A Cookbook for Reducing BIMA Data using MIRIAD

Michael W. Regan

May 12, 1995

Contents

1. Introduction

The purpose of this document is to take someone who is familiar with interferometry through the steps required to make a map of BIMA data. You are assumed to also have a basic understanding of the MIRIAD package and to have it already installed at your site. For an introduction to MIRIAD you can read the MIRIAD User's Guide.

The format of this document is based on the MIRIAD shell, although this is just for clarity and is probably not the interface you ultimately will use to calibrate your data. Users who are well-versed in Miriad usually make a script that contains all the steps used in their calibration. This lets you recreate the process quickly if needed and maintain documentation of how the calibration was performed. An example of a data reduction script is shown in the last section of this document.

Throughout this document a consistent file naming convention will be used. Table 1 explains the meaning of the various file names.

Filename	Definition
source	source observation
phasecal	Phase calibrator
ampcal	Amplitude calibrator
pbcal	Passband calibrator
X.ll	The line length corrected version of file X
X.nw	A version of file X that has had the wideband recalculated
X.aver	A version of file X that has been averaged in time
cal.gvis	The output of `gmake`, a visibility gain dataset
cal.gfit	The output of `gfiddle`, a polynomial approximation of the gains
X.pcal	The output of `passmake`
X.cal	Calibrated version of file X
X.pb.cal	Calibrated and passband corrected version of file X
X.cm	Clean channel map of object X
X.cl	Clean components of object X
X.mom	0th moment map (total intensity) of object X

2. Calibration

2.1. Correcting for variations in the line length

Before you can begin to reduce your data you must correct the phases of all of your datasets for variations in the "line length", which is the length of the electrical path between the receivers on the antennas and the correlator in the computer room. Under ideal conditions these variations are caused primarily by diurnal changes in the temperature of the cables between the receivers and the correlator. These temperature variations typically cause small phase drifts of not more than about 2π over a 10 hour track (except in A+ array). However, there has been another source of phase drifts with rates as high as 4π per hour that is probably caused by the electronics. This can only be removed by the line length correction, so it is critical that the line length correction be applied in this case.

During observations, a line length measurement is made for every integration. This ranges from once every 15 seconds (A+ array) to every 120 seconds (C array). Because of noise in the measurement process each individual measurement has an error of 5-7 degrees rms. This precludes the application of the correction online.

The current method for applying the correction converts the antenna-based phase measurements into baseline-based phase differences. Then `selfcal` can be used to determine the average phase offset to use for each antenna. This method is slightly confusing but required the least amount of new code to be written.

Step 1: Run `uvcal` with:

```
vis= [all your data files (source, phasecal, ampcal, pbcal)]
out= [any name]
options=phasem,nochannel
```

> The goal of the first step is to create a dataset with the baseline-based phase differences placed in wideband channel 4. The option phasem is the flag to do this calculation. The option nochannels is set to make the output dataset only contain the four wideband channels to save time and disk space.

Step 2: Run `selfcal` with:

```
vis= [output from Step 1]
interval=5
line=wide,1,4
```

> Step 2 calculates the antenna based phase drifts due to the line length variations. A five minute interval is the normal default since this gives a $\sqrt{5}$ decrease in the rms of the phase determination but still allows you to track fast changes if they occur. The line parameter tells `selfcal` to use the phase differences placed into

the dataset by uvcal. Selfcal creates a gain table in the input dataset which contains the calculated corrections for the line length variations.

Step 3: Run copyhd once for all of your data files with:

```
in=[output from Step 1]
out=[raw data file]
items=nsols,gains,ngains,interval,ntau,nfeeds
```

Step 3 copies the gain table from the line length file into each of the raw data files.

Step 4: Run uvcat once for all of your data files with:

```
vis=[raw data files]
out=datafile.ll
```

Step 4 creates a new version of the raw data files with the line length corrections applied. This is needed since there can only be one gain table in the file and we still need to correct for other phase variations and determine the gain of the antennas. The output data files are now ready for further calibration.

2.2. Removing bad data from your phase calibrator dataset

Before you can calibrate your data you need to remove any glitches or other bad data from your phase calibrator. Once the bad data have been flagged you need to create a new version of the dataset with the wideband channels recalculated from the individual narrowband channels excluding the channels that have been flagged as bad.

Step 1: Run uvflag on your line length corrected phase calibrator with:

```
vis=phasecal.ll
edge=2,2 (if old mode 4, use 2,2,2)
flagval=flag
```

Step 1 flags the edges of the corfs. These regions are, in general, bad since they are at the edges of the filters. Note that in old mode 4 you also need to flag the central channels since they are actually the ends of the two corfs that make up the 200 MHz corf.

Step 2: Run uvspec with:

```
vis=phasecal.ll
line=channel,512,1
axis=channel,amp
device=/xw
```

In Step 2 you are looking at your phase calibrator to decide the amplitude above which the data are bad and should be clipped. For most calibrators (1-2 Jy) 0.2 K is a good cutoff. This will have to be adjusted upward if a strong calibrator is used (i.e. 3c273). The correct cutoff will catch outliers but not cut out any points that are within the normal noise distribution.

Step 3: Run `uvflag` with:

```
vis=phasecal.ll
select=-amp(0.001,0.2) [or whatever you determined from step 2]
flagval=flag
```

In Step 3 you are flagging all the outlier visibilities in the individual channels. The `select` keyword is set to flag any values not in the range of 0.001 to 0.2. The lower value is used to flag any zero or very low amplitude points.

Step 4 Run `uvflag with:  [Optional]`

```
vis=phasecal.ll
select=shadow(6.1)
flagval=flag
```

In Step 4 you are flagging those visibilities that occurred when an antenna was shadowed by another antenna. This is most common in C array with low declination sources.

Step 5: Run uvwide with:

```
vis=phasecal.ll
out=phasecal.nw
```

In Step 4 you are creating a new dataset that has the wideband amplitudes and phases recalculated by taking the vector sum of the individual channel amplitudes and phases excluding any flagged channels. This way the wideband data will not contain any of the bad data that was flagged earlier. This is important since only the wideband data is used for calibration.

Step 6: [OPTIONAL and TEMPORARY] Run uvaver with:

```
vis=phasecal.nw
line=wide,2,1
interval=10 [see below]
out=phasecal.aver
```

Step 5 is only needed if you are going to use the GUI method of calibration described in Section 2.5.2. It is needed because the routine gmake does not calculate the time of a gain measurement correctly. This step will no longer be needed once gmake is fixed. The interval should be large enough to encompass an entire scan of your phase calibrator. Uvaver will vector average the visibilities and create a single visibility.

2.3. How to determine the flux of your phase calibrator (optional)

Due to the current instability of the amplitudes, flux determination is unreliable. If you are using a calibrator that is in the set of NRAO calibrators, you can use the most recent flux available in the table and get about the same accuracy as you will get using the method outlined in this section. If your calibrator is not in the table or if you don't believe the flux in the table, you can use this method.

The basic method is to determine the flux of your phase calibrator by comparing the visibility amplitude of your phase calibrator to the visibility amplitude of a source with a known flux. The sources with known fluxes are usually planets. In A array and in A+ array planets get resolved out and you must use a strong quasar whose flux is well determined as the known flux source. By comparing the expected flux on each baseline with the amplitude of each baseline we can determine the amplitude gain (Jy/K) for each baseline. With this known amplitude gain we can convert the amplitudes of the phase calibrator to Janskys.

Step 1: Run selfcal on both your phase calibrator (phasecal.nw) and flux calibrator (ampcal.ll) with:

```
vis=[phasecal.nw or ampcal.ll]
```

```
line=wide,1,1
interval=10
```

In Step 1 the goal is to take out any phase variations in your phase and flux calibrators that could cause the amplitudes to be biased low when the vector average is taken. Note that the phase variation can be different in the two sidebands so you cannot use the phase gains from one sideband for the other sideband.

Step 2: Run nuflux with:

```
vis=phasecal.nw,ampcal.ll
line=wide,1,1
taver=10 or 600
primary=[name of ampcal,flux of ampcal or Tb if planet]
```

This step does the actual flux calculation. It will calculate the flux for each time interval. If you want a flux calculated for each scan of your phase calibrator, then use taver=10. If you want a flux calculated from the vector average of all of the scans of your phase calibrator, then use a value of taver greater than the length of your track. The parameter primary tells nuflux which source is the one with a known flux. If the second value is not entered, it will calculate the flux based on its value of the brightness temperature if it is a planet or on the most recent flux in the cals.fluxes table if it is a quasar. It is best to use the sideband that has the lowest noise. For CO 1-0 this will always be the lower sideband. For other frequencies it could be the upper sideband. If both sidebands have the same noise level then you can run Steps 1 and 2 twice and average the fluxes.

2.4. Finding low level glitches in your data (tvflag)

By using uvspec and uvflag it is possible to flag all of the outlier visibilities but there are some low level correlator glitches that will not be flagged. At times a window will be biased upwards by a small amount due to a glitch in the correlator. If this bias is below the upper limit that will be found in uvspec then it will not be detected. Different baselines seem to have this problem at different times with baseline 5-6 currently being the worst. The routine tvflag allows you to inspect your data visually and find bad correlator windows and sections. This method can be skipped if you are just doing a quick calibration to look for a detection because the low level glitches do not have a large effect on your map but otherwise it should be done.

Step 1: Run tvflag with:

```
vis=source.cal
server=xmtv@[hostname]
line=channel,512,1
```

There is only one step needed when running tvflag. The server keyword is the name of the display server. The routine xmtv must have been started and running the in background before you start tvflag. Tvflag will display a greyscale image of all the data for a given baseline. Time runs in the vertical direction and channels run in the horizontal direction. A bad correlator window will be brighter in both the upper and lower sidebands and the same time. To flag a time range bad you use the ''+'' target to zoom until the region you want is on the screen. Then move to the MIRIAD window and type ''s''. Then to select the region with the mouse: point it at the lower left position, push the left button, drag it to the upper rightmost position and then release the button. Then move back to the MIRIAD window and type ''b'' to flag the region bad. A list of the commands can be obtained by typing ''?'' in the MIRIAD window. Once you have finished marking all the bad regions you type ''e'' to exit saving the changes to the file. Tvflag will flag the channels you have marked as bad.

2.5. Removing bad data from your source dataset

Bad data should be removed from your source dataset before you attempt to make a map. There is a difference of opinion on whether large amplitudes should be flagged before or after calibration. The advantage of flagging before calibration is that a single level can be used for all of the baselines. If there is a large variation in the amplitude gains of the antennas flagging after calibration requires different cutoffs for each baseline to flag outliers.

Step 1: Run uvflag on your line length corrected source dataset with:

```
vis=source.ll
edge=2,2 (if old mode 4 use 2,2,2)
flagval=flag
```

Step 1 flags the edges of the corfs. These regions are, in general, bad since they are at the edges of the filters. Note that in old mode 4 you also need to flag the central channels since they are actually the ends of the two corfs that make up the 200 MHz corf.

Step 2: Run uvspec with:

```
vis=source.ll
line=channel,512,1
axis=channel,amp
```

In Step 2 you are looking at your source dataset to decide where a good upper limit to the data is. For most weak sources 0.3K to 0.5K is a good cutoff.

Step 3: Run uvflag with:

```
vis=source.ll
select=-amp(0.001,0.4) [or whatever you determined from step 2]
flagval=flag
```

In Step 3 we are flagging all the outlier visibilities in the individual channels.

Step 4 Run uvflag with: [Optional]

```
vis=source.ll
select=shadow(6.1)
flagval=flag
```

In Step 4 you are flagging those visibilities that occurred when an antenna was shadowed by another antenna. This is most common in C array with low declination sources.

2.6. Determining the amplitude gain and correcting for phase variations

The key step in calibration is to determine the gains of the antennas and to correct for atmospheric and instrumental phase variations. In MIRIAD there are two different methods for performing this antenna-based calibration. Each method has its advantages and disadvantages. This document will describe how to use each method and leave to the user the decision of which method best fits their needs.

Both calibration methods perform a self-calibration of the phases and amplitudes for each time interval to determine the antenna-based amplitude and phase gains. Therefore, both methods are new to the six-element array and the accuracy of both methods improves as the number of antennas increases. Amplitude gain determination requires at least four antennas, and phase gain determination requires at least three antennas. Phase and amplitude gain determinations are both quite uncertain when operating at the minimum number of antennas. If you do not have at least four antennas during your entire track, you will need to calibrate using the old baseline-based method which is not described here.

2.6.1. *The* selfcal *method*

Calibration using the selfcal method is very quick. Basically you determine the gains using your phase calibrator(s) and the routine selfcal. You then will copy the gain table from your phase calibrator(s) into your source dataset. This is all that is required since all MIRIAD routines will read the gain table and apply the gains before processing the visibilities. If you want to permanently apply the gains you can make a copy of your source dataset and have the gains applied (uvcat, uvcal) to the new version.

Step 1: Run selfcal on the flagged version of your phase calibrator dataset with:

```
vis=phasecal.nw
line=wide,1,2
interval=30
refant=5
```

```
options=amp,apriori,noscale
fluxes=[name,flux]
```

This step perform the self-calibration and puts the output in a gain table in the phase calibrator dataset. You want to make sure to use the correct sideband for the line parameter since the phase variations are different for the different sidebands. Most observations are in the upper sideband, but if you are using the lower sideband you will want to have line=wide,1,1. The interval should be the sum of your time on source and time on your phase calibrator. This way you will get one gain determination for each scan of your phase calibrator. Currently, selfcal determines the start of the calibration interval independently from where your data is. This can lead to gain determinations combining data separated by 30 minutes in time. A new version of selfcal will avoid this problem. The reference antenna should be a new antenna (not #1 or #3) near the center of the array. The amp option tells selfcal to perform amplitude calibration. The option of apriori tells selfcal to assume there is a known flux for the calibrators. The option of noscale tells selfcal not to rescale the amplitudes to 1.0. The fluxes parameter is only used if you want to override the default value of the flux for a source. You can enter as many source name-flux pairs as you need, and selfcal will use those fluxes rather than the values in the cals.fluxes table.

Step 2: Run copyhd with:

```
in=phasecal.nw
out=source.ll
items=nsols,gains,ngains,interval,ntau,nfeeds
```

Step 2 copies the gain table from the phase calibrator file into your source dataset.

Step 3: Run uvcat with:

```
in=source.ll
out=source.cal
```

This step is optional. If you want to permanently apply the gains to your source dataset this step will copy the dataset and apply the gains to the new version. There is no way to look at the uncalibrated data in source.cal.

2.6.2. The graphical user interface method

Although the underlying principles used for the calibration is the same in this method as in the selfcal method, the philosophy of the user interaction is completely different. The GUI method also exhibits a higher degree of skepticism toward the signal to noise ratio of the data than the selfcal method. A disadvantage of the GUI method is that it cannot be run in the background. Step 2 (gfiddle) must be run interactively on a graphics terminal, precluding the use of a script to calibrate your data in a batch mode.

The GUI method has three steps. The first step is to create a visibility file with the self-calibration gain solutions. This file is identical to the gain table created by selfcal except that it is a standard MIRIAD visibility dataset that can be viewed by any of the routines that can look at UV data (uvplt, uvspec ...).

This file is used as input to a second routine, gfiddle, which generates polynomial approximations to the gains. This is done on the assumption that the variations in phase that are caused by the atmosphere and the instrumentation can be described by a low order (< 5) polynomial. Under this assumption it is better to fit a polynomial to the individual gain determinations since each gain determination has an uncertainty associated with it, and, by fitting a polynomial, you increase your effective signal-to-noise ratio. For example, if the phase variation were perfectly described by a second order polynomial and you had 12 scans of your phase calibrator each with a signal to noise ratio of 5, the phase rms of the gfiddle solution would be only 3 degrees vs an rms of 11 degrees for each individual selfcal gain determination.

Once the polynomial expressions for the gain variations have been determined the gains are applied to the dataset by running gapply which creates a new version of the dataset with the gains applied to the dataset.

Step 1: Run gmake on the flagged and averaged version of your phase calibrator dataset with:

```
vis=phasecal.aver
interval=1
out=cal.gvis
refant=5
options=amp,apriori,noscale
fluxes=[name,flux]
```

This step performs a self-calibration for each wideband channel and puts the output in a MIRIAD visibility dataset. The interval is set to 1 to work around a bug in gmake to make sure you only get the correct time tag for the gain determination. The reference antenna should be a new (not #1 or #3) antenna near the center of the array. The amp options tells gmake to perform amplitude calibration. The option of apriori tells gmake to assume there is a known flux for the calibrators. The option of noscale tells gmake not to rescale the amplitudes to 1.0. The fluxes parameter is only used if you want to override the default value of the flux for a source. You can enter as many source name-flux pairs as you need and gmake will use those fluxes rather than the values in the cals.fluxes table.

Step 2. Run gfiddle with:

```
gvis=cal.gvis
out=cal.gfit
device=/xw
```

Gfiddle is an interactive program that fits polynomials to the gains determined by gmake. The input is a .gvis file output from gmake. The output is a .gfit file which is a polynomial approximation of the .gvis file evaluated at a default value of 1 per minute. gfiddle allows the user to remove outlier points by typing ''d'' when the cursor is on the bad value. The recommended procedure is to fit 0th order polynomials to the amplitude gains and ''low order''(1st-5th) polynomials to the phase determinations. The strength of your phase calibrator is one constraint on fitting high order polynomials. For a weak calibrator, ($\sim$1 Jy) third order is about as high as you want to go since the individual gain determinations have a high inherent uncertainty. If you are lucky and are using a very strong calibrator (> 10 Jy), you can use the point-to-point interpolation (selfcal-like) option. The reason that a 0th order polynomial is used for the amplitude gain is that the first order variations in the amplitude gain caused by variations in the atmospheric transmission is corrected for at the telescope. In addition, the amplitude gain determination in any one scan is much more uncertain than the phase gain determination. Therefore, to best utilize all the data to decrease the uncertainty in the amplitude gain determination, it is best to always fit a zeroth order polynomial to the amplitudes.

Step 3. Run gapply to apply the gains to your source dataset with:

```
gvis=cal.gfit
vis=source.ll
out=source.cal
options=jyperk
```

This step applies the gain to a new copy of the source dataset. Note that gvis is not set to the output of gmake but to the file that is output from gfiddle. This is to confuse the uninitiated. The option of jyperk writes a uv variable for each baseline with the calculated amplitude gains. This is used by invert when determining the theoretical rms of your map when you use the systemp option.

2.7. Correcting for variations in the IF passband

The channel-to-channel variations in the amplitude and phase gain should be taken out by the online calibration. This leaves what should be only large scale variations across the passband. To correct for these, observations of a strong flat spectrum source are usually done during an observing track. Low order polynomials are then fit to the variations in the passband and applied to your source dataset.

Step 1: Run passmake on your narrow-band observations of your line length corrected passband calibrator with:

```
vis=passband.ll
pcal=pbcal.pcal
```

This step creates a .pcal dataset that contains the average phase and amplitude for each channel normalized to the wideband amplitude and phase.

Step 2: Run passfit on the output from Step 1 with:

```
pcal=[output from Step 1]
order=1
device=[your output device]
```

In this step you look a each baseline and can decide to reject bad points or just continue on to the next step. Usually you will just hit the ''e'' key for each baseline to go to the next baseline. You can use higher order fits if you have a strong enough passband calibrator. For the strongest calibrators you can use an order as high as 3.

Step 3. Run calapply on your calibrated source dataset with:

```
vis=source.cal
pcal=pbcal.pcal
out=source.pb.cal
```

This step applies the passband variations determined in Step 2 to your calibrated source dataset.

3. Making a Map

Once you have your data properly calibrated you will now want to make a map. There are many permutations on ways to make maps of interferometric data. We will only discuss the simplest cases here.

3.1. Creating a dirty map of your source (invert)

To transform you observations from the visibility plane to the xy plane a Fourier transform must be taken of the visibilities. In MIRIAD this is performed by the routine invert. The input to invert is a set of UV visibilities which invert will Fourier transform to create a ''dirty'' map.

Step 1. Run invert with:

```
vis=source.pb.cal
map=dirty.mp
beam=dirty.bm
imsize=512
cell=.5
sup=0
line=velocity,[# of channels],[starting velocity],[channel width]
options=systemp
slop=.2
```

```
In general the product of the imsize and cell should be twice as
large as the region you are interested in.  Thus, if you are creating
a map as big as the primary beam the product should be around 4′.
The parameter sup is used to control the type of weighting that
is used in the inversion process.  If sup=0 is used then natural
weighting is used giving you the most sensitivity to extended
structure and the best signal to noise ratio.  If sup is unset
then uniform weighting is used giving you the highest resolution at
the expense of sensitivity.  With the line parameter you control the
channel size, starting channel and number of channels that will be
in you channel map.  The linetype of velocity will usually be used
since this will average overlapping channels from different corfs and
can be set to a different width than the observing channel width.  By
setting options=systemp invert will weight the visibilities by the
expected variance based on the system temperature and, if you used
gapply, the antenna gains.  This will give more weight to the good
antennas and is recommended when you have a wide variation in system
temperatures.  The slop parameter controls how many bad channels
there can be in a visibility before invert will throw out the whole
visibility.  If you flag the edges of your corf windows and you run
with slop=0 all the visibilities would be ignored by invert while
using slop of 0.2 allows 20% of the channels to be flagged before the
visibility is rejected.
When invert runs it calculates the theoretical noise in the map.
This value is only correct if the correct Jy/K is in the dataset for
each baseline.  Currently, only the GUI method calculates this.
```

3.2. Removing the sidelobes from the map (clean)

Once you have run invert you have created a channel map. This map still contains the sidelobes of the synthesized beam and must be ''cleaned'' and thus it is known as the ''dirty'' map. Usually you will want to place a lower limit on how many features in the map clean should attempt to clean. Cleaning noise peaks does not improve the noise level in the map and takes a lot more time. Thus, you will need to find the noise level in the map and use that to determine a cutoff for clean.

Step 1: Run histo with:

```
in=dirty.map
region=box(20,20,150,400)
```

> There are two reasons why you need to specify a box as input to histo. The first is that you want to avoid regions where there is emission from your source. So in the above example we have avoided the center of the map. In addition, you want to avoid the edges of the map since sometimes these regions have weird aliasing problems. This is why we start at 20,20 in the above example.

Step 2: Run clean with:

```
map=dirty.map
beam=dirty.bm
out=source.cl
cutoff= [2 time the value of sigma output by histo]
niters=5000
```

> Here you input the dirty map and dirty beam that were output from invert. The output of clean is a cube each plane of which contains the clean components of that channel in you dirty map. The standard cutoff value to use is twice the noise level in the dirty map. The default value of niters is 250 which works well for small maps but when cleaning 512 x 512 maps you need a larger number of iterations to get down to the cutoff value. In general, you want to hit the cutoff value before you are stopped by the niters cutoff. If you know that there is only emission in a region smaller than the inner quarter of your map you can use the region keyword to restrict where clean will look for clean components. This can greatly speed up clean.

3.3. Making a clean map of your source (restor)

Now that the clean components in your map have been identified you need to subtract the dirty beam at these locations and add back in a Gaussian beam (''clean beam'') to create a ''clean'' map.

Step 1: Run restor with:

```
model=source.cl
beam=dirty.bm
map=dirty.mp
out=source.cm
```

Restor does not have parameters that change under most conditions. This setup will create a ''clean'' map by removing the sidelobes from the ''dirty'' map using the clean components.

Step 2: Run histo with:

```
in=source.cm
region=box(20,20,150,400)
```

You need to run histo again on your clean map to find the noise level. This will be needed to create a total intensity map and to determine the contour levels on your channel map.

Step 3: Run moment with:

```
in=source.cm
out=source.mom
mom=0
clip= [ 1 to 2 times the value of sigma found in step 2]
```

This step will take the zeroth moment of your data which results in a total intensity map. The clean map is used as input and only those points with an absolute magnitude greater than a clip point are used when determining the moment. This is done to avoid the contribution of a large amount of noise to your map which would increase the noise level in the resulting intensity map.

Step 4: Run histo with:

```
in=source.mom
region=box(20,20,150,400)
```

You need to run histo one more time to find the noise level. This will be needed to determine the contours on your total intensity map.

3.4. How to look at your map

Now that you have created a clean map of your source you'll want to look at it and start doing some science. The most powerful routine for this in MIRIAD is cgdisp. This routine has many options and modes only the basic version of how to print a total intensity map will be shown here.

Step 1. Run cgdisp with:

```
in       = source.mom,source.mom
type     = contour,grey
region   = quart
slev     = a,[1 sigma of map from histo]
levs1    = -4,-2,2,4,6,8,10,12,14,16,18,20
range    = [2 sigma],[maximum value in map],lin
device   = xw
labtyp   = hms,dms
options  = beambr
```

This example will plot a contour map overlaid on a greyscale version of the map. This helps see the difference between hills and valleys, which can be quite hard to differentiate in a contour only map.
The input file is listed twice so that you get both a contour and a greyscale. The region keyword is set to only plot the inner quarter of the map which is the region that was cleaned. By setting slev=a,XX, where XX is the one sigma value of the map, you multiply the contour levels specified in the slev keyword by the value of one sigma in the map. This lets you specify contour levels in units of sigma without having to calculate the absolute contour value for each contour. The range keyword controls the range of the greyscale. You would like it to start at the lowest contour and be black at the peak in the map. The labtyp keyword is set to label the axes in hour,

minute, seconds for RA and degree, minute, seconds for DEC. Finally, the options keyword is set to print a beam in the bottom right corner of the map.

4. Example Data Reduction Script

```
! /bin/csh -fv
uvcal vis=3c454.3,3c454.3.3,2038+513,ngc6946 options=phasem,nochannel,nocal\
       out=linel
selfcal vis=linel line=wide,1,4 interval=5 refant=5
copyhd in=linel out=ngc6946 items=nsols,gains,ngains,interval,ntau,nfeeds
copyhd in=linel out=2038+513 items=nsols,gains,ngains,interval,ntau,nfeeds
copyhd in=linel out=3c454.3.3 items=nsols,gains,ngains,interval,ntau,nfeeds
copyhd in=linel out=3c454.3 items=nsols,gains,ngains,interval,ntau,nfeeds
uvcat vis=ngc6946 out=ngc6946.ll
uvcat vis=2038+513 out=cal.ll
uvcat vis=3c454.3 out=3c454.3.ll
uvcat vis=3c454.3.3 out=3c454.3.3.ll
uvflag vis=cal.ll edge=2,2,2 flagval=flag
uvflag vis=cal.ll 'select=amp(0.2)' flagval=flag
uvflag vis=3c454.3.ll 'select=amp(0.2)' flagval=flag
uvwide vis=cal.ll out=cal.nw
uvaver vis=cal.nw line=wide,2,1 interval=10 out=cal.aver
/smc/mregan/miriad/selfcal vis=3c454.3.3.ll line=wide,1,1 interval=5
/smc/mregan/miriad/selfcal vis=cal.nw line=wide,1,1 interval=10
#
nuflux vis=cal.nw,3c454.3.3.ll primary=3c454.3,10.5 taver=29 line=wide,1,1 \
  select='-ant(4)'
gmake vis=cal.aver interval=1 options=noscale,apri,amp line=wide,2,1 \
      out=cal.gvis refant=6 fluxes=2038+513,1.6
# Ran Gfiddle, Gains seem to be very high 255 to 535 Jy/K
# Ant   1       3       4       5       6       7
# order 3       3       2       1       0       2
# sigma 15      12      11.4    10      0       7
uvflag vis=ngc6946.ll edge=3 flagval=flag
gapply vis=ngc6946.ll gvis=cal.gfit out=ngc6946.cal options=jyperk
uvflag vis=ngc6946.cal 'select=amp(90)' flagval=flag
```

```
#
#
uvflag vis=3c454.3.3.ll 'select=amp(1.0)' flagval=flag
selfcal vis=3c454.3.3.ll interval=1 line=wide,1,2
uvcat vis=3c454.3.3.ll out=3c454.3.3.cal
passmake vis=3c454.3.3.cal pcal=3c454.3.pcal
#
# ran passfit order=2, ant 4 problem doesn't affect PB
#
# ran tvflag to flag bad baseline 5-6 times, found 12 bad times
#
calapply vis=ngc6946.cal pcal=3c454.3.pcal out=ngc6946.pb.cal
```