BIMA Memoranda Series #22

Mapping BIMA Data with MIRIAD: A Cookbook

Eugène de Geus
University of Maryland
November 1991

Abstract

This cookbook is designed to help the new or slightly experienced user make maps from BIMA data using the MIRIAD data reduction package. Different methods are given for determining the success of the calibration procedure, and some detailed guidelines are presented on how to make dirty and clean maps. A familiarity with MIRIAD is assumed, such as how to run MIRIAD programs and how the observations from a single track on an object are organized into the different MIRIAD datasets. Advanced topics such as `MINT` are not covered here.

Life After Calibration

Now that your data has undergone some massaging, the first thing you want to be sure of is that during the massage you didn't break off an arm or a leg. Some things that can happen are: (1) in running `calapply` you didn't allow extrapolation of the amplitude calibrator, so most (if not all) of your data is now flagged bad, (2) in running `uvflag` you accidentally flagged bad more or a different region of uv-space than you intended. Furthermore, you would like to get some idea of the quality of your data. Three different methods to look at the data are available at this stage of the game: `tvflag`, `uvplt`, and `uvspect`. Of course you can also press on and actually make the dirty maps using `invert`, and `clean` them, but subtle errors in *e.g.* `uvflag` may not be so obvious from these maps. After you have made sure that all is well, you do want to go ahead and make the maps, clean them and use various plotting routines to visualize your results.

Looking at the Calibrated Data

At this point it is useful to understand how to access different sections of your data. Two keywords are very useful in this respect: `line` and `select`, and it would be worth your while to read up on them in the Miriad User's Guide (in version 1.0 of March 1991, this corresponds to pages 21 - 27). In short: `line` allows you to access subsets of your data in frequency (or velocity); `select` allows you to access subsets of your data in uv-space, time, amplitude, etc. They will be used extensively when looking at the calibrated data, so make sure you're familiar with them.

tvflag/uvflag

This program gives you the best bird's-eye view of your data, and it allows you to flag single visibilities or ranges of visibilities on-line. First you have to open a display window and a panel with buttons (in Sunview: `msss` and `panel`, in XWindows: `mxas` and `xpanel`). As options for `tvflag`, set `vis=`*calibrated_source_name*, and set `server=msss@`*your_machine_name*; `line` and `select` should at first be unset, so that you can look at all your data.

What will you see?

When this program is run, each baseline is displayed, one at a time, with the x-axis representing channel number (increasing to the right), the y-axis representing time (increasing upward), and the color intensity representing the amplitude of the visibility. The antennae that constitute the current baseline are labeled at the bottom of the plot (if room permits). In addition, if room permits, a color wedge is displayed to the right side of the visibility data which represents the sum of the data over all displayed channels. Also, a wedge of the sum of the data over all displayed times will appear above (or below) the data as room permits. At the top of the `msss`-window one can switch from *remote*-mode to *local*-mode, which allows one to change the colour-scheme and the transfer function of the display. In order to operate the panel-buttons, however, one has to be back in *remote*-mode. The panel-buttons are self-explanatory, and for more information on them, as well as some details on how to flag data, read the miriad-help on `tvflag`. The exit-button on the panel will move you to the display of data from the next baseline, and, after all have been displayed, exit the program. Quit will exit without going to the next baseline.

What should you be looking for?

Both real signal and bad data can be typified as having high amplitude (points or sections with intensities much larger than neighbouring points). If there is overall a large difference between the amplitudes on the upper and lower sidebands, go back and check the fit to the gains-calibrator. If you see a horizontal band of high amplitudes (more common around 115 GHz), you may have temporarily run into some bad weather. Usually you do see some deterioration of the amplitudes in the beginning or towards the end of your observations, which is due to your source being at low elevations; this is normal, and unless it gets really bad (more than a factor of 2 in T_{sys}: see the `listobs` output) it is ok to leave this in. If you see a vertical band of high amplitudes, you may have signal from your source (the band will then usually have irregular edges and change shape and intensity with time), or you may have a couple of bad channels (usually the band is more regular). Check to see if the corresponding section in the other sideband has it too: if it does it is possibly a bad section in the correlator, although you *always* want to be careful and check where you would expect to see your source. The edges of the frequency windows are usually bad, so you can expect the amplitudes to get high there. The summation-wedges at the edges help distinguish bad sections.

Flagging?

I find `tvflag` very useful for spotting problems with sections of the data (whole time-ranges or channels that are bad), but except from flagging the odd visibility, I prefer to do the flagging with `uvflag` . I take note of what looks bad in `tvflag`, and then use the `select` and `line` keywords in `uvflag` to set the proper flags. For the edges of the windows, `uvflag` has a special option to flag the edges of all frequency windows (keyword `edge=`*#-channels*) and (in mode 4) the `corfs` in the middle of the windows. I usually flag 8 edge channels bad, but that's in mode 2, where you have a total of 128 channels per window. In mode 4 (64 channels per window) you may want to be a little more conservative and flag 4 edge channels. Of course if the edges are very bad you may have to go with more than that. Using `uvflag` makes it much easier to keep track of the flagging.

uvplt

Normally the visibilities with the largest amplitudes are spurious and they can add a considerable amount to the noise-level of your final maps. So clipping of high-amplitude visibilities is usually a good idea. `uvplt` allows you to plot visibilities as a function of different parameters (like time or uv-distance), and is therefore particularly useful for determining the level at which you want to clip your data in amplitude. Select your reduced datafile as the input; for `line` take the `channel` option and choose a couple of channels covering the channel where you expect signal from your source (you don't want to clip your source out by setting the level too low), set `xaxis=uvdist` and `yaxis=amp`. This will plot the amplitudes versus uv-distance for each visibility in the selected channels.

What to look for?

Each baseline of observations covers a range in uv-distance and can be recognized as such. The points should all be distributed around some mean amplitude, with a few points going up to large values. The amplitude at which you should clip is rather subjective, but if you consider the distribution of points in amplitude to be a Gaussian, you would like to clip at the 4 or 5 σ level. In `uvflag` now, use `select=amplitude(`*clip-level*`)`. Sometimes one baseline is particularly high in amplitude which will be quite obvious in the `uvplt`-plot. This can mean that baseline really was bad (throw it out), or your gains-calibrations somehow went wrong (check it).

uvspect

`uvspect` allows you to look at the amplitudes of your visibilities in a more traditional single-dish manner: amplitude (flux) as a function of channel, frequency, or velocity (depending on `line`). The plot will also show the phases (as dots) in the same plot. You can integrate your data over time using keyword `aver`, but you should remember that it will perform a *vector*-average, so if a source is not at or near the center of your field, its phase can change significantly during the course of the full observation, and its amplitude will be averaged out (and the phase will become random). So sometimes it can be useful to look at 1 hour averages, instead of full averages. Be sure to use `select` to select one baseline at a time (usually the shortest for the largest sensitivity), because otherwise all three get averaged into one plot, and a long baseline, with low sensitivity, might wash out the signal. Also, set `fmode=u`, so that only unflagged data are plotted. If you want to plot velocity on the x-axis, but you forgot exactly what your frequency set-up implied in terms of v_{lsr}, run `uvlist` on your source file with `options=spect`. You will get a summary of the frequencies and velocities of your different windows.

What to look for?

`uvspect` is great for catching data where you forgot to apply the gains-calibrator. The y-axis will then have amplitudes $\lesssim 0.1$ (which means it is still in Kelvin), whereas calibrated data should have amplitudes > 1.0 (in units of Jansky). Note that the units are not given in the labelling of the y-axis, you have to look at the numbers. If you flagged some channels to be bad, they will show up as having zero amplitude and zero phase in your plot. So errors in the flagging process can also be detected using `uvspect`.

The Making of the Maps

Now that you have checked your calibrated data using the techniques above, you are ready to start making maps. The first step is to grid the visibility data and then to Fourier transform them into a map. This is done by the program `invert`, which is discussed in more detail below. The result (a single map or a cube of channel maps depending on the input parameters) is often referred to as the "dirty" map. For a first look at your data it is usually wise to make a single map, integrated over a reasonable range in velocity and fairly small in size. Such a map is a lot faster to make, easier to handle and more convenient to display. Then later when you know you have signal and are ready to start analyzing your source, you can start filling up your disk with giant cubes. If you have a reasonable signal from your source (say $> 10\sigma$), the sidelobes from your synthesized beam will introduce "structures" in your map. Of course, because you know the properties of the synthesized beam exactly, you can take these structures out using an iterative procedure known as CLEAN (yes, the Miriad program name is `clean`). In order to produce the final maps, this should be followed by `restore`. A second way to clean your maps is by using the maximum entropy method (`maxen` in Miriad). This cookbook will deal with `clean` more extensively than with `maxen`.

invert

The program `invert` takes all calibrated visibility datasets that have been specified as the input (using keyword `vis=`), interpolates the data in uv-space onto a regular grid, and does the Fourier transform, to create a two-dimensional map or a three-dimensional data cube. The output is written into a file specified with keyword `map`. The two-dimensional map containing the synthesized beam pattern (normalized to 1) is written into a separate file, identified by the keyword `beam`. The pixel size of the output image (in arc seconds) has to be given by the keyword `cell`, and the number of pixels is specified by the keyword `imsize` (which must be a power of 2). Note that `cell` $\approx \frac{FWHM(beam)}{3\ or\ 4}$ is appropriate, because you need to oversample by a factor of at least 2 to satisfy the Nyquist theorem. Oversampling by too large a factor (*i.e.* making `cell` too small) can result in maps that are smaller than the primary beam, so that you could miss interesting information. On the other hand, if your resolution is very high (*e.g.* with A-array data), you will have to go with a small value for `cell`, because you *have* to oversample.

At this point it is important to ask what you want to do with the maps and how well you understand your source already. For a first quick look at your maps (see if you detected something, get a feeling for the size and morphology of the region) it is usually enough to make a fairly small map (just the size of the primary beam, which is typically $2'$ FWHM). For instance, if you have B-array data you will get a beam size of about $6''$, allowing you to use values for `cell` of $2''$ and `imsize` of 64, but with A-array data your beam will be more like $3''$ which means you'll have to use values for `cell` of $1''$ and `imsize` of 128. When you are ready to make full blown maps that you also want to clean, you should double the number for `imsize`. The properties of the third dimension of the data cube are controlled by the keyword `line` (standard use). The remaining keywords have reasonable defaults, but circumstances may call for other keywords to be filled in as well. (Upon special request, contact your local MIRIAD guru, it is possible to run a version of MIRIAD that allows maps of 512^2, but these sizes rapidly become unwieldy) .

Example: I would like to make a map from visibility data taken in B-array and in C-array. The B-array data has been backed-up on tape, but I saved the calibrated source data sets in a sub-directory `save`. *The data obtained in the C-array configurations are stored in sub-directories named after the date of observation. All the filenames used as input are combined in an "include"-file (called invert.inp). I first want to make a single small map of the velocities between -82.5 and -77.5 km/s for a quick look. After that I would like to obtain a 3-dimensional cube with size 256x256 pixels of 2″ x 2″ each, and 100 velocity channels starting at -98.0 km/s with a width of 0.25 km/s in steps of 0.25 km/s. The mode of observation was 2, and the signal is expected in the upper sideband (i.e. windows 3 and 4). The commands are:*

```
invert  vis=@invert.inp  map=map_name_mp beam=map_name_bm imsize=128 cell=2
line=velocity,1,-80.0,5.0,5.0 select=window(3,4)
```

```
invert  vis=@invert.inp  map=map_name_mp beam=map_name_bm imsize=256 cell=2
line=velocity,100,-98.0,0.25,0.25 select=window(3,4)
```

The contents of the include file invert.inp are:

save/s127co_02oct.r
save/s127co_09nov.r
save/s127co_01dec.r
save/s127co_02dec.r
save/s127co_16dec.r
save/s127co_17dec.r
save/s127co_16jan.r
25may/s127co_25may.r
03jun/s127co_03jun.r
08jul/s127co_08jul.r
19aug/s127co_19aug.r

More options (*i.e.* more keywords) are available, but I will mention only those three that I would consider to be useful for inexperienced users.

- `fwhm` (in arc seconds) allows you to increase your beam size by convolving with a Gaussian of the specified fwhm in order to optimize the signal-to-noise for structures of larger angular size.

- `sup` (in arc seconds) specifies the sidelobe suppression area. The what? The sidelobe suppression area. `sup` can be all integer numbers between, and including, 0 and *imsize*. If `sup`=*imsize* (the default), you suppress the sidelobes of the beam over the whole image. This is done during the stage of re-gridding the data in uv-space, by assigning equal weight to each new grid point regardless of how many original uv-data points contributed to it. Hence the often-used name"uniform weighting". This process goes at the expense of the signal-to-noise ratio. If `sup`=0, you don't suppress sidelobes at all. This means that during the stage of re-gridding the data, each new grid point is assigned a weight proportional to the number of original uv-data points contributing to it. Hence the name "natural weighting". The signal-to-noise ratio is better in this case, but the beam is generally not as nice (surprise: it has sidelobes...). If you use something in between these two extremes, you will obviously get some trade-off between sidelobe suppression and signal-to-noise ratio.

- `options` (as if the other keywords aren't *options*). This is a hodgepodge of processing options for which no separate keywords have been made. The only option which I think might be of interest this stage is `options`=*systemp*, which ensures that each visibility is weighed inversely proportional to the square of the system temperature. This results in noisy visibility data being used with a lower weight, and, thus, should improve the noise of the resulting map.

Inspecting the Newly Acquired Maps

Now that you have made the "dirty" maps, you would like to do several things. First you want to see what your beam looks like by plotting it, second you want to find out what the rms noise level of the maps is, and third you want to look at the maps. Two plotting routines are available inside MIRIAD (`pgdisp` and `implot`) and of course there is WIP. For quick plotting of the beam, I prefer to use `implot`. In order to find the rms level of the maps, a few programs are available, namely `imhist`, `histo`, `imstat` and `imlist`. Knowledge of the rms level of the maps is important before you plot the maps, because it tells you at what contour levels to plot, and also before you run `clean`, because it determines the level at which you can stop cleaning. For a quick look at the maps `pgdisp` is useful (or if you have some macros set up, WIP is actually faster). Looking at the maps is important, because it tells you whether or not you have signal, and whether or not it is useful to run `clean` at all.

implot

Use `in=`*map_name*`_bm` and make sure to set the keyword `device` properly (*e.g.* /sun or /x for screen-plotting or *outfile*/ps for writing to a ps-file which can then be sent to the printer).

The keyword `region` specifies the sub-section of the map which you want plotted, which defaults to the whole image. The keyword `region` is well described in the Miriad User's Guide, in the section on General Task Concepts. The keyword `units` allows you to specify the units for the plot-axes: **p** for pixels relative to the center (default), **s** for arc seconds relative to center, and **a** for absolute coordinates. Keyword `conflag` specifies the contour options, and for the purpose of plotting the beam, I use **p** which means a percentage of the maximum in the map, combined with **n** which means negative contours should be plotted as well. The actual percentage-value is given in keyword `conargs` . Other keywords are available, but they are more useful when plotting the data maps rather than the beam.

Example: In order to make a hardcopy plot of the ten-percent contours (positive and negative) in the inner quarter region of the beam, one would use:

`implot in=`*map_name*`_bm device=beam.ps/ps region=box(65,65,192,192) units=s conflag=pn conargs=10`

imhist/histo

These programs give you some statistical information on a selected region of your image such as minimum, maximum, mean and r.m.s. deviation. If you want to know the rms and you use `region` to select more than one plane of your cube, you have to be careful that none of the planes have signal. If you cannot select planes without signal (*e.g.* with a continuum observation), you will have to use a sub-region of your map where you can be reasonably sure that it is empty. `imhist` will produce a plot of the histogram of the pixel values, besides writing the information to the screen. `histo` will just write to the screen.

Example. First I would like to get some statistics on the inner quarter of planes 2 to 5 of my map, and then I would like to compare that to the noise-level in a subregion of plane 6 (note how we circumvent the "from-to" format for selecting the plane in the latter example):

`imhist in=`*map-name*`_mp region=boxes(65,65,192,192)(2,5)`

`histo in=`*map-name*`_mp region=boxes(65,65,83,91)(6,6)`

imlist/imstat

The advantage of `imlist` (with `options=`*stat*) and `imstat` over `imhist` and `histo` is that it provides you with the statistical information (*e.g.*maximum, minimum and rms) on a plane by plane basis, each on a separate line, with `imstat` having the additional feature of producing a plot of the desired parameter as a function of the third coordinate of your cube (*e.g.* channel or velocity). So by looking at the output listing or the plot you will immediately notice if a plane has some signal (both the maximum and the rms go up, so you know not to trust the rms in that plane). `imlist` does more than produce statistical information on your data, because using different values for keyword `options` allows you to look at the header of the map (*head*), dump the data in some specified format (*data* and `format=`*desired-format*), and list the history of processing for the map (*history*).

Example. I would now like to see the statistics of the inner quarter of all planes of my map, and write the output to a log-file:

```
imlist in=map_name_mp region=boxes(65,65,192,192) options=stat log=log_file
```

or, if I also want to plot the rms:

```
imstat in=map_name_mp region=boxes(65,65,192,192) log=log_file
```

pgdisp

The plotting routine I most commonly use for displaying my maps is `pgdisp` (formerly `cgplot`). You can plot both contours and grey scales on top of each other, but for a quick look contours are usually enough. Specify the input map for contouring with `cin` (`gin` for grey scale map). The `region` keyword is standard use, usually I only look at the inner quarter of the maps (for 128^2 maps: box(33,33,96,96), and for 256^2 maps: box(65,65,192,192)). Keyword `slev` consists of one character (**a** meaning absolute contour levels, or **p** meaning percentage), a comma and a value by which to scale the contour levels (with a it is an absolute number, with p it is in percent). The next keyword, `levs`, determines at what levels to draw the contours in units of the scaling specified in `slev`. Normally, I put `slev` in absolute units of 1σ of the noise, and use `levs` starting at 2.5 or 3, depending on how strong you expect the signal to be. Make sure to set the `device` properly (see `implot`). For the labelling of the axes you have several options, and the online help explains them quite clearly. I usually take arcsecond offsets (**ao**). The keyword `options` raises its ugly head again, this time for the specification of further plot annotation and formatting. I usually use **f**(ull, for complete plot annotation), **e**(qual, for plots with equal scales in x and y), and when I plot multiple planes of the data cube on one page, I also annotate the plot with the velocity of that plane (**v**elocity). If you do want to plot multiple planes of a cube, and don't want them to come out one per page, specify **nxy** to be the number of panels in x and y. This should provide you with enough tools to make nice plots with `pgdisp`.

Example: I just created a cube of 50 planes, 256^2 each, with an rms per plane of 0.8 Jy/beam, for which `imlist` *revealed that 25 planes probably have signal. In order to make a plot on the screen (I'm in XWindows) of the inner quarter of these 25 planes, with 3x2 planes per page, arcsec offsets on the axes, full plot annotation, equal axis-scales and velocity information per plane, I would do:*

```
pgdisp cin=map_name_map device=/x region=box(65,65,192,192)(8,37) slev=a,0.8
levs=-18,-15,-12,-9,-6,-3,3,6,9,12,15,18 nxy=3,2 labtyp=ao options=f,v,e
```

In the plots you might look for structures that are above the noise, and that appear to mimic the shape of the beam (including the sidelobes). These are usually the indications of a real detection. If your maps appear to be empty, try making maps integrated over a different number of channels (`line` in `invert`) or try smoothing your data to a lower angular resolution (using `fwhm` in `invert`), or... ask yourself why you wanted to observe this particular source in the first place.

The CLEANing of the Maps

Your maps are now ready to be CLEANed of the effects of the side lobes of the beam. This process can be made as complicated as one likes, and, again, getting acquainted with some of CLEAN's intricacies may not be such a bad idea. You can, however, get some very nice results already with rather simple inputs. The principle of the CLEAN algorithm (devised by J. Högbom in 1974) is based on the assumption that the radio source can be represented by a number of point sources in an otherwise empty field. The result of the radio observation is a convolution of the beam (including its horrible sidelobes) with all of the point sources. So the CLEAN algorithm employs an iterative approach to find locations and strengths of the point sources called the CLEAN components (starting with the strongest and working down), convolves them with a CLEAN beam (usually a Gaussian), adds the residuals back into the map, and writes them in a final image. In MIRIAD this procedure is done in two steps (*i.e.* two programs): first you determine the CLEAN components (using `clean`) and then you do the convolution with the CLEAN beam and the writing into a final image (using `restore`).

The description of the CLEAN procedure given here is highly simplified, but this cookbook is not intended to teach you the backgrounds or intricacies of CLEAN. A very clear treatise on the principles and problems of CLEAN is Tim Cornwell's section on deconvolution in the "Course Notes from the NRAO Summer School on Synthesis Imaging" (chapter 7, 1985). Getting some familiarity with the backgrounds and especially the limitations of CLEAN will prove to be very beneficial. It aso gives a very good description of the maximum entropy algorithm, which can be applied instead of CLEAN.

clean

This program takes the dirty map and beam as inputs (keywords `map` and `beam`), and it requires an output name (keyword `out`). CLEAN will work on a point in the map and look for effects due to the beam as far away as 1/4 of the full size of the map. So, it is *impossible* to CLEAN a region larger than the inner quarter of your map!! Now, by default it will CLEAN exactly the inner quarter of the map, unless you use the keyword `region` (standard use) to specify a *smaller* region. Once you have some idea of what your source looks like, it might be a good idea to limit the CLEAN-box to the region just around your source. It is usually also smart to provide CLEAN with some kind of limitation on the number of iterations it will do. The most basic limitation is simply the maximum number of iterations allowed (keyword `niters`). Since the CLEAN-algorithm starts with the strongest source in the map, working its way down, a more sensible limitation is provided with the keyword `cutoff`: if the next strongest peak in the map is below this value CLEAN will stop. I've found that a value for `cutoff` of about 1.5$\times$rms to work very well, because obviously you don't want to start CLEANing the noise. The map you are left with after having taken out all the CLEAN components is the so-called "residual" map. Usually I also provide a hard limit with `niters`, just in case I made a mistake (in determining the rms, or simply a typo) so that `clean` doesn't gobble up a huge amount of CPU time for nothing. The `mode` keyword allows you to select which CLEAN algorithm will be used. The default for `mode` is "any", which means that the program itself decides which algorithm to use. The bad news about that is that it is not clear (from the on-line help) what that choice is based on, the good news is that the program does tell you which algorithm it chose to apply. Depending on the choice of algorithm, more keywords can be filled out. For more information on the different CLEAN algorithms and the significance of the additional keywords, see Cornwell's paper.

Example. I want to CLEAN the map I made before. Just the inner quarter is fine, but all planes need to be CLEANed. The rms of my map turned out to be 1 Jy/beam. I am lazy so I'll let the program figure out which algorithm is best for me. So I use:

clean map=*map_name*_mp beam=*map_name*_bm out=*map_name*_cln cutoff=1.5

niters=1000

maxen

The maximum entropy method to clean maps employs the idea that the data should produce an image containing positive total flux with a compresses range in pixel values. This is contrary to CLEAN which assumes the map has zero total flux. Maximum entropy works best when it is provided with a separate constraint on the flux or morphology of the source (*e.g.* in the form of an input map). In the absence of such additional information it is very hard to voice a preference for either algorithm, and I will not do so. So try to form your own opinion based on the specific needs of your data.

In any case, the basic inputs for `maxen` are: the dirty map, the dirty beam, the output (keyword `out`), and the rms of the map (a very important input on which the behavior and convergence of `maxen` depend). There are also a number of possibilities to specify more knowledge about the source (*e.g.* keyword `flux` which is the total flux of the resulting map), and options to specify exactly what the measure of entropy should be. I will not discuss these at all.

Example. I want to deconvolve the map I made before using `maxen`*. Just the inner quarter is fine, but all planes need to be done. The rms of my map turned out to be 0.75 Jy/beam. So I use:*

maxen map=*map_name*_mp beam=*map_name*_bm out=*map_name*_max rms=0.75

restore

Now that `clean` has calculated the clean components, or `maxen`has deconvolved the dirty image and the beam, you need to convolve them with a "clean" beam (usually a Gaussian), and add them to the map of residuals. The program `restore` will do that for you. The inputs are the following, `map` is the input "dirty" map, `beam` is the input "dirty" beam. Contrary to what was used in `clean` and `maxen`, however, `model` is now the file with the deconvolved components (which was keyword `out` in `clean` and `maxen`). The properties of the clean beam can be user-specified with keyword `fwhm`, which normally requires two numbers namely the full-widths at half maximum of the major and minor axes of the beam, and keyword `p`, which is the position angle of the restoring beam measured in degrees east from north. If `fwhm` has one number, the beam is circular, and the position angle becomes meaningless. If no values are given to `fwhm`, the parameters of the restoring beam are calculated by fitting a 2-dimensional gaussian to the dirty beam. The final, restored image is then written to the file specified by keyword `out`. By using the keyword `mode` you have more control over what exactly gets written out (*e.g.* you can choose to write only the residual maps), but I refer to the on-line help for its options.

Example. I want to restore the maps that just got cleaned, using the gaussian fit to the dirty beam:

`restore map=`*map_name*`_mp beam=`*map_name*`_bm model=`*map_name*`_cln out=`*map_name*`_final`

After `restore` you are of course eager to look at your maps, so go ahead, use `implot`, `pgdisp` or WIP. For `implot` and `pgdisp` I refer to their description in the section immediately following the making of the maps. For WIP I refer to an excellent manual written by Jim Morgan. As for my preference: `pgdisp` for quick looks after you've created the maps, and WIP for making publication quality plots, because it gives you a lot of freedom to make maps look exactly as you like it.

Summary

The processes of quality control for calibrated data and the making of maps have been described in this cookbook. Some programs that have been described here may prove to be beneficial to your understanding of the data storage and handling, but you may find yourself skipping them when you've gone through the process a couple of times. This will be especially true for the programs concerning quality control. This may be ok, you're becoming an experienced user, but keeping track of the quality of your data is an extremely important, albeit tedious part of making good maps, and so should never be ignored.

Some of the choices for keywords are based on my experience (or lack of it) and on the properties of the sources that I look at, so try to understand what you are doing and how certain programs or keywords may have to be applied differently in your specific case. For questions and problems you can go to the MIRIAD on-line help which is quite good, read the manual, or talk to your local MIRIAD expert.