Miriad Continuum Data Processing Tutorial

Dave Rayner

September 25, 2001

Contents

1 Introduction

This paper provides a tutorial introduction to processing simple ATCA "continuum" data with MIRIAD. The tutorial aims both to demonstrate the fundamental steps for calibrating and imaging synthesis data, and also to show what real ATCA data looks like.

The tutorial assumes a basic knowledge of synthesis telescopes and imaging; you should know what are meant by visibilities, gain, bandpass, baseline, uv-plane, Fourier transform, and deconvolution. The final part of the tutorial deals with self-calibration. No prior experience with Miriad is assumed.

1.1 A quick rant

In general, there is no one "right" method for calibrating ATCA data! Nearly everyone has their own preferences for options and what tasks to do in what order, which can make processing ATCA data quite confusing for beginners (and so-called experts!). The calibration process followed in this tutorial differs from that described in the *Miriad User's Guide*, for example. Although the finer points of synthesis imaging are still a bit of a "black art", the improvements which can be obtained by tweaking the calibration and imaging process are really only second order, and all sensible processing strategies should yield very similar results. Synthesis imaging is like drinking; everyone has their own theory on the best way to do it, but the final result is usually pretty similar.

There are, however, a number of very "wrong" ways of calibrating ATCA data. Everyone stumbles into one of these sometime. The most innocuous processing mistakes reduce the sensitivity of the image, while the nastiest will cause realistic-looking but totally erroneous features.

The processing strategy followed below is designed to be robust in the presence of unexpected errors, but even so, when it comes to processing ATCA data, a "cook-book" is no substitute for understanding the principles of synthesis imaging. At the same time, most people only come to understand the principles of synthesis imaging by first following a cook-book! So when you follow this processing procedure, make sure you understand what is actually being done at each step, and why.

Also, check the results of each calibration and imaging step, by examining plots of the visibilities, gains or the images as appropriate. Too much checking never hurt; too little might.

1.2 For more information

- Miriad User's Guide: `http://www.atnf.csiro.au/computing/software/miriad/userhtml.html`

- Synthesis Imaging in Radio Astronomy II, Taylor, Carilli and Perley.

2 The Miriad package

The Miriad package is a collection of tasks for processing radio-astronomical data. Data processing in Miriad proceeds stepwise; at each step, you run a task which will either modify a data-set (eg. delete corrupted visibilities), create a new data-set (eg. create an image from a visibility data-set), or display some aspect of a data-set (eg. plotting visibilities vs. time). Often, the output data-set of one task becomes the input data-set for the next step.

Miriad tasks are really unix executables, which are invoked with a series of named arguments. They can be run from the unix command-line or by other programs (eg. shell scripts - see Appendix A), but are usually run using the Miriad shell (see below). In the examples which follow, Miriad task names are usually written in CAPITALS.

Miriad data-sets are stored on disk as unix directories - you can copy, rename or delete miriad data-sets like any other directory. However, you should not modify the internal contents (the *items*) of a data-set, except via the Miriad tasks.

Note that miriad data-sets do not contain revisions; you can not "undo" the changes! In most cases this is not a serious draw-back, because many tasks generate new output data-sets. Thus, when you make a mistake you usually only have to re-do the last or last few calibration steps.

2.1 The Miriad Shell

The Miriad shell is a convient environment for specifying arguments and options for miriad tasks, and executing them. It is an old, text-style interface[1]. To start the Miriad shell, type:

```
miriad
```

The essential Miriad shell commands are:

help Get help on a topic, eg. `help miriad`, or `help uvplt`.

inp *task* Select a task. Displays a list of arguments for the specified task, eg. `inp uvplt`. With no *task* argument, **inp** displays the list of arguments for the currently selected task.

key=value Set the argument for the current task, eg. `vis=1934-638.1384`.

unset *key* Unset the argument ie. let it take a default value.

go Execute the currently selected task.

[1] This is partly because no-one has ever gotten around to writing anything better, partly because AIPS++ is going to take over the world, and partly because many people, once they are familiar with Miriad, use unix shell scripts or other programs to run Miriad tasks. And also because the Miriad shell usually does a pretty good job.

tget *task* Select *task*, and retrieve the inputs that were used last time it was run. Very useful.

Miriad shell understands the unix directory tree; it maintains a working directory, which you can change using `cd`, and will handle aguments such as `in=../mydata.uv`.

Any command that the Miriad shell does not recognise is passed out to the host unix shell, eg. `ls -l /n/mydir/miriaddata/`.

2.2 Default values

One of Miriad's greatest failings is that many of the tasks have a large number of obscurely-named input parameters - try `inp cgdisp` and you'll see what I mean. Luckily, nearly all parameters will take a sensible default value if left unset. Thus, in the examples which follow, only those arguments which are actually given a value will be listed. To find out what all the arguments do, use the help eg. `help cgdisp`.

Needless to say, many of the tasks provide more much functionality than is described in this brief introduction.

2.3 Pgplot

Most of the display tasks in Miriad use the pgplot graphics package. These tasks give some control over the plot by the `device` key. `device=/xs` is the most common `device` setting, which outputs to a pgplot window on the screen. For more options, see `help device`.

2.4 Kview and Viewer

Miriad has a few tasks for displaying images, but nothing as useful or sophisticated as kview or the aips++ viewer. Only when you need to do something Miriad-specific, like define a sub-region from an image, would I suggest using the Miriad tasks.

I prefer kview, firstly because when I type `kview` it works, and secondly because kview can read Miriad images. The kview documentation is embedded in the Karma documentation[2], which means you have to sift through a lot of crap to find what you want. On the other hand, kview is very easy to use.

The aips++ viewer looked stunning in demonstration, though I haven't used it myself. I don't think viewer will read miriad images directly; you have to export them with the task FITS first. The viewer documentation[3] is part of the aips++ documentation, which means you have to sift through a lot of crap to find what you want.

[2]http://www.atnf.csiro.au/karma/user-manual/index.html
[3]http://aips2.nrao.edu/released/docs/gettingresults/gettingresults/node3.html

3 A tutorial example: Gravitational-lens candidate PKS 0252-549

PKS 0252-549 was observed with the ATCA in 1996, as part of a search for gravitational lenses. Observations were conducted at 1384 MHz, 2368 MHz, 4800 MHz and 8640 MHz. This tutorial describes processing the 1384 MHz data.

Some of the display tasks produce multi-page output; only the first page is usually displayed.

3.1 Loading ATCA data into Miriad

The task to convert ATCA data[4] into a Miriad data-set, use ATLOD. For example, to load data from CD-ROM, change to your data directory using `cd`, then:

```
Task:   atlod
in      = /cdrom/cdrom1/DATA/*.C619
options = birdie,xycorr,reweight,noauto
out     = C619.uv
```

3.2 Splitting

Although you could work on C619.uv, it's more convenient to split it into several data-sets, each containing a unique pointing and frequency.

```
Task:   uvsplit
in      = C619.uv
```

UVSPLIT creates files in the current working directory of the form *sourcename.freq* eg. `1934-638.1384`. I usually create a subdirectory for each frequency (eg. `1384MHz`), and distribute the data-sets accordingly.

3.3 Bandpass calibration

The ATCA "continuum mode" correlator configuration produces 33 partially-overlapping channels covering a total of 128 MHz band-width. The ATLOD task drops the edge-channels and consolidates the remainder to 13 contiguous 8 MHz channels. Because the gain on each antenna will be frequency-dependent, the channels appear to have different flux levels. To account for this effect, we do a bandpass calibration.

For continuum observations, bandpass calibration is usually performed using an observation of the ATCA primary calibrator, 1934–638, for which the spectral index is known.

[4] ATCA data is writtin in the "RPFITS" format.

3.3.1 Examining and flagging visibilities

The first thing you should always do is to look at the data. Was the telescope off-source? Was there interference? A detailed observing log will help here.

To view visibilities in Miriad use UVPLT (see Figure 1).

```
Task:   uvplt
vis      = 1934-638.1384/
stokes   = xx,yy
axis     = time,amp
device   = /xs
nxy      = 3,5
size     = 3,3
```

If you give `nxy=3,5`, you'll need a big pgplot window. At this stage the visibilities are best treated as *feed-based*; the xx visibilities are the correlations between the X-polarized feed on one antenna and the X feed on another.

The visibility data for the primary-calibrator should be almost constant. There appear to be a few bad points on baseline 4–5, and to a lesser extent on 1–2. Baseline 4–5 was the shortest in the array, and so most subject to interference.

The best tool for flagging[5] visibilities is BLFLAG:

```
Task:   blflag
vis      = 1934-638.1384/
device   = /xs
stokes   = xx,yy
axis     = time,amp
select   = ant(4)(5)
```

A long-thin pgplot window is best for BLFLAG. Press **h** when the cursor is in the pgplot window, and BLFLAG lists available commands. Basically, you just click the bad data.

The `select=ant(4)(5)` selects just the 4–5 baseline. See `help select` for more ways to select antennas and baselines.

3.3.2 Determine the bandpass

Now you can determine the bandpass with MFCAL.

```
Task:   mfcal
vis      = 1934-638.1384
interval = 0.1
```

[5] geek-speak for "marking as corrupted"

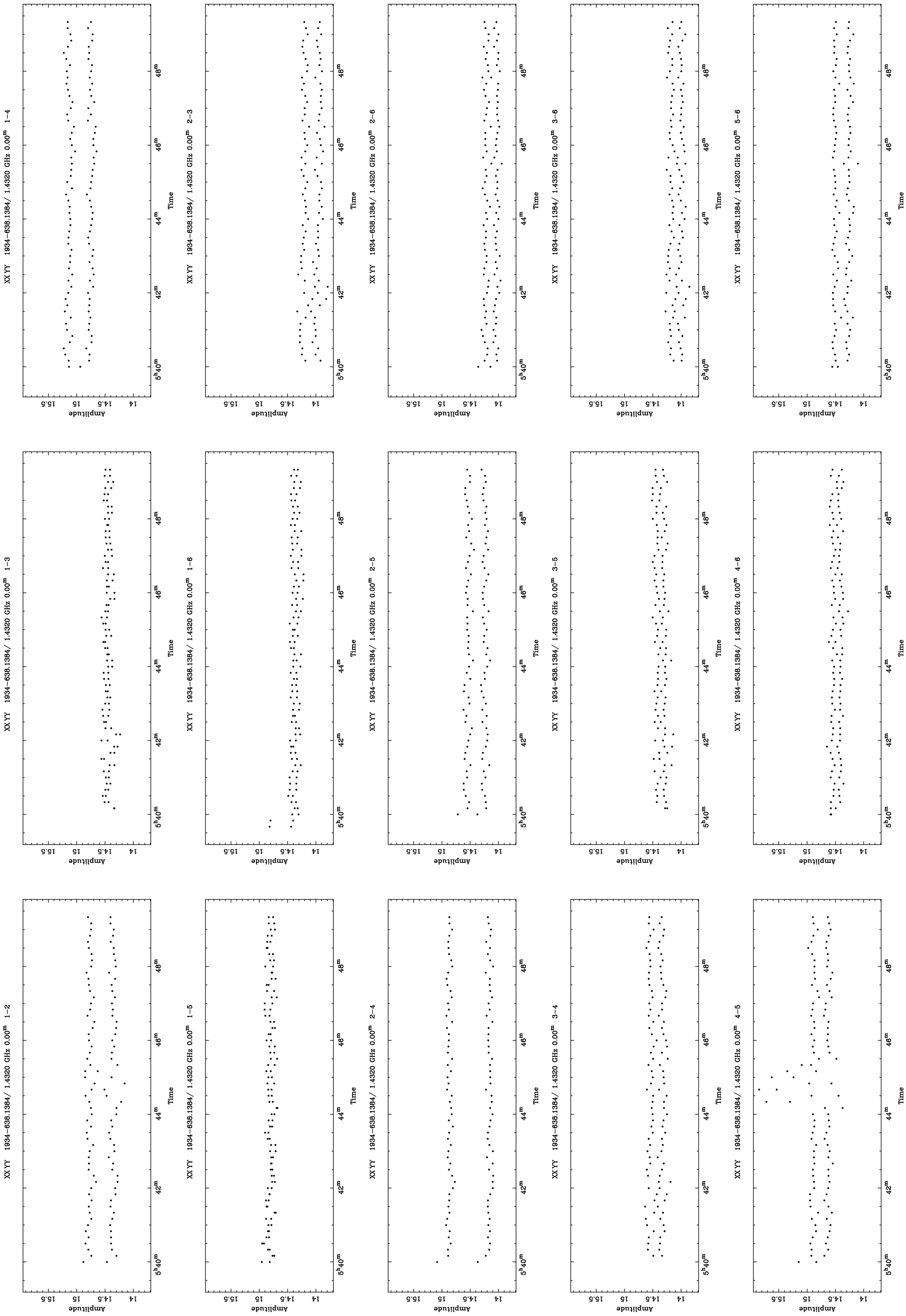

Figure 1: UVPLT of the primary calibrator, 1934–638

Mfcal usually issues a few warnings about correlations being flagged and the flux scale used.

Mfcal actually solves for two antenna-based gain terms; a frequency-dependent, time-independent *bandpass*, and frequency-independent, time-dependent *gains*. Both are complex. The `interval` is the time-step for the gains; setting it to 0.1 minute, which is less than the ATCA integration time of 10 seconds, forces mfcal to solve for the gains once per integration.

To view the resultant bandpasses, use gpplt (see Figure 2):

```
Task:   gpplt
vis      = 1934-638s.1384
device   = /xs
yaxis    = amp
options  = bandpass
nxy      = 2,3
```

If there are large spikes in the bandpass, you may need to flag whole channels, rather than just visibilities. Or maybe there was some interference which you didn't flag.

Also, at this stage you should look at the visibility data again, just to check the calibration gave a sensible result. Use `axis=time,amp` and `axis=time,phase`; the phases should be close to zero. The results are shown in Figures 3 and 4.

There are some visibilities at the start of the track which did not calibrate very well, as well as the baseline 1–2 visibilities which were subject to interference. A few bad visibilities like these won't make any difference to the calibration, so I left them unflagged. If you're feeling enthusiastic, and you flag them with blflag, don't forget to re-run mfcal afterwards! Otherwise, the bandpass and gains won't be improved.

3.4 Polarization leakage correction

If you run uvplt with `stokes=xy,yx` (see Figure 5)

```
Task:   uvplt
vis      = 1934-638.1384
stokes   = xy,yx
axis     = time,amp
device   = /xs
nxy      = 3,3
size     = 2,2
```

you'll see there are correlations between the X feed on one antenna and the Y feed on another; the visibilities have non-zero amplitudes. Because 1934–638 is known to be unpolarized, this implies that some X-polarized radiation is "leaking" into the Y-polarized feed, or vice-versa. If you are interested in making polarization images (and you should make at least one – a Stokes V image – as a test; see later), then you will need to correct for these leakages.

The task which determines the leakages is gpcal. Gpcal is similar to mfcal in usage, except that it solves for time-independent leakages and time-dependent gains, instead of bandpass and gains. Gpcal will over-write the gains written by mfcal.

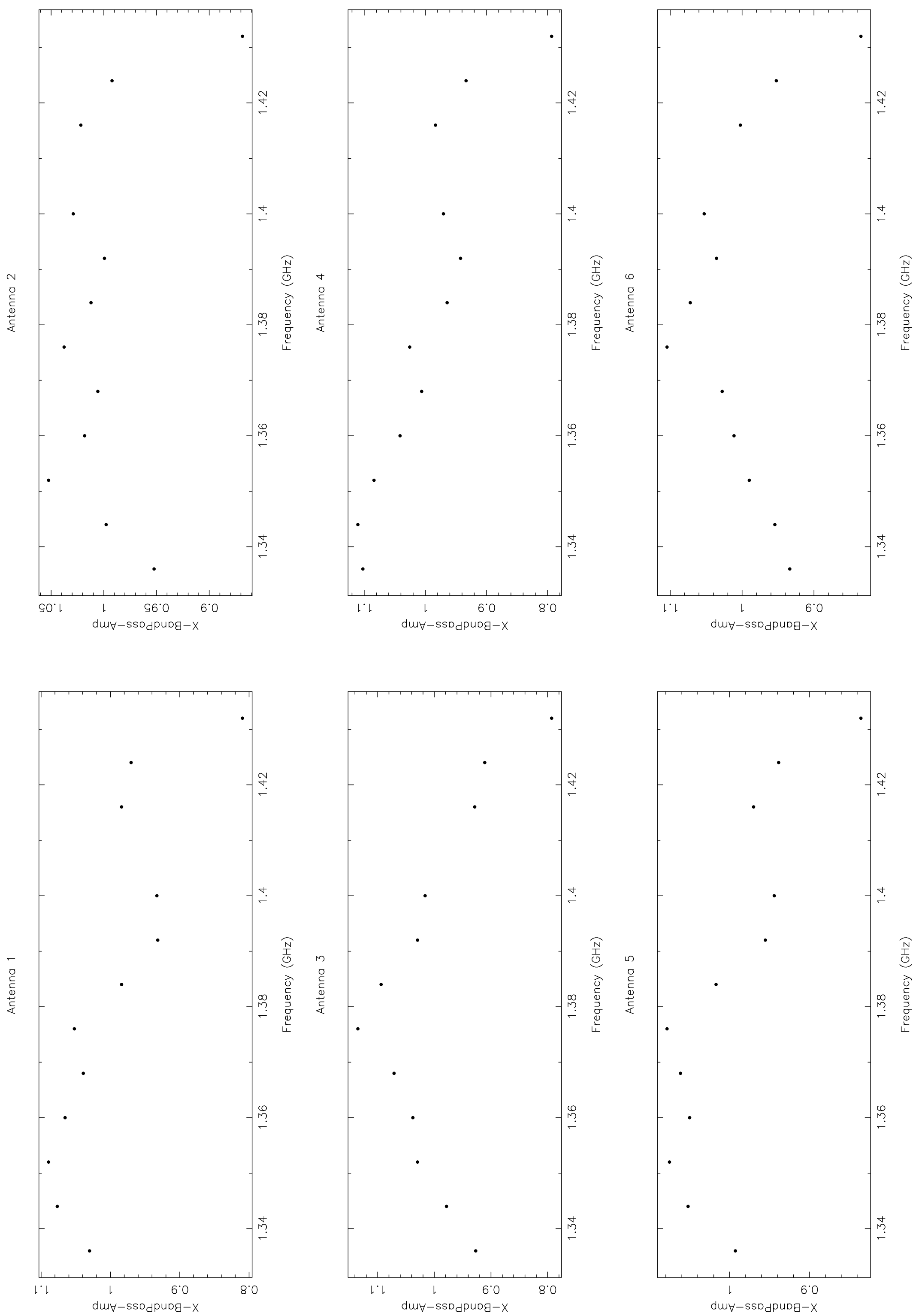

Figure 2: GPPLT showing the bandpass derived from MFCAL.

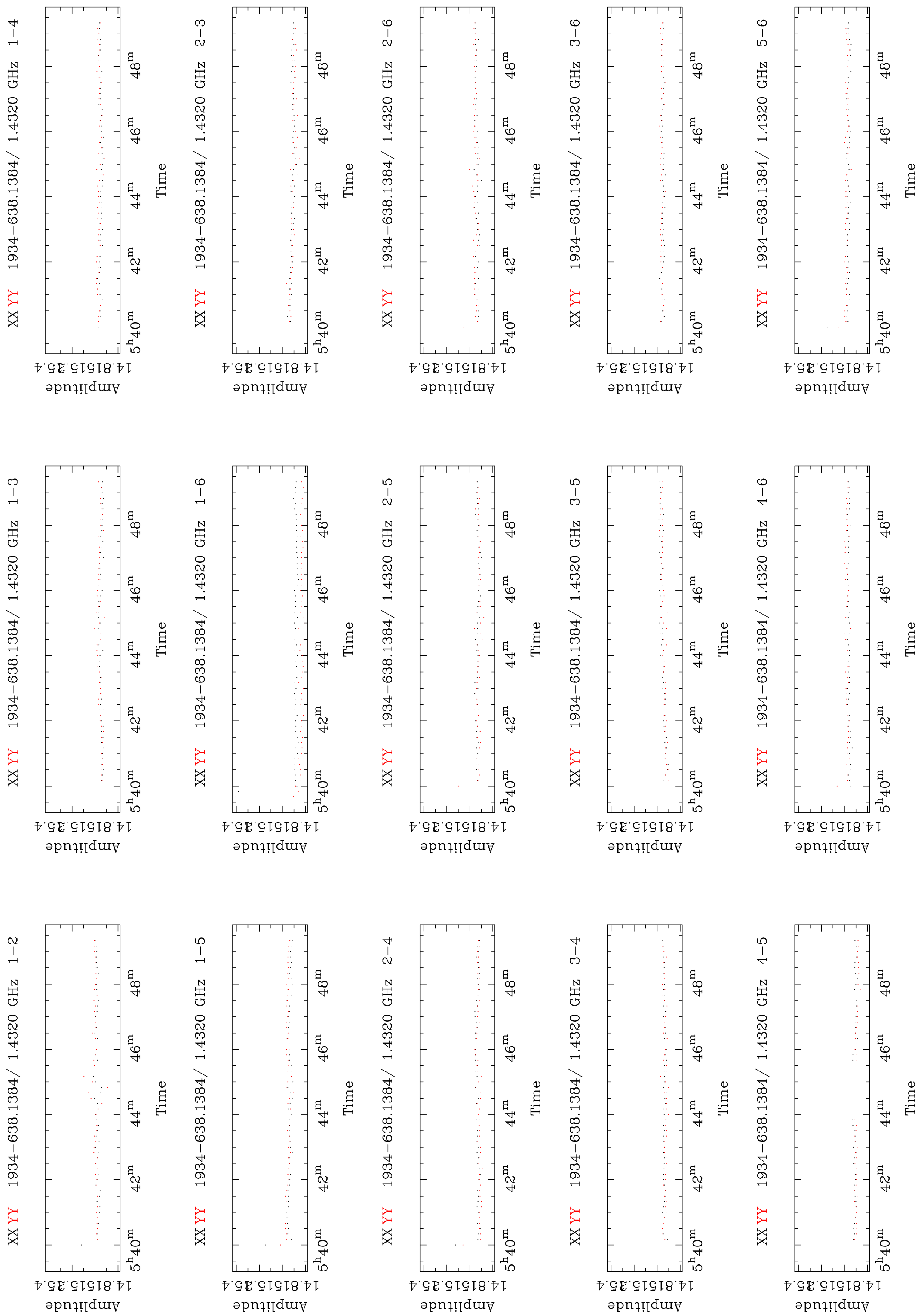

Figure 3: UVPLT of the 1934–638 amplitudes, after running MFCAL

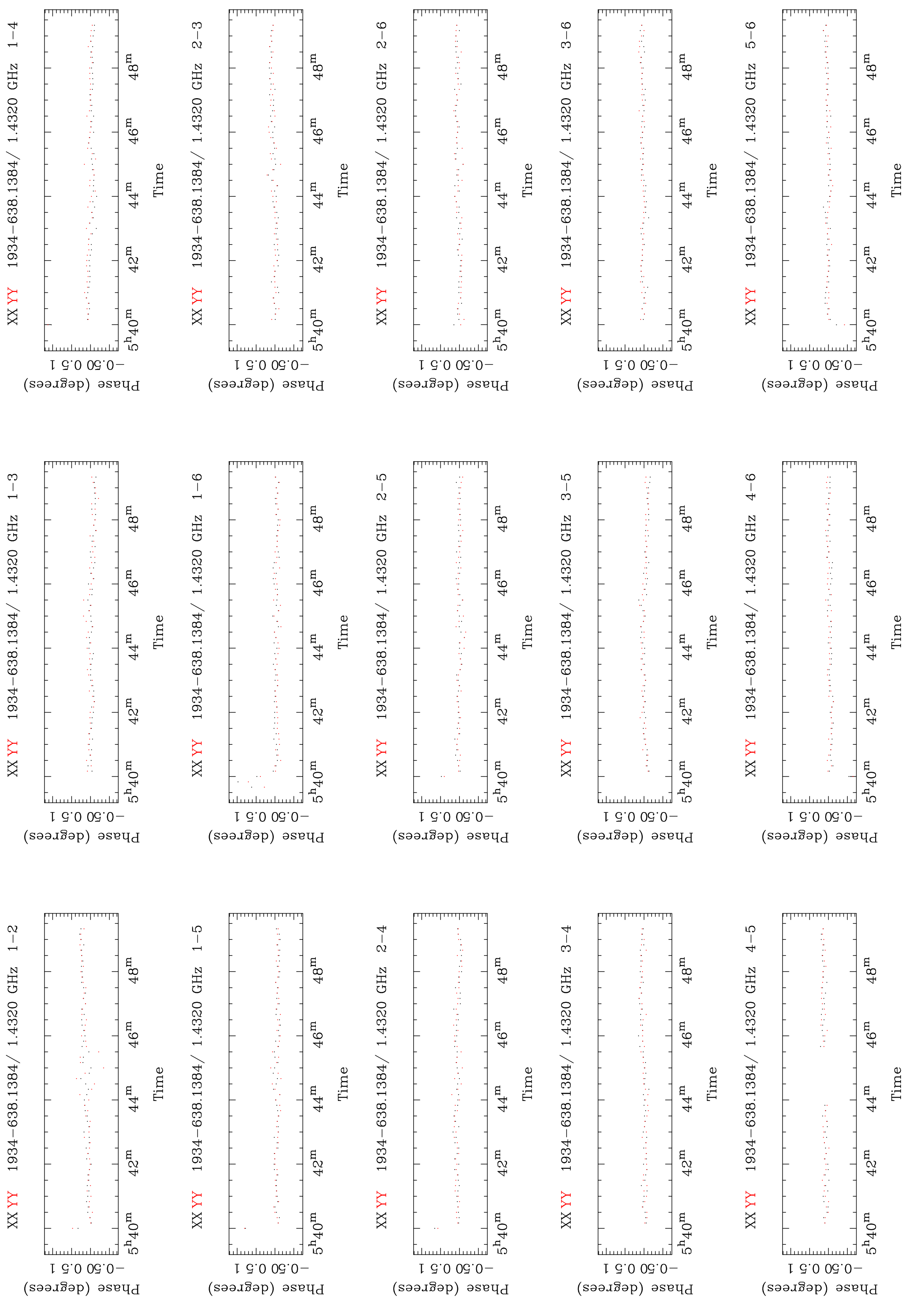

Figure 4: UVPLT of the 1934–638 phases, after running MFCAL

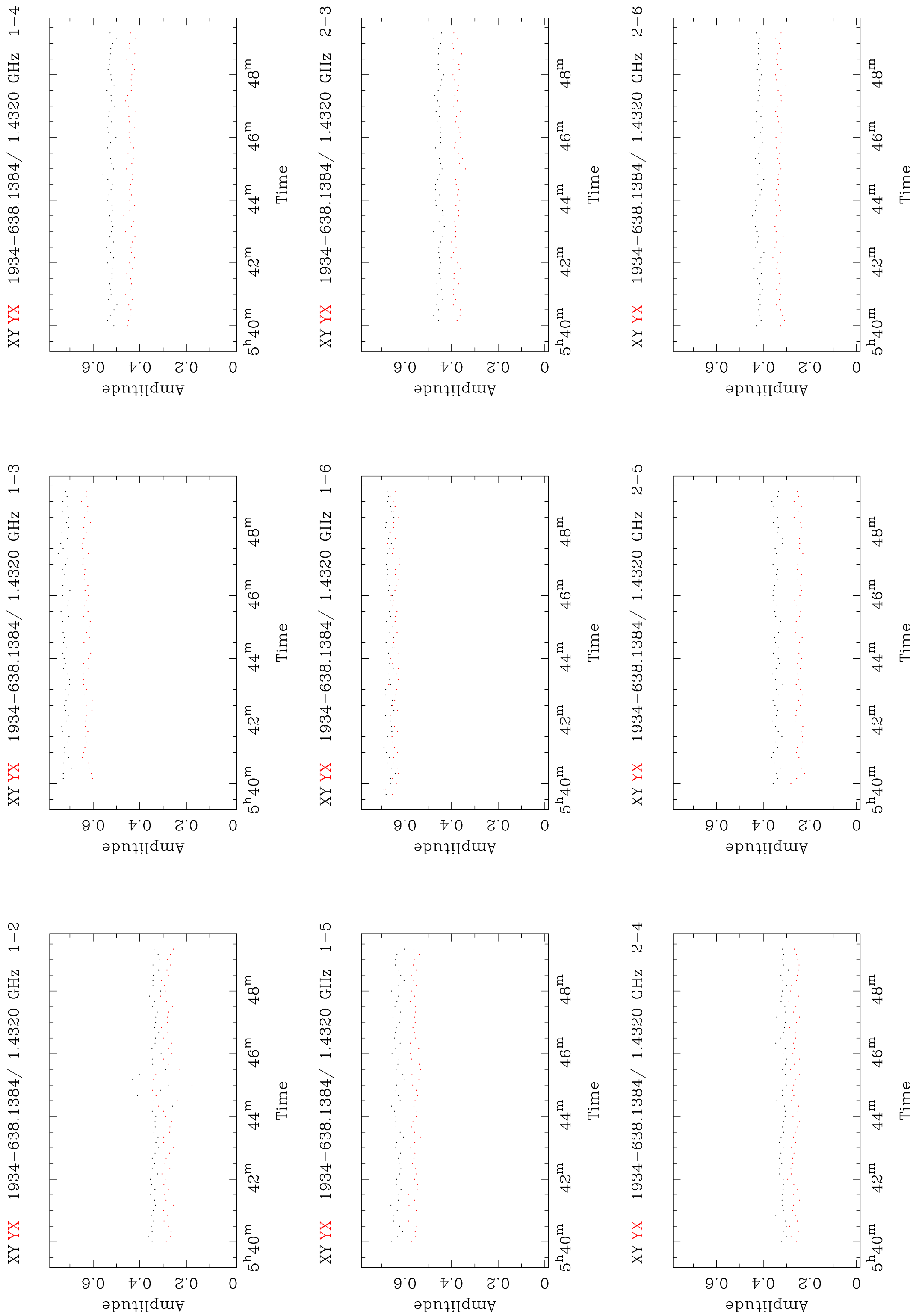

Figure 5: UVPLT of the 1934–638 xy and yx, showing significant polarization leakages.

```
Task:   gpcal
vis      = 1934-638s.1384/
interval = 0.1
```

GPCAL does some internal data-checking. It can issue a copious number of warnings when it discards visibilities, plus the usual ones about visibilities being flagged.

Again, check that the leakage corrections which GPCAL derived are reasonable by running UVPLT again with `stokes=xy,yx` – see Figure 6. The amplitudes should be close to zero on all baselines.

3.5 Flux density calibration

The absolute flux density scale for an observation is determined from an observation of a calibrator of known flux density – the primary calibrator. At the ATCA, 1934–638 is almost always used as the primary calibrator.

Both MFCAL and GPCAL know the flux of 1934–638, and will scale the gains accordingly. Thus, you get the absolute flux density calibration automatically in Miriad.

3.6 The secondary calibrator

Usually 1934–638 is only observed for the first 5–10 minutes of an experiment. Thus the time-dependent gains determined from 1934–638, although essential for setting the absolute flux density, are useless for determining the gain variations during a 12 hour synthesis observation.

To overcome this, most experiments make use of a "secondary calibrator", typically chosen from a catalogue of strong, compact sources, which is within a few degrees of the target[6].

The secondary calibrator for this example is 0252–712. Again, first look at the data with UVPLT, and flag out any bad visibilities. There were some really bad points on baseline 4–5 – again, presumably interference – and some funny recurring errors most noticeable on baseline 3–6. After a few runs of BLFLAG, my data-set looked like Figure 7.

Most of the visibility amplitude plots are fairly constant with time, the exceptions being baselines 1–2 and 4–5. It is very common for secondary calibrators to show extended structure on short baselines, when observed at 20 cm. GPCAL determines the gains assuming the calibrator is unresolved, and baselines which show structure will cause the derived gains to be in error.

What to do? Well, if there are only two baselines which show structure, just leave them out of the solution. Because the gains are *antenna-based*, leaving out two baselines will not degrade the solution very much, provided there are still baselines to every antenna[7].

So calibrating the secondary proceeds as follows: first, copy the bandpass, polarization leakages, and the absolute flux scale from 1934–638 to 0252-712:

[6] "target" is geek-speak for "the region of the sky you're actually interested in"

[7] If the calibrator is resolved/confused on a significant number of baselines, or all baselines to one antenna, then you may have to make a model for the source. That is beyond the scope of the current notes!

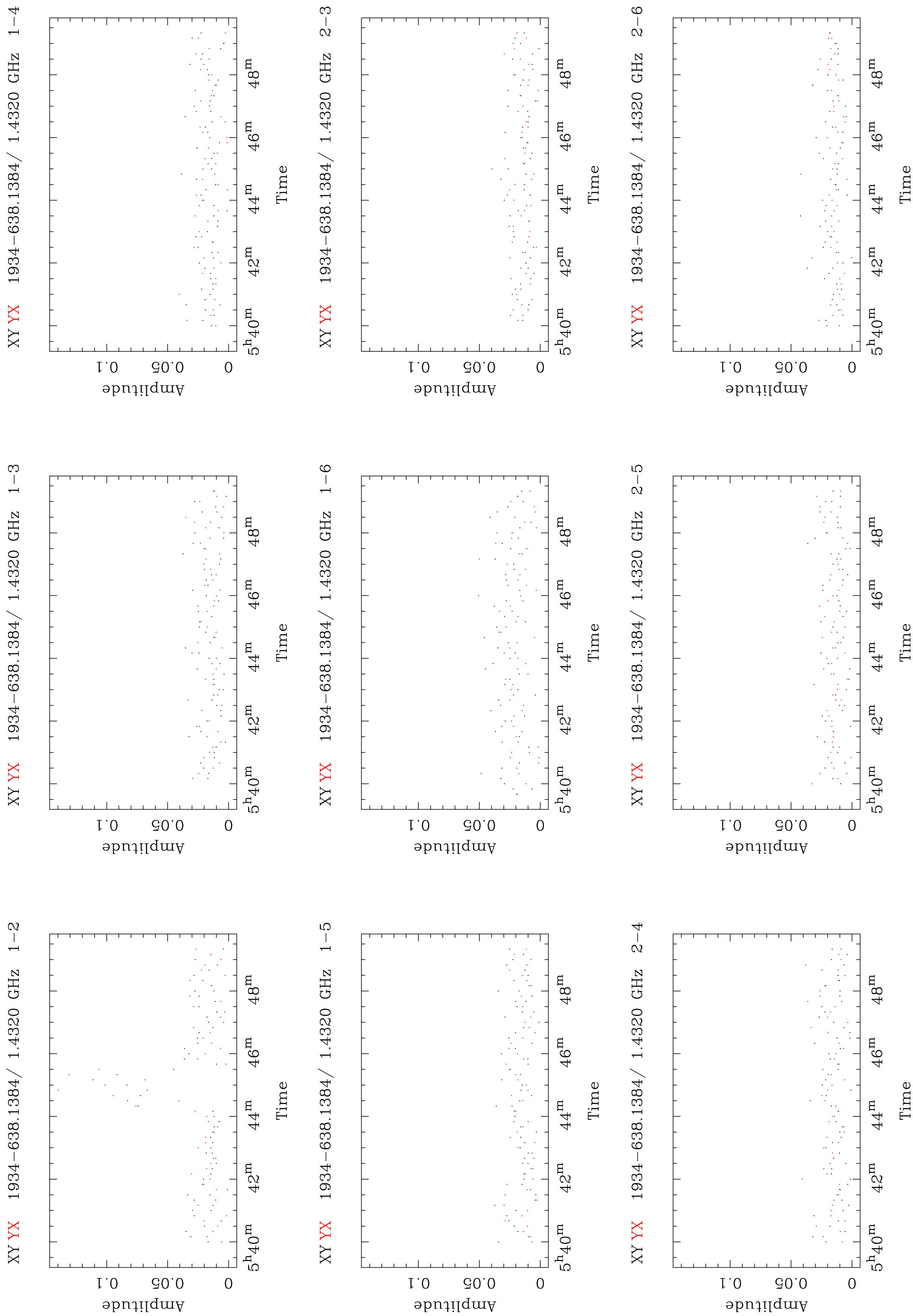

Figure 6: UVPLT of the 1934–638 xy and yx, after running GPCAL

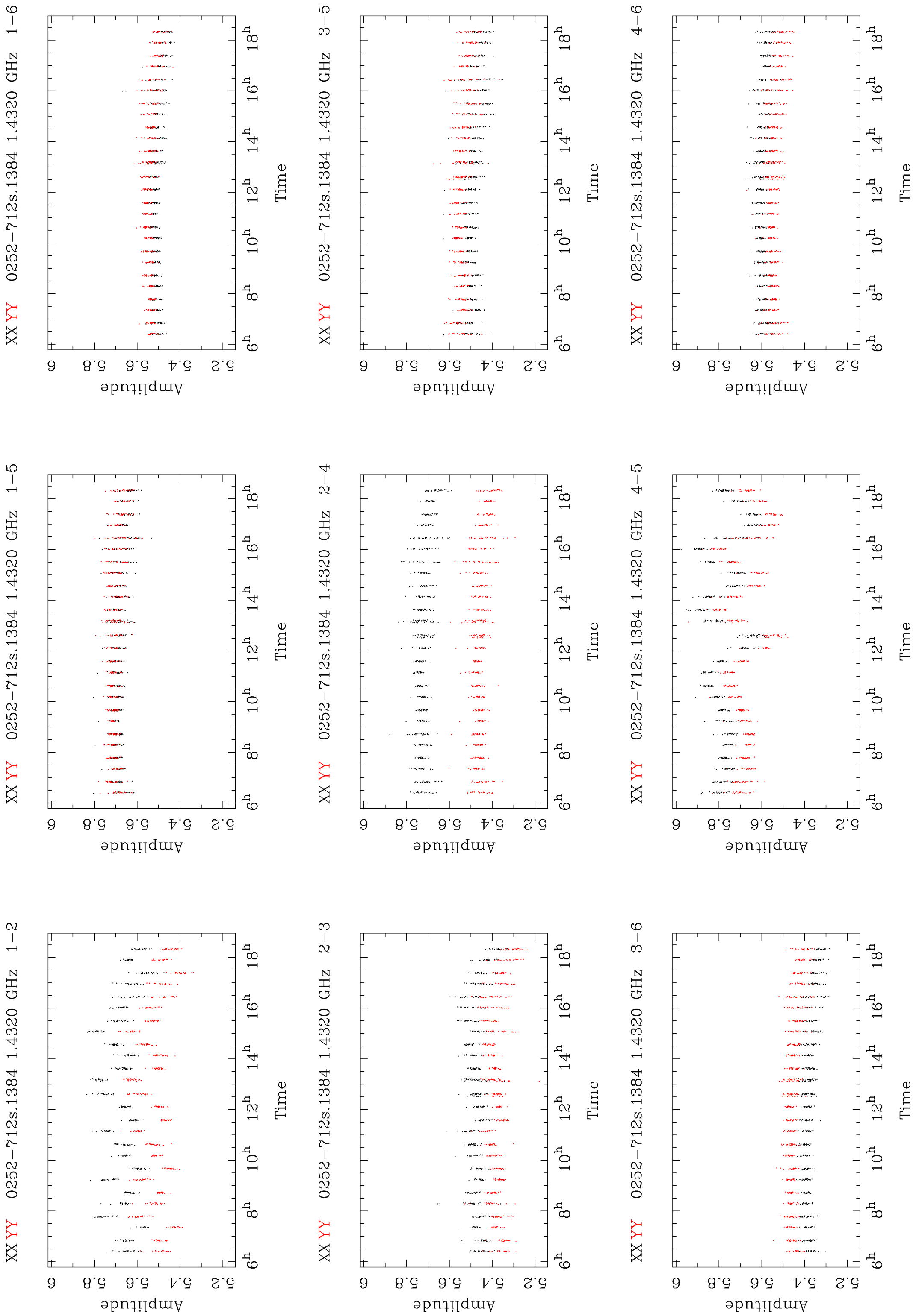

Figure 7: 0252–712 visibilities, after running BLFLAG

```
Task:   gpcopy
vis      = 1934-638.1384
out      = 0252-712.1384
```

Then determine time-variable gains using the observation of the secondary calibrator:

```
Task:   gpcal
vis      = 0252-712.1384
select   = -ant(1)(2),-ant(4)(5)
interval = 0.1
options  = noxy,nopol,qusolve
```

The `select=-ant(1)(2),-ant(4)(5)` excludes the two baselines which we found showed structure. When run with `options=qusolve`, GPCAL attempts to solve for the polarization of the calibrator. Only use this option if you have observations of the secondary calibrator spread over ~ 8 hours or more. For most secondary calibrators, the `Percent Q` and `Percent U` reported by GPCAL will be $\lesssim 5\%$.

Look at the data again, to see how the calibration went (see Figure 8 and 9):

```
Task:   uvplt
vis      = 0252-712.1384
stokes   = i
select   = -ant(1)(2),-ant(4)(5)
axis     = time,amp
device   = /xs
nxy      = 3,3
size     = 2,2
```

Now I've got the polarization leakages worked out, it makes sense to plot Stokes I, and I'll usually plot Stokes parameters rather than feed-based corellations from now on.

I've solved one gain solution for every 10 second integration. Miriad's gain interpolation routines are pretty simple, however, and only use the nearest two gain solutions. This means that, even if you spend several minutes on each calibrator scan, when you interpolate the gain solution across the target visibilities, only the first and last 10 seconds would be used! To overcome this, average the gain solutions derived from the secondary calibrator[8]:

```
Task:   gpaver
vis      = 0252-712.1384
interval = 10
options  = scalar
```

[8]There is some debate about whether you should use vector or scalar averaging in GPAVER. My experience suggsets that you should *always* use scalar averaging. If the atmospheric phase is poor, vector averaging will give you rubbish, whereas scalar averaging will at least give you the gain amplitudes. If the phase stability is good, both vector and scalar averaging will give almost identical results.

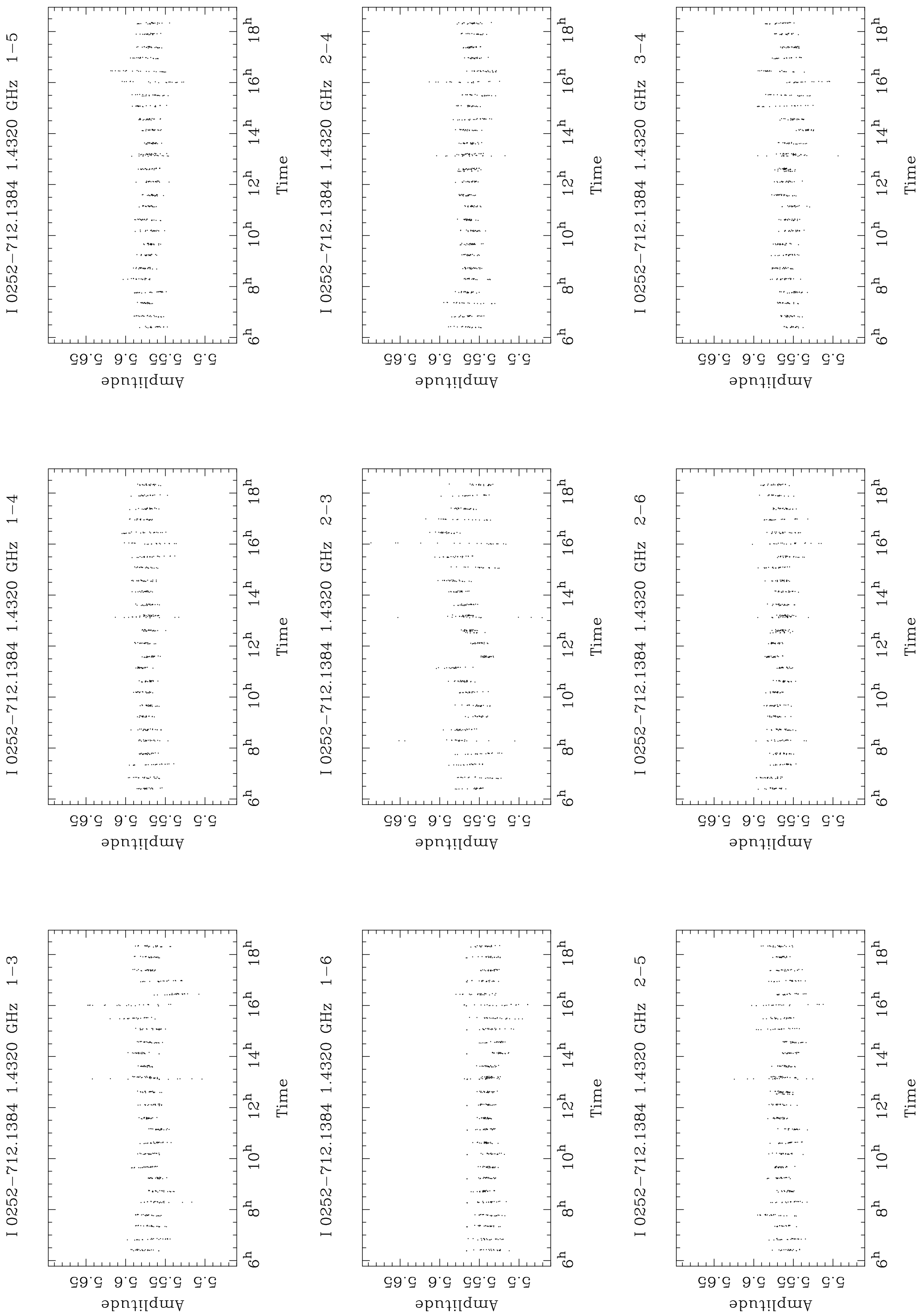

Figure 8: 0252–712 visibility amplitudes, after running GPCAL

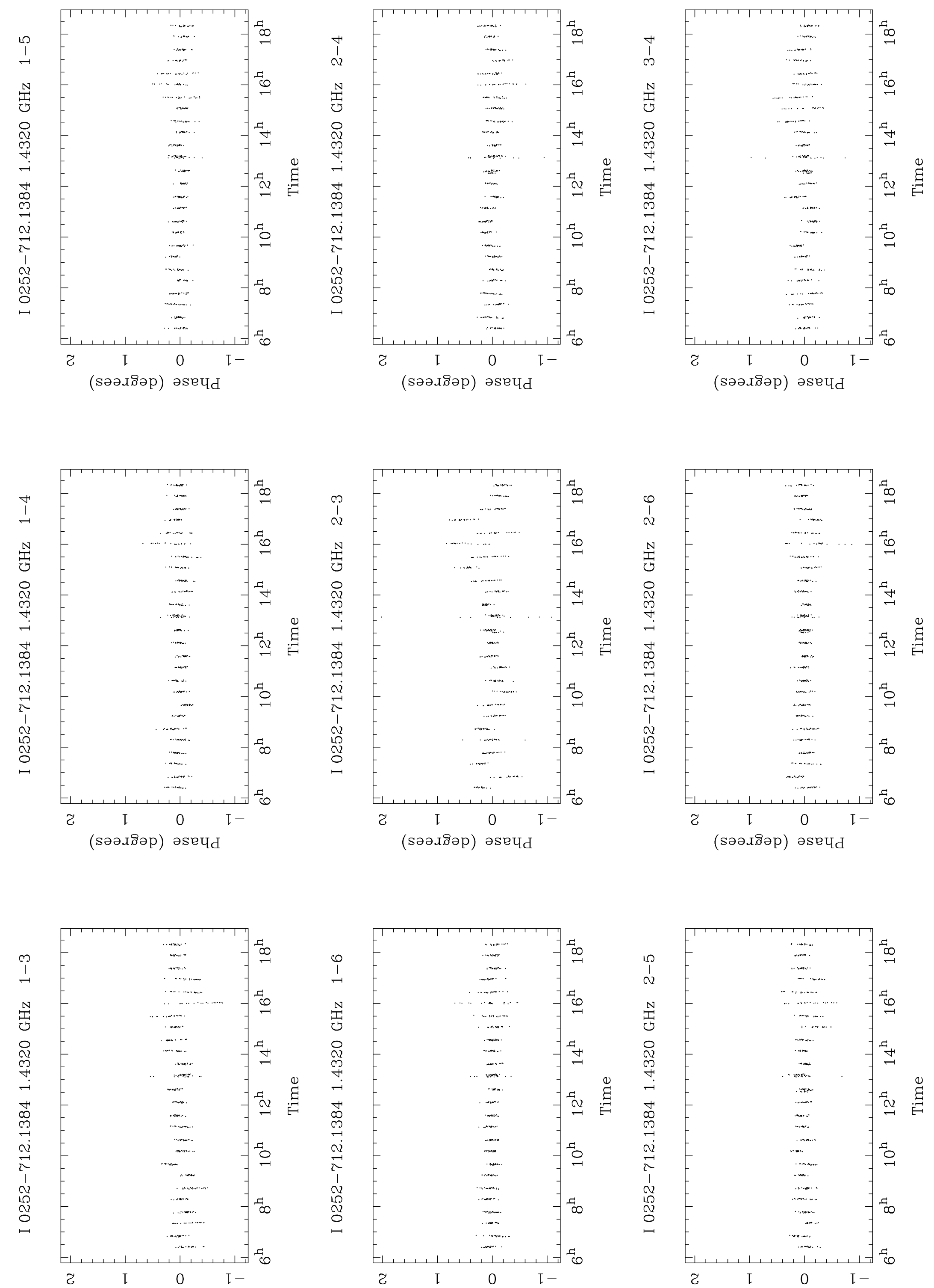

Figure 9: 0252–712 visibility phases, after running GPCAL

After running GPAVER, the secondary calibrator visibilities will have a lot more scatter, especially the phase. This isn't important; the aim is to get a good gain solution for the target, not to accurately image the secondary calibrator.

3.7 0252–549s – the target at last!

As I hope you can guess by now, the first thing you should do is to look at the data with UVPLT. This one is a little more exciting (see Figure 10).

```
Task:   uvplt
vis      = 0252-549s.1384
stokes   = xx,yy
axis     = time,amp
device   = /xs
nxy      = 5,3
size     = 1.5
```

There are some corrupt visibilities on baseline 1–2 and 4–5; you should flag at least the spike on baseline 4–5 using BLFLAG.

Now copy the gains, leakage terms and bandpass from the secondary calibrator to the target:

```
Task:   gpcopy
vis      = 0252-712.1384
out      = 0252-549s.1384
```

and you're ready to Fourier transform the visibilities and make an image!

Note: Appendix B discusses bandwidth decorrelation in the visibility domain. It certainly isn't necessary to read this, but it examines Miriad's handling of visibility data in a little more depth, and illustrates the value of multi-frequency synthesis, which is used in the next section.

3.8 Imaging

To make a "dirty-image", the visibilities have to be interpolated onto a rectangular grid, weighted, and then Fourier transformed. The Miriad task which performs both these functions is INVERT:

```
Task:   invert
vis      = 0252-549s.1384
map      = 0252-549s.imap
beam     = 0252-549s.beam
robust   = -2
stokes   = i
options  = mfs,double
```

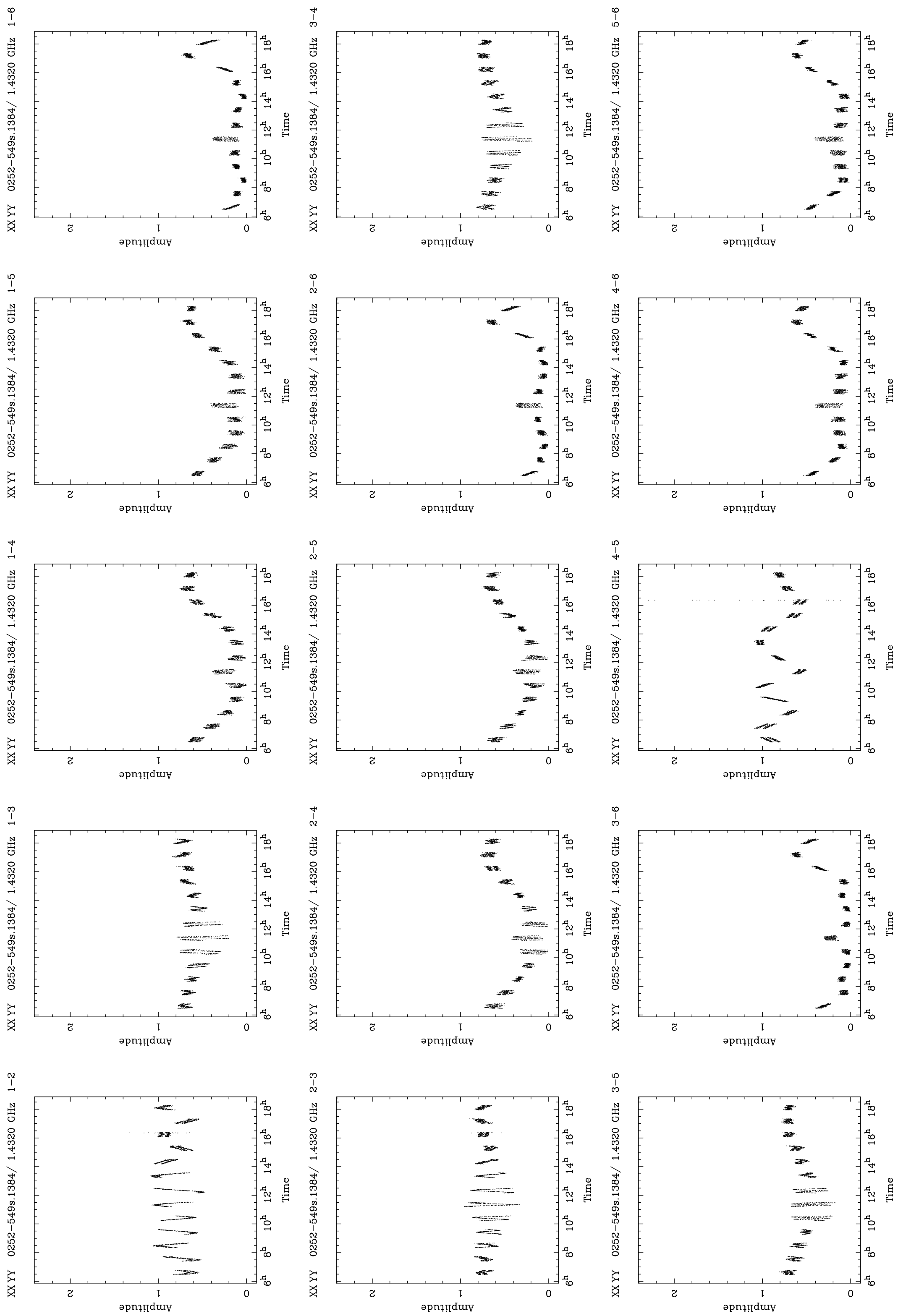

Figure 10: Plot of the gravitational lens candidate 0252–549 visibility amplitudes.

The `options=mfs` invokes multi-frequency synthesis, forcing all channels to be gridded onto the correction position in the uv-plane. The `robust=-2` uses *uniform weighting*, which sacrifices sensitivity for improved synthesized beam shape. See Lecture 7 in Taylor, Carilli and Perley.

To view the dirty-image, either load it into kview, or run CGDISP (see Figure 11):

```
Task:   cgdisp
in       = 0252-549s.imap
type     = p
range    = -0.1,0.1
device   = /xs
labtyp   = arcs
options  = wedge
csize    = 1.5
```

Figure 11 shows the image to be dominated by two bright sources. It's a pretty awful image; the sidelobe response from the bright sources limits the sensitivity across the whole image. We'll have to deconvolve.

3.9 Deconvolution

Deconvolution algorithms attempt to remove the side-lobes around strong sources, by filling in the uv-plane. See Chapter 8 of Taylor, Carilli and Perley for more information. For images like Figure 11, it is the deconvolution step which will make the biggest difference to the image sensitivity.

The safest way to deconvolve is to use CLEAN. It isn't a very good idea to just set CLEAN loose on an image without constraints – before running CLEAN, you should specify which regions you think contain emission. This is most easily done by creating a region-definition file with CGCURS.

```
Task:   cgcurs
in       = 0252-549s.imap
type     = p
range    = -0.1,0.1
device   = 1/xs
labtyp   = arcs,arcs
options  = region
```

When you run CGCURS, instructions appear in the Miriad window. Basically, use the left mouse button to start defining a region, and the right mouse button to finish. An example session is shown in Figure 12. CGCURS creates or appends the file `cgcurs.region`. You probably want to copy this file to another name lest you overwrite it later by accident.

Then run CLEAN, specifying the region to use:

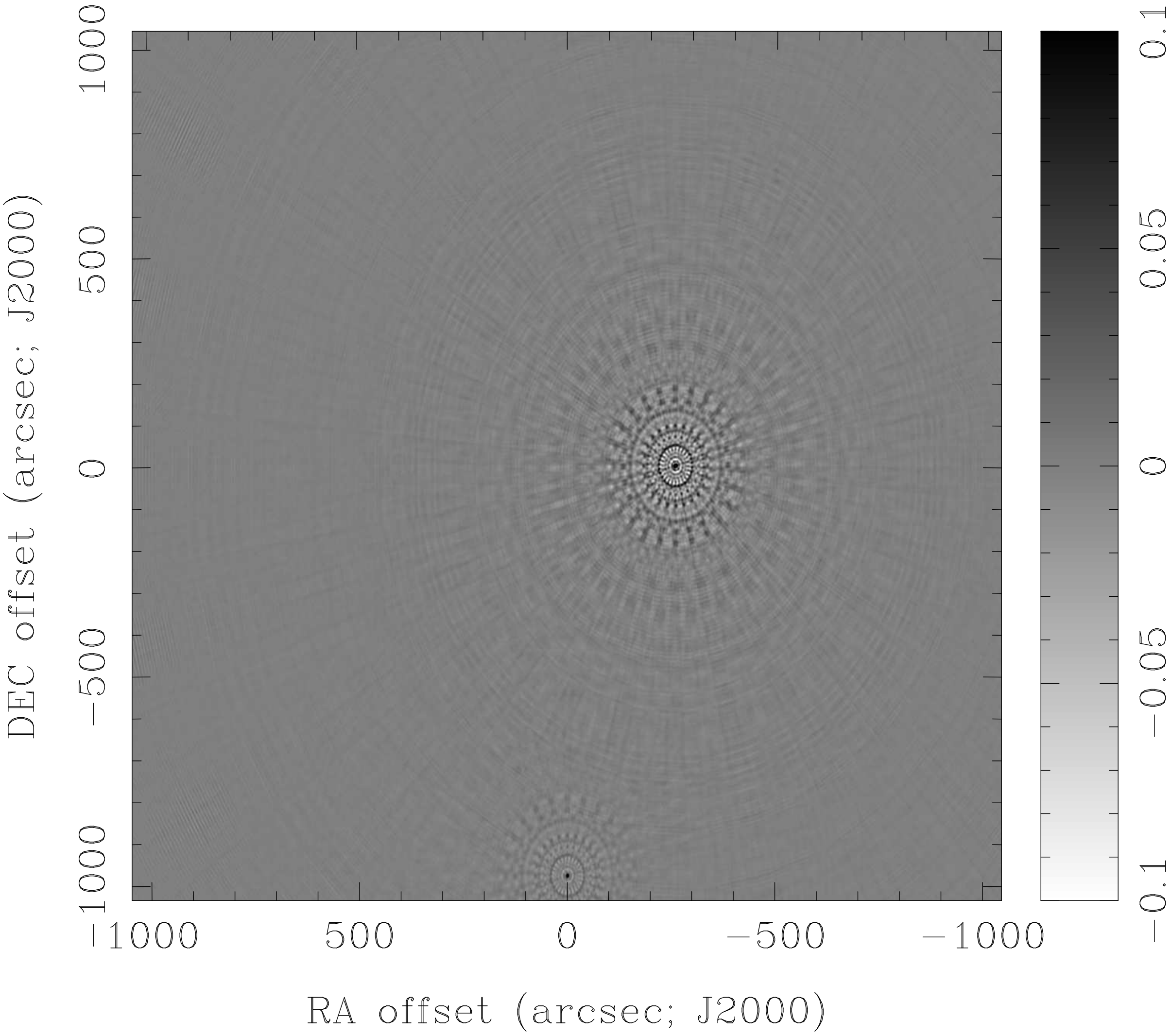

Figure 11: The 0252-549 “dirty image”.

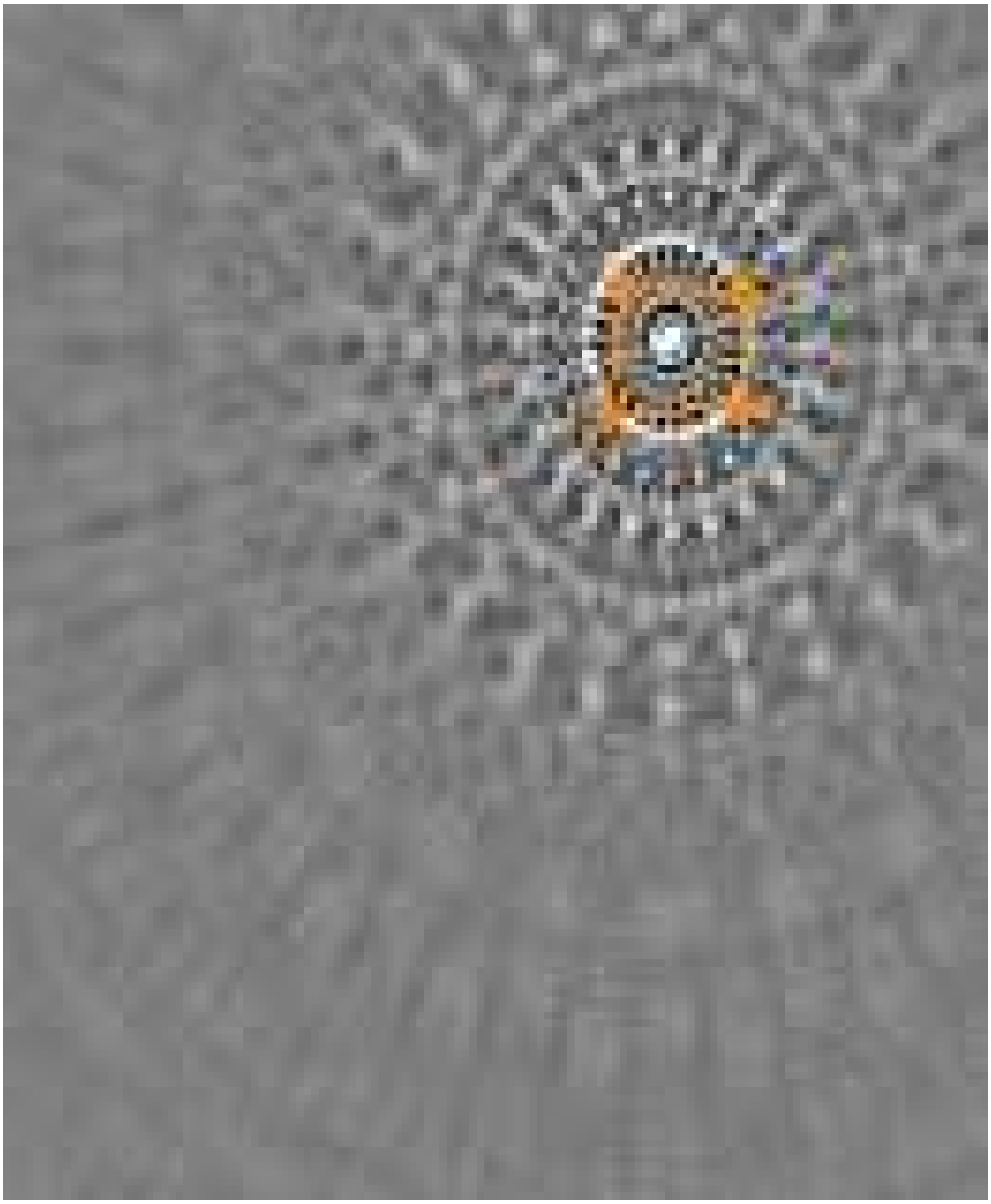

Figure 12: Defining the emission regions using CGCURS.

```
Task:   clean
map      = 0252-549s.imap
beam     = 0252-549s.beam
out      = 0252-549s.icmp
niters   = 30
region   = @cgcurs.region
```

CLEAN's `out` data-set will be a clean component model.

How do you know how many iterations of to use? I start off by giving a conservative number of iterations, usually `niters=20` or `30`, and then I look to see how much side-lobe structure is still there. If CLEAN has removed all the side-lobes, *or* if the remaining structure doesn't look like side-lobes anymore, then it's time to stop cleaning.

To look at what's left after you've cleaned[9], you subtract the CLEAN component model output from the original dirty image. Of course, you have to convolve it with the dirty-beam first. Both these steps are performed by RESTOR:

```
Task:   restor
model    = 0252-549s.icmp
beam     = 0252-549s.beam
map      = 0252-549s.imap
mode     = residual
out      = 0252-549s.ires
```

You can view the residual data-set using CGDISP or kview. The residuals after 30 iterations, for the area around the strong North-East source, are are shown in Figure 13. Clearly there is still emission left to clean; the source was about 600 mJy in the dirty image, and there is about 40 mJy left of this in the residuals. There are also features which are *not* side-lobes; the extension to the south-west, and the linear negative feature running diagonally across the image.

When non-sidelobes start to dominate the residuals, it is time to stop cleaning. At this stage, we are *not* trying to get the best possible image; what we are after is a reliable clean-component model, which can be used for self-calibration (as discussed in the next section). If we clean too deeply here, we would end up with a clean-component model which includes the negative feature – this is definitely not desirable!

To work out which features are sidelobes and which are not, it may help to look at the beam (as output from INVERT) with CGDISP or kview; for compact structures, side-lobes in the residual image will have the same form as the beam.

3.10 Self-calibration

Self-calibration algorithms attempt to improve the visibility calibration by using a model for the target itself to determine the calibration. For more information, see Chapter 10 of Taylor, Carilli and Perley.

[9] the "residuals"

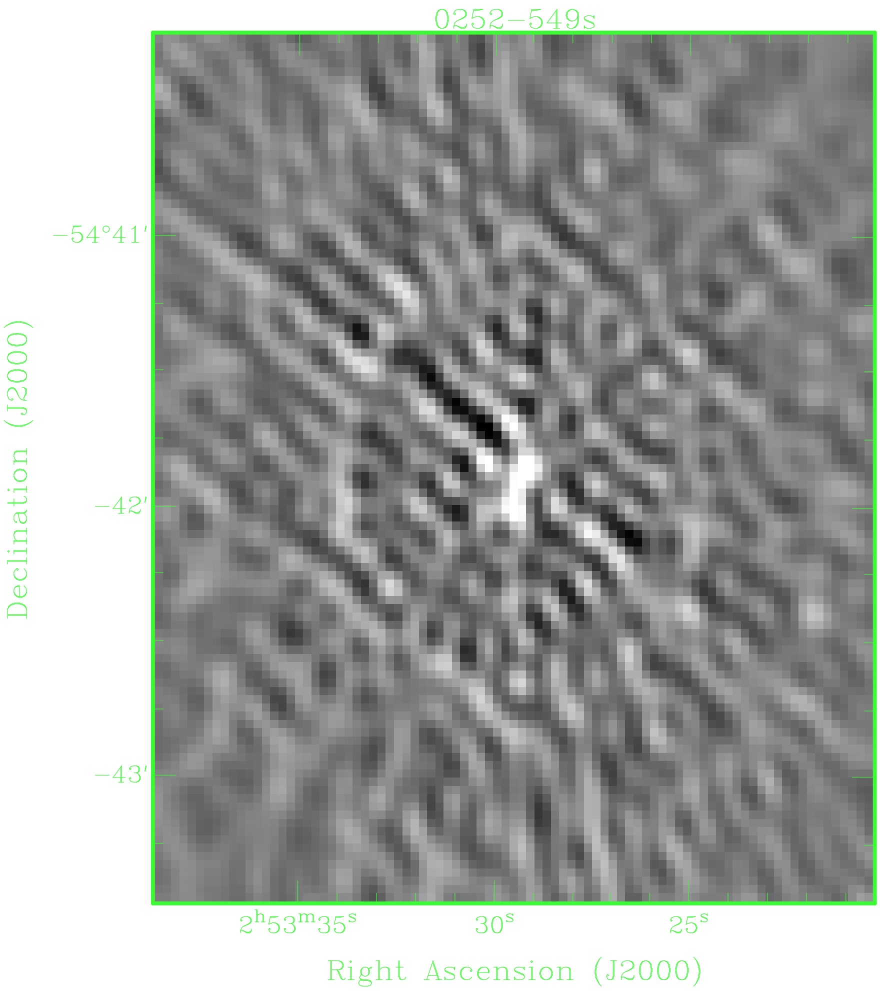

Figure 13: Kview plot of the residuals after 30 iterations, for the area around the strong North-East source. The grey-scale is from -35 mJy to +35 mJy.

We've already seen several types of images for the visibility data-set – dirty images, clean-component images, residual images. When to comes to selecting an image for publication or analysis, the acceptable image to use is the clean-component image, convolved with a Gaussian to remove high-spatial frequencies, and with the residual image added in as well. This combination is often called the "restored" image. To produce this image, use RESTOR again, but this time with `options=clean`:

```
Task:   restor
model    = 0252-549s.icmp
beam     = 0252-549s.beam
map      = 0252-549s.imap
mode     = clean
out      = 0252-549s.icln
```

You can view this image with CGDISP or kview (see Figure 14). Although it's an improvement on the dirty-image, the dynamic range is still very low. At this stage, we should try self-calibration.

There is one very important rule which you should observe if you are thinking about self-calibration: do *not* self-calibrate if there are no strong sources in the image! Self-calibrating background noise can actually create very realistic looking sources! If you can see the source structure in the visibilities, then self-calibration is fine. If not, try time-averaging (`average=2`, say) in UVPLT. If using an averaging interval reveals source structure, then you can self-calibrate with the same averaging interval. If you still can't see anything, even when you average with an interval equal to time between secondary calibrator observations, then forget it.

The source structure in 0252–549 is clearly visible in UVPLT, so self-calibration is fine. Before self-calibration, you should *apply* the existing gain table, leakages and bandpass. What this means is that, rather than just storing the gains, leakages and bandpass as tables, you actually use these to modify the visibility data, and create a new data-set. The reason for this is that Miriad data-sets only have one gain table, and if you determine a new one with SELFCAL, you would over-write any existing gain table. This is done by a fudge; run UVAVER with none of the averaging options set:

```
Task:   uvaver
vis      = 0252-549s.1384
out      = 0252-549_a.1384
```

Then run SELFCAL:

```
Task:   selfcal
vis      = 0252-549_a.1384
model    = 0252-549s.icmp
interval = 0.1
options  = mfs,phase
```

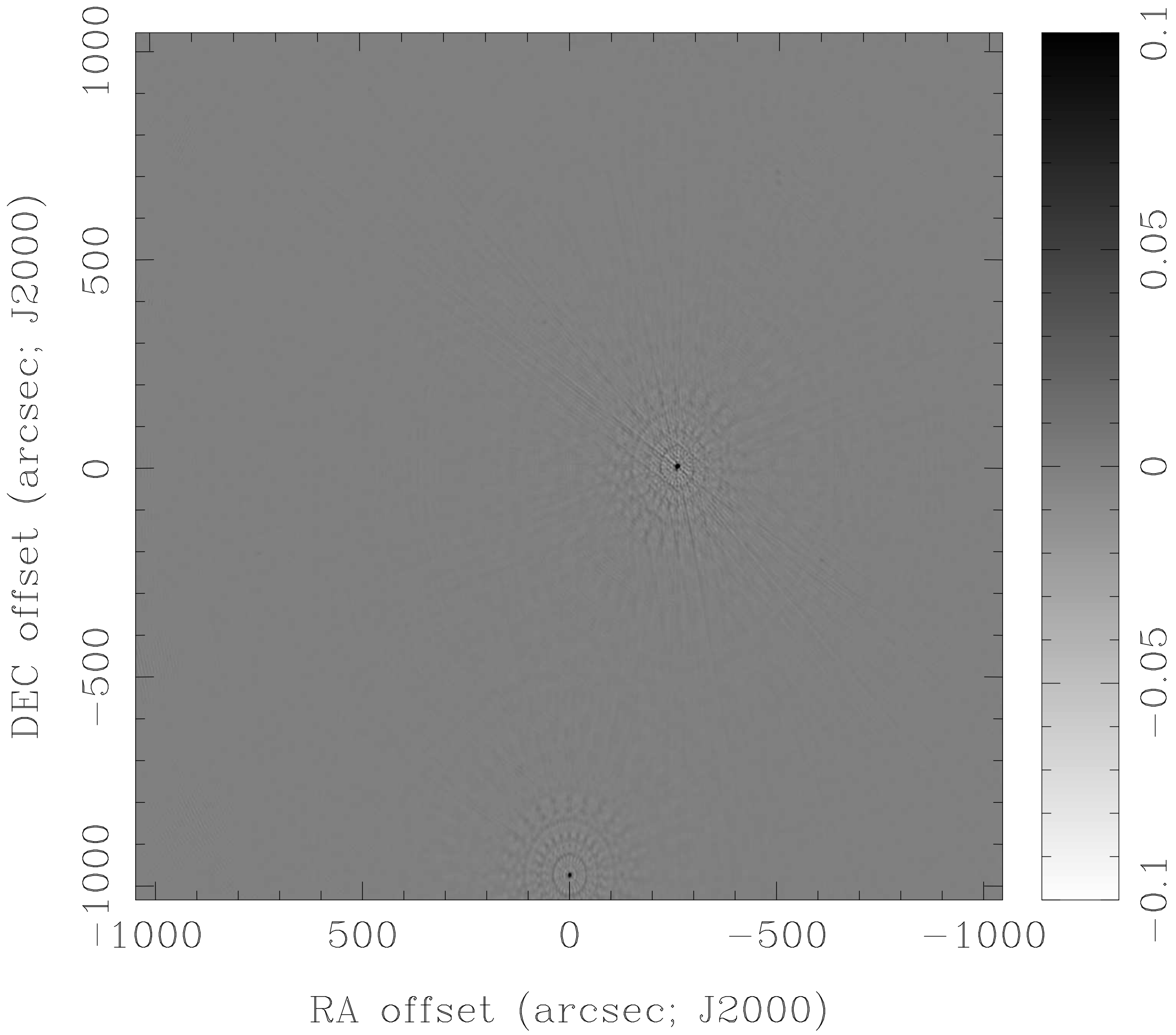

Figure 14: Target image after deconvolution with CLEAN. The greyscale is the same range as Figure 11.

Selfcal allows you to selected whether to self-calibrate only visibility phases, or both phases and amplitudes. The usual practice is to self-cal phase-only first, then re-image and then self-cal both amplitudes and phases.

So, now with the hopefully improved gains, we have to go through the whole imaging process again…

```
Task:   invert
vis      = 0252-549_a.1384
map      = 0252-549_a.imap
beam     = 0252-549_a.beam
robust   = -2
stokes   = i
options  = mfs,double

Task:   clean
map      = 0252-549_a.imap
beam     = 0252-549_a.beam
out      = 0252-549_a.icmp
niters   = 30
region   = @cgcurs.region

Task:   restor
model    = 0252-549_a.icmp
beam     = 0252-549_a.beam
map      = 0252-549_a.imap
mode     = residuals
out      = 0252-549_a.ires
```

Finally arriving at the residuals again (Figure 15). Contrast this to Figure 13; after self-calibration, the residuals look like side-lobe structure again. Good! That means we can keep cleaning.

I repeated the last clean and restor steps, but used `niters=60` in clean. Now you should be able to identify other weak sources in the residual image.

But after 60 iterations, the residuals around the north-western source are starting to be dominated by the extensions to the south-west and north-east (see Figure 16). So now it's time to start try amplitude self-calibration. We use the most recent clean-component model.

```
Task:   selfcal
vis      = 0252-549_a.1384
model    = 0252-549_a.icmp
interval = 0.1
options  = mfs,amp

Task:   invert
```

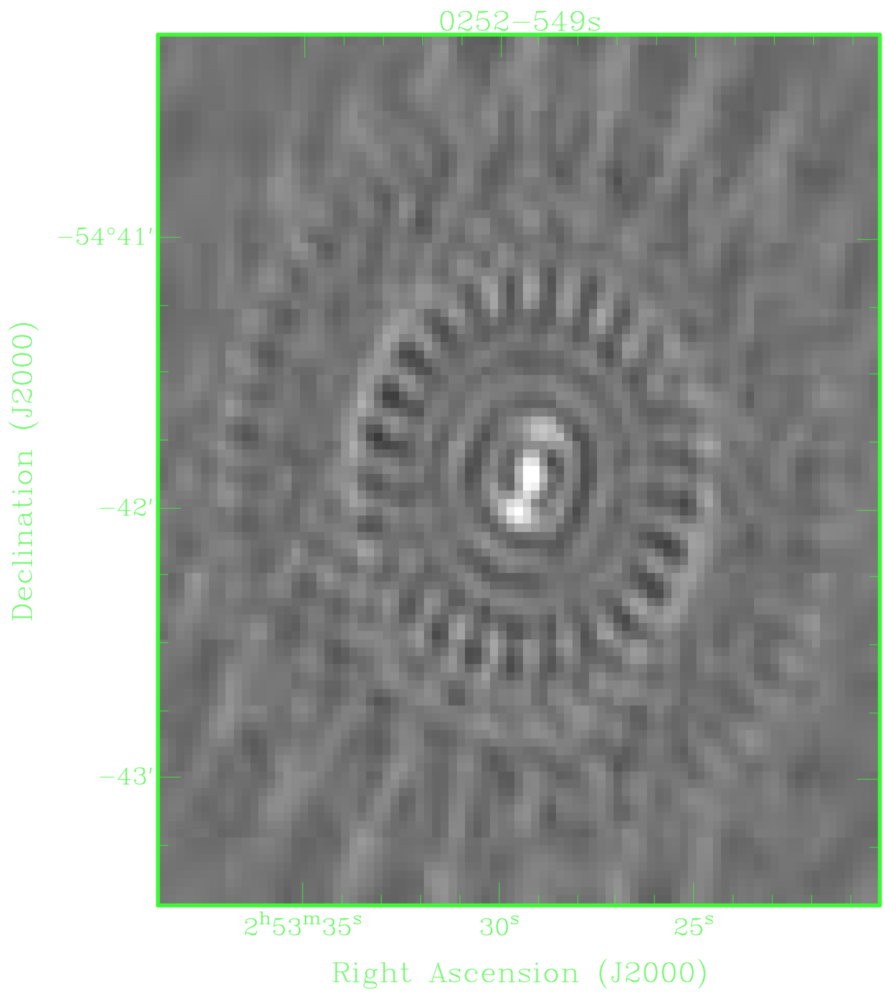

Figure 15: Kview plot of the residuals after phase-only self-calibration, and 30 iterations of CLEAN. The grey-scale is from -30 mJy to +35 mJy. Compare this with 13

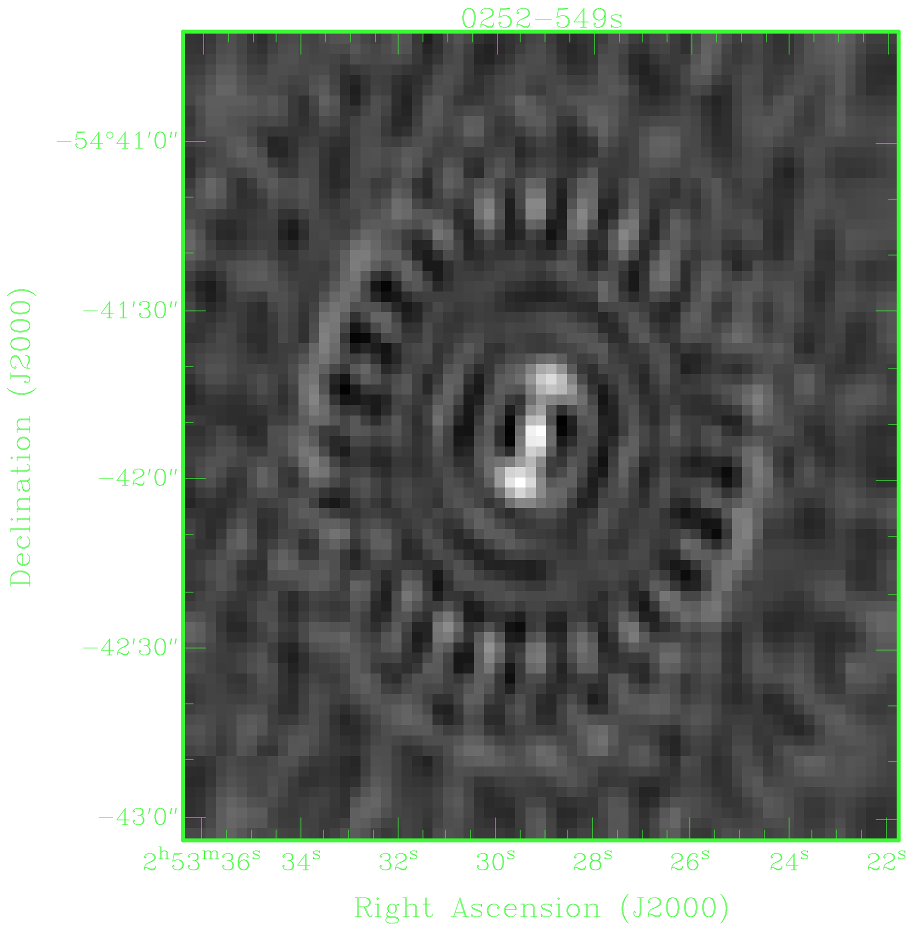

Figure 16: The residuals after phase-only self-calibration, and 60 iterations of CLEAN. The grey-scale is from -10 mJy to +30 mJy.

```
vis      = 0252-549_a.1384
map      = 0252-549_a.imap
beam     = 0252-549_a.beam
robust   = -2
stokes   = i
options  = mfs,double

Task:   clean
map      = 0252-549_a.imap
beam     = 0252-549_a.beam
out      = 0252-549_a.icmp
niters   = 60
region   = @cgcurs.region

Task:   restor
model    = 0252-549_a.icmp
beam     = 0252-549_a.beam
map      = 0252-549_a.imap
mode     = residuals
out      = 0252-549_a.ires
```

And we're back at the residuals again! (see Figure 17). Once again, the self-calibration has removed the residuals which did not have the form of side-lobes, and we can continue to clean.

Now that you can see other sources in the residual image (Figure 18), you can use CGCURS again to select extra regions to clean. And then with the new clean-component model, do an amplitude SELFCAL. For observations where there is good uv-coverage, and significant extended emission, you can go around this INVERT, CLEAN, SELFCAL loop several times. You're probably beginning to see why people use scripts to run Miriad…

3.10.1 When to stop?

Sooner or later, however, you will reach the limit of this technique. Either you will have run out of sources to clean, or (much more likely) you'll get to the point where the residuals around strong sources don't look like side-lobes, and they don't look like real source structure. With a single ATCA observation, it is very difficult to obtain an on-source dynamic range of more than about 100. On-source dynamic range is the peak flux density of a source, divided by the *rms* of the remaining crap in the image *in the region around the source.*

At this point, the best thing to do is to ignore the strong sources, and concentrate on the weaker sources for which there are still uncleaned sidelobes. In fact, it often helps to include a slightly larger region around strong sources; the clean-components there will be rubbish, so the on-source dynamic range will not improve but it might help to increase the sesitivity away from the strong source.

When cleaning and self-calirbation no longer make any improvement to the residual images, create a final restored image (Figure 19).

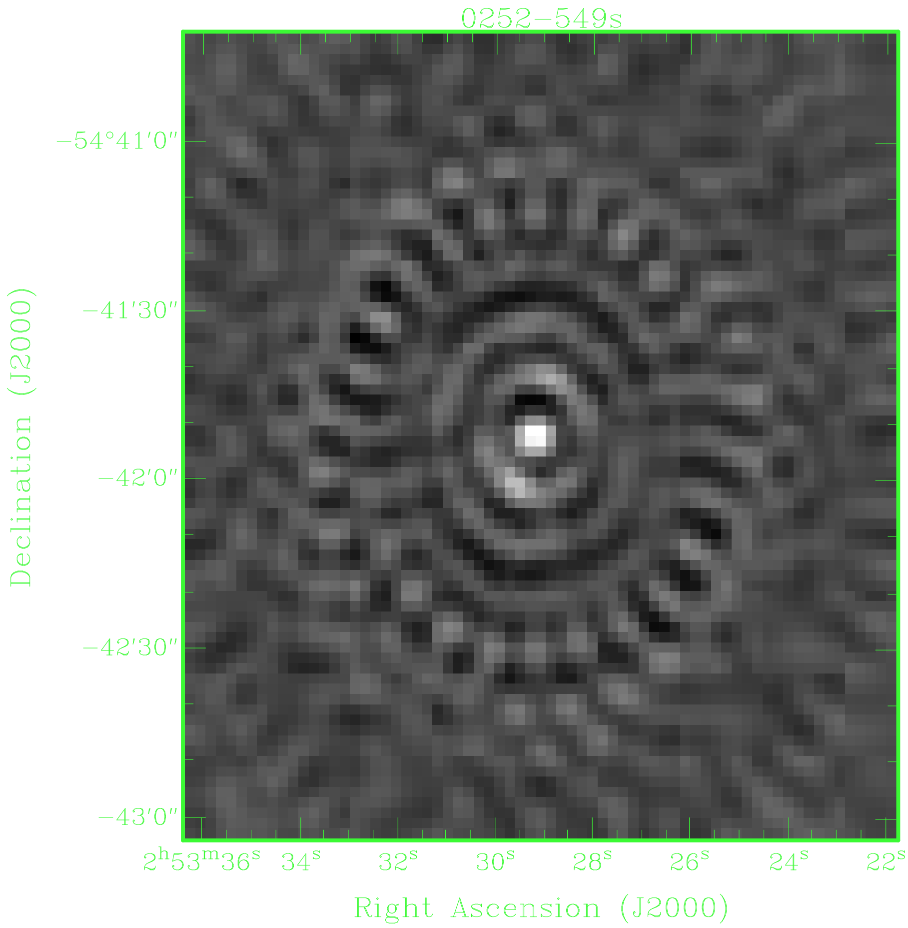

Figure 17: The residuals after phase and amplitude self-calibration, and 60 iterations of CLEAN. The grey-scale is from -10 mJy to +25 mJy.

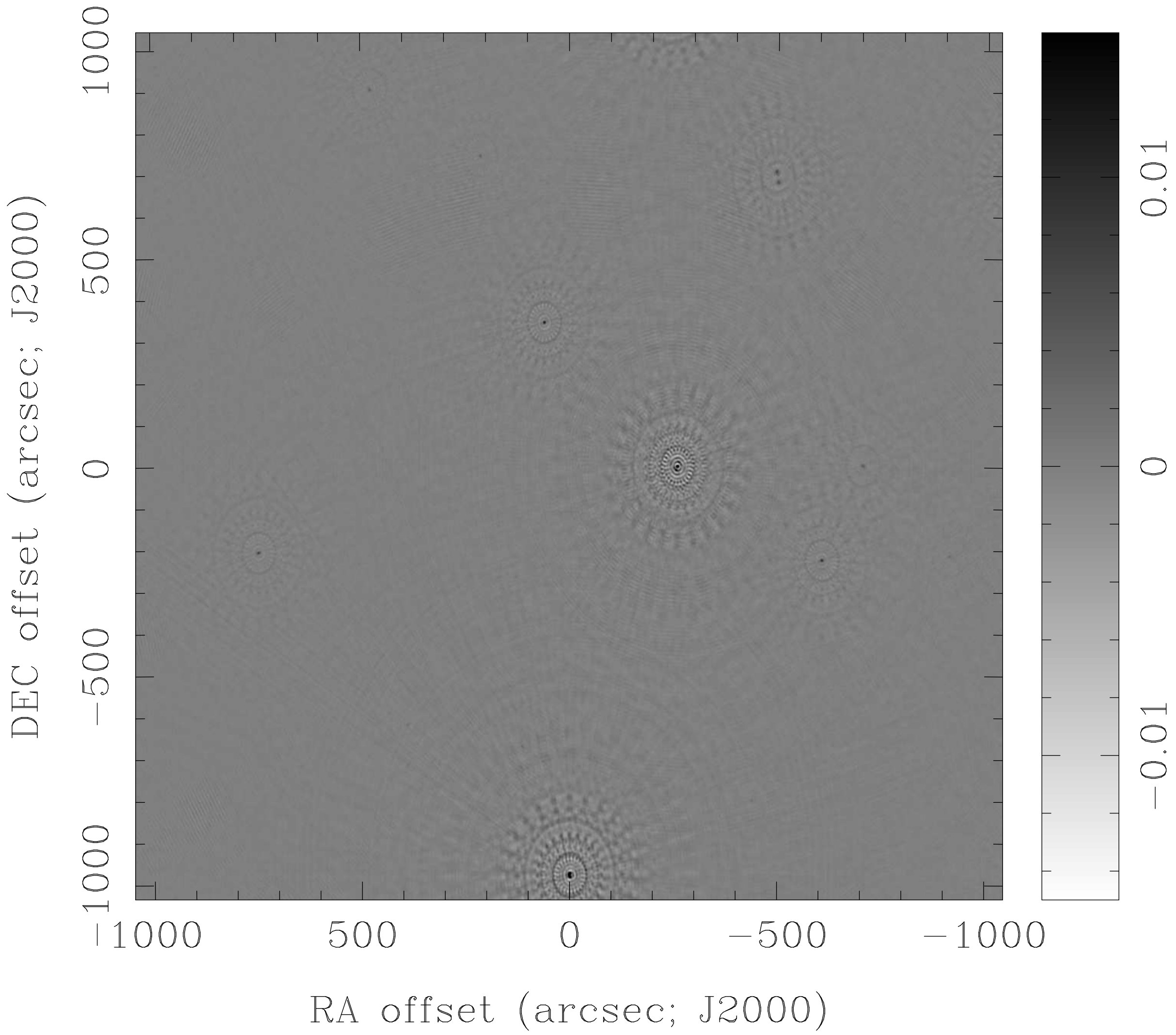

Figure 18: The residual image of the entire field after 60 iterations of clean. Several background sources are now visible.

```
Task:   restor
model    = 0252-549_a.icmp
beam     = 0252-549_a.beam
map      = 0252-549_a.imap
mode     = clean
out      = 0252-549_a.icln
```

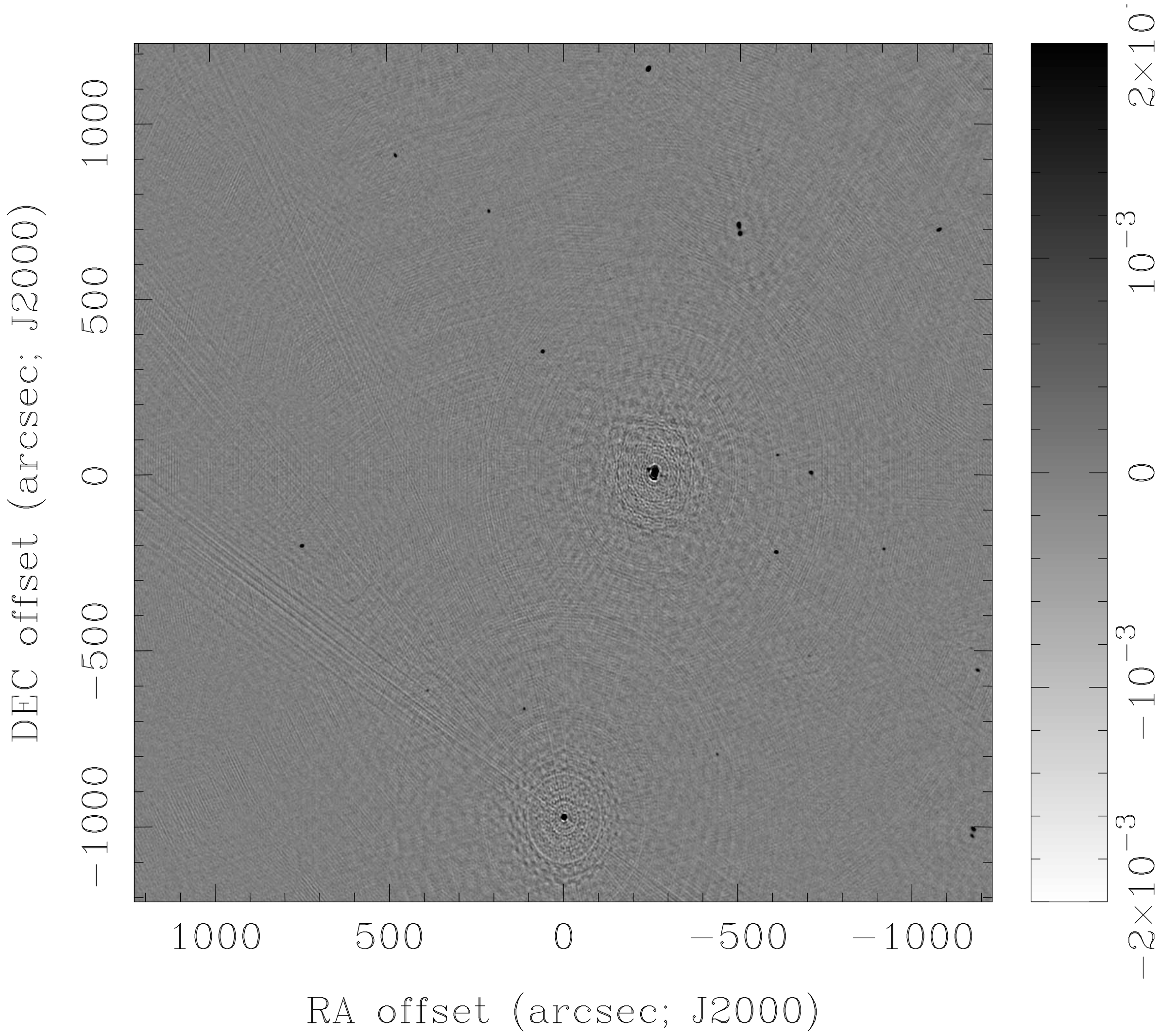

Figure 19: The final restored image of 0252–549.

4 Summary

Making an image is only half the fun, of course. Once you have a decent image, there are a number of analysis tools available to help you support your latest astrophysical theory. Some things you might want to do:

- IMSTAT returns statistics over a region in an image.
- IMFIT allows you to fit model components to an image.
- IMSAD finds sources in your image for you.
- CGDISP can make several types of plots, including contour plots, and image overlays.

A Invoking Miriad programs from the unix shell

Miriad programs are invoked using the syntax:

```
taskname firstkey=firstargument secondkey=secondargument ...
```

Often, the unix shell will recognize and try to interpret special characters eg. $*$ or (). To protect these, you should enclose the key=value pairs in quotes, like:

```
maths 'in=<vmap>+(0.00029*<imap>)' 'out=vmap_corrected'
```

B Bandwidth decorrelation

This Section discusses bandwidth decorrelation in the visibility domain. First, take a long, thin PGPLOT window, and try the following (two baselines from which are shown in Figure 20):

```
Task:   uvplt
vis      = 0252-549s.1384
stokes   = i
axis     = time,amp
device   = 1/xs
nxy      = 1,2
size     = 1.5
```

Now, recall the discussion about bandpass earlier; the ATCA "continuum mode" correlator configuration produces 33 partially-overlapping channels covering a total of 128 MHz bandwidth, and ATLOD consolidates these into 13 contiguous 8 MHz channels. By default, UVPLT averages the data for all channels together, and produces one "dot" for the average.

You can specify a sub-range of channel in most Miriad tasks using the `line` keyword. See `help line` for more information, but the following will plot *only* the data for channel 7, which is in the middle of the band (two baselines are shown in Figure 21).

```
Task:   uvplt
vis      = 0252-549s.1384
line     = chan,1,7,1,1
```

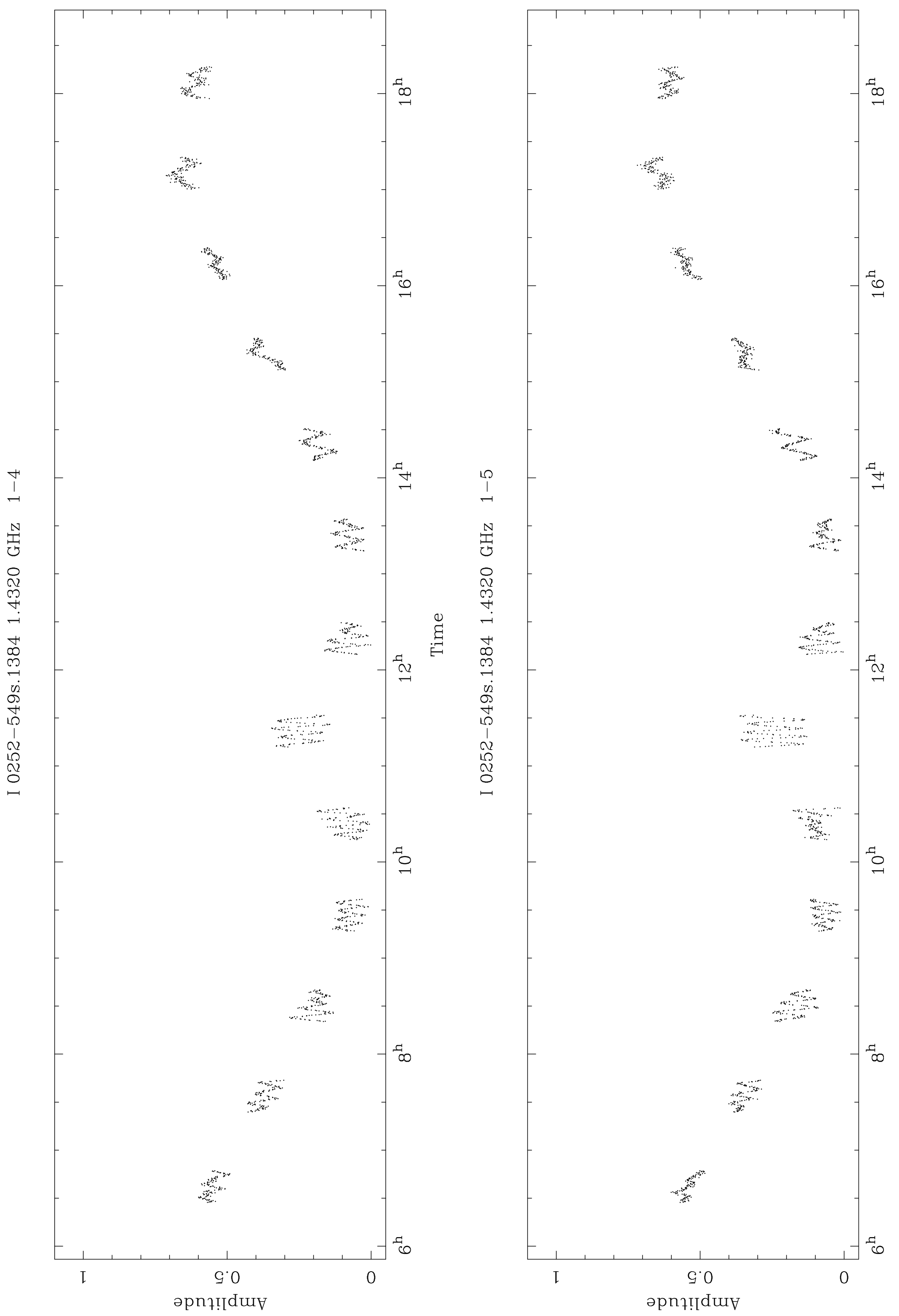

Figure 20: Plot of 0252–549 visibilities, with all channels averaged together.

```
stokes   = i
axis     = time,amp
device   = 2/xs
nxy      = 1,2
size     = 1.5
```

Comparing the two, you'll see that the amplitude of the visibilities for the full bandpass is often significantly lower than that for a single channel! What is going on?!

This is an example of *bandwidth decorrelation*, in the visibility domain. As you can see from Figure 21, the visibilities change rapidly with baseline length (in both amplitude and phase). Thus, if you were to average the visibilities from several different baselines together, they would decorrelate. But this is exactly what UVPLT is doing when you ask it to average across the whole bandpass; the baseline length is measured in wavelegths, and so the different channels all have slightly different baselines. You can see this effect by using UVPLT to plot the uv-coverage. First, plot the uv-coverage with all channels averaged together (the default) (Figure 22):

```
Task:   uvplt
vis      = 0252-549s.1384
stokes   = i
axis     = uc,vc
options  = nobase,equal
device   = /xs
nxy      = 1,1
size     = 1.5
```

Then try the `nofqav` option, which will plot the uv position of each channel of each visibility separately[10] (Figure 23):

```
Task:   uvplt
vis      = 0252-549s.1384
stokes   = i
axis     = uc,vc
options  = nobase,equal,nofqav
device   = /xs
nxy      = 1,1
size     = 1.5
```

You can see that the spread of frequencies in the bandpass makes a significant difference to the baseline lengths.

Another interesting visibility plot is phase vs. time, for the first and last channel in the bandpass:

[10]and consequently produce a plot with a very large number of points

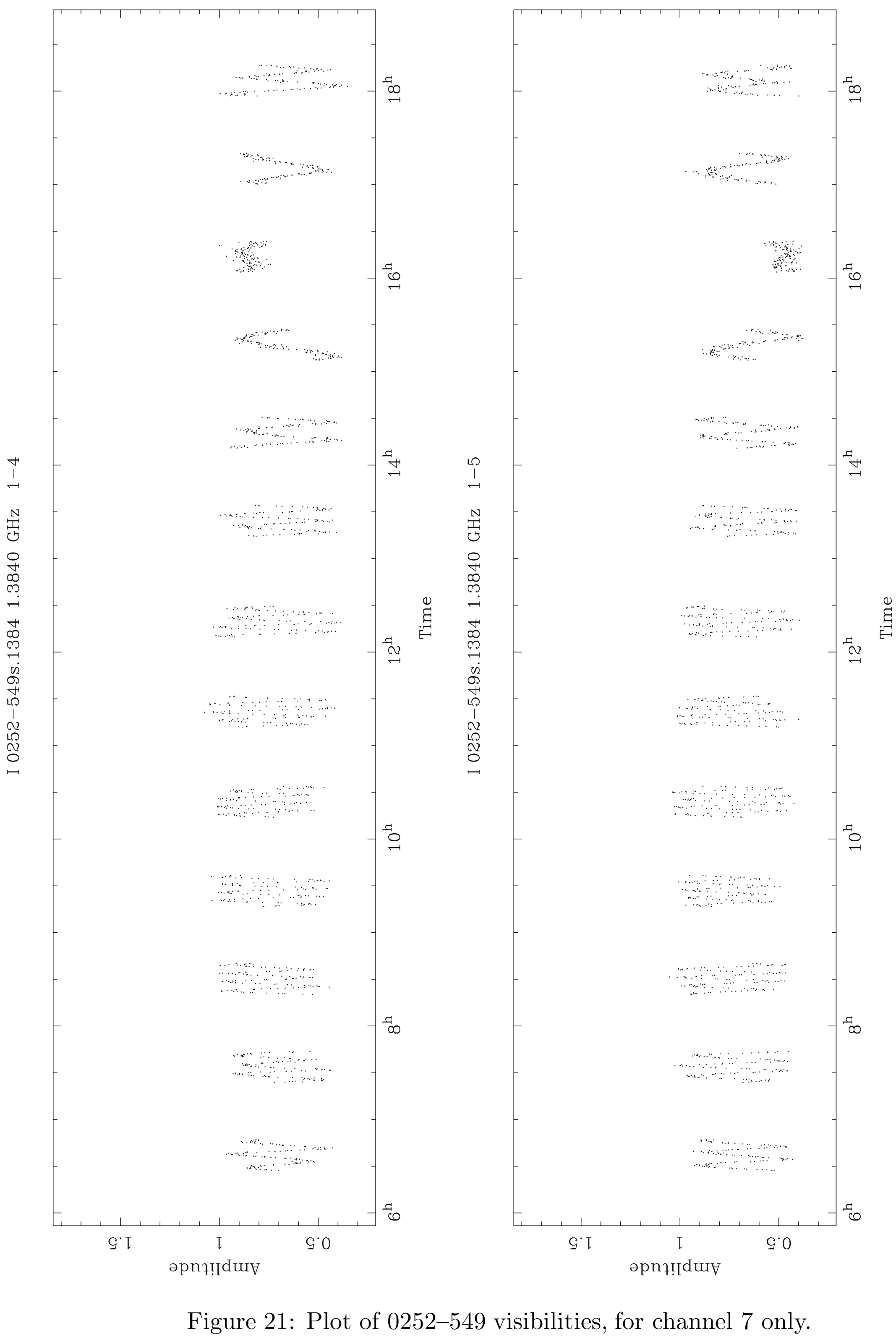

Figure 21: Plot of 0252–549 visibilities, for channel 7 only.

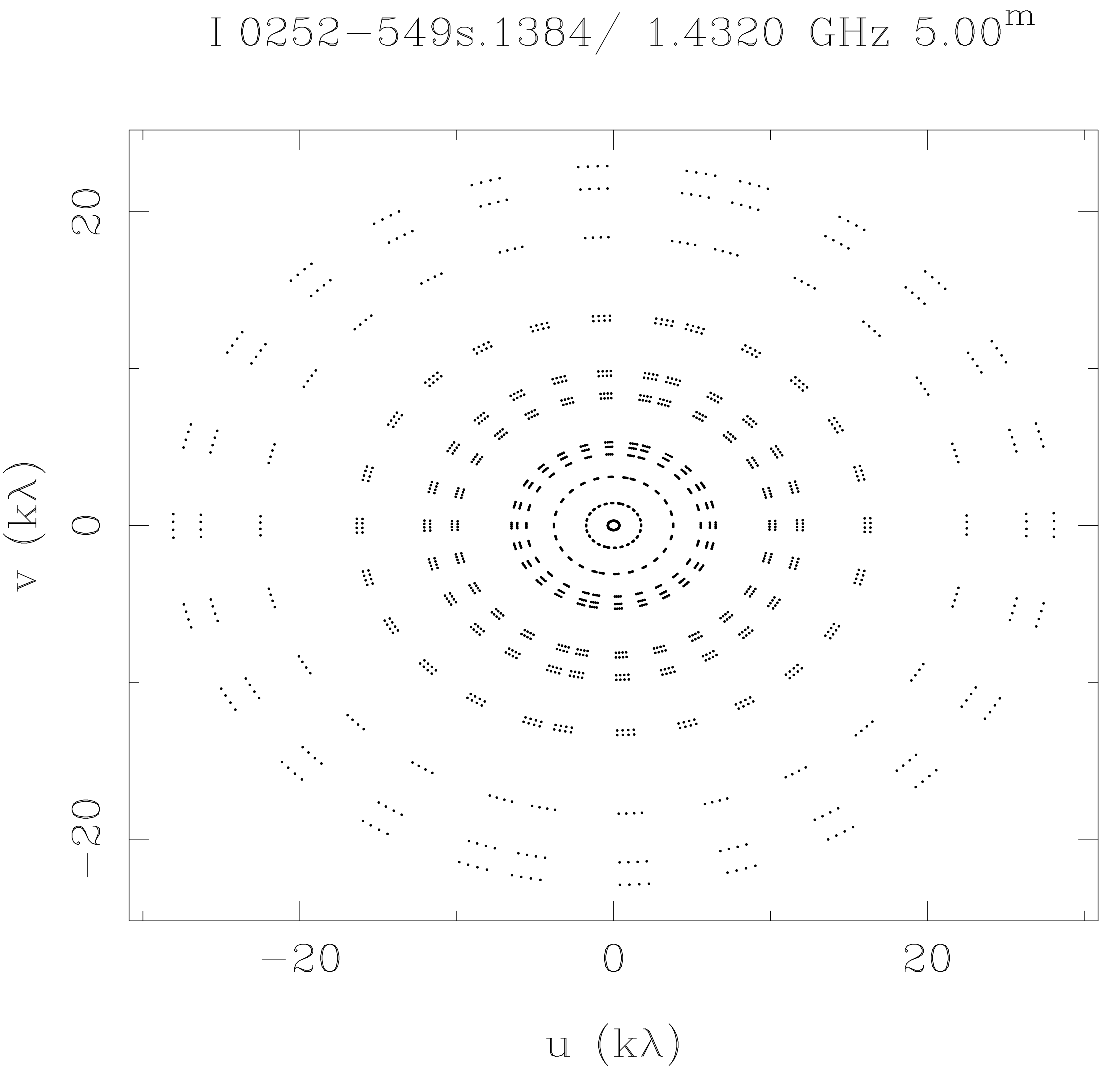

Figure 22: Plot of 0252–549 uv-coverage, all channels averaged.

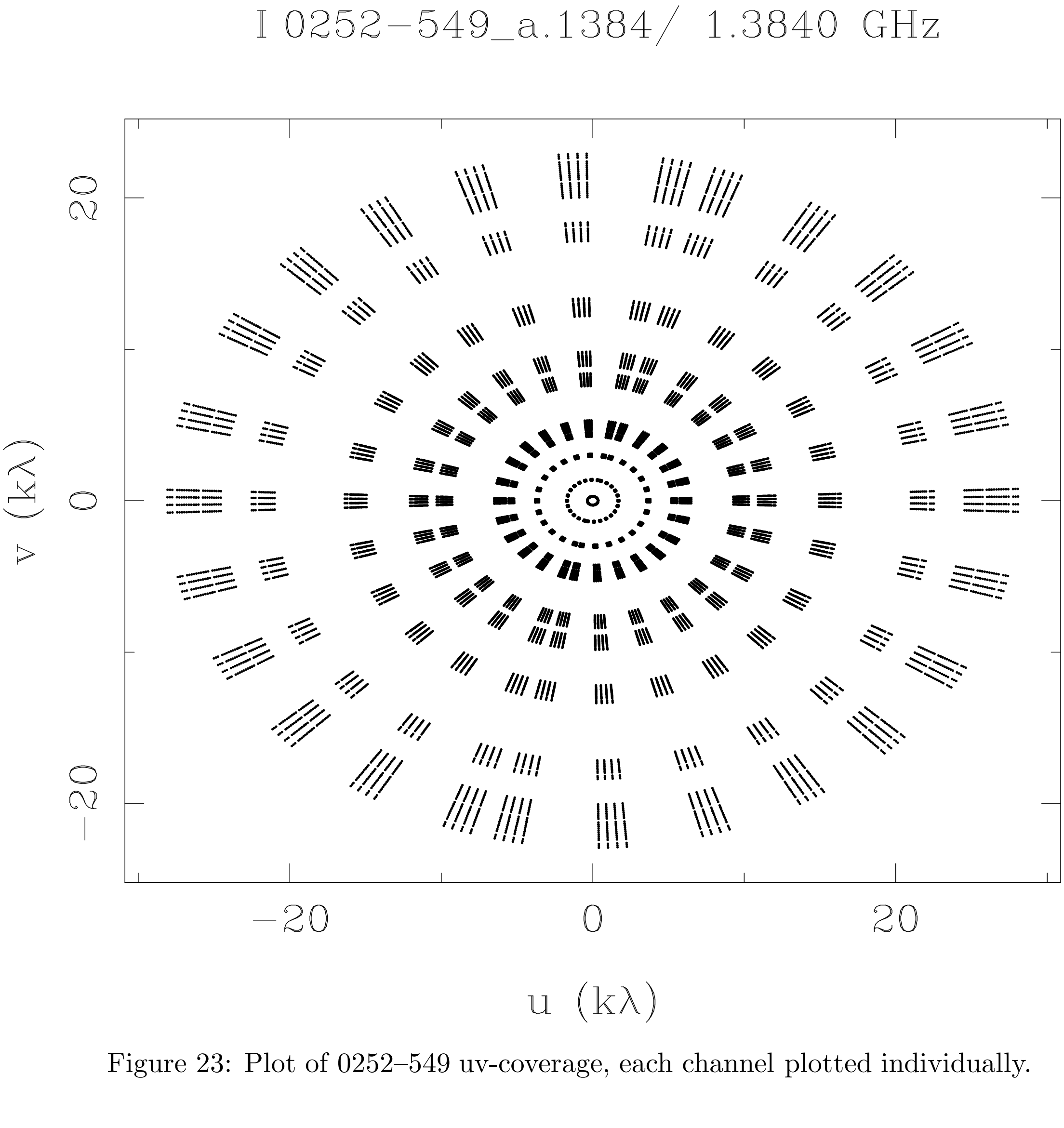

Figure 23: Plot of 0252–549 uv-coverage, each channel plotted individually.

```
Task:   uvplt
vis      = 0252-549s.1384
line     = chan,2,1,1,12
stokes   = i
axis     = time,phase
device   = 2/xs
nxy      = 1,2
size     = 1.5
```

This shows you why decorellation occurs; the phases vary significantly across the band.