ideal on the night of the reconfiguration. Consequently improved solutions will often be derived at a later time. If improved solutions become available, then 3-mm observers should correct their data accordingly.

  Updated solutions are stored in parameter files in the *MIRIAD* system directory `$MIRCAT`. These solutions are distributed via the *MIRIAD* update mechanism, and have file names of the form "`dantpos.`*yymmdd*". Here where "*yymmdd*" is the date of the relevant array reconfiguration.

  Task `atfix` corrects the data for an improved antenna location solution via the `dantpos` parameter. If an improved solution file is present, you can instruct `atfix` to read this directly using the indirect parameter input mechanism (see Section 2.5). For example, to use the antenna location solution appropriate for a hypothetical array reconfiguration that happened on 16 October 2002, use

  ```
  dantpos=@$MIRCAT/dantpos.021016
  ```

- One of the flaws of the current ATCA RPFITS files is that antenna locations are not saved when antennas are not in use. While this might not seem a serious flaw (who cares about the antennas if they are not in use?), the unused antennas can still cause shadowing – true antenna locations are needed for shadowing calculation. This was a particular issue with the interim 3-antenna 3-mm system when observing in compact arrays.

  By giving a value to the `array` parameter, `atfix` will fill in any antenna locations that are missing in the input visibility file. *NOTE*: This just fills in missing antenna locations, it does not perform any flagging of shadowed data.

  The value given to this parameter is either an array configuration name (e.g. EW352 or 750A) or a list of six station names ("W106" and "N3" are examples of station names). When giving the station names, these must be in the order of the antennas (i.e. CA01, CA02, CA03 etc), regardless of any possible array shuffles.

  NOTE: When antennas are in a shuffled order, or for arrays using the north spur, you should generally give the list of station names, as the standard array configuration names assume an antenna orderwhich is probably not correct.

  If in doubt the on-line configuration history should be consulted.

Typical inputs to `atfix` are:

ATFIX	
in=vela.uv	Input visibility dataset.
out=vela.fixed.uv	Output corrected visibility dataset.
dantpos=@$MIRCAT/dantpos.021016	Antenna location correction.

23.2.3 Additional monitoring variables

In addition to the plethora of uv variables that can be examined by an observer, the following are particularly relevant for 3-mm data.

smonrms	ATCA seeing monitor rms (in microns)
refpnt	Reference pointing solution (in arcseconds).
axisrms	RMS tracking accuracy
axismax	Maximum tracking accuracy

These variables can be plotted or listed with `varplt`, e.g.

VARPLT	
in=vela.uv	Input visibility dataset.
device=/xs	PGPLOT plotting device.
yaxis=axisrms	Plot rms tracking error.

Reference pointing solutions

The variable `refpnt` gives the offset from the global solution, in azimuth and elevation, of a reference pointing solution.

Seeing monitor measurement

The seeing monitor is an independent interferometer which constantly measures the atmospheric phase stability to a geostationary satellite. The baseline length is 220m, and the satellite elevation is 60°. The measurement is expressed in microns of excess path.

Tracking accuracy

With the ATCA beam at 3mm being comparatively small (35″), it is possible for tracking errors to be significant. This may occur during windy weather (gusts can blow the antennas off the directed tracking position). In additional poor tuning of the antenna drive systems can sometimes cause poor tracking, particiularly soon after a change in source.

Note that tracking errors should be distinguished from pointing errors. Tracking errors are the inability of the antenna drive system to follow the requested path. Pointing errors correspond to the difference between the position that the astronomer requests and the actual position. Tracking errors are caused by flaws in the drive servo system, whereas pointing errors are a sum of tracking errors and errors in the antenna pointing model.

Since October 2003, tracking errors have been saved in the *MIRIAD* dataset. The uv variables corresponding to the rms and maximum tracking error in a cycle are `axisrms` and `axismax`.

Both the maximum and rms give two tracking error values for each antenna, nominally corresponding to the tracking error in the azimuth and elevation axes. However the ATCA on-line system produces a single composite value, which is replicated for the two axes in the dataset.

When the tracking error is less than 2′, the ATCA on-line system believes that this error sufficiently small to consider that the antenna is tracking. However this tolerance is not generally good enough for 3-mm observations. The task `uvflag` can be used to flag based upon rms tracking error. To do this, use the selection `pointing` subcommand.

UVFLAG	
in=vela.uv	Input visibility dataset.
flagval=flag	Flag selected points.
select=pointing(15,1000)	Flag when the rms tracking error is between 15″ and 1000″.

23.2.4 Bandpass and antenna gain calibration

When calibrating 3-mm data, it is possible to use a similar approach as with lower frequency data – see Chapter 11. In this case, because polarimetry is not currently possible with the 3-mm system, only task `mfcal` is needed to perform bandpass and antenna calibration. Flux density bootstrapping can be accomplished using `mfboot` described below.

However, the recommended approach at 3mm is different from this. As noted in Section 23.1.4, calibration at 3mm is more difficult because of weak calibrators, poorer system sensitivity and more challenging atmospheric conditions. This section outlines an alternative calibration approach exploiting the dual frequency bands of the ATCA. The steps are as follows:

1. The first step is not a step at all! Unlike the approach of Chapter 11, the approach described here *does not* split the data into separate single source, single frequency files before calibration. Rather the current approach assumes that you have a single *MIRIAD* dataset containing bandpass calibrator, secondary calibrator, flux density calibrator and target source. We focus on data having two simultaneous frequency bands here, but the approach works equally well for one or many simultaneous frequency bands within the dataset.

2. The first real step is akin to a bandpass calibration step. When dealing with two frequency bands, each with two polarisation products (XX and YY), it is generally appropriate to assume that the bandpasses of the bands and the phase offsets between bands and polarisations are constant with time.

   To determine the bandpasses and the phase offsets, you need to use `mfcal` on an observation of a strong continuum source. Typically this would be 1253-055, 1921-293 or 0537-441. Typical inputs are as follows:

MFCAL	
in=vela.fixed.uv	Input multi-source, dual-band dataset.
select=source(1921-293)	Select the strong continuum "bandpass" calibrator.

The important output of this step is a bandpass calibration table which will contain the relevant phase offsets. The output also contains antenna gain calibration tables, but this will be overwritten below.

In doing this step, you may well want to avoid using some edge channels in the bandpass calibration process. To achieve this, you can use the `edge` parameter of `mfcal`. The `edge` parameter sets the number of channels to drop for all bands equally. It does not allow you to set different number of channels to drop in the different bands. If the channels and bandwidth characteristics of the different bands differ significantly, the `edge` parameter probably does not give you sufficient flexibility. In this case, you could use `uvflag` to explicitly flag edge channels.

3. The next step is to determine the antenna gain calibration, again using `mfcal`. To do this, you will want to select those data for the secondary calibrator which will give the best overall sensitivity appropriate to your observation.

   - If the calibrator is a continuum source, you need only select this source and use both bands. Task `mfcal` will weight the two bands according to their relative bandwidth. Consequently if one of the bands is quite narrow relative to the other, then it will contribute little to the overall gain solution.
   - If the calibrator is an SiO maser, you will want to select only those channels that contain strong SiO signal.
   - With either a continuum calibrator or an SiO maser, it is possible to either determine antenna gain calibrations for the two polarisation products independently, or to determine a single joint solution for the two polarisations. The latter will give a $\sqrt{2}$ improvement in sensitivity in the solution process. The latter is also the desirable approach if the calibrator is strongly polarised (some SiO masers are more than 50% polarised). Doing a joint solution is generally a reasonable approximation given that the antenna gains are dominated by changes common to both polarisations: in amplitude, it is from the physical antenna gain change, and in phase it is from the atmosphere and instrumental contributions common to both polarisations.

   Note in this calibration scheme, it is implicitly assumed that the antenna gain changes are common to both frequency bands. This is a reasonable approximation given that the frequency separation between the bands is at most 2.7 GHz and potentially much less. Consequently the fractional frequency separation in at most a few percent, and so the calibration change between bands can be assumed to be minimal.

Typical inputs for `mfcal` follow. This assumes a continuum source as the secondary calibrator and does a joiubt solution for the two polarisations:

MFCAL	
vis=vela.fixed.uv	Input multi-source, dual-band dataset.
select=source(1622-297)	Select secondary calibrator.
options=nopassol	*Do not* solve for bandpass.
stokes=i	Do a joint solution for both polarisations.

To derive a separate solution for the two polarisation products, the `stokes` keyword would be left unset. Note the use of `options=nopassol`. This causes `mfcal` to not attempt to solve for the bandpass and phase offsets again, but rather to apply the previously determined bandpass and phase offset solutions.

Typical inputs when using an SiO maser would be as follows: Typical inputs to `mfcal` are as follows:

MFCAL	
vis=vela.fixed.uv	Input multi-source, dual-band dataset.
select=source(oceti)	Select SiO secondary calibrator.
options=nopassol	*Do not* solve for bandpass.
stokes=i	Do a joint solution for both polarisations.
line=chan,20,45	Select the range of channels where the SiO signal is strong.

23.2.5 Flux density bootstrapping

The best task to correct the flux scale of an observation is `mfboot` - `mfboot` performs an analagous function to `gpboot`. Task `mfboot` also supercedes the now obsolete task `plboot`. It works by scaling the antenna gain table so that the flux density scale of the observed data agrees with some model.

Task `mfboot` models a planet as a simple disk with constant brightness temperature. It has built-in ephemerides of the planets, and so knows their apparent size and orientation. It also has a model of the brightness temperature variation with frequency. Hence `mfboot` can generate a nominal visibility function of a planet. This model for Mars is not sophisticated, but should be accurate to better than 10% if appropriately used.

Task `mfboot` can also be used when bootstrapping from a point source. Although `mfboot` contains models for sources such as 1934-638, these models are not relevant for 3mm observers. If using a point source, generally the user would need to give `mfboot` the flux density of the source.

It is implicitly assumed in the bootstrapping approach described here that the flux density calibrator is observed near simultaneously and at the same elevation as the secondary calibrator. This is required because the antenna gain changes with both elevation and with thermal changes. The most reliable approach to tying the secondary and the flux density calibrators flux scales together is through observations with the same elevation and thermal environment.

Task `mfboot` will apply the bandpass and antenna gain solutions to the observed data before it does a comparison with the models. While it is not generally important that the flux density calibrator data is phase calibrated, it is important that it has the antenna amplitude calibration, derived from the secondary, is applied. This is critical in tying the flux density scales of secondary and flux density calibrator.

We will discuss the various input parameters for `mfboot`:

- `vis`: These give the input visibility dataset.
- `select` and `line`: This selects the data to be used as the flux density calibrator.
- `flux`: If using a point source as a flux density calibrator, use this parameter to set the flux density of the source (in Jansky). If using a planet, you may set this parameter (in Kelvin) to override the brightness temperature value that `mfboot` would use by default.
- `mode`: The comparison between the model and observed data can be done using either the scalar average (effectively the amplitude of the data), the vector average (effectively the real part of the data), or the triple product of the data. The amplitude and triple product are insensitive to phase errors, and so are normally the most appropriate. The default is to use the triple product for point sources and the amplitude for planets. For planets that are not worse than mildly resolved, you might consider using the triple product for planets.

  When using scalar mode with significantly resolved planets, where there is little flux on long spacings, you might consider using the `clip` parameter so as to exclude data that would simply contribute noise bias.
- `clip`: If Mars is used as a flux density calibrator, you will want to ignore those visibilities where the deviation of the visibility function from a simple disk model is significant. In practise, it is best to ignore the visibility data outside the main central lobe of the visibility function. The way to do this is by using the `clip` parameter. For Mars it is recommended that you set `clip=0.4`, which will ignore data where the visibility function model is less than 40% of the model zero spacing value.

- `device`: `mfboot` can produce a plot of the the data (after bootstrapping) and model visibilities to give you a quantitative feel for the quality of the fit.
- `options`: The only available option is `noapply`, which causes `mfboot` to perform all steps *except* applying the bootstrap factor.

In the general case, to use `mfboot` with the calibration scheme recommended above is straightforward enough: you simply run `mfboot` on the dataset, having selected the flux density calibrator. For example:

MFBOOT	
vis=vela.fixed.uv	Input multi-source, dual-band dataset.
select=source(uranus)	Select flux density calibrator.

23.2.6 Information on planets

Given the observing time and frequency, task `plplt` plots the expected visibility function for a planet. You will most likely want to check this visibility function before observing and discovering that the planet is resolved out! **NOTE** that `mfboot` and `plplt` do not include the effect of the ATCA's primary beam – the planet is implicitly (and silently) assumed to be small compared with the primary beam. This may not be the case at 3-mm. If there is a significant change in the visibility function between baseline lengths of 0 and 22 metres, then primary beam effects are important, and you probably do not want to use the planet (most likely it would be resolved out anyway).

Task `planets` can give you basic information about a planet, including rise and set times (but note these are quoted at the horizon, not the ATCA's elevation limit), the planet's angular size at a particular time, and its RA and DEC (note that the ephemeris used for RA and DEC is not sufficiently accurate to point or phase track the telescope).

Information on using the ATCA to observe a planet is available from

```
http://www.narrabri.atnf.csiro.au/observing/sched/solsys.html
```

23.3 Handling large spectral bandwidths

When using a single spectral window, and observing spectral lines, the bandwidth of the ATCA can be problematically narrow, or the channel widths can be too coarse. This is most likely to be an issue when observing spectral lines of extragalactic sources at high frequencies.

A partial work around for this situation is to use the two ATCA frequency bands to observe in adjacent somewhat overlapping windows, and using the same bandwidths and channel counts for the two windows. With the ATCA this is possible with some correlator configurations when observing with bandwidths of 16, 64 or 128 MHz. *MIRIAD* allows you to stitch these two spectral windows together in a reasonably straightforward fashion (although some quirks do become apparent). In so doing, it handles any overlap region between the two windows in a sensible fashion. The steps to achieve this are as follows:

1. Make sure you set a rest frequency in `atlod` (or during the subsequent processing). If you really want to view the spectrum in frequency (rather than velocity), then you might set a dummy rest frequency as the average of the centre frequencies of the two spectral windows.
2. During the flagging process, flag any channels in the overlap region between the two spectral windows which you *do not* wish to contribute to the final output stitched spectrum. These would be channels where the bandpass gain is quite low in one, but reasonable in another band. Task `uvflag`'s `edge` parameter is the easiest way to flag these channels.
3. Perform the calibration as described above, keeping both bands within the one file.

4. Subtract any continuum, using task `uvlin` as normal. Task `uvlin` handles multiple spectral windows within a single dataset.

   Task `uvlin` processes each frequency band separately. You will want to give some thought when setting the range of channels to use in the fitting of the continuum. If the spectral of interest in near the edge of the individual spectra, you will want to set the channels to use in `uvlin`'s fitting process carefully. You may wish to use a zeroth order fit, and just fit using channels at the edges away from the overlap region. Alternatively you may wish to run `uvlin` after the two spectra are stitched together instead. Some experimentation (and certainly thought) will be required.

   Note that when handling multiple bands, both *MIRIAD*'s `line` and `uvlin`'s `chans` parmaters number channels consecutively from 1 to the total number of simultaneous channels. The distinction between simultaneous bands is ignored in this numbering. But when fitting to the continuum, the boundaries between bands act as a break point.

5. It is nearly time to stitch the two spectral windows together. To do this you will need to use the "velocity linetype" One of *MIRIAD*'s quirks is that you cannot apply on-the-fly bandpass calibration and use a "velocity linetype" within the same task. You will need to make a copy of the dataset with the bandpass (and presumably other calibration) applied directly to the data. If you have used `uvlin`, you will have done this already. If not, create a copy of your dataset using `uvaver`.

6. With a dataset with calibration applied, you can now stitch the two spectral windows together. See Sections 5.4 and 16.8 for information on using velocity linetypes. Do the copying and linetype conversion with `uvaver` (or possible `uvlin` if you have not already used it) to generate a new stitched copy of the data. By using a velocity linetype, when creating an output spectrum, *MIRIAD* copies channels by their velocity. Any channels that overlap between the two spectral windows are averaged together.

Chapter 24

ATCA Bin Mode Observations

24.1 ATCA Correlator Bin Modes

The ATCA correlator has a mode whereby each correlator cycle (which is normally the same as an integration cycle) can be broken into a number of time bins. This capacity can be used in two different modes:

- Pulsar bin mode: This mode is useful for a periodic signal with a period much shorter than the correlator cycle, i.e. pulsars. In pulsar bin mode, the bins produced by the correlator correspond to the integration during a particular phase of the pulsar cycle.
- High time resolution bin mode: This mode is useful for observations which, for some reason, require higher time resolution than the normal correlator cycle time (the minimum correlator cycle time is about 10 s). In this mode, the correlator cycle is broken up into a number of finer time bins, effectively working around the minimum correlator cycle time limit.

24.2 High time resolution bin-mode observations

$\mathcal{MIRIAD}$'s `atlod` task has an option, `options=hires` which converts a RPFITS file observed in high time resolution bin-mode into an apparently normal dataset with fine time resolution. You should note the following caveats:

- Although the timestamp of each correlation reflects the observing time of that correlation, the $u-v-w$ coordinate is that for the midpoint of the correlator cycle (i.e. the $u-v-w$ coordinate is not recomputed at the high time resolution).
- Certain $\mathcal{MIRIAD}$ tasks that need to detect the end of an integration do this by checking whether consecutive timestamps differ by some tolerance. In high resolution time mode, the time difference between consecutive integrations might be smaller than this tolerance. There may be some resulting confusion.

24.3 Pulsar bin-mode observations

When observing pulsars with the ATCA, the correlator is capable of sampling synchronously with a pulsar. In this way, during a single integration, the correlator will produce a number of *bins*, each of which sample at a particular range in phase of the pulsar cycle. Much of the time, when reducing a bin-mode experiment, you will want to process all the bins without regard to the bin number. In this case, to a large degree, processing a bin-mode observation differs little from a normal observation. In particular:

- No special action is needed in loading the data with `atlod`.
- During the flagging process, tasks `tvflag` and `blflag` will simply average together all the pulsar bins before display. When you flag a point on the display as bad, all the bins which are used to generate that point are flagged as bad.
- Generally the calibrators in such an experiment will be observed in a binning mode. The calibration software combines the different bins together before determining their calibration solutions.

Obviously, however, there are times that you will want to distinguish between the bins. The `bin` visibility selection sub-command is the main mechanism for this. The general form is:

`select=bin(`*n1,n2*`)`

This selects bins *n1* to *n2* inclusive. If only a single bin is given, then only that bin is selected – numbering of the bins starts at 1.

The most common places where you will want to select a subset of bins is in the imaging and possibly self-calibration steps (the pulsar, during its 'on' phase, may make an excellent phase self-calibration source).

Although almost all visibility-data tasks support bin-mode observations, there are a few exceptions. These give warnings when they are used on bin-mode data. *These warnings should be heeded!*. The most notable task which fails to properly handle bin-mode experiments is `uvaver` when time averaging of the data is being performed. When used to time-average, `uvaver` will average all the bins together. Note, however, that `uvaver` works correctly (i.e. treating the bins as separate) when no time averaging is involved.

24.4 Tasks Specific to Pulsar Bin-Mode Observations

We now mention a few bin-mode specific tasks.

- Pulse profiles – PSRPLT: Task `psrplt` produces a pulse profile (amplitude against bin number). It can also produce a greyscale with axes of bin number and frequency and the greyscale intensity being pulsar amplitude.
- De-dispersion, rebinning and timing correction – PSRFIX: Task `psrfix` performs three related functions. De-dispersion consists of a channel-dependent shift of bins.

Appendix A

Setting Up Your Account

Setting up your account to use *MIRIAD* and the ATNF Visualisation Software varies from system to system. If in doubt, ask a local! We will assume you are using the `csh` shell.

If you are using the 'standard' login scripts on the ATNF machines (Marsfield, Narrabri, Parkes), you will be prompted whether you wish to use *MIRIAD* and the ATNF Visualisation Software when you first log in. If you fail to do this, then you can edit the file `.login.packages` in your login directory, and change `-miriad` to `+miriad`. After this change, *MIRIAD* will be set-up after your next login.

If you are not using the standard login scripts on ATNF machines, you will need to include three lines in your `.login` script. include;

```
source /applic/miriad/bin/MIRRC
set path = ($path $MIRBIN)
source /appplic/karma/.login
```

Other sites should have similar arrangements – only the directory should change.

The operations performed by these lines are simple enough. The `MIRRC` script, in the first line, sets up environment variables (including `MIRBIN`) to point to the appropriate directories. The second line includes the *MIRIAD* executables in your executable search path (you may choose to place it somewhere other than the end of the search path). The third line sets up the ATNF Visualisation Software.

You may also wish to set the environment variable `MIRDEF` to point to a directory where the keyword files generated by the `miriad` front-end should be stored. For example:

```
% setenv MIRDEF /mydirectory/mirdef
```

By default these keyword files are stored in the directory `~/mirdef`, if it exists when `MIRRC` is invoked, or in your current working directory.

The login initialisation script, `MIRRC.sh`, exists for users of the Bourne or Bash shells. This would typically be invoked in your `.profile` script. Like its `csh` cousin, `MIRRC.sh` does not modify the `PATH` environment variable. You will want to include `$MIRBIN` in your path.

Appendix B

Image Items

In this Appendix we will describe the items that can be present in an image dataset. $\mathcal{MIRIAD}$ utilities such as `prthd`, `imlist`, `itemize`, `puthd`, `copyhd` and `delhd` may be used to browse or modify these 'header variables'.

Note that the units used in $\mathcal{MIRIAD}$ are not always FITS-like, *e.g.* frequencies are stored in GHz in $\mathcal{MIRIAD}$, whereas in FITS one finds Hz. For example, $\mathcal{MIRIAD}$ keeps velocity in km s^{-1}, sky coordinates in radians and frequencies in GHz.

Table B.1: Item names in $\mathcal{MIRIAD}$ image datasets

Item	Type	Units	Description/Comments
bmaj,bmin	real	radians	Beam major and minor axis FWHM.
bpa	real	degrees	Beam position angle, measured east from north.
bunit	character		The units of the pixels.

btype	character		The type of the image. Possible values:
			`intensity` – Normal map.
			`beam` – Point spread function.
			`depolarisation_ratio`
			`fractional_polarisation`
			`optical_depth`
			`polarised_intensity`
			`position_angle`
			`rotation_measure`
			`spectral_index`

cdelt1,cdelt2,...	double	⇒ ctype	The increment between pixels.

cellscal	character		Latitude/longitude cell scaling convention with frequency/velocity.
			Possible values are:
			`CONSTANT`
			`1/F`

crpix1,crpix2,...	real	pixels	The pixel coordinate of the reference pixel.
crval1,crval2,...	double		The coordinate value at the reference pixel.
		radians	if ctype represents a celestial coordinate.
		km s^{-1}	if ctype `VELO` or `FELO`
		GHz	if ctype `FREQ`
ctype1,ctype2,...	character		The type of the nth axis (as FITS CTYPE).
datamin,datamax	real	⇒ bunit	The minimum and maximum pixel values.
epoch	real	years	The epoch of the coordinate system.
history	text		A text item containing the history of processing performed to the data set.
image	real	⇒ bunit	The pixel data.
llrot	double	radians	Eastward rotation relative to north of the sky grid relative to the pixel grid.
lstart,lstep,lwidth	real		
ltype	character		The linetype used in making the map.
mask	integer	#	Bitmap used to determine which pixels in the image have been blanked.
mostable	binary		Mosaic information table.
naxis	integer	#	Number of dimensions.
naxis1,naxis2,...	integer	#	Number of pixels along each dimension.
niters	integer	#	The total number of deconvolution iterations performed on the image.
object	character		source name
observer	character		observers name
obsra,obsdec	double	radians	The apparent RA and DEC of the observation centre.
obstime	double	Julian date	The mean observing time.
pbfwhm	real	arcsec	Gaussian primary beam size (deprecated).
pbtype	character		Primary beam type. Standard values include: `HATCREEK, VLA, ATCA, WSRT`
restfreq	double	GHz	The rest frequency of the observed data.
telescop	character		Telescope name. Standard values include: `HATCREEK, VLA, ATCA, WSRT`
vobs	real	km s^{-1}	Velocity of the observatory with respect to the rest frame, during the observation.

Appendix C

UV Variables

A *MIRIAD* uv dataset is composed of a collection of items and '$u-v$ variables'. The variables are parameters that are known at the time of the observation, and include measured data, and the description of the observation set up (*e.g.* correlator set up and observing centres).

Table C.1 gives a list of the items that are used to build up a *MIRIAD* uv dataset.

The *Programmers Guide* contains more detailed information on how a visibility dataset is constructed, this Appendix only reports which variables can be found in the item `visdata`. The text item `vartable` contains an ordered (for quick indexing) list of all the variables which exist in the `visdata` item.

A list of all items in a visibility dataset is summarised in Table C.1 below. A list of all the uv variables can be obtained with the *MIRIAD* program `uvlist`.

The storage **types** (2nd column) in the table below are:

```
A -- ascii (NULL terminated)
R -- real (32 bit IEEE)
D -- double (64 bit IEEE)
C -- complex (2 * 32 bit IEEE)
I -- integer (32 bit twos complement)
J -- short (16 bit twos complement)
```

They are the same as the data type in the first column of the `vartable` item in a *MIRIAD* uv dataset.

Variables with two dimensions have the first dimension varying fastest, the usual FORTRAN notation.

Table C.1: *MIRIAD* items in a uv visibility dataset

Item name	Type	Description
obstype	ascii	value: ‘`cross`’, ‘`auto`’ or ‘`mixed`’
history	text	history text file (in principle editable)
vartable	text	lookup table for all uv variables (do not edit!)
visdata	mixed	data stream of uv variables
flags	integer	optional flags for narrowband data
wflags	integer	optional flags for wideband data
gains	mixed	antenna gain table
nfeeds	integer	number of feeds on each antenna
ntau	integer	Number of delay/spectral index terms per antenna in ‘gains’
nsols	integer	number of records in ‘gains’
ngains	integer	number of antenna gains in each record of ‘gains’
interval	double	gain interpolation time tolerance (days)
leakage	complex	polarization leakage parameters
freq0	double	reference frequency for delay terms
freqs	mixed	frequency set up description table for ‘bandpass’
bandpass	complex	bandpass function gains
nspect0	integer	number of windows in the bandpass function
nchan0	integer	total number of channels in the bandpass function

Name	Ty	Units	Comments
airtemp	R	centigr.	Air temperature at observatory
antaz(nants)	D	deg.	azimuth of antennas
antdiam	R	meters	Antenna diameter
antel(nants)	D	deg.	elevation of antennas
antpos(nants, 3)	D	nanosec	Antenna equatorial coordinates
atten(nants)	I	dB	Attenuator setting (Hat Ck)
axismax(2,nants)	R	arcsec	Maximum tracking error in a cycle. axismax(1,?) is azimuth error, axismax(2,?) is the elevation error.
axisrms(2,nants)	R	arcsec	RMS tracking error. axisrms(1,?) is azimuth error, axisrms(2,?) is the elevation error.
baseline	R	-	The current antenna baseline Baseline is stored as $256 * ant1 + ant2$ or $2048 * ant1 + ant2 + 65536$ The uv coordinates are calculated as $uvw = xyz(ant2) - xyz(ant1)$. Note that this is different from the AIPS/FITS convention (where $uvw = xyz(ant1) - xyz(ant2)$). When writing this variable, software must ensure that $ant1 < ant2$. `baseline` is also known as `preamble(4)`
bin	I	-	Pulsar bin number.
cable(nants)	D	nanosec	measured length of IF cable (Hat Ck)
calcode	A	-	ATCA calcode flag
chi or chi(nants)	R	radians	Position angle of the X feed relative to the sky. This is the sum of the parallactic angle and the `evector` variable. If only one value is present, all antennas are assumed to have identical values.
coord(2)	D	nanosec	uv baseline coordinates `coord` is also known as `preamble(1:2)`
corbit	R	-	Number of correlator bits (Hat Ck)
corbw(2)	R	MHz	Correlator bandwidth setting (Hat Ck) Must take the values

			1.25, 2.5, 5.0, 10.0, 20.0, 40.0 & 80.0 MHz.
corfin(4)	R	MHz	Correlator LO setting before Doppler tracking (Hat Ck) This is the LO frequency at zero telescope velocity Must be in the range 80 to 550 MHz.
cormode	I	-	Correlator mode (Hat Ck). Values are: 1 : 1 window /sideband by 256 channels 2 : 2 windows/sideband by 128 channels 3 : 4 windows/sideband by 64 channels, single sideband 4 : 4 windows/sideband by 64 channels, double sideband
coropt	I	-	Correlator option (Hat Ck) 0 means cross-correlation 1 means auto-correlation Same as the `obstype` item?
corr(nchan)	J or R	-	Correlation data `corr` is really a complex quantity, but the data stream variable can be stored otherwise for efficiency.
cortaper	R	-	On-line correlation taper (Hat Ck) This is the value at the edge of the window The value is from 0-1.
ddec	R	radians	Offset in declination from `dec` in `epoch` coordinates. The actual observed DEC is calculated as dec + ddec.
dec	R or D	radians	Declination of the phase center/tangent point. See `epoch` for coordinate definition.
delay(nants)	D	nanosec	delay setting at beginning of integration (Hat Ck)
delay0(nants)	R	nanosec	delay offset for antennas (Hat Ck)
deldec	R or D	radians	Declination of the delay tracking center. See `epoch` for coordinate definition.
delra	R or D	radians	Right ascension of the delay tracking center. See `epoch` for coordinate definition.
dewpoint	R	centigr.	Dew point at weather station (Hat Ck)
dra	R	radians	Offset in right ascension from `ra` in `epoch` coords. The actual observed RA is calculated as ra + dra/cos(dec).
epoch	R	years	A badly named variable – this defines the mean equinox and equator for the equatorial coordinates `ra`, `dec`, `dra` and `ddec`. The epoch of the coordinates is actually the observing time. Values less than 1984.0 are Besselian with coordinates in the FK4 system. Values greater than 1984.0 are Julian with coordinates in the FK5 system.
evector or evector(nants)	R	radians	Position angle of the X feed, to the local vertical. If only one value is present, all antennas are assumed to be identical.
focus(nants)	R	volts	Focus setting (Hat Ck)
freq	D	GHz	Rest frequency of the primary line
freqif	D	GHz	? (Hat Ck only?)
inttime	R	seconds	Integration time
ischan(nspect)	I	-	Starting channel of spectral window
ivalued(nants)	I	?	Delay step (Hat Ck) Used in an attempt to calibrate amp and phase vs. delay.
jyperk	R	Jy/K	The efficiency Jy/K, calculated during calibration
latitud	D	radians	Geodetic latitude of the observatory.
lo1	D	GHz	First local oscillator (Hat Ck) `lo1` is in the range 70 GHz - 115 GHz.
lo2	D	GHz	Second local oscillator (Hat Ck)
longitu	D	radians	Longitude of the observatory.
lst	D	radians	Local apparent sidereal time.
mount or mount(nants)	I	-	The type of antenna mounts. If only one value is given, all antennas are assumed to be the same. Possible values are: 0: Alt-az mount. 1: Equatorial mount.

			2: X-Y. 3: orbiting. 4: bizarre.
name	A	-	ATCA raw RPFITS file name.
nants	I	-	The number of antennas Following variables use a dimension of `nants`: antpos(nants, 3) focus(nants) phaselo[1-2](nants) phasem1(nants) systemp(nants, nspect) wsystemp(nants, nwide) temp(nants, ntemp) tpower(nants, ntpower) axisrms(2,nants) The antennas are always numbered starting at 1.
nbin	I	-	Total number of pulsar bins.
nchan	I	-	The total number of individual frequency channels The following variables have the dimension of nchan: corr(nchan)
npol	I	-	The number of simultaneous polarisations
nschan(nspect)	I	-	Number of channels in spectral window
nspect	I		Number of spectral windows Following variables use a dimension of `nspect`: ischan(nspect) nschan(nspect) restfreq(nspect) sdf(nspect) sfreq(nspect) systemp(nants, nspect)
ntemp	I	-	Number of antenna thermisters Following variables use a dimension of `ntemp`: temp(nants, ntemp)
ntpower	I	-	Number of total power measurements The following variable depends on `ntpower`: tpower(nants,ntpower) `ntpower` is currently 1, could be more later.
nwide	I	-	Number of wideband channels Variables which depend on `nwide` are: wfreq(nwide) wwidth(nwide) wcorr(nwide) wsystemp(nants,nwide)
obsdec	D	radians	Apparent declination of the phase centre/tangent point at time of observation.
observer(*)	A	-	The name of the observer
obsra	D	radians	Apparent right ascension of the phase centre/tangent point at time of observation.
on	I	-	Either 0 or 1, for on and off pointing, for auto-correlation data.
operator(*)	A	-	The name of the current operator
pbfwhm	R	arcsec	(Deprecated) Primary Beam at Full Width Half Maximum For Hat Ck, it is approximately 11040.0/`lo1`.
pbtype	A	-	Primary beam type to be used in imaging.
phaselo1(nants)	R	radians	Antenna phase offset (Hat Ck)
phaselo2(nants)	R	radians	Second LO phase offset (Hat Ck)
phasem1(nants)	R	radians	IF cable phase (Hat Ck)
plangle	R	degrees	Planet angle
plmaj	R	arcsec	Planet major axis (note units)
plmin	R	arcsec	Planet minor axis
pltb	R	Kelvin	Planet brightness
pntdec	R or D	radians	Declination of the pointing center. See `epoch` for coordinate definition.

pntra	R or D	radians	Right ascension of the pointing center. See `epoch` for coordinate definition.
pol	I	-	Polarization type of the correlation data. Values follow the AIPS/FITS convention, viz: 1: Stokes I 2: Stokes Q 3: Stokes U 4: Stokes V -1: Circular RR -2: Circular LL -3: Circular RL -4: Circular LR -5: Linear XX -6: Linear YY -7: Linear XY -8: Linear YX
precipmm	R	mm	Mm of precipitable water vapour in the atmosphere.
pressmb	R	millibar	atmospheric pressure.
project(*)	A	-	The name of the current project
ra	R or D	radians	Right ascension of the phase center/tangent point. See `epoch` for the definition of the coordinate system.
rain	R	mm	The current amount of water in the rain guage. The rain guage is emptied at 9:00 AEST (ATCA).
refpnt(2,nants)	R	arcsec	Reference pointing offsets. refpnt(1,?) is azimuth offset, refpnt(2,?) is the elevation offset.
relhumid	R	%	Relative Humidity at observatory
restfreq(nspect)	D	GHz	Rest frequency for each spectral window. This may be zero for continuum observations.
sctype	A	-	Scan type
sdf(nspect)	D	GHz	Change in frequency per channel
sfreq(nspect)	D	GHz	Sky frequency of first channel in window
smonrms	R	μm	ATCA seeing monitor rms value
source(*)	A	-	The name of the source
systemp or systemp(nants) or systemp(nants,nspect)	R	Kelvin	Antenna system temperatures

telescop(*)	A	-	The telescope name. Some standard values are:
			`'ATCA'`
			`'HATCREEK'`
			`'VLA'`
			`'WSRT'`

temp (nants, ntemp)	R	centigr.	Antenna thermistor temperatures (Hat Ck)
themt(nants)	R	Kelvin	temperature of the hemt amplifier (Hat Ck)
tif2(nants)	R	Kelvin	temperature of IF amplifier (Hat Ck)
time	D	days	The time (nominally UT1) stored as a Julian date. For example, noon on Jan 1, 1980 is 2,444,240.0! `time` is also known as `preamble(3)`
tpower (nants, ntpower)	R	volts	Total power measurements (Hat Ck)
tscale	R	-	Optional correlation scale factor Used only when `corr` is stored as J (16 bits).
tsis(nants)	R	Kelvin	temperature of the SIS mixers (Hat Ck)
ut	D	radians	The time since midnight Universal time (nominally UT1).
veldop	R	km s^{-1}	The sum of the radial velocity of the observatory (in the direction of the source, with respect to the rest frame) and the nominal systemic radial velocity of the source.
veltype(*)	A	-	Velocity rest frame. Possible values for `veltype` are: VELO-LSR: rest frame is the LSR VELO-HEL: rest frame is the barycentre VELO-OBS: rest frame is the observatory FELO-LSR: rest frame is the LSR (deprecated) FELO-HEL: rest frame is the barycentre (deprecated)

version(*)	A	-	The current hardware/software version Current options: oldhat, newhat
vsource	R	km s^{-1}	Nominal radial systemic velocity of source. Positive velocity is away from observer.
wcorr(nwide)	C	-	Wideband correlations. The current ordering is: wcorr(1:2) are the digital LSB and USB. wcorr(3:4) are the analog LSB and USB.
wfreq(nwide)	R	GHz	Wideband correlation average frequencies
wind	R	km/h	Wind speed in km/h
winddir	R	deg	Wind direction (note: originally encoded as 'N', 'SE', 'W', etc.)
windmph	R	mph	Wind speed - in imperial units!
wsystemp or wsystemp(nants) or wsystemp(nants,nwide)	R	K	System temperature for wide channels.
wwidth(nwide)	R	GHz	Wideband correlation bandwidths
xsampler (3,nants,nspect)	R	percent	X sampler statistics (ATCA).
xtsys(nants,nspect)	R	Kelvin	System temperature of the X feed (ATCA).
xyamp(nants,nspect)	R	Jy	On-line XY amplitude measurements (ATCA).
xyphase (nants,nspect)	R	radians	On-line XY phase measurements (ATCA).
ysampler (3,nants,nspect)	R	percent	Y sampler statistics (ATCA).
ytsys(nants,nspect)	R	Kelvin	System temperature of the Y feed (ATCA).

Appendix D

Preparing Your Data in AIPS

If you feel that you would rather load and flag your data within *AIPS* but calibrate within *MIRIAD*, a number of steps need to be performed:

1. The *AIPS* task to load RPFITS data is `ATLOD`. Copious information on this is given in Neil Killeen's 'Analysis of Australia Telescope Compact Array Data'. However, before loading your data into *AIPS*, if you have measured all four polarisation correlations, it is best to do a preliminary run of `ATLOD` using the `optype ='sysc'` option. This run writes a text file (`XYPHS_`*xx*, where *xx* is your *AIPS* number) containing the XY phase for each antenna into the FITS area (`/DATA/FITS`). Text files of system temperature are also written. Although the various selection parameters of `ATLOD` are still active, you probably want to see all the data. Most of the other `ATLOD` parameters are unimportant for this.

AIPS/ATLOD	
optype = 'sysc'	Load the data
freqsel	Select all data
ifsel	
source	
timer	

   You should plot the phases (and the system temperatures) with the Unix program `pltsys` – which prompts you for the name of the text file to plot as well as other information. Examine these plots carefully to assess their quality. You should use these plots to choose your reference antenna (for calibration purposes). Choose the reference antenna to be the antenna having the cleanest, most stable XY phase measurements.

   Determine some mean value of the XY phase for each antenna from the plots. The command `pltsys` prints out both the average and median XY phase. As there are often outliers, the median is more likely to reflect the true XY phase value. Getting a good value is only important for the reference antenna. Do not be too concerned if there are large jumps in the XY phases on antennas other than the reference antenna.

2. *No XY Phase Correction in* `ATLOD`*:* You are now confronted with the decision of where to correct the XY phase of the reference antenna. Your choice will depend on taste, circumstances and the quality of the XY phase measurements. There are three main options:

   - Correct using the *MIRIAD* task `atxy` (described later). This is generally the best option. This allows you to correct for the variation of XY phase with time. This will be essential if there is significant time variability. To use this approach, you will need to keep a copy of your XY phase text file, `XYPHS_`*xx*.
   - Correct in *AIPS* `ATLOD`. This has been the approach once recommended, and is still useful if the XY phase of at least one antenna is quite constant with time. It also has the advantage of getting the XY correction step over and done with early. However the user input can be error prone and tedious if the observation switches frequency with time.

- When only XX and YY correlations have been measured, absolute XY phase becomes irrelevant. Indeed it is not measured. In this case you do not need to apply any XY phase correction.

There are a number of other possibilities, which will not be described here.

If you are going to correct the XY phases in $\mathcal{MIRIAD}$, or if you are not going to correct XY phase at all, then you should now load your data *without applying any XY phase*. It is probably worth your while to pretend that the polarisations are circular rather than linear with the usual fudges, as not all the $\mathcal{AIPS}$ software will recognise linears (most of the calibration software will). You must *not* convert to Stokes parameters. The appropriate `ATLOD` parameters are

AIPS/ATLOD	
optype = 'load'	Load the data
aparm(1) =-1	Label as circular
cparm(5) =0	Do *not* apply any XY phases.

3. *Correcting XY Phases with* `ATLOD`*:* If you want to correct the XY phases with $\mathcal{AIPS}$ `ATLOD`, the XY phases on at least one antenna should be reasonably constant with time (vary by no more than a few degrees). In this case, give `ATLOD` the values of the XY phases that you determined from the plots discussed above. Input these values into `ATLOD` with the `xyphase` array. You must enter one value per antenna for each frequency. If you have more than one frequency, you must enter XY phases for all six antennas, even if you do not have antenna 6 in the array during the observation (the XY phase value is not important for this antenna, of course). Note that even if the values are close to zero, you still should apply them. *Applying a value of zero is different from not applying anything.* Again, you must *not* convert to Stokes parameters.

AIPS/ATLOD	
optype = 'load'	Load the data
aparm(1) =-1	Label as circular
cparm(4) =1	Use `xyphase` array and not the on-line values
cparm(5) =1	Apply XY phase to Y gains
xyphase	Assign the XY phases here

4. Now flag the data in the way you would normally do with the $\mathcal{AIPS}$ tasks `SPFLG`, `TVFLG`, `IBLED`, and `UVFLG`. Using `SPFLG` is highly recommended, particularly at 20 and 13 cm, to check for interference.

   For continuum work, to save disk space and to speed access to the data, you may consider averaging your channels together to form "channel-0" datasets, using task `AVSPC`. While this causes very little degradation for 3 cm observations, forming "channel-0" results in bandwidth smearing in 13 and 20 cm observations, and so is probably inadvisable for high dynamic range work there. For high dynamic range work at 6 cm, it is debatable whether averaging is detrimental. If in doubt, do not average. It is always possible to form a channel-0 dataset later anyway.

   One other consideration in determining whether or not to average is whether or not you are going to apply XY phase corrections with task `atxy`. For obscure reasons, if you used $\mathcal{AIPS}$ `ATLOD`, `atxy` needs to know the "sideband indicator" of the data. The sideband indicator, which is ±1, is copiously reported by `ATLOD`, both in its output to the terminal, and in the history file. The sideband indicator also happens to be the sign of the channel frequency increment. This is how `atxy` normally determines them. However, if you form a channel-0 dataset, the sign of the frequency increment is lost! So if you give `atxy` a channel-0 dataset, you will also have to tell it the sideband indicators. You must give it a sideband indicator for each IF. Provided the sideband indicator remains constant with time, this is little more than an annoyance. However if the sideband indicator varies with time, you are in some trouble. Overall it is best not to form channel-0 datasets if you used $\mathcal{AIPS}$ `ATLOD` and you are going to use `atxy`.

   After flagging (and possibly averaging), write your data as a FITS file using `FITTP`. It is probably most convenient to write out a multi-source file. At this stage you have no calibration – only flagging tables (which you can apply in $\mathcal{MIRIAD}$ later).

5. Read the data into *MIRIAD* using `fits`. Task `fits` does *not* apply *AIPS* flagging tables (`FG` tables). Instead you have to use another task to do this – `fgflag`. Tasks `fits` and `fgflag` are discussed in Chapters 8 and 10 respectively, although they are usually fairly straight forward. An exception is for spectral line observations, where the velocity system should be defined with `fits` – see Chapter 16.

FITS	
in=MULTI.FITS	FITS multi-source file to be loaded into *MIRIAD*
op=uvin	Read uv data in
out=multi.uv	The output *MIRIAD* dataset.

FGFLAG	
vis=multi.uv	Apply *AIPS* flagging table to the data.

At this stage it is worth running `uvindex`. This produces a summary of your dataset, which you should probably save in a log file. Inspect this summary carefully, particularly the frequencies (especially in `fits` complained about inconsistent frequency definitions). If the frequency information looks incorrect, read Chapter 8 more carefully and/or seek help.

UVINDEX	
vis=multi.uv	Dataset to summarise.
log=multi.log	Output log file.

6. Skip this step if you have only measured two polarisation products. Otherwise now is the time to apply your XY phases to the data if you have not already done so with *AIPS* `ATLOD`. As mentioned above, the task to do this is `atxy`. If you used *AIPS* `ATLOD`, you should also have XY phase text file that it produced. Let us discuss the various input parameters:

   - `vis:` The name of the input dataset. Generally this will be a multi-source dataset.
   - `xyphase:` You need to give the name of the XY phase text file that *AIPS* `ATLOD` produced. Generally this will be of the form `XYPHS_`*xx*, where *xx* is your *AIPS* user number. Task `atxy` can also be used to correct the XY phases of data loaded with *MIRIAD* `atlod` where `options=xycorr` was not used. In this case, the on-line XY phase measurements are contained within the dataset (as the variable `xyphase`), and you do not require a input text file.
   - `refant:` It is best to correct for a time varying XY phase on only one antenna. All other antennas are assumed to have an XY phase of zero (in the *AIPS* XY phase convention – see below). The antenna which is corrected for a time varying XY phase should be the antenna with the cleanest, most constant XY phase (as determined by the `pltsys` or `varplt` plots). This will be the antenna you will use as the reference antenna for calibration purposes. It is not necessarily the same as the antenna used as the reference during the observation – although it will often happen that it is the same. You give this antenna via the `refant` keyword. Note that if you have broken a dataset up into sub-files before using `atxy` (e.g. break it up into a calibrator dataset and a program source dataset), then you *must* correct the same antennas in all datasets.
   - `interval:` This gives one or two numbers, both in minutes, which determine the length of a solution interval (the time interval over which an XY phase is solved for; should be comparable to the time scale on which the XY phases are constant over – use `varplt`). The first number gives the maximum length of a solution interval, whereas the second gives the maximum gap within a solution interval. A new solution interval is started when either the maximum time length is exceeded, or a gap larger than the maximum gap is encountered. The default maximum length is 30 minutes. The default maximum gap is the same as the maximum length.
   - `break:` If significant steps in the XY phase occurs during the observation (usually caused by resetting the delays), then it is best to prevent a solution interval spanning this time. To do this, you list the times where there was a break in the XY phase. The times are given in the normal *MIRIAD* time format (i.e. either *hh:mm:ss* or *yymmmdd:hh:mm:ss*, such as 93oct18:19:21:00, for 7:21 pm on 18 October, 1993).

- `sideband:` As noted above, `atxy` needs to know the sideband indicator for for data loaded with *AIPS* `ATLOD`. For multi-channel data, this will be the sign of the frequency increment. However this sign will be lost if you form a channel-0 dataset. In this is what you have done, you will need to tell `atxy` the sideband indicators for each IF band. Note that `atxy` cannot cope with a channel-0 dataset if the sideband indicators change with time.
- `out:` The name of the output dataset. Apart from application of the XY phase, this will be a copy of the input dataset.

Typical inputs to `atxy` are given below

ATXY	
vis=multi.uv	The input dataset.
xyphase=XYPHS_56	The input *AIPS* `ATLOD` XY phase text file, or
xyphase	leave unset if *MIRIAD* `atlod` was used.
refant=3	Correct antenna 3 in time varying way.
interval=#	Solution interval. Default is 30 min
sideband	Sideband indicator. Leave unset for multi-channel data.
out=multi.uvxy	Output XY phase corrected data.

Appendix E

Using xmtv and xpanel

E.1 Using xmtv

TV-related tasks use the `server` keyword to indicate the TV type and its host. For `xmtv`, this is given in the form:

`xmtv@`*host*

Here *host* is the host name of the workstation running `xmtv`. For example

```
server=xmtv@tucana
```

indicates that `xmtv` is running on the machine `tucana`.

The TV server display can be resized, zoomed, panned and also offers the capability to change the lookup table and transfer function. However, the current *MIRIAD* TV server implementation is not very well integrated with concepts such as extracting quantitative information from the display.

Note that the computer where you are running *MIRIAD* tasks may differ from the computer running `xmtv` (e.g. you may be running *MIRIAD* tasks on a compute server such as `kaputar` or `raptor`, and running `xmtv` on your local workstation). The *host* part of the `server` parameter is the machine running `xmtv`.

As `xmtv` uses a specific internet port, normally only a single instance of `xmtv` can run on a given machine at a given time. This is why you should start `xmtv` on your *local* workstation, rather than a compute server. If this is not possible, then you should attempt to run it on the compute server. If you get a message such as:

```
XMTV:MakeLink - bind error (INET): Address already in use
initializeSocket - MakeLink error: Address already in use
```

then the internet port is already in use. In this case, you can tell `xmtv` to use a different port (e.g. port 5001) with

```
% xmtv -port 5001 &
```

(substitute another number for 5001 if it still fails). If you use a port number other than the default, you will need to give this in the `server` keyword, e.g. for port 5001 on machine `tucana`, use

```
server=xmtv:5001@tucana
```

E.2 The Control Panel

A few tasks (notably `tvflag`, but also `tvdisp` in its movie mode) can use a "control panel". This is a panel of buttons that pops up, with each button performing some interactive function associated with the TV device. Although always used in unison with the TV device, this control panel is strangely a separate executable, which must also be started by hand. To do this, give the command

```
% xpanel &
```

to the same computer that you are using for `xmtv`.

Appendix F

Glossary

The following is a loose list of terms used throughout the *MIRIAD* manuals, and perhaps not always explained in detail.

@file An @file (at-file) or 'include file' is a text file containing the value of a keyword. If multiple lines are used, a newline is equivalent to a comma.

breakpoint A discontinuity (often in time or frequency), *e.g.* meant to signify that fitting data should be done in separate intervals. Used in calibration software.

BIMA Berkeley-Illinois-Maryland Association, but also: the VAX in Maryland's Astronomy Program.

C-shell In UNIX this is one of the possible shells that can be used to communicate with UNIX. Other shells are `tcsh`, the T-shell, `sh`, the Bourne Shell and `bash`, the GNU Shell. See also: **script**.

dataset A *MIRIAD* dataset is a hierarchical set of (data) items, and has been implemented as a host system directory.

deconvolution algorithm Algorithm by which remove instrumental point spread function from an image. *e.g.* `clean` and `maxen`).

doc file See 'help file'.

FITS Flexible Image Transport System. Standard data format to interchange data between different computers. Can be used for image as well as visibility data.

header variable Misnomer for a *MIRIAD* **item**. Do not confuse a header variable with a uv variable.

help file File with extension `.doc` that explains workings of a *MIRIAD* task. Names of parameters and their defaults are also in here. Help files live in directory **$MIRPDOC**. The command `mirhelp` can be used to get any help on *MIRIAD*.

host Your local computer. The term *host* is often used as in *host interpreter* and *host commands*, and is really meant to warn you that in this context commands may differ depending on which host/machine (often VMS vs. UNIX) you work.

keyword file File containing *'keyword=value'* assignments, one per line. Each user interface can read such files, although the extension is often assumed to be `.def`. See Chapter 2.

image centre pixel This is the central pixel in an image. If pixel numbers vary from 1 to N, the centre pixel is pixel $\mathrm{int}(N/2)+1$. This pixel should be distinguished from the 'image reference pixel' (see below).

image reference pixel This is the pixel in an image which is used as a reference point for the image coordinate system. If the image was produced by task `invert`, or derived from an image produced by `invert`, the reference pixel will correspond to the same point in the sky as the observing centre (see 'observing centre' and 'image centre pixel').

include file Other name for @file

item Name tagged data, often of the same datatype, that is part of a dataset. For images **item**s are what we commonly refer to as the 'header variables', but also the `history` and actual `image` data are items in a dataset.

observing centre A *MIRIAD* uv file does not distinguish between the pointing centre, phase centre and delay centre. All are assumed to be the same point on the sky, and are fixed by the observation process. Documentation refers to this point simply as the observing centre.

override When an item takes precedence over a uv variable. Only applies to *MIRIAD* visibility datasets.

PGPLOT Package used to get graphical displays on the screen or hardcopy device.

PostScript file An ASCII text file in the PostScript language. PostScript is a page description language, and has become an industry standard for printing high-quality text and graphics. One of the PGPLOT device drivers creates a PostScript file, which can be sent to a printer to get hardcopy output.

script A set of commands bundled together in a text file, which can be executed. Depending on the interpreter, command flow and various other programming styles are possible.

shell A supposedly easier to handle front-end command processor used to communicate with a lower level, and often more difficult to handle program or operating system.

spectral window See 'window'.

window A window, or spectral window, in *MIRIAD* usage, means a sequence of frequency channels which are separated by the same frequency increment. The Hat Creek correlator is capable of observing eight spectral windows simultaneously, whereas the ATCA can observe two simultaneously. This is somewhat analogous to the *AIPS* 'IF band' concept.

task A *MIRIAD* task is a regular executable in the sense of the host operating system, but only accepts input parameters via a set of `keyword=value` assignments and/or `-f keyfile` keyword files, that cannot contain blanks. Help on the program can be obtained via `mirhelp` the command.

uv Referring to the `u`, `v`, and formally `w` variables in synthesis observations, commonly called 'uv'. Most often this term is a shorthand notation where we could have used the word 'visibility'.

uv variable Not to be confused with a header variable, a $u - v$ variable is a named quantity with a certain type (integer, real, ...) and dimension, that can change value during the course of an observation.

Appendix G

Trouble Shooting

This appendix attempts to suggest a few ways of troubleshooting while running *MIRIAD*.

MIRIAD programs usually announce trouble by starting a line with the `###` characters. If a 'Fatal Error' message follows, the program is also aborted, although in case of a 'Warning' or 'Informational' message, the task continues to run, at your own risk.

Some messages are related to the fact that errors occur in low-level subroutines, which have no built-in way to escape back to the user level, and be descriptive about the problem. In such cases somewhat obscure system-like messages like *'No such directory'* could be encountered, and other error messages are just what we sometimes jokingly call *'pilot errors'*, where you forgot to supply the program with an essential piece of information.

G.1 CAVEATS

- Datasets are directories, and cannot be created when they already exist in UNIX. You should not manipulate the component files inside a *MIRIAD* dataset using the hosts normal utilities. Doing so can potentially 'corrupt' the dataset.
- Parameter values cannot contain spaces.

G.2 Error messages

- *### Warning: Parameter vis not used or not exhausted.* The parameter *vis* to that particular task has been given multiple values, probably separated by commas, or the file-wild card expansion has been used, or from a @file. The program, however, did not read and use all of them. For file name the common error is that the program can only handle one file at a time.
- *### Fatal Error: No such file or directory.* A typical UNIX error message. Your program was either supplied with a non-existing file name or dataset name (remember, datasets are really directories). Also, and this is more obscure, it could be that for strange reasons one of the files (or items) which is supposed to be in the dataset directory, is missing. The file `header` must always be present in a *MIRIAD* dataset directory.
- *### Fatal Error: Keyini: Input parameter too long for buffer* The text you supplied to this parameter had too many characters. Try using an @ (include) file. If string space is still exhausted your system manager may have to increase a buffer length in the `key.for` file and recompile at least the program it failed on.
- *### go: Failed to exec the subprocess* A typical UNIX response – totally illegible for normal humans. This means that while trying to execute your task, UNIX found it could not do it. There

could be a lot of reasons why this was so, one of them being the wrong machine type of the executable, but also not enough memory to start the task etc.

- *Command not found.* See Section G.3 below.
- *Found no documentation on task* `xxx` No .doc file; either the default task in `miriad` is wrong, or someone did not install a document file. Check if the file `$MIRPDOC/xxx.doc` exists, where `xxx` is the taskname.

G.3 Command not found

A very common error made by beginning users is that *MIRIAD* has not been set up properly, resulting in a *'Command not found'* message.

On UNIX this means that the `$MIRBIN` directory must be in your search path for executables. If any of the following commands produces an *Undefined variable* message or the like, check your `.login` file to see if it was set up properly:

```
% echo $MIR           # check which MIR you are using
% echo $MIRBIN        # see if the MIRBIN makes sense
% echo $PATH          # see if MIRBIN is inside the PATH
% rehash              # rehash the search (assume csh is used)
% ls $MIRBIN          # see what is present in MIRBIN
```

If not, check a `.login` file from one of your colleagues to see how it should be set up. Another possibility is that the program has not been loaded into `$MIRBIN`, but the documentation is present, or the other way around. In both cases the *MIRIAD* system manager needs to be consulted.

G.4 Not a directory

Since *MIRIAD* datasets are implemented as directories, some of the system messages may complain when you have mixed up a file with a dataset.

Appendix H

Bibliography

Bracewell R.N., The Fourier Transform and its Applications

Briggs D., 1995, "High fidelity deconvolution of moderately resolved sources" PhD thesis
`http://www.aoc.nrao.edu/dissertations/dbriggs/` .

Christiansen W.N., Högbom J.A., Radiotelescopes.

Clark B.G., 1980, "An efficient implementation of the algorithm CLEAN" *Astron. Astrophys.* 26, 89.

Cornwell T.J., 1983, "A method of stabilizing the clean algorithm" *Astron. Astrophys.*, 121, 281.

Cornwell T.J., 1988 "Radio-interferometric imaging of very large objects" *Astron. Astrophys.*, 202, 316.

Cornwell T.J., Holdaway M.A., Uson J.M., 1993 "Radio-interferometric imaging of very large objects: implications for array design" *Astron. Astrophys.*, 271, 697.

Cornwell T.J., Uson J.M., Haddad N., 1992, "Radio-interferometric imaging of spectral lines - The problem of continuum subtraction" *Astron. Astrophys.*, 258, 583.

Frater R.H. ed., 1992, Journal of Electrical and Electronics Engineering, Australia: Special Issue – The Australia Telescope, 12
`http://www.narrabri.atnf.csiro.au/observing/iree.pdf` .

Greisen E., 1993, "*Non-linear coordinate systems in AIPS*" *AIPS* memo No. 27
`ftp://ftp.aoc.nrao.edu/pub/software/aips/TEXT/PUBL/AIPSMEMO27.PS` .

Hamaker J.P., Bregman J.D., Sault R.J., 1996, "Understanding radio polarimetry. I. Mathematical foundations" *Astron. Astrophys.* 117, 137.

Högbom J.A., 1974, "Aperture Synthesis with a Non-Regular Distribution of Interferometer Baselines" *Astron. Astrophys. Suppl.* 15, 417.

Killeen N.E.B., 1996, "Effective spectral resolution of the ATNF correlator" AT Memo 39.3/057
`http://www.atnf.csiro.au/observers/memos/` .

Killeen N.E.B., Lo K.Y., Crutcher R., 1992, "Zeeman measurements of the magnetic fields at the Galactic center" *Astrophys. J.*, 385, 585.

Kraus J.D., Radio Astronomy

Rayner, D. 2000, "Circular polarization users guide", AT Memo 39.3/102
`http://www.atnf.csiro.au/observers/memos/` .

Reynolds J.E., 1994, "A revised flux scale for the AT Compact Array" AT Memo 39.3/040
`http://www.atnf.csiro.au/observers/memos/` .

Rohlfs K., Wilson T.L., Tools of Radio Astronomy.

Sault R.J., 1991, "Some simulations of self-calibration for the AT" AT Memo 39.3/058 `http://www.atnf.csiro.au/observers/memos/` .

Sault R.J., 1994, “An analysis of visibility-based continuum subtraction” *Astron. Astrophys. Suppl.*, 107, 55.

Sault R.J., 2003, “ATCA flux density scale at 12mm” AT Memo 39.3/124
`http://www.atnf.csiro.au/observers/memos/` .

Sault R.J., Bock D.C.-J., Duncan A.R., 1999, “Polarimetric imaging of large fields in radio astronomy” *Astron Astrophys. Suppl.*, 139, 387.

Sault R.J., Killeen N.E.B., Zmuidzinas J., Loushin R., 1990, “Analysis of Zeeman effect data in radio astronomy” *Astrophys. J. Suppl.*, 74, 437.

Sault R.J., Killeen N.E.B., Kesteven, M.J., 1991 “AT Polarisation Calibration” AT Memo 39.3/015
`http://www.atnf.csiro.au/observers/memos/` .

Sault R.J., Hamaker J.P., Bregman J.D., 1996, “Understanding radio polarimetry. II. Instrumental calibration of an interferometer array” *Astron. Astrophys.*, 117, 149.

Sault R.J., Noordam J.E., 1995, “Eliminating solar interference in radio interferometric spectrometry” *Astron. Astrophys. Suppl.*, 109, 539.

Sault R.J., Staveley-Smith L., Brouw W.N., 1996, “An approach to interferometric mosaicing” *Astron. & Astrophys. Suppl.*, 120, 376.

Sault R.J., Teuben P.J., Wright M.C.H., 1995, “A retrospective view of Miriad” in *Astronomical Data Analysis Software and Systems IV*, ed. R. Shaw, H.E. Payne, J.J.E. Hayes, ASP Conf. Ser., 77, 433-436.

Sault R.J., Wieringa M.H., 1994, “Multi-frequency synthesis techniques in radio interferometric imaging” *Astron. Astrophys. Suppl.*, 108, 585.

Stanimirovic S., 2002, “Short-Spacings Correction from the Single-Dish Perspective” ASP Conf. Series, 278, p 375.

Steer D.G., Dewdney P.E., Ito M.R., 1984, “Enhancements to the deconvolution algorithm CLEAN” *Astron. Astrophys.*, 137, 159

Subrahmanyan R., 2005, “Photogrammetric measurement of the gravity deformation in a Cassegrain antenna” *IEEE Trans Antennas Propogation*, 53, 2590-2596.

Taylor G.B., Carilli C.L., Perley R., eds, Synthesis Imaging in Radio Astronomy II, Astronomical Society of the Pacific Conference Series Volume 180.

Thompson A.R., Moran J.M., Swenson G.W., Interferometry and Synthesis in Radio Astronomy.

Ulich B.L. 1980, “Improved correction for millimeter-wavelength atmospheric attenuation”, *Astrophys. Letters*, 21, 21.

van Hoerner S., 1967, “Design of large steerable antennas” *Astron. J.*, 72, 35.

Wieringa M.H., Kesteven M.J, 1992, “Measurements of the ATCA primary beam”, AT Memo 39.3/024
`http://www.atnf.csiro.au/observers/memos/` .

Index

.login, G-2

abspixel, image coordinate units, 6-2
amplitude, uv select, 5-6
ansiread, 8-2
ansitape, 8-2
antenna configuration files, ATCA, 9-2
antenna configuration files, VLA, 9-2
antennae, uv select, 5-6
APCLN, 14-11
arcsec, image coordinate units, 6-2
atfix, 10-1, 10-2, 11-3, 23-1, 23-4, 23-5
ATLOD, 11-7, D-1
atlod, 7-3, 8-1–8-4, 10-17, 11-7, 12-1, 12-9, 13-3, 13-5, 16-2–16-4, 22-1, 22-2, 23-3, 23-9, 24-1, 24-2, D-3, D-4
atxy, 11-7, D-1–D-4
auto, uv select, 5-7
avmaths, 16-5

background, 2-3
blflag, 10-2–10-4, 24-2
boxes, region of interest, 6-2

calibration, application, 5-2
calibration, bandpass, 12-4
calibration, bandpass slope, 12-11
calibration, calibration model, 5-1
calibration, copying, 11-8, 12-6
calibration, deleting, 11-8
calibration, gain solution averaging, 12-14
calibration, general, 5-1
calibration, glitches, 12-15
calibration, inspection, 12-8
calibration, interpolation, 12-13
calibration, interpolation tolerance, 12-14
calibration, planet flux, 12-11
calibration, plotting, 11-8
calibration, polarimetric, 11-1, 12-6, 12-7
calibration, solvers, 11-5
calibration, strategy, 12-1
calibration, XY phases, 11-7, D-1–D-3
cg, 18-8
cgcurs, 6-1, 10-11, 14-4, 17-1, 17-10, 17-11, 18-2, 18-3, 18-9, 21-16
cgdisp, 17-1–17-3, 17-6, 17-7, 17-9, 17-10, 18-4, 20-4, 20-6
cginit, 17-1, 17-6
cgslice, 17-1, 17-10, 17-11
cgspec, 17-1, 17-2, 17-10, 19-1–19-3

channel, linetype, 5-3
clean, 7-3, 14-1, 14-3–14-10, 15-1, 15-5, 21-7, F-1
closure, 10-16
command line switch, 2-8
conterr, 16-6, 16-7
contsen, 16-6
contsub, 16-5
convol, 18-7
copyhd, 4-2
CVEL, 16-3, 16-5

datasets, archiving, 8-9
datasets, deleting, 4-1
datasets, managing, 4-3
datasets, transporting, 8-9
ddec, uv select, 5-6
deconvolution, 14-1
deconvolution, Clark CLEAN, 14-2
deconvolution, CLEAN stripes, 14-3
deconvolution, Cotton-Schwab CLEAN, 14-3
deconvolution, Högbom CLEAN, 14-2
deconvolution, maximum entropy, 14-6
deconvolution, multi-frequency synthesis, 14-8
deconvolution, restoring, 14-10
deconvolution, SDI CLEAN, 14-3
delhd, 4-2, 11-8, 11-9
demos, 21-7–21-9, 21-11
disks, 4-3
dra, uv select, 5-6

ellint, 18-1, 18-2, 21-1

fft, 18-11, 18-12
fgflag, 8-6, 10-10, D-3
FITS, B-1
fits, 2-10, 8-5–8-9, 10-10, 13-5, 16-2, 16-3, D-3, F-1
FITS files, 8-5
FITS files, FG tables, 10-10
FITS files, images, 8-9
FITS files, visibility data, 8-5
frequency, uv select, 5-6

gaufit, 18-2, 18-3
gethd, 4-2
gpaver, 12-14, 12-15
gpboot, 12-10, 12-11, 22-1, 22-2, 23-8
gpbreak, 12-15
gpcal, 11-5–11-8, 12-1, 12-6–12-8, 12-10, 12-14–12-16
gpcopy, 11-6, 11-8, 12-6, 12-11, 21-4, 21-11

gpnorm, 12-8–12-10
gpplt, 11-8, 12-7–12-9
gpscal, 11-5, 15-2, 15-4, 15-6, 15-7, 21-8

hanning, 8-3, 16-2, 20-2
help, mirhelp, 2-8
histo, 2-7, 2-9, 18-1, 18-9, 21-12

im2uv, 9-3
image analysis, 18-1
image display, 17-1
image display, contour diagrams, 17-1
image display, cursor, 17-1
image display, pixel map, 17-1
image display, region-of-interest, 17-1
images, region of interest, 6-1
imaging, 13-1
imbin, 18-7, 18-8
imblr, 18-11, 18-12
imcat, 18-4
imdiff, 18-11
imfit, 18-2, 18-3, 21-7
imframe, 18-4
imgen, 2-7, 4-2, 18-8, 20-8, 20-9, 21-16, 21-18
imheq, 18-11, 18-12
imhist, 18-1
imlist, 18-2, 21-9
immerge, 21-12–21-18
impol, 18-8–18-10
impoly, 18-11, 18-12
impos, 18-2, 18-3
imrm, 18-8–18-10
imspec, 19-1, 19-2
imspect, 19-1, 19-2
imstat, 18-1, 18-2, 21-12
imsub, 18-4, 20-2
increment, uv select, 5-6
invert, 5-4, 7-3, 13-1, 13-3–13-6, 14-1, 14-3, 14-4, 14-7, 14-9, 15-5, 16-8, 16-9, 18-5, 18-8, 21-4–21-7, 21-9–21-12, 21-16, F-1
item, F-2
item, definition, 4-1
itemize, 4-2, 6-2

keyword, files, 2-8
kms, image coordinate units, 6-2

lastexit, 2-1
line, 5-3
linear feeds, 11-1
linetype, uv, 5-3
linmos, 21-1, 21-11
load, miriad, 2-5

man, 8-2, 8-4, 8-5, 8-10
mask, region of interest, 6-2
maths, 18-4–18-6, 18-8, 20-8–20-10, 21-17
maxen, 7-3, 14-1, 14-6–14-10, 15-1, 15-5, 21-7, F-1
maxfit, 18-2
maximum entropy, 14-6
mfboot, 12-7, 12-11, 12-13, 23-6, 23-8, 23-9
mfcal, 5-1, 11-5, 11-6, 12-4, 12-6, 12-8, 12-10, 12-12, 12-13, 12-16, 23-6, 23-7
mfclean, 7-3, 13-1, 13-4, 13-5, 14-1, 14-8–14-10, 15-1, 15-5, 21-6
mfspin, 21-1
millimetre data, 22-1, 23-1
MIRBIN, environment, G-2
mirbug, 2-5
mirhelp, 2-8
miriad, front-end, 2-1
moment, 20-1, 20-8
mosaicing, 21-1, 21-2
mosaicing, calibration, 21-3
mosaicing, combining single dish data, 21-11
mosaicing, imaging, 21-5
mosaicing, individual approach, 21-10
mosaicing, joint approach, 21-4
mosaicing, joint deconvolution, 21-6
mosaicing, observing strategies, 21-2
mosaicing, self-calibration, 21-7
moslst, 21-9
mosmem, 14-10, 21-6–21-9, 21-12, 21-15–21-18
mospsf, 21-7, 21-9
mossdi, 21-6, 21-7
mossen, 21-9, 21-11, 21-17
mostess, 21-9
mt, 8-4, 8-5, 8-10
multi-frequency synthesis, deconvolution, 14-8
MX, 14-11

on, uv select, 5-7
or, uv select, 5-7, 5-8

pbplot, 21-1
pgflag, 10-3, 10-7–10-9
pgplot, F-2
planets, 12-13, 23-9
plboot, 23-8
plplt, 12-13, 23-9
pmosmem, 14-10, 21-9
pointing, uv select, 5-6
polarimetry, fractional polarisation, rotation measure, 18-8
polarimetry, rotation measure image, 18-9
polarization, 5-10
polarization, uv select, 5-7
polygon, region of interest, 6-2
PostScript, F-2
primary beam, correction, 21-1
primary beam, models, 21-1
prthd, 4-1, 6-2, 10-12, 10-13, 21-15
psrfix, 24-2
psrplt, 24-2
pulsar bin mode, 24-1
puthd, 4-2, 8-7, 8-8, 12-14, 16-4, 21-1, 21-9, 21-15, 21-16

PUTHEAD, 8-6, 8-8

quack, 10-10
qvack, 10-10

region, of interest, 6-1
regrid, 18-4, 18-5, 21-12
relcenter, image coordinate units, 6-2
relpixel, image coordinate units, 6-2
reorder, 17-7, 18-4, 20-1, 20-10
restor, 7-3, 14-4, 14-7, 14-10, 14-11, 21-7, 21-12, 21-15, 21-17
rm, 4-1
RPFITS files, 8-1

save, miriad, 2-5
script, F-1
select, 5-5
select, $u-v$ data selection, 5-5
self-calibration, 15-1
self-calibration, frequency handling, 15-4
selfcal, 5-1, 11-5, 15-2, 15-4–15-7, 21-8, 21-9
shadow, uv select, 5-6
shifty, 18-11
single dish, combining with mosaics, 21-11
smooth, 18-7
source, uv select, 5-6
spectral index, 18-5
spectral line, 16-1
spectral line, continuum subtraction, 16-5
spectral line, display, 19-1
spectral line, fits and velocity definitions, 16-2
spectral line, image velocity axis, 16-8
spectral line, imaging, 16-8
spectral line, listing velocity information, 16-3
spectral line, recomputing velocity information, 16-3
spectral line, rest frequency, 16-4
spectral line, velocity definition, 16-1
spectral line, velocity linetype, 16-4
SPLIT, 8-6
stokes, 5-10
Stokes parameters, 11-1

TABED, 8-8
tar, 8-10
telepar, 9-3
time, uv select, 5-5
TMPDIR, environment, 2-6
tpcp, 8-5
tvdisp, E-2
tvflag, 10-2, 10-3, 10-5–10-7, 21-3, 24-2, E-2

unset, miriad, 2-4
uv, linetypes, 5-3
uvaflag, 10-11
uvaver, 5-2, 7-2, 10-13, 10-15, 12-1, 15-2, 21-3, 23-10, 24-2
uvcat, 5-2, 5-9
uvedit, 8-8, 10-17
uvflag, 10-3, 10-9, 12-1, 12-12, 23-6, 23-7, 23-9
uvflux, 12-8, 12-9
uvgen, 9-1–9-3
uvindex, 8-8, 10-12, D-3
uvlin, 16-5, 16-7, 16-8, 23-10
uvlist, 10-2, 10-12, 16-3, C-1
uvmodel, 9-3, 16-5
uvnrange, uv select, 5-6
uvplt, 2-10, 10-14, 10-15, 12-7–12-10, 22-2
uvputhd, 8-8
uvrange, uv select, 5-6
uvredo, 8-8, 10-17, 16-2–16-4
uvsector, 10-11
uvspec, 5-2, 10-16
uvsplit, 10-3, 12-2, 12-3, 21-3, 21-10

varlist, 10-17
varplt, 8-4, 10-17, 12-7, 12-10, 16-3, 22-2, 23-5, D-3
vartable, visibility item, C-1
velfit, 20-8
velimage, 20-8, 20-9
velmodel, 20-8, 20-9
velocity, linetype, 16-4
velocity,linetype, 5-3
velplot, 20-2–20-8
velsw, 16-2, 16-9
visdata, visibility item, C-1
visibility datasets, calibration, 5-1
visibility datasets, closure phase plotting, 10-16
visibility datasets, correcting, 10-1
visibility datasets, flagging, 10-2, 10-9, 10-10
visibility datasets, generating, 9-1
visibility datasets, listing, 10-12
visibility datasets, modifying, 10-17
visibility datasets, plotting, 10-14
visibility datasets, spectral plotting, 10-16
visibility datasets, variables, 10-16
VTESS, 21-9

wide, linetype, 5-3
window, uv select, 5-6

XY phase, 11-7
XY phase, convention difference, 11-7
XY phases, D-1–D-3

Zeeman analysis, 20-10
Zeesim, 20-10
zeesim, 20-10–20-12
zeestat, 20-10–20-12