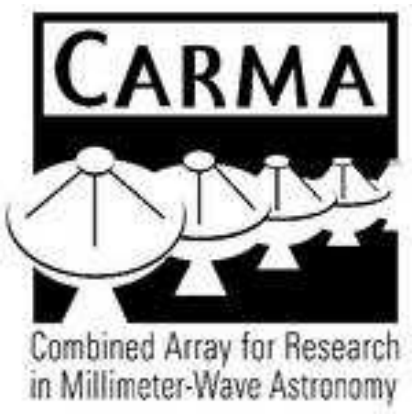

CARMA Memorandum Series #999

HEMOSI: Heterogeneous Mosaicing Simulations

Peter Teuben
University of Maryland
Melvyn Wright
UCB

October 2, 2008

ABSTRACT

HEterogeneous MOsaicing SImulations (**hemosi**) using MIRIAD are described. A Python script implements the algorithm also found in earlier shell scripts. The user supplies an image and an antenna location file and through a set of parameters to the scripts defines groups of antennas. We also discuss some advanced usage of the Python scripting language for MIRIAD, some of its Pros and Cons, and some scaling issues with MIRIAD we found in using the script for these types of simulations. We also suggest a new method to have MIRIAD tasks communicate with the shell.

Change Record

Revision	Date	Author	Sections/Pages Affected
	Remarks		
1.0	2008-Jul-30	P. Teuben	
	Initial version, based on hex7-15.csh, mosaic.py and hetero.py		
2.0	2008-Aug-08	P. Teuben	
	Various keywords changed meaning: config=ant, ants=pb, tsys=systemp, plot=device; added new keyword factor=, filename conventions changed in the run directory		
2.2	2008-Aug-15	P. Teuben	
	Re-running script implemented differently and clarified		

1. Introduction

In previous memos (e.g. Wright 2004, CARMA memo 27) simulations of interferometric arrays have been presented using MIRIAD shell scripts. The directory `$MIR/demo/carma` in particular contains shell scripts that produced CARMA memo 27 and some earlier memos referenced therein. The script described in this memo, **hemosi.py** can be found in `$MIR/examples/mosaic` and expands on the ideas presented in earlier scripts, most notably in providing a more programming friendly environment using Python for this relatively well defined problem.

? Explain Antenna type vs. Primary Beam type?

2. Description

MIRIAD (Sault et al. 1995) contains a number of programs useful in generating data for heterogenous mosaicing simulation. The script employs the following procedure.

1. **uvgen** : a template visibility file for each antenna/primary beam type (K) is generated. Each file has multiple pointings (N) and they are generated with only realistic noise (i.e. no source).
2. **demos** : the input model map is de-mosaiced into N fields as defined by the previously generated visibilities.
3. **uvmodel** : for each field and PB type the corresconding model is added onto the visibilities.
4. **invert** : all K*N visibilities from the previous step are accumulated and inverted into a single linearly mosaiced dirty map and K*N dirty beams
5. **mosmem** (or **mossdi**) : using a Maximum Entropy or a Steer CLEAN algorithm the mosaic is deconvolved. For a single field single antenna setting one can also choose **maxen** or **clean**. Some comments on **mfclean**
6. **restor** : produces a "cleaned" map by restoring the synthesized beam with the clean components.

An important feature of the script is that it can be re-run skipping the first 5 steps and only recomputing the deconvolution method, and with different parameters.

3. HEMOSI

3.1. Running

Running the script requires the MIRIAD environment to be loaded in your shell[1]. In addition, the PYTHONPATH environment variable should point to the location where `Miriad.py` is located (MIRIAD provides this by default).

When you run the script, a number of parameters are passed via the commandline, much like running MIRIAD commands. On the screen you will see python script output and the MIRIAD commands that are issued, but you will not see the sometimes lengthy output from the individual MIRIAD commands. They are gathered in the `miriad.log` file inside a run directory. With the `log=t` keyword you will be able to view its progress in an another window.

Here is an example of a simple simulation of a 6 element OVRO array in the CARMA C configuration (since the first 6 antennae are the orginal OVRO antenna, the CARMA+SZA 23 antenna file can be used for this):

```
  % hemosi.py image=n4254.3 ant=$MIRCAT/carma_CZ.ant dir=n4254_CZ pb=ovro,6 jyperk=43 nring=1
   ---  HEMOSI: HEterogenous MOsaicing SImulations  ---
hemosi.py: PYRAMID Version 2.1 (8-aug-2008)
Found 23 antenna in carma_CZ.ant
Telescopes:  ['ovro', 'hatcreek', 'sza', 'carma', 'sza10', 'sza6']
Jy/K: [43.0, 126.0, 383.0, 73.0, 128.0, 220.0]
MOSAIC FIELD, using hexagonal field with nring=1 and grid=20 (1 pointings)
....
MIRIAD%  uvgen ant=/home/astromake/opt/miriad/cvs/cat/carma_CZ.ant
         baseunit=-3.33564 radec=23:23:25.80,30 lat=37.28 ellim=10
         harange=-2,2,0.013 source=$MIRCAT/no.source telescop=ovro
         jyperk=43 systemp=80,290,0.26 freq=115 corr=1,1,0,8000
         out=n4254_CZ/uv_0 center=0.00,0.00
MIRIAD%  puthd in=n4254_CZ/single/crval1 value=23:23:25.80,hms
MIRIAD%  puthd in=n4254_CZ/single/crval2 value=30.0,dms
MIRIAD%  puthd in=n4254_CZ/single/crval3 value=115.0
MIRIAD%  puthd in=n4254_CZ/single/cdelt1 value=-0.5,arcsec
MIRIAD%  puthd in=n4254_CZ/single/cdelt2 value=0.5,arcsec

MIRIAD%  demos map=n4254_CZ/single vis=n4254_CZ/uv_0 out=n4254_CZ/uv_0_demos

MIRIAD%  uvmodel vis=n4254_CZ/uv_0 model=n4254_CZ/uv_0_demos1
         out=n4254_CZ/uv_1_0 select=ant\(1,2,3,4,5,6\)\(1,2,3,4,5,6\)
         options=add,selradec

MIRIAD%  invert vis=@n4254_CZ/vis.all map=n4254_CZ/xy.mp
         beam=n4254_CZ/xy.bm imsize=257 select=-shadow\(3.5\) sup=0
```

[1]For certain demos you will need a non-default MIRIAD installation with a larger value for MAXANT

```
        options=mosaic,double,systemp
...
```

Placing all files in a single run directory makes for easy data management and cleanup. You will also find that within a run directory files always carry the same name. For example, the final cleaned image is always called **xy.cm**.

It may be advantageous to encode certain parameters in the name of your run directory, or in a hierarchy of run directories.

An important NOTE: the `hemosi.py` script is organized in such a fashion to create vis data once, if not present, and deconvolve many times with different parameters.

3.2. Keywords

The **hemosi.py** script accepts a *keyword=value* command line interface. Use `--help` to obtain help on the current keywords and defaults, as they keyword listing below can easily get out of sync with the actual code. Most keywords have a reasonable default, listed here in square brackets.

- **dir** : Run directory in which all datasets are written. Depending on the type of simulations you are doing, choosing your run directory hierarchy and/or naming convention can be beneficial for further analysis. The current directory "`dir=.`" is also allowed, but not a recommended practice. NOTE: Only for the first run visibility files will be generated, but for any subsequent run only the deconvolution will be run. [run1]
- **ant** : antenna config file (NEU in meters). This file will also be copied into the run directory. [CZ.ant]
- **pb** : Primary beam types and antenna numbers, plus PB types for all cross type baselines. First a list of primary beam types, followed by the number of antennas that are in the array need to be listed, followed by the cross correlation primary beam types of unequal antennas. For example, the initial CARMA-15 array would be encoded as
  `pb=ovro,6,hatcreek,9,carma`,
  and the 2008 CARMA-23 array with 8 added SZA antennas is encoded as
  `pb=ovro,6,hatcreek,9,sza,8,carma,sza10,sza6`.
  The convention for the order of the PB types of the cross antennae list is to make the second index runs fastest, and is also larger than the first, much like the output of the `uvlist` program in MIRIAD, thus carma=1-2, sza10=1-3 and sza6=2-3. If only one primary beam type is given (a homogenous array), it can still be optionally followed by the number of

antennas that will be used from the antenna file. If none is given, it will assign each this PB type to each position. [ovro,6,hatcreek,9,sza,8,carma,sza10,sza6]

- **systemp** : System temperature(s). Either one or three numbers can be given. Either a constant receiver temperature is given, or else a receiver and sky temperature, plus a zenith opacity, from which elevation dependant system temperatures are calculated. See also the description under the `uvgen::systemp` keyword. Currently there is no support for antenna dependant system temperatures. Just before `invert` is run, the system temperatures are patched with fake values using **jyperk** in order to run a weighted invert. See also MIRIAD's `obstau` program for more information on reasonable values for this parameter. [80,290,0.26]

- **jyperk** : Jy/K scaling for all primary beam types. This determines the noise scaling. An entry for each primary beam type needs to be given. [43,126,383,73,128,220]

- **gnoise** : Gain noise (percentage) used to emulate pointing errors.

- **dec** : Declination where object should be placed. [30.0]

- **freq** : Observing frequency (in GHz) [115.0]

- **image** : Model image to map. This needs to be a MIRIAD image (2D or 3D). The scaled image (see below) will be copied into the run directory. The value at the reference pixel will also be changed, to account for the requested declination. (RA – check) [casc.vla]

- **nchan** : Number of channels to use from the model image. If the input model is a cube, this number can be less or equal the number of channels in the model cube. [1]

- **cell** : New cell size of the model. This parameter allows one to change the size of the input model, to match the requested field of view of the mosaiced pattern. [0.5]

- **factor** : Factor by which the input model image is scaled to Jy. Depending on the header, the image can be JY/BEAM or JY/PIXEL, but this factor merely scales the numbers. [1]

- **size** : Size of the cleaning and plotting region (-size..size) in arcsec. [50.0]

- **nring** : Number of rings in the hexagonal mosaic. Notice a single (non-mosaic) pointing counts as 1 ring. [2]

- **grid** : Gridsize (in arcsec) for the mosaic. The program will warn if you are not close to Nyquist sampling of the smallest primary beam. [20.0]

- **center** : optional center file that overrides (nring,grid). By default the **nring** and **grid** parameters define

- **method** : Deconvolution method. Valid options are: `mossdi, mossdi2, mosmem, joint`, or `default`. If there is only one field and one antenna type, the old-style clean and maxen programs can also be choosen. [mosmem]

- **niters** : Maximum number of iterations in mossdi/mosmem/.... [200]
- **flux** : Expected flux in the image (for mosmem/maxen). **check if this works on re-run ** [0]
- **device** : The pgplot device name. Some choices can make your script interactive. The name is allowed to contain a format directive for an integer that gets automatically incremented for each time a device= is used in a MIRIAD program. Examples are `%d/xs` if you want a persisent X window for each plot, or `pgplot_%d.ps/ps` if you want to keep postscript files in the run directory. [/null]
- **log** : Show live view of the logfile in an xterm? [f]

Not all parameters which could become command line keywords are exported as such. For example, the observatory is hardcoded at a latitude of 37 degrees (CARMA), the observing HA range is from -2 to 2 hours, etc.etc. It is fairly straightforward to promote these to command line parameters.

3.3. Input Files

The script needs only a few input files:

1. An antenna configuration file, as requested by the **ant=** keyword. These are simple ASCII files, of which many examples can be found in the directory `$MIRCAT`. A few notes on this file. The format is currently hardcoded to be in a topocentric North-East-Up frame of reference, in meters. Another popular format used by CARMA is geocentric XYZ in nanoseconds. This is only a minor change to the UVGEN interface, but not implemented in the script. If you ever need to convert one to the other, here's a simple recipe using MIRIAD:

```
uvgen ant=$antpos baseunit=1 out=$vis lat=37.28 source=$MIRCAT/point.source
puthd in=$vis/veltype value=VELO-LSR type=ascii
listobs vis=$vis > listobs.log
grep ^Ant listobs.log | awk '{print $7,$6,$8}' > $antpos.NEU
```

   Of course, far easier is it to have an option to support a `baseunit=1` option in the script. Also note the latitude of the observatory is needed to properly compute the topocentric coordinates, another reason antenna files are better kept in topocentric format.

2. A model image dataset, as requested by the **image=** keyword. This can be a two- or three-dimensional MIRIAD image dataset. Note that some care is needed in certain header elemented to ensure MIRIAD programs. Some often forgotten ones are ..... Use `puthd`

3. Optionally a mosaic pattern can be given by a small text file (the **center=** keyword), though by default a hexagonal pattern optimal for Nyquist sampling is choosen using the the **nring=** and **grid=** keywords.

3.4. Output Files

The script produces a lot of output files, but they are all created in the run directory as specified by the **dir=** keyword.

- **single** : the model, fully scaled (factor= and cell=) and RA/DEC modified
- **xy.bm** : dirty beam, usually multiple planes denoting different pointings and primary beam types. For example a 7 pointing mosaic with a OVRO+HATCREEK array will have 7 x 3 = 21 planes.
- **xy.psf** : mosaic beam computed from mospsf, the linear combination of the beam in xy.bm
- **xy.mp** : dirty map, produced by invert
- **xy.cc**: clean components (CC) (output produced by mosmem or mossdi).
- **xy.cm**: clean map, which is the dirty map (MP) minus clean components (CC) convolved by dirty beam (BM) plus the clean components (CC) convolved by the gaussian beam (PSF).
- **uv_K**: visibilities, one for each primary beam type, containing only noise
- **uv_K_demosI**: demos fields (I counts field, K primary beam types) of the model
- **uv_K_I**: visibilities of the model per field (I) and primary beam type (K)

4. MIRIAD lessons

In experiments with 5 rings (61 fields) and 3 antennae types (6 primary beam types) invert will need 366 files. Some real observations (M51) have

Scaling issues. The `select=ant(...)(....)` resulted in very long strings. select.for and uvmodel.for needed changes. The number of files processed can be large, which needed a change in uvdat.for (400 - 4000 and 2000 - 20000).

invert also issues with too many files, this was solved using an include file in the python code. (using an include file `@uv.inc`)

5. Python

5.1. Pros and Cons of Python

Pros and Cons of using the Python scripting language, instead of the more common C-shell. The historic reason of using C-shell (`csh`) for MIRIAD scripts is because this is traditionally the more common login shell on Unix systems, despite the presence of the much more programmable and powerful `bash` (Bourne Again Shell) shell. The usage of python at CARMA and many other observatories and data processing system had already led to us to investigate this option a few years ago, but users are conservative and have mostly continued to write scripts in `csh`.

Pros: Python is very programmable, can do math in the shell, simple way to make a *keyword=value* parsable command line. Once data are in python, some sophisticated plotting and analysis can be done, e.g. using Python modules `matplotlib`, ...etc.

bla bla

Cons: what you see is not what you get, so you have to construct the commands in perhaps somewhat awkward ways using the *miriad(program(arg1,arg2,...))* style and define your personal functions for MIRIAD *program*s. This can get out of hand and make it difficult to exchange code with other scripts.

```
...
def selfcal(vis):
    cmd = [
        'selfcal',
        'vis=%s' % vis,
        'options=amp'
        ]
    return cmd
...
miriad(selfcal(gains))
...
```

Example already in this script is the existence of `uvgen` and `uvgen_point`.

Although the `grepcmd` is very convenient, just like in the regular shell, these are very dangerous constructs to use, as programs change their output in unextected ways and the resulting script is usually not very robust againsts such changes.

For example, using `csh -fvx` a shell script is run in verbose mode, and can usually be easily debugged. Use missing histo example?

bla bla

5.2. Algorithm

Leaving off all fluff, the essential algorithm of the code is as follows:

```
# generate vibilities for each PB type (there are k1 of those)
  for k in range(0,k1):
    miriad(uvgen(ant,dec,harange,freq,nchan,uv_k[k],center,telescopes[k],jyperk[k],systemp))

# copy model image, and fix up the header
  miriad(imgen0(image,base2,factor))
  miriad(puthd(base2,'crval1',ra,units='hms'))
  miriad(puthd(base2,'crval2',dec,units='dms'))
  miriad(puthd(base2,'crval3',freq))
  miriad(puthd(base2,'cdelt1',-cell,units='arcsec'))
  miriad(puthd(base2,'cdelt2',cell,units='arcsec'))

# de-mosaic the model into images as each field would see it
  for k in range(0,k1):
    miriad(demos(base2,uv_k[k],demos_k[k]))

# uvmodel add each field to the previously generated PB type visibilities,
#                                and select only the appropriate baselines
  vis_all = ""
  fp = open("%s/vis.all" % rundir, "w")
  for k in range(0,k1):
    for i in range(1,npoint+1):
      vis_i = "%s_%d_%d"% (uv,k,i)
      demos_i = demos_k[k]+"%d"%i
      miriad(uvmodel(uv_k[k],demos_i,vis_i,antselect_k[k]))
      # hack systemps for weights for invert
      miriad(puthd(vis_i,'systemp',2.0*jyperk[k],type='real'))
      if len(vis_all)==0:
        vis_all=vis_i
      else:
        vis_all=vis_all + ',' + vis_i
      fp.write("%s\n" % vis_i)
  fp.close()

# add amp noise to emulate pointing errors
  if gnoise > 0:
    print "Adding pointing errors: gnoise=%g" % gnoise
    visgain = 'gains.uv'
    for k in range(0,k1):
      miriad(uvgen_point(ant,dec,harange,freq,nchan,visgain,telescopes[k],jyperk[k],systemp,gnoise))
      miriad(selfcal(visgain))
      for i in range(1,npoint+1):
         vis_i = "%s_%d_%d"% (uv,k,i)
         miriad(gpcopy(visgain,vis_i))
```

```
# invert
  vis_all = "@%s/vis.all" % rundir
  miriad(invert(vis_all, base1+".mp", base1+".bm", imsize, select))

# deconvolve
  miriad(mosmem(map1,beam1,cc,region,niters=niters))

# find out effective beam
  miriad(mospsf(beam1,psf1))
  bmaj = string.atof(grepcmd('imfit in=%s object=beam' % psf1, 'Major axis (arcsec)', 3))
  bmin = string.atof(grepcmd('imfit in=%s object=beam' % psf1, 'Minor axis (arcsec)', 3))
  bpa  = string.atof(grepcmd('imfit in=%s object=beam' % psf1, '  Position angle',    3))

# restore clean components
  miriad(restor(map1,beam1,cc,cm,[bmaj,bmin],bpa))
```

6. Model Making

easy is uvgen and patch the resulting image, that way you are ensured of proper header variable in the image (cube). Can create a velocity map using `velmodel` (***seems to have bug***) , or NEMO's `ccdvel`), and then unfold an image into cube using `velimage`.

Maps can have JY/PIXEL or JY/BEAM, or worse, something MIRIAD doesn't know how to deal with. Use histo to get the flux, use **factor=** to scale it down to the Jy you expect. That way the signal to noise is realistic for the observatory.

According to Sicking (1997) number of effective points per beam is

$$\frac{4\pi B_x B_y}{P_x P_y}$$

for given beam size $B_{x,y}$ and pixel size $P_{x,y}$.

bla bla

7. Debugging

Running a Python script such as this one can be challenging to debug. A common type of error

```
% hemosi.py image=cube6 ...
...
MIRIAD%  imframe in=run2/single out=run2/single.bigger frame=-1024,1024,-1024,1024
###: Error 256 from imframe
```

and this cryptic error can only be understood by looking at the MIRIAD error message, which is now hidden in miriad.log, e.g. viax

```
% tail run2/miriad.log
...
imframe version 1.1 31-Aug-98
### Fatal Error [imframe]:  No space left on device
```

A better way perhaps, especially if you are monitoring the progress of the script, is to use the **log=t** option, which keeps a live view of the logfile in a scrolling xterm.

8. Examples

8.1. Single Pointing Homogenous Array

The simplest example is a single pointing using a single telescope type. For simplicity we will use imgen to create a model image, but this image needs a few extra header items for hemosi to work:

```
% imgen out=point object=point
% puthd in=point/naxis  value=3
% puthd in=point/naxis3 value=1
% puthd in=point/ctype3 value=FREQ
% hemosi.py ant=$MIRCAT/carma_C.ant image=point pb=ovro,6 jyperk=43 nring=1 systemp=35
```

If the header

8.2. Single Pointing Heterogenous Array

```
hemosi.py ... pb=ovro,6,hatcreek,9,carma jyperk=42,126,73 nring=1
```

8.3. Multiple Pointing Homogenous Array

```
hemosi.py ... pb=hatcreek jyperk=43 nring=3 grid=30
```

8.4. Multiple Pointing Heterogenous Array

```
hemosi.py ...pb=ovro,6,hatcreek,9,carma jyperk=42,126,73 nring=3 grid=30
```

8.5. Pointing Error figure

```
hemosi.py ant=carma_CZ.ant image=point nring=1 dir=run1a gnoise=0
hemosi.py ant=carma_CZ.ant image=point nring=1 dir=run1b gnoise=5
hemosi.py ant=carma_CZ.ant image=point nring=1 dir=run1c gnoise=10
hemosi.py ant=carma_CZ.ant image=point nring=1 dir=run1d gnoise=20

imcat in=run1a/xy.cm,run1b/xy.cm,run1c/xy.cm,run1d/xy.cm out=gnoise.mir options=relax

implot in=gnoise.mir device=gnoise.ps/vps nxy=2,2 units=s conflag=g range=-0.02,0.05
```

This run takes about 5 minutes on a current 2008 style workstation. QUESTION: WHat is the relationship between pointing errors (e.g. via uv variables) and gnoise?

See also Figure 1.

```
Config  DEC  HA[hrs]  Beam[arcsec] scale Model_Flux,Peak  Image_Flux,Peak Residual:Rms,Max,Min[Jy] Fidelity
carma_CZ 30 -2,2,0.013 -1.000 4.42 2.3 1 1.000 1.000 3.222 1.037 0.004 0.019 -0.047 258.109   (run1a)
carma_CZ 30 -2,2,0.013 -1.000 4.42 2.3 1 1.000 1.000 3.507 1.040 0.004 0.021 -0.050 248.593   (run1b)
carma_CZ 30 -2,2,0.013 -1.000 4.42 2.3 1 1.000 1.000 4.157 1.048 0.005 0.023 -0.058 220.073   (run1c)
carma_CZ 30 -2,2,0.013 -1.000 4.42 2.3 1 1.000 1.000 7.136 1.085 0.007 0.026 -0.103 152.274   (run1d)
```

8.6. Saturn images - CARMA memo 27

In CARMA memo 27 a 603x603 0.1 arcsec pixel VLA image of Saturn was introduced and used. Peak is 0.036 Jy/pixel, total flux is 732.1 Jy. Here we will show how to use hemosi to run an example from this memo.

9. Design your own array

The following changes are probably needed if you want to design a new array:

- The antenna file is simply an ASCII file, currently in topocentric East-North-Up (ENU) with meters as the units this occurs if you use imgen
- If there is a new antennae that MIRIAD does not know about , modify `$MIRSUBS/obspar.for` and recompile the library, and if needed all programs. Then test your new telescope:

```
telepar telescop=foobar
```

In the example in the next subsection we have cheated and bypassed this by only mapping the inner few beams (e.g. for VLBA/VLBI type observations).

9.1. Micro managing your baselines

Here is an example where each baseline was given different characteristic. Although formally each antennae would need a different size, in this example we choose to use a nominal antennae and only map the inner portions of this array where the primary beam did not matter.

```
nchan = 44
step  = 0.02
jyperk = [6.6, 1.0, 1.8, 1.8, 2.0, 0.6, 1.8]

def antpos(file):
    fp = open(file,'r')
    lines = fp.readlines()
    fp.close()
    x=[]
    y=[]
    for line in lines:
        w = line.split()
        x.append(float(w[0]))
        y.append(float(w[1]))
    return (len(x),x,y)

(nant,x,y) = antpos('merlin.antpos')

up = open("uvall.inc","w")
visno = 0
for n in range(nchan):
    f = 1.0 + step*n
    bl = 0
    for i in range(nant):
        for j in range(i+1,nant):
            # now working on baseline i+1,j+1, for channel n
            # the order is 1,2 1,3 1,4 ....    6,7
            visno = visno + 1
            out = 'uv.%d' % visno
            ant = 'ant.%d' % visno
            fp = open(ant,"w")
            fp.write("%g %g 0.0\n" % (f*x[i],f*y[i]))
            fp.write("%g %g 0.0\n" % (f*x[j],f*y[j]))
            fp.close()
            date = '79jan01.%03d' % visno
```

```
            cmd = 'uvgen source=Taurus freq=4.2,0.01 ant=%s out=%s harange=-4.5,
4.5,0.001 baseunit=-3.33564 lat=52 corr=1,1,0,10 systemp=25.0 jyperk=%g radec=0.
,30. time=%s' % (ant,out,(jyperk[i]*jyperk[j]),date)
            print cmd
            # cmd = cmd + ' > /dev/null'
            os.system(cmd)
            bl = bl + 1
            up.write("%s\n" %out)

up.close()
os.system('uvcat vis=@uvall.inc out=uvall')
```

Several comments:

1. there will be nant*(nant-1)*nchan uv files, thus uvcat needs a macro include file, as the wildcard usage will not have enough string space
2. uvgen needs to use a different date for each file, since that is used as the seed for the random number generator. Amusing results were obtained if the same seed was used.

10. Sample Error Messages

Typical failure of **hemosi** is a message

```
###: Error 256 from uvmodel
```

which means you should look at the last few lines in the `miriad.log` file in the run directory. The following error messages are found to be common ones:

- **### Fatal Error [demos]: Frequency could not be determined, in pbInit** : this occurs if you use imgen
- **### Fatal Error [uvmodel]: No visibilities selected** : typically happens if you have specified more antennas then there are in the antenna file. Check your ant=

11. TODO

should we keep running it from local directory and forcing all these rundir references, vs. chdir (dangerous if it's not a local subdirectory, need to keep remembering abs path)

Instead of the current method to get numbers out of a miriad program's output

```
...
  bmaj = string.atof(grepcmd('imfit in=%s object=beam' % psf1, 'Major axis (arcsec)', 3))
```

it would be better to have a contractual agreement what kind of number miriad will output. As a suggestion, suppose – similar to the **key** routines to input values from the command line – we would have a mechanism to output values to the calling program. Let us call them the **yek** routines:

```
Miriad.imfit(in=%s,object=beam,yek='bmaj,bmin,bpa')
bmaj=yek('bmaj')
bmin=yek('bmin')
bpa=yek('bpa')
```

12. Conclusion

Python is great! Gotta love that new PGPLOTDeviceName() class where you can use `device=pgplot%d.ps/vps` names.

The brittle places in shell scripts where textual output is captured, parsed and decided upon is isolated in python functions, aiding the debugging. Unless there are contracts betweem Miriad routines and such parsers, this is a very dangerous path to write pipeline code. However this is also an excellent place to make our legacy code ("code without tests") more robust.

REFERENCES

MIRIAD: http://carma.astro.umd.edu/miriad

NEMO: http://www.astro.umd.edu/nemo

Sault, R. J., Teuben, P. J., and Wright, M. C. H. 1995. *A Retrospective View of MIRIAD* - 1995ASPC...77..433S

Sicking, F. 1997. PhD Thesis U of Groningen. (Chapter 3, Appendix A, p59)

Wright, M.C.H. - "Antenna Configuration Evaluation" (CARMA memo 24, May 2004)

Wright, M.C.H. - "Imaging Simulations with CARMA-23" (CARMA memo 27, July 2004)

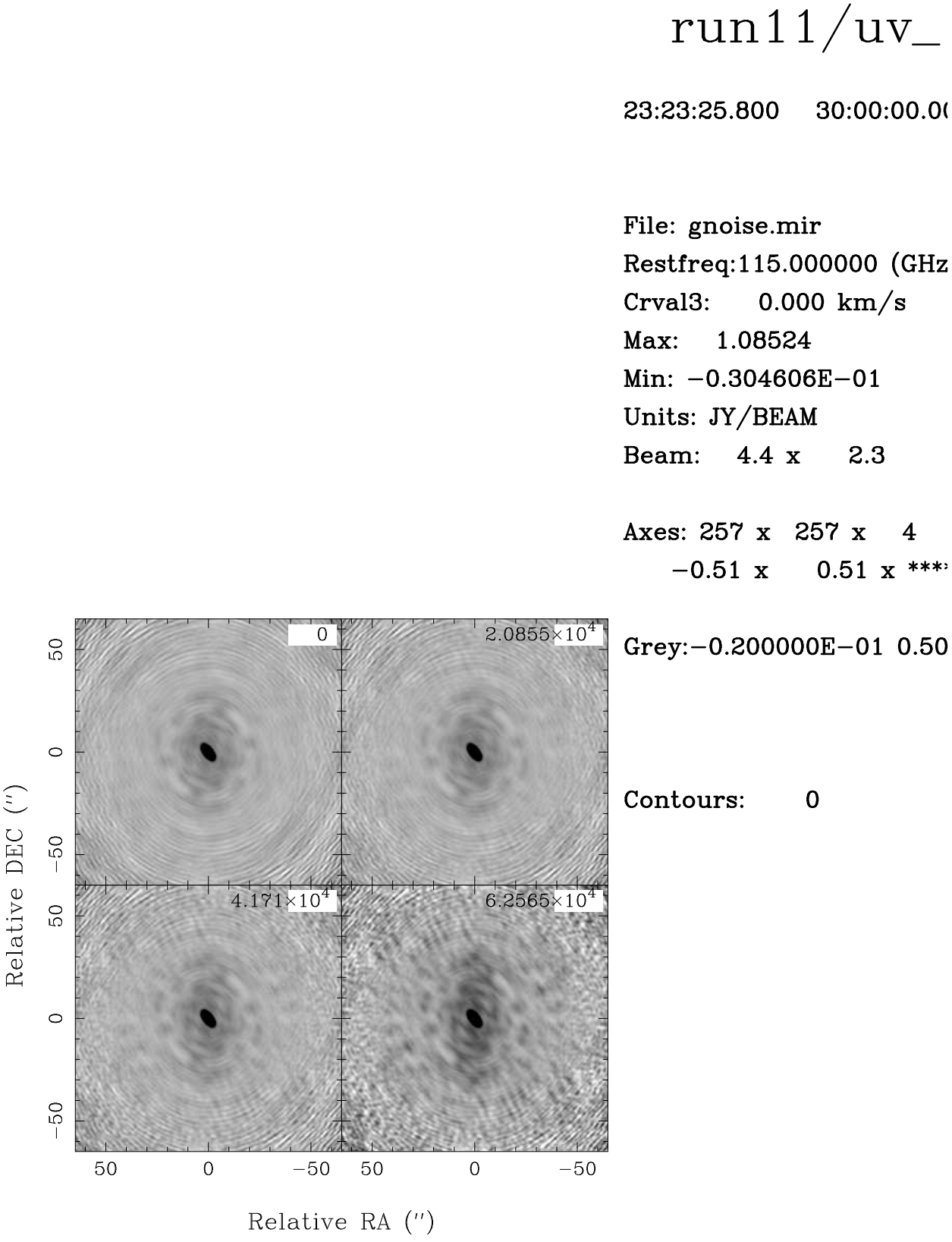

Fig. 1.— Point source in CARMA-23 CZ imaging array (4.4 by 2.3 arcsec) with gnoise=0, 5, 10 and 20% resp.