Imaging

NAASC data analysis workshop

Jim Braatz

Based on material from David Wilner, Scott Schnee, Remy Indebetouw

Atacama Large Millimeter/submillimeter Array
Expanded Very Large Array
Robert C. Byrd Green Bank Telescope
Very Long Baseline Array

Overview

- Goals of this talk:
  - Gain some intuition for interferometric imaging
  - Introduce deconvolution in CASA (clean)
  - Introduce various imaging methods available in CASA

- More formal description of imaging available in NRAO Synthesis Imaging Workshop lectures

From Sky Brightness to Visibility

1. An interferometer measures the interference pattern observed by pairs of apertures.
2. The interference pattern is directly related to the source brightness. In particular, for small fields of view the complex visibility, V(u,v), is the 2D Fourier transform of the brightness on the sky, T(x,y)

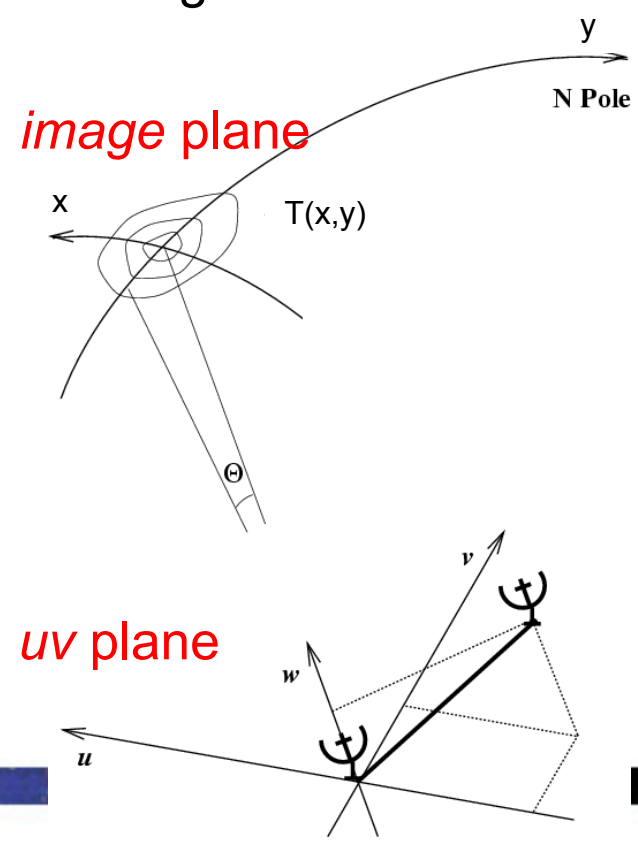
(van Cittert-Zernike theorem)

Fourier space/domain

$$V(u,v) = \int\int T(x,y) e^{2\pi i(ux+vy)} dxdy$$

$$T(x,y) = \int\int V(u,v) e^{-2\pi i(ux+vy)} dudv$$

Image space/domain

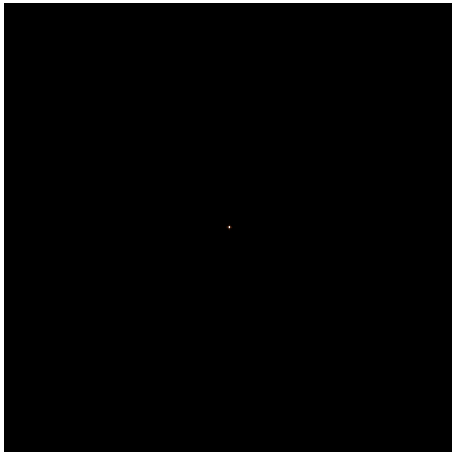


Some 2D Fourier Transform Pairs

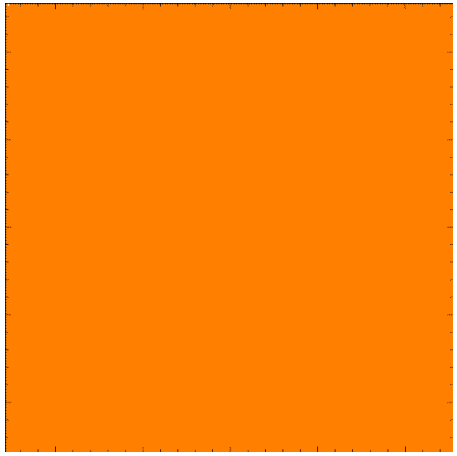
T(x,y)

Amp{V(u,v)}

δ Function

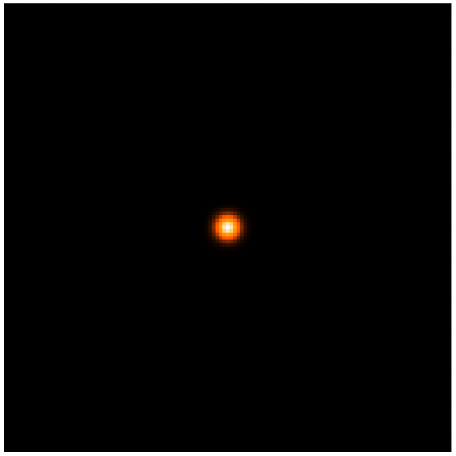


⇌

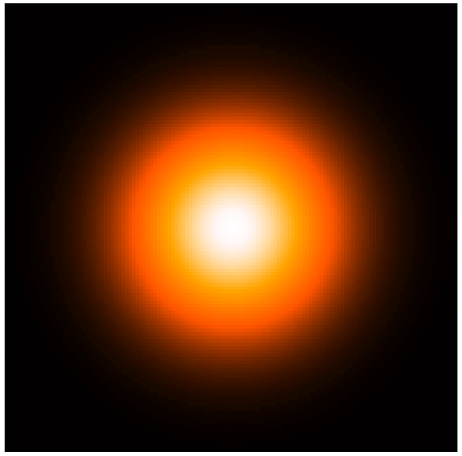


Constant

Gaussian



⇌



Gaussian

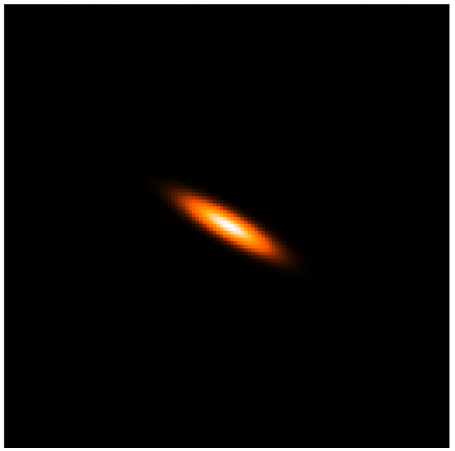
narrow features transform to wide features (and vice-versa)

More 2D Fourier Transform Pairs

T(x,y)

Amp{V(u,v)}

elliptical Gaussian

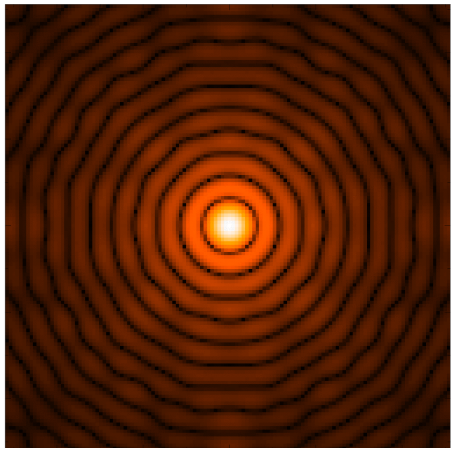


$\rightleftharpoons$

elliptical Gaussian

Disk

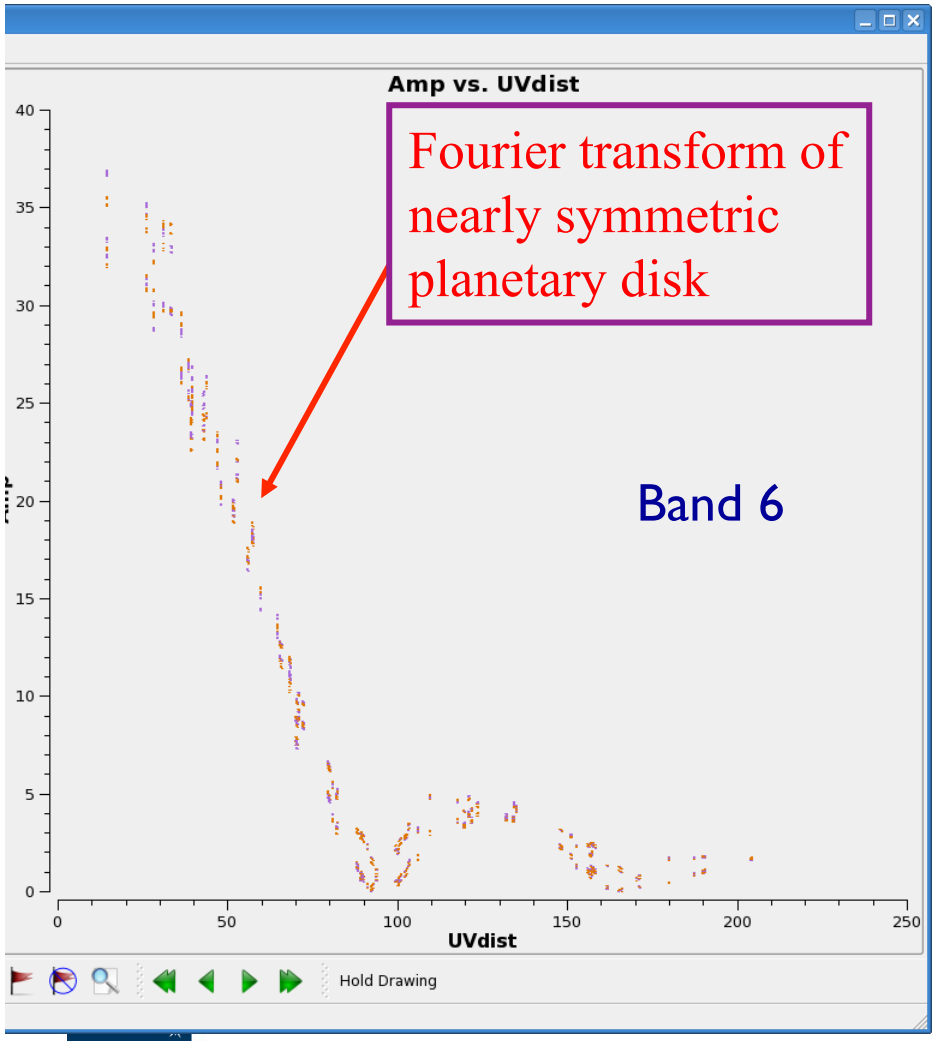
$\rightleftharpoons$

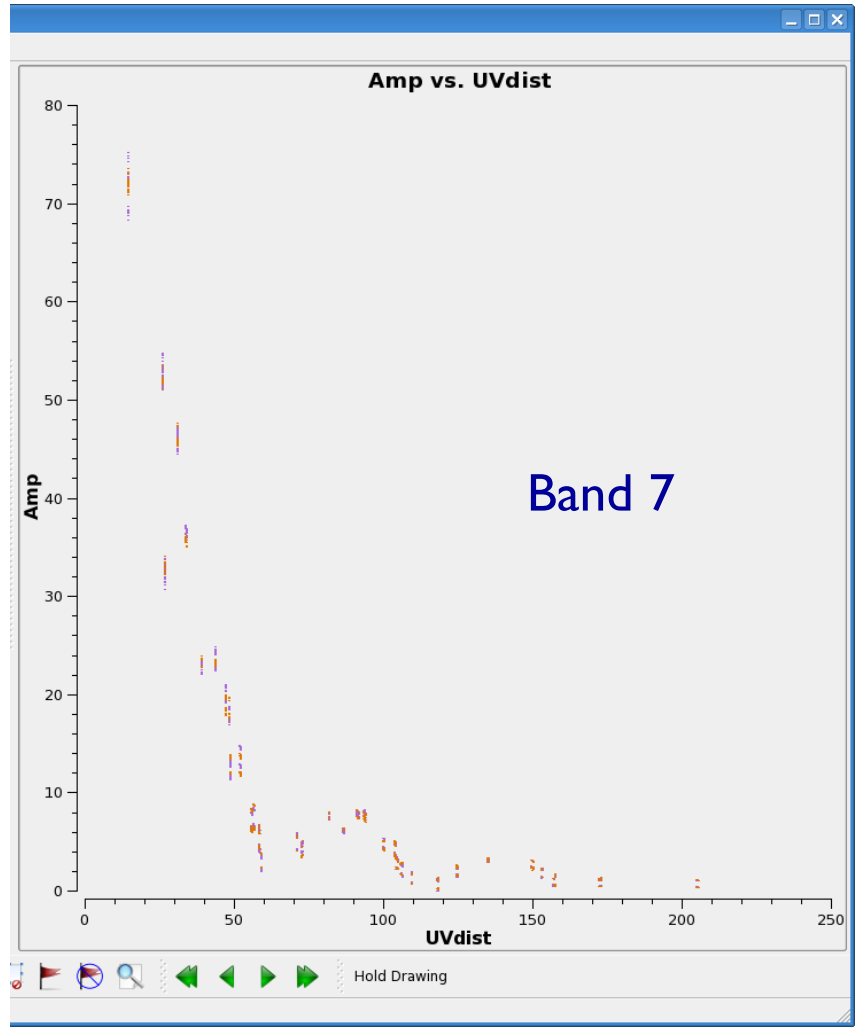


Bessel

sharp edges result in many high spatial frequencies

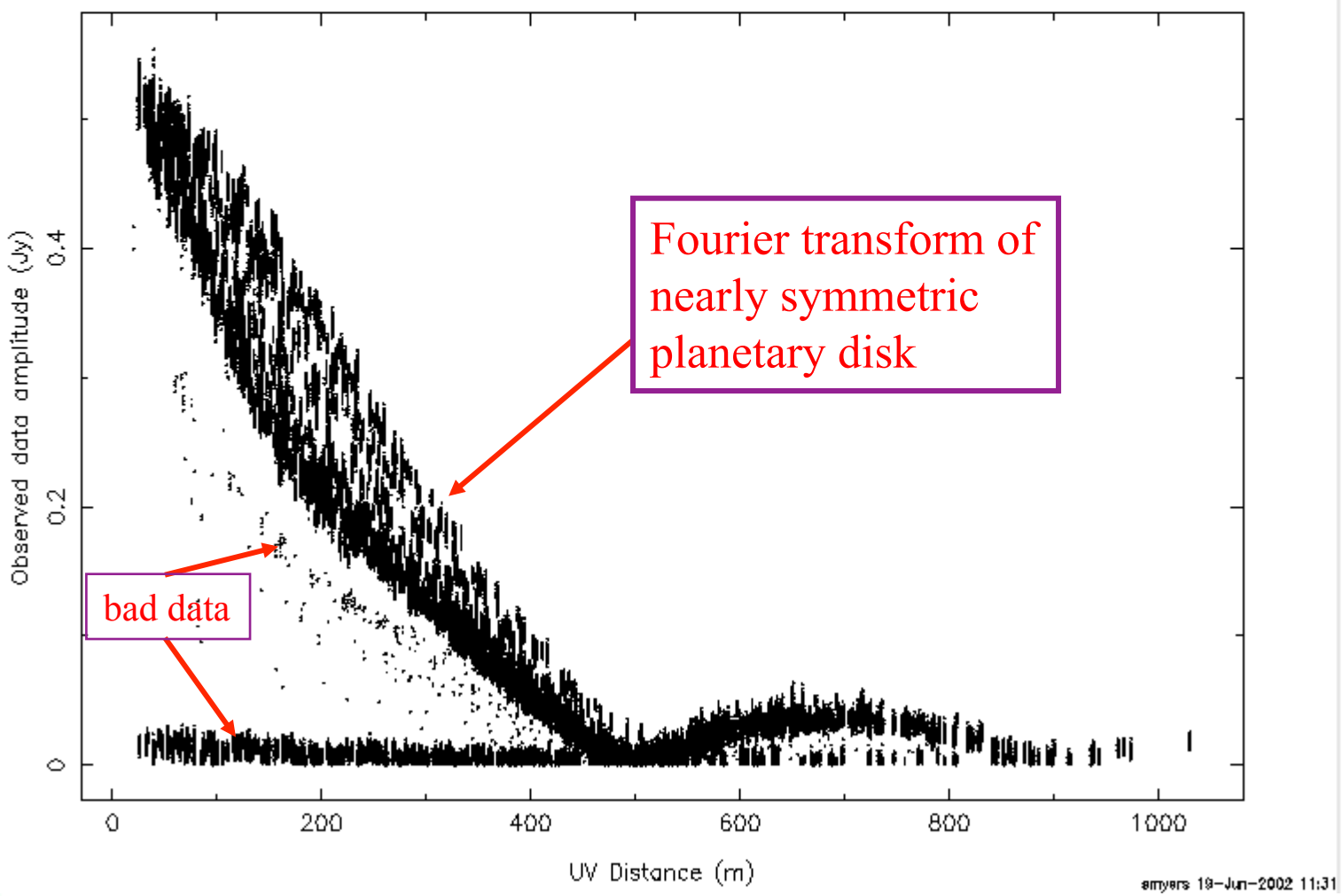
ALMA observes planetary disk





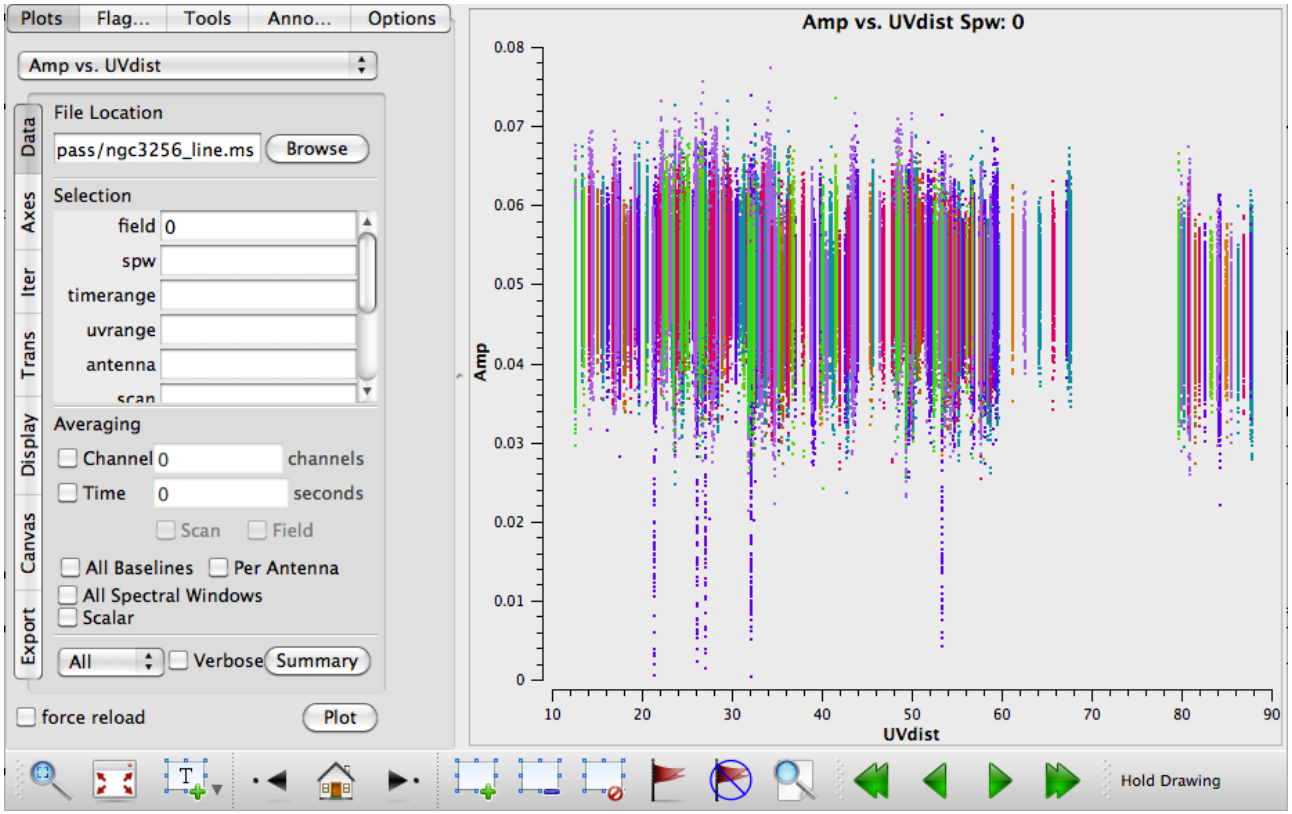
Real Example: VLA observes Jupiter

- A 6cm VLA observation of Jupiter:



How to analyze (imperfect) interferometer data

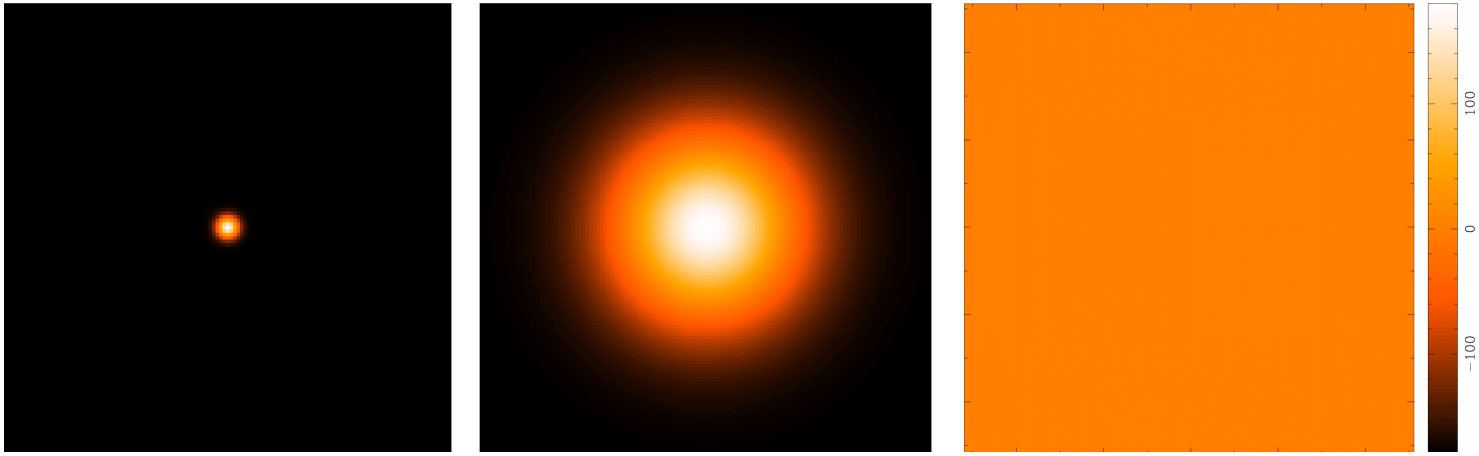
- uv plane analysis
  - Ok for "simple" sources, e.g. point sources, disks
  - Recall our BP calibrator from yesterday



Amplitude and Phase

- complex numbers: (real, imaginary) or (amplitude, phase)
  - amplitude tells “how much” of a certain frequency component
  - phase tells “where” this component is located

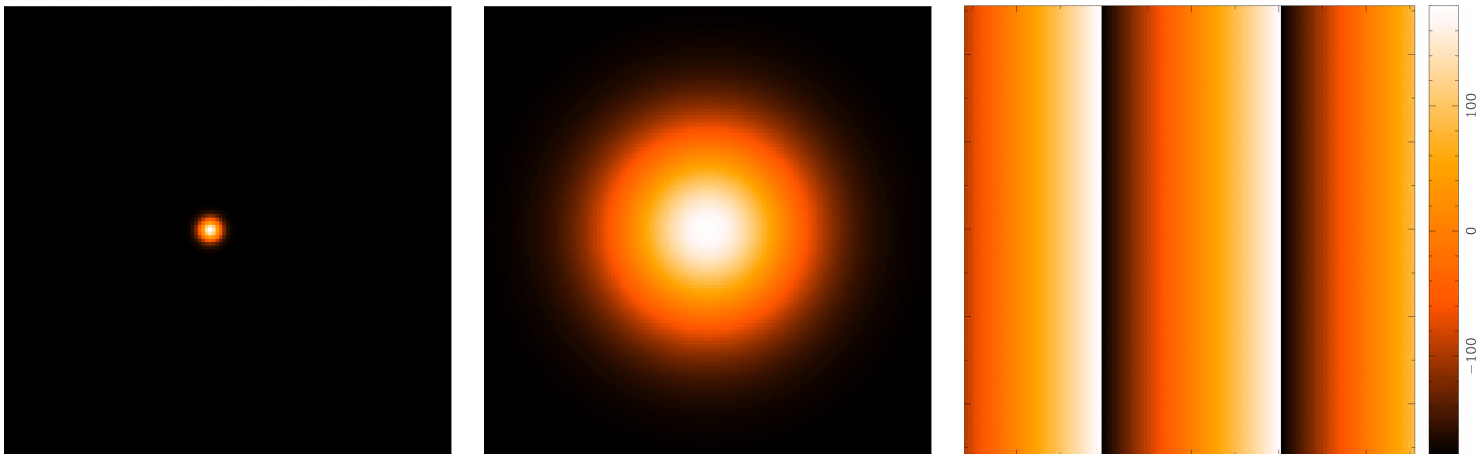
T(x,y) Amp{V(u,v)} Pha{V(u,v)}



Amplitude and Phase

- complex numbers: (real, imaginary) or (amplitude, phase)
  - amplitude tells “how much” of a certain frequency component
  - phase tells “where” this component is located

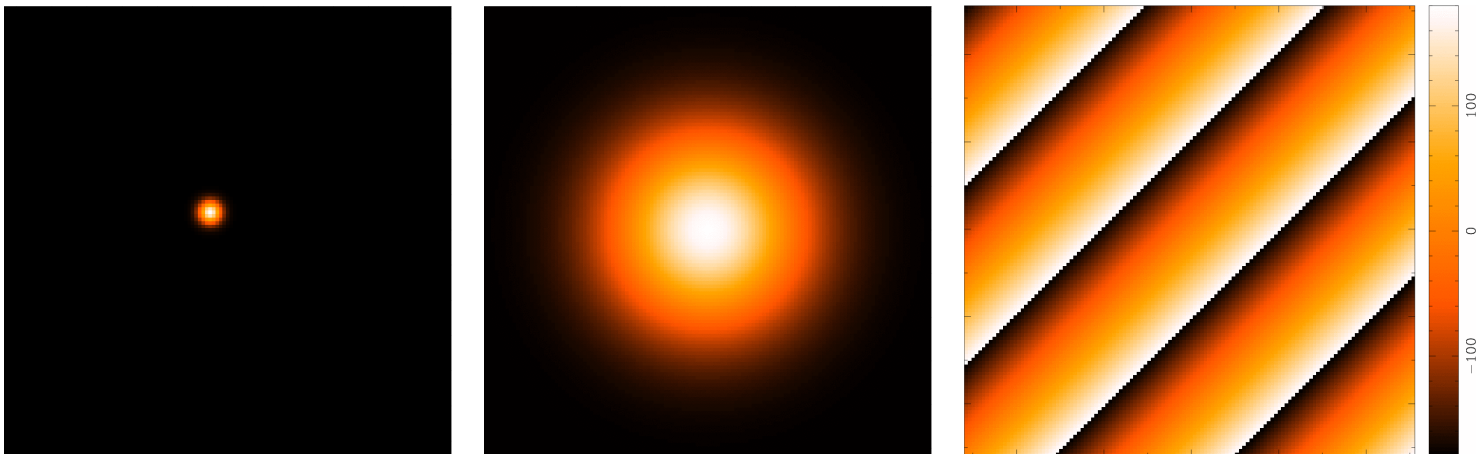
T(x,y) Amp{V(u,v)} Pha{V(u,v)}



Amplitude and Phase

- complex numbers: (real, imaginary) or (amplitude, phase)
  - amplitude tells “how much” of a certain frequency component
  - phase tells “where” this component is located

T(x,y) Amp{V(u,v)} Pha{V(u,v)}



Dirty Images from a Dirty Beam

- We sample Fourier domain at discrete points

  $B(u,v) = \sum_k (u_k, v_k)$

- the inverse Fourier transform is

  $T^D(x,y) = FT^{-1}\{B(u,v) \times V(u,v)\}$

- the convolution theorem tells us

  $T^D(x,y) = b(x,y) \otimes T(x,y)$

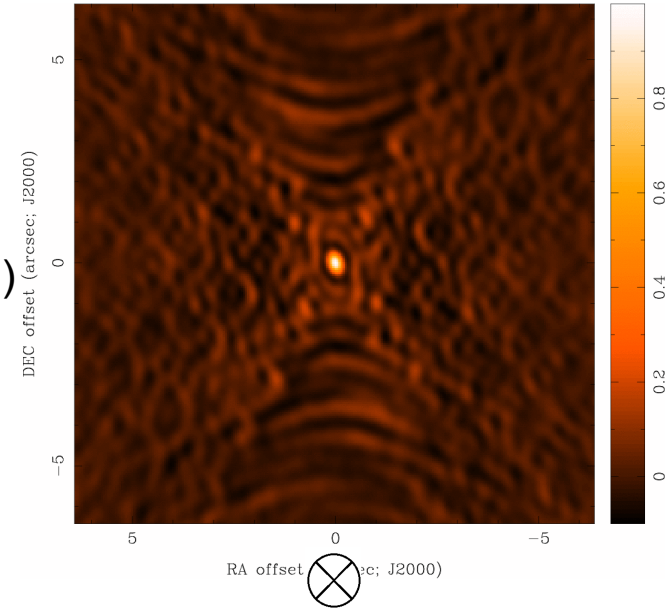
  where $b(x,y) = FT^{-1}\{B(u,v)\}$ (the point spread function)

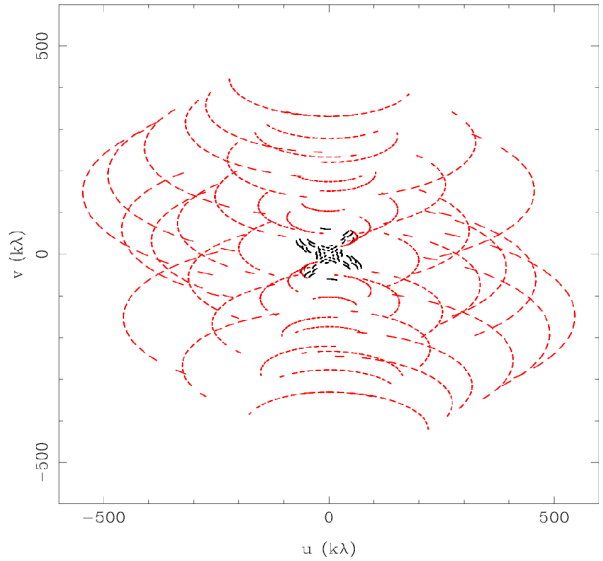
Fourier transform of sampled visibilities yields the true sky brightness convolved with the point spread function

(the "dirty image" is the true image convolved with the "dirty beam")

Dirty Beam and Dirty Image

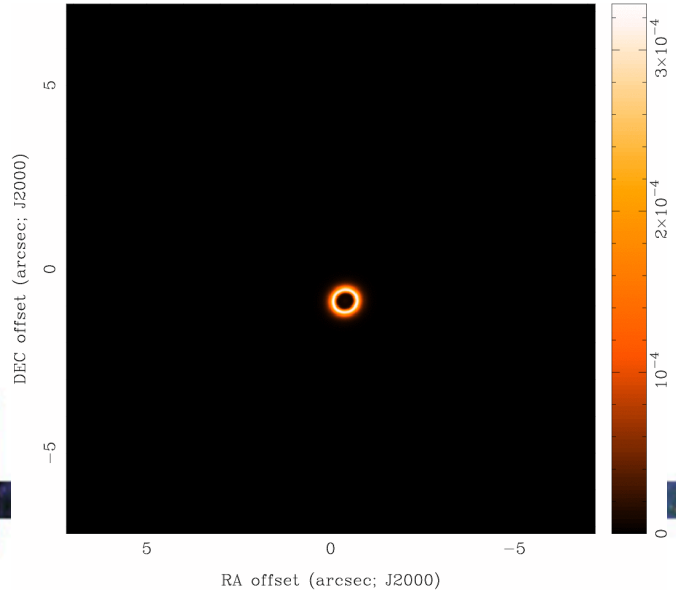
b(x,y)
(dirty beam)

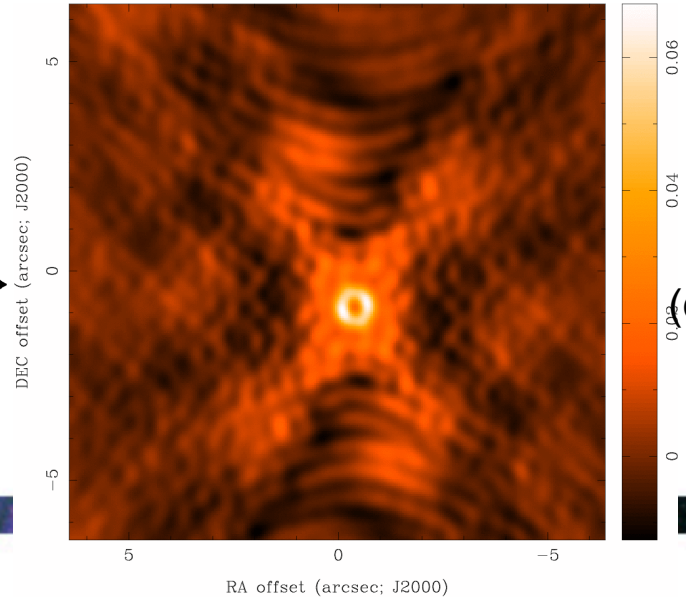




B(u,v)

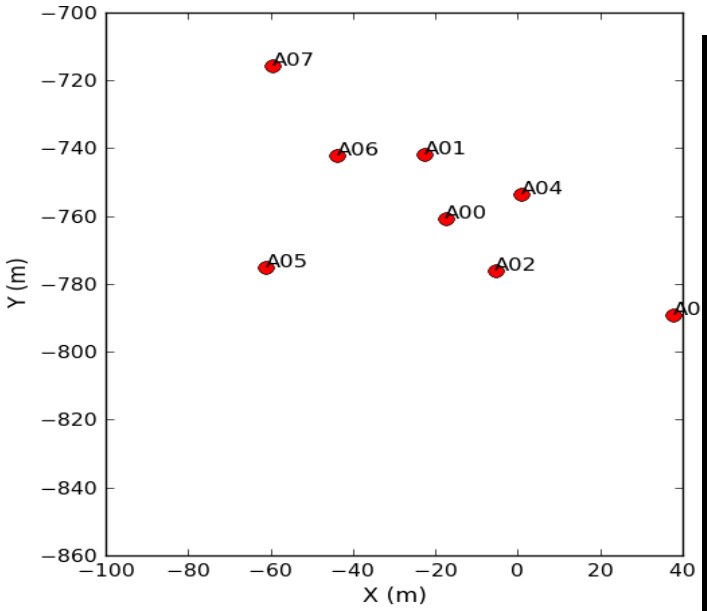
T(x,y)

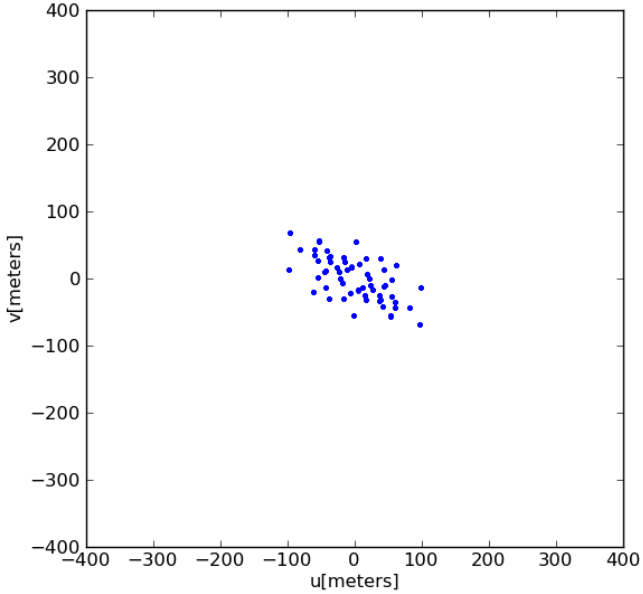


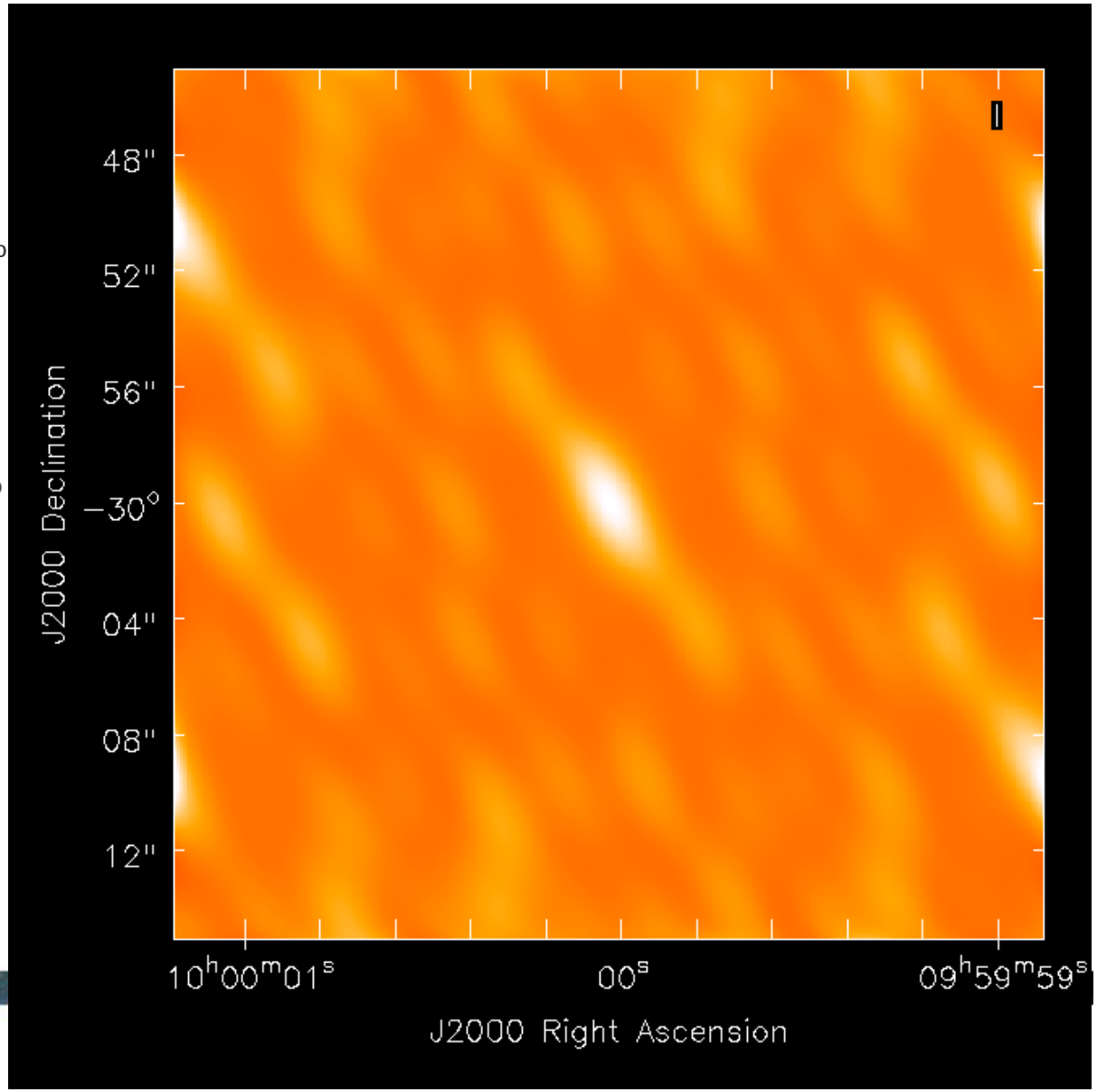


$T^D(x,y)$
(dirty image)

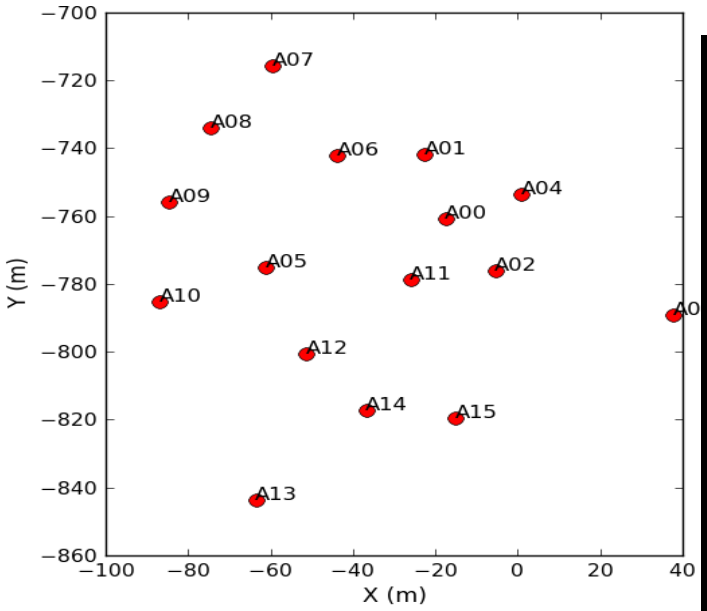
8 Antennas

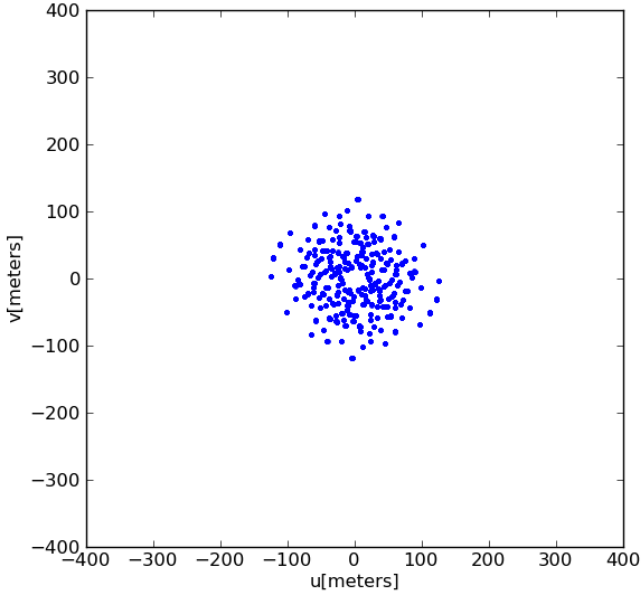

A07
A06
A01
A04
A00
A05
A02
A0
Y (m)
X (m)

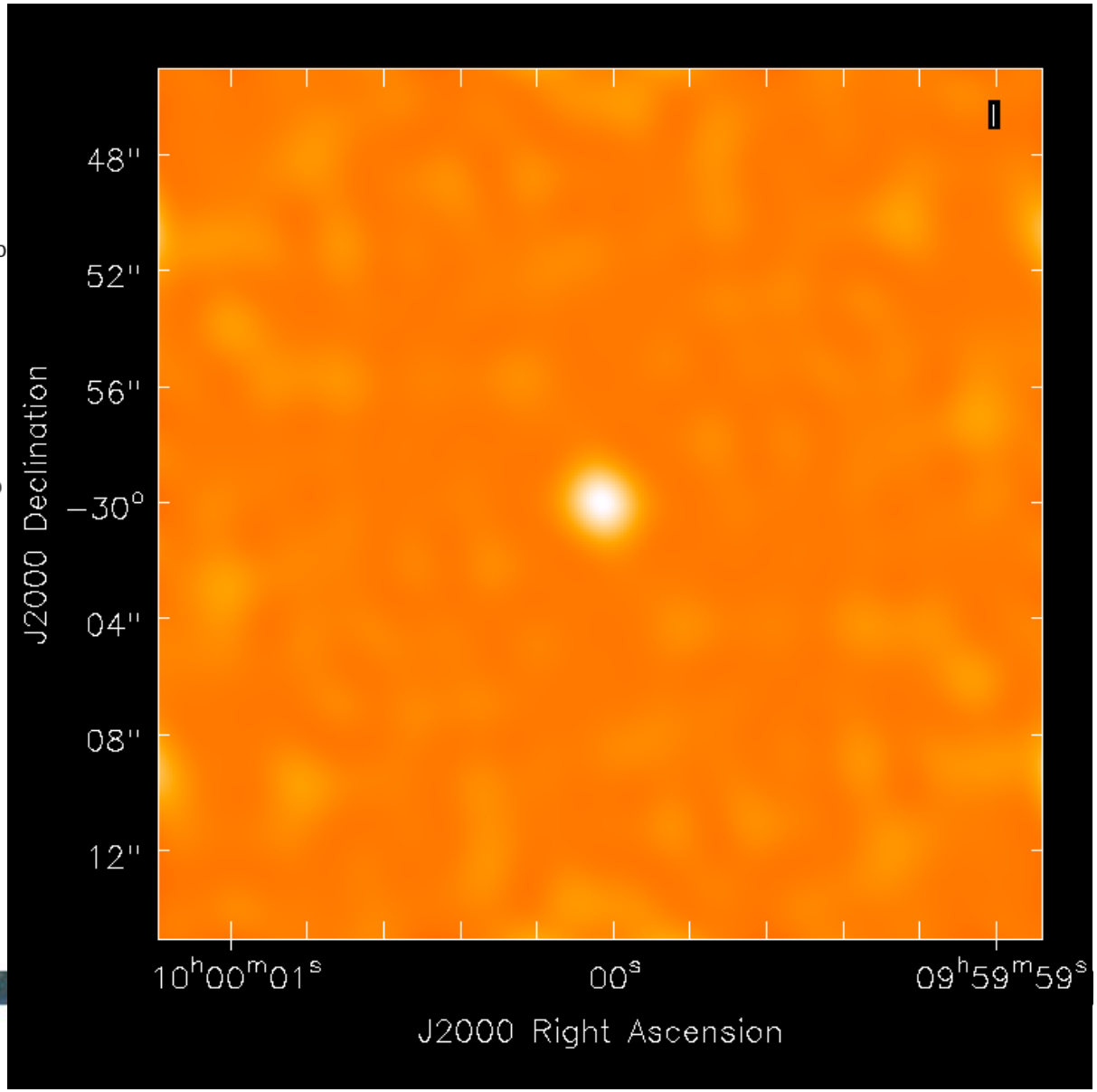

v[meters]
u[meters]


J2000 Declination
J2000 Right Ascension

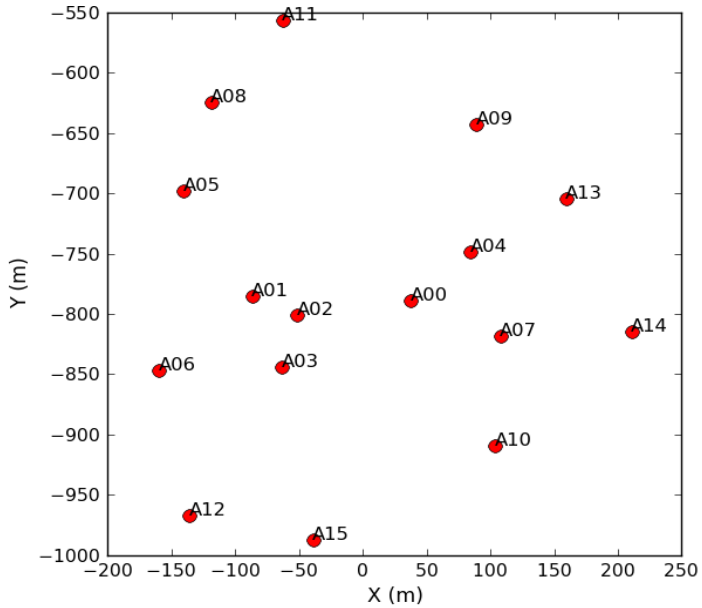
16 Antennas - Compact

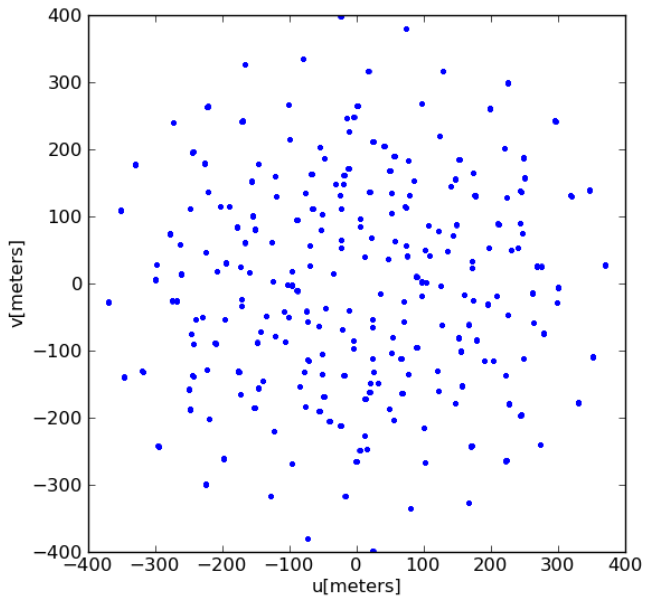


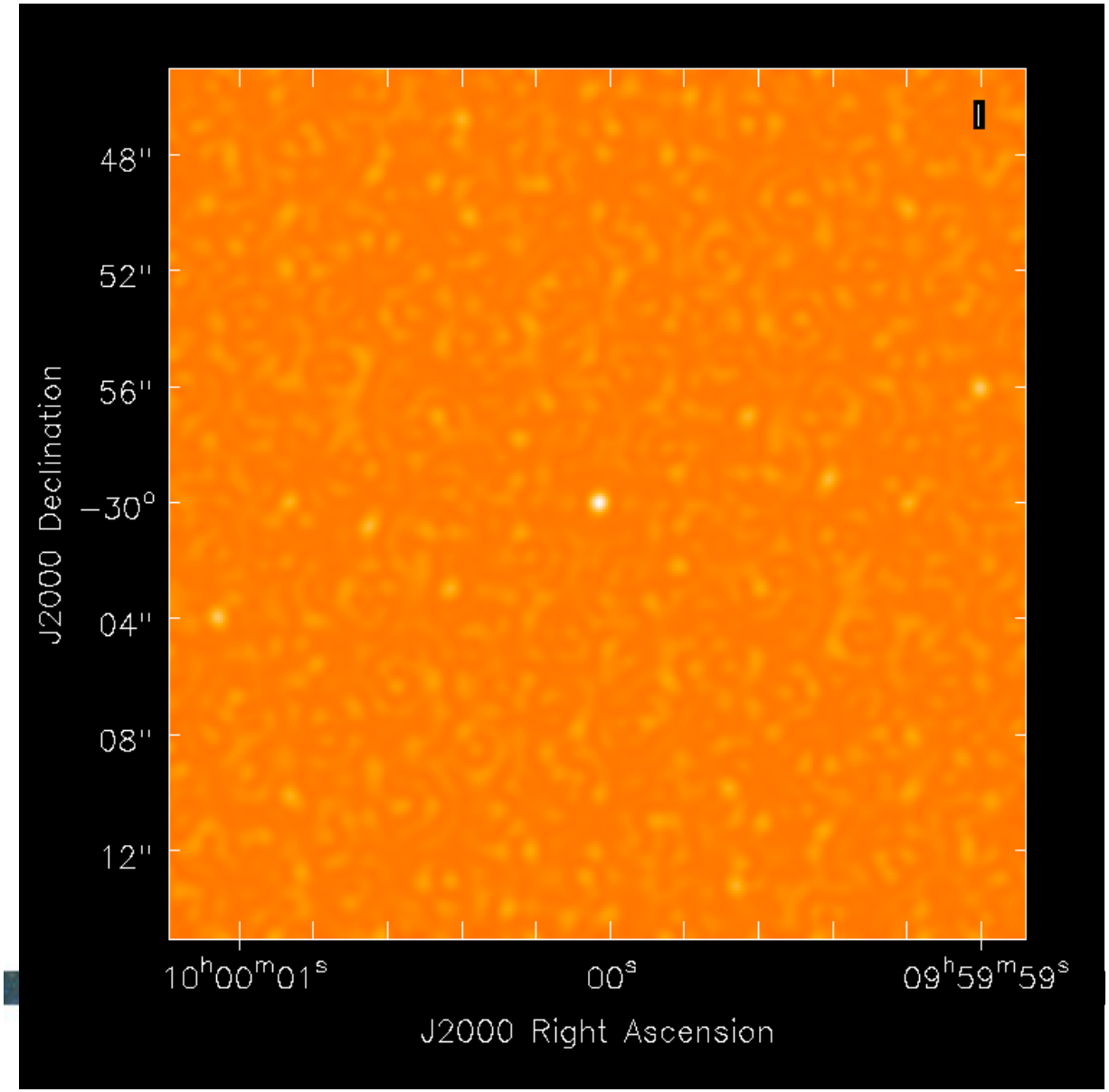




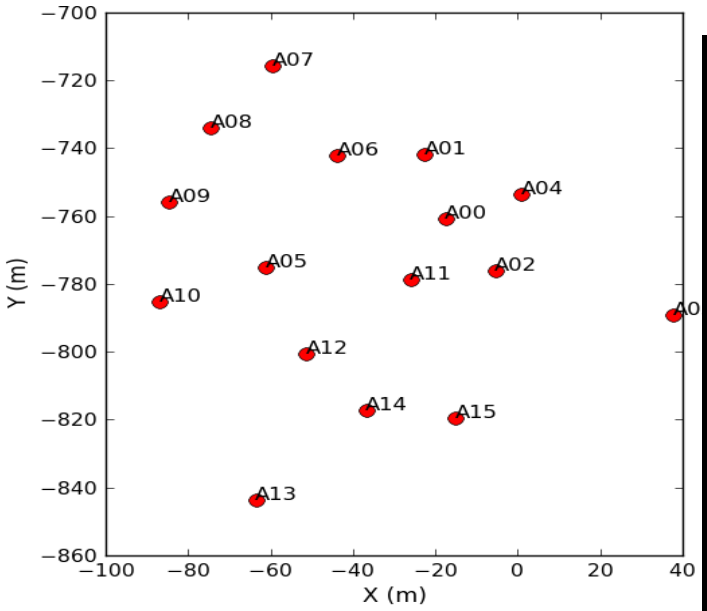
16 Antennas - Extended

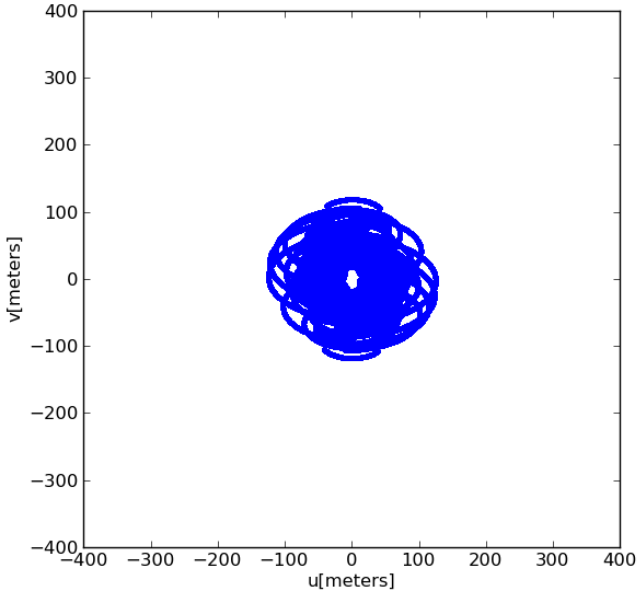


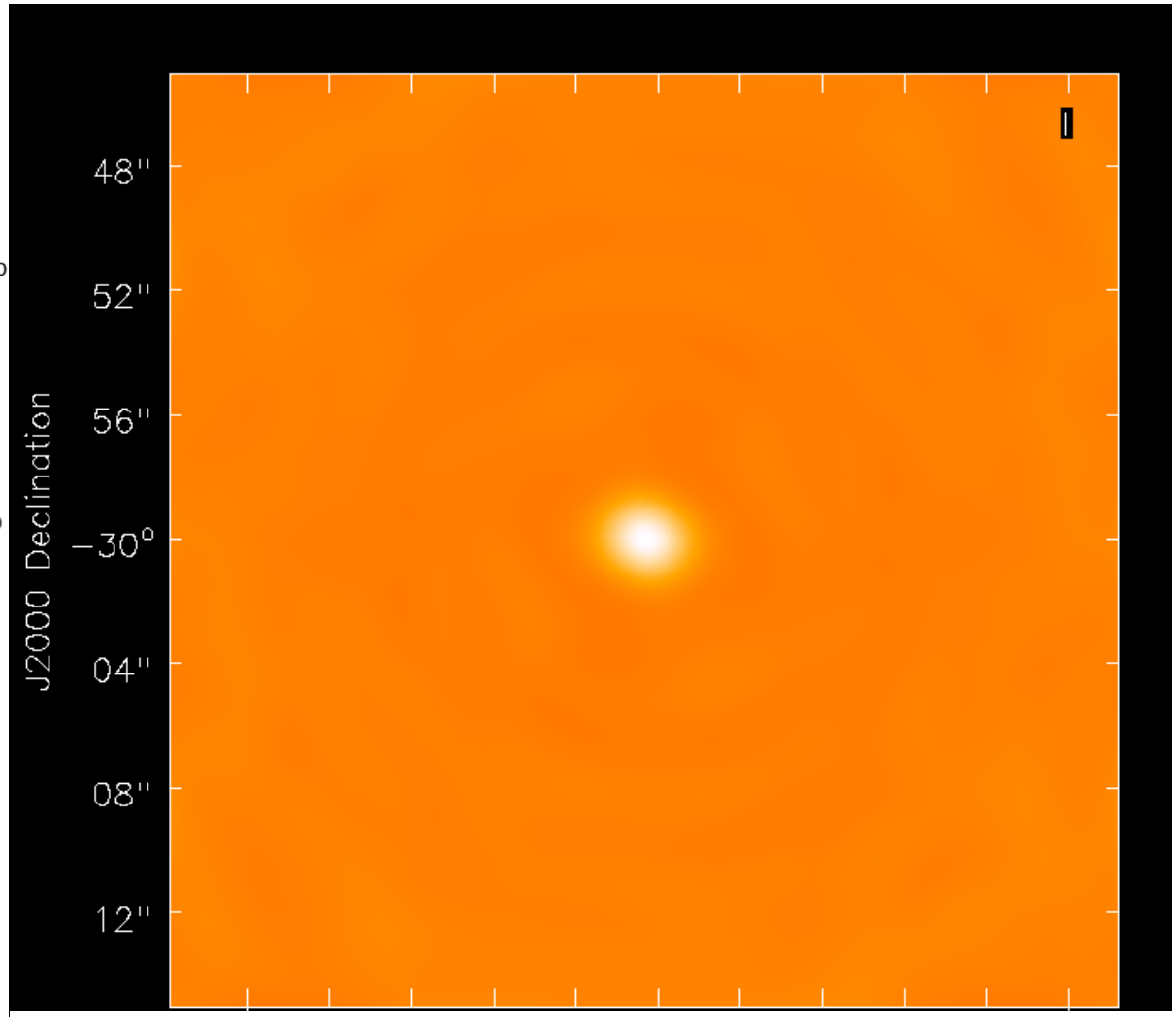




16 Antennas – Compact – 8 hours





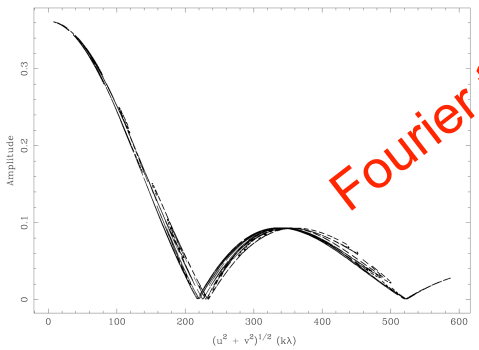


- Sky response = PSF = dirty beam = Fourier transform of uv distribution

How to analyze (imperfect) interferometer data?

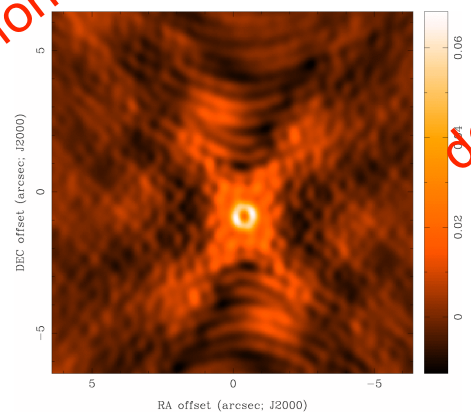
- image plane analysis
  - dirty image $T^D(x,y)$ = Fourier transform $\{V(u,v)\}$
  - deconvolve $b(x,y)$ from $T^D(x,y)$ to determine (model of) $T(x,y)$

visibilities



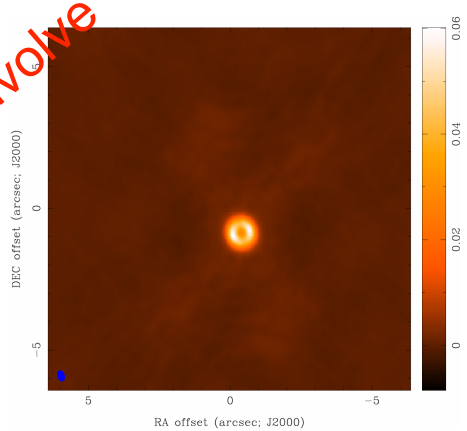
Fourier transform ⇌

dirty image



deconvolve →

sky brightness

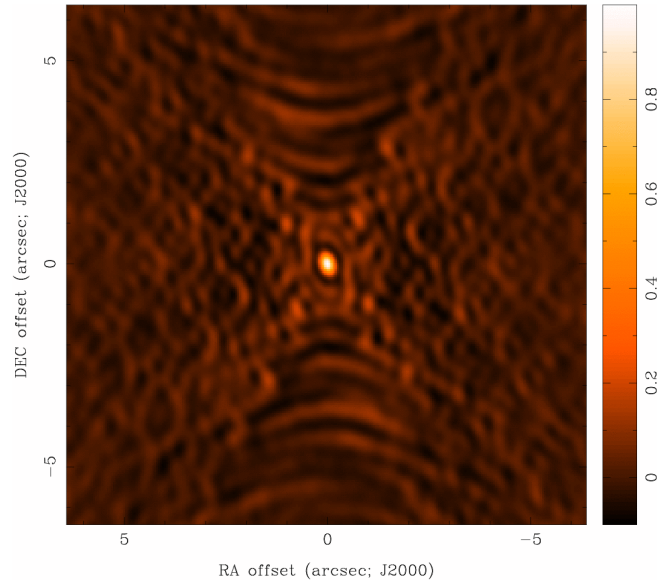


Basic CLEAN Algorithm

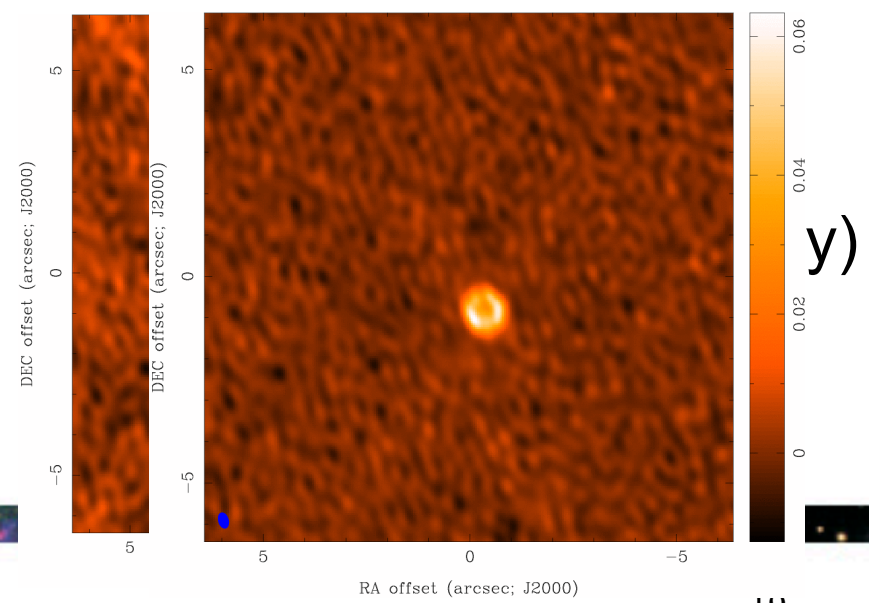
① Initialize a *residual* map to the dirty map

1. Start loop
2. Identify strongest feature in *residual* map as a point source
3. Add this point source to the clean component list
4. Convolve the point source with b(x,y) and subtract a fraction g (the loop gain) of that from *residual* map
5. If stopping criteria not reached, do next iteration

② Convolve *Clean component* (cc) list by an estimate of the main lobe of the dirty beam (the "Clean beam") and add *residual* map to make the final "restored" image



b(x,y)



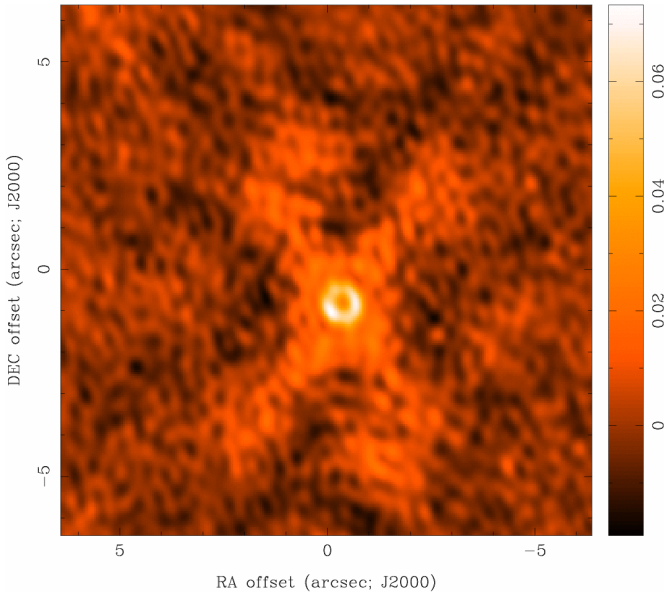
y)

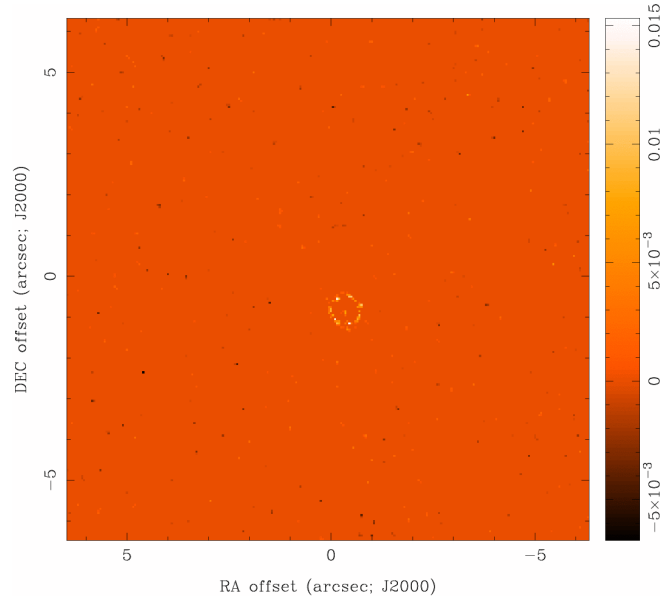
Basic CLEAN Algorithm (cont)

- stopping criteria
  - r*esidual* map max < multiple of rms (when noise limited)
  - r*esidual* map max < fraction of dirty map max (dynamic range limited)
  - max number of clean components reached (no justification)
- loop gain
  - good results for g ~ 0.1 to 0.3
  - lower values can work better for smoother emission, g ~ 0.05
- easy to include *a priori* information about where to search for clean components ("clean boxes")
  - very useful but potentially dangerous!

CLEAN

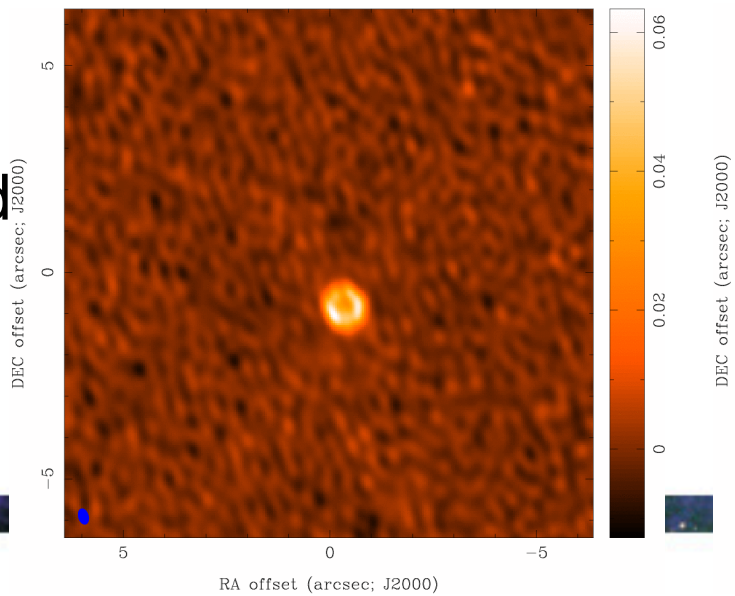
$T^D(x,y)$

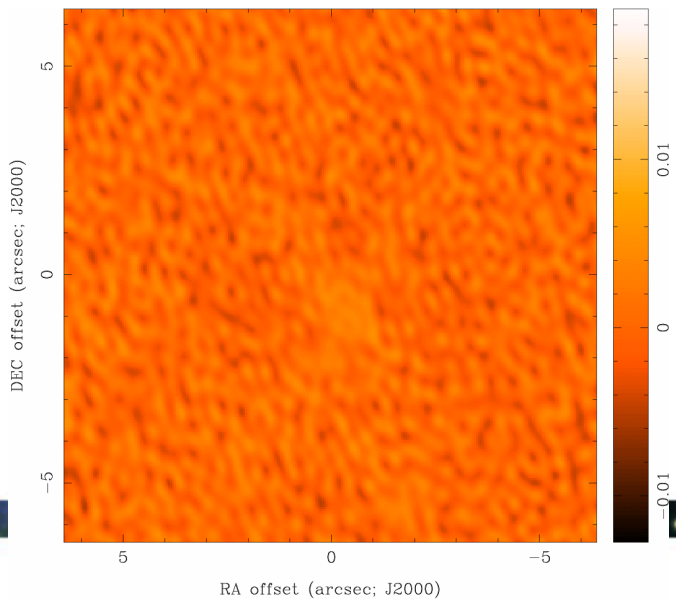




CLEAN model

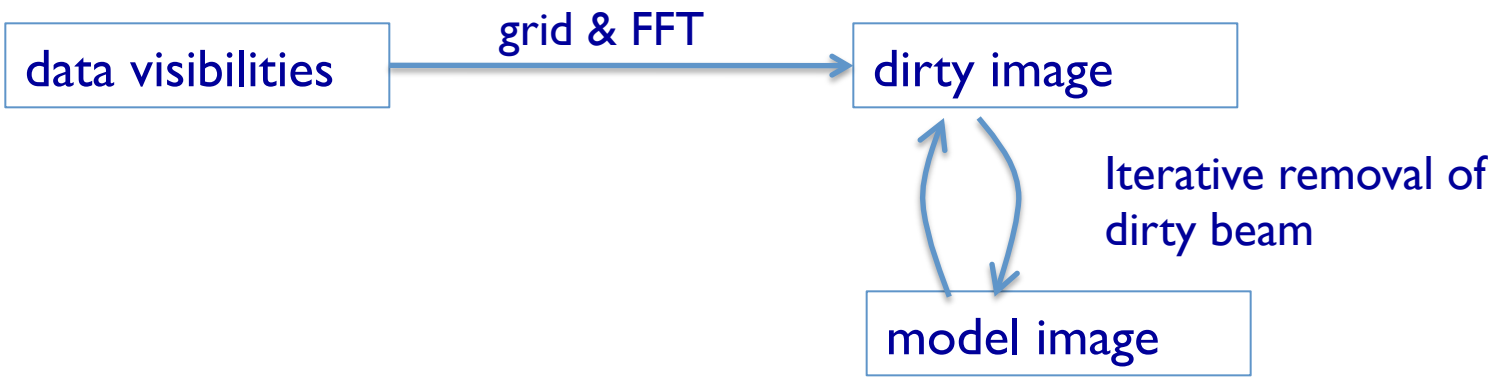
restored image





residual map

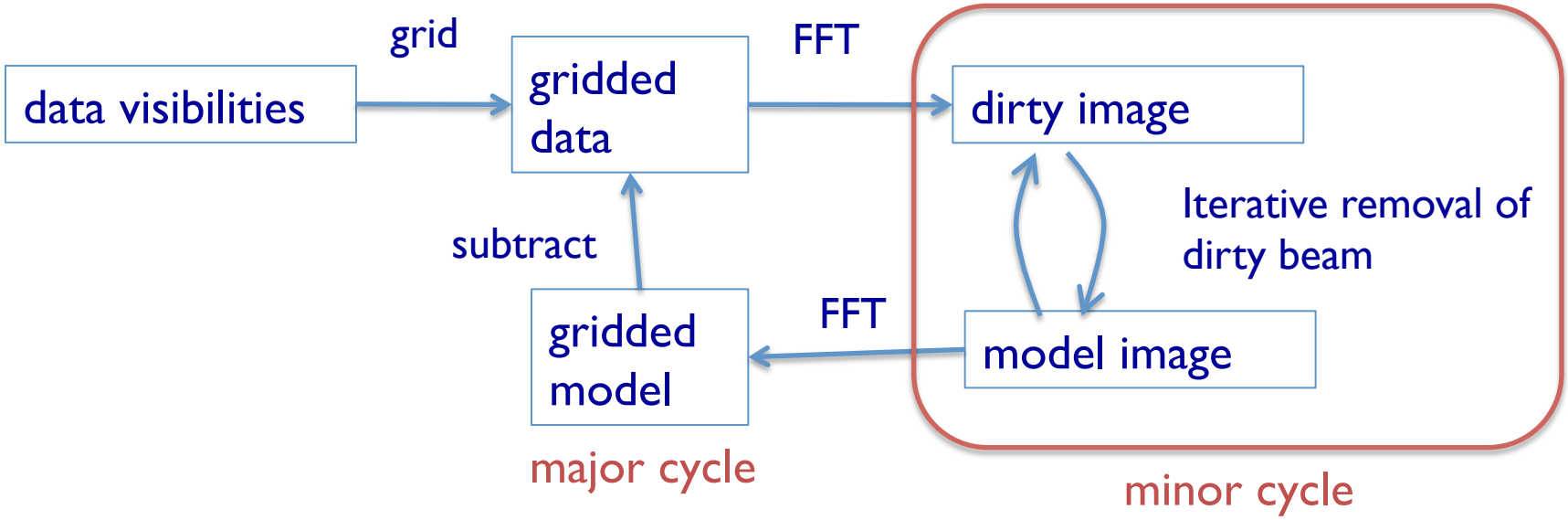
Deconvolution algorithms : Hogbom



subtracts full PSF in image domain

For complex images, errors can build

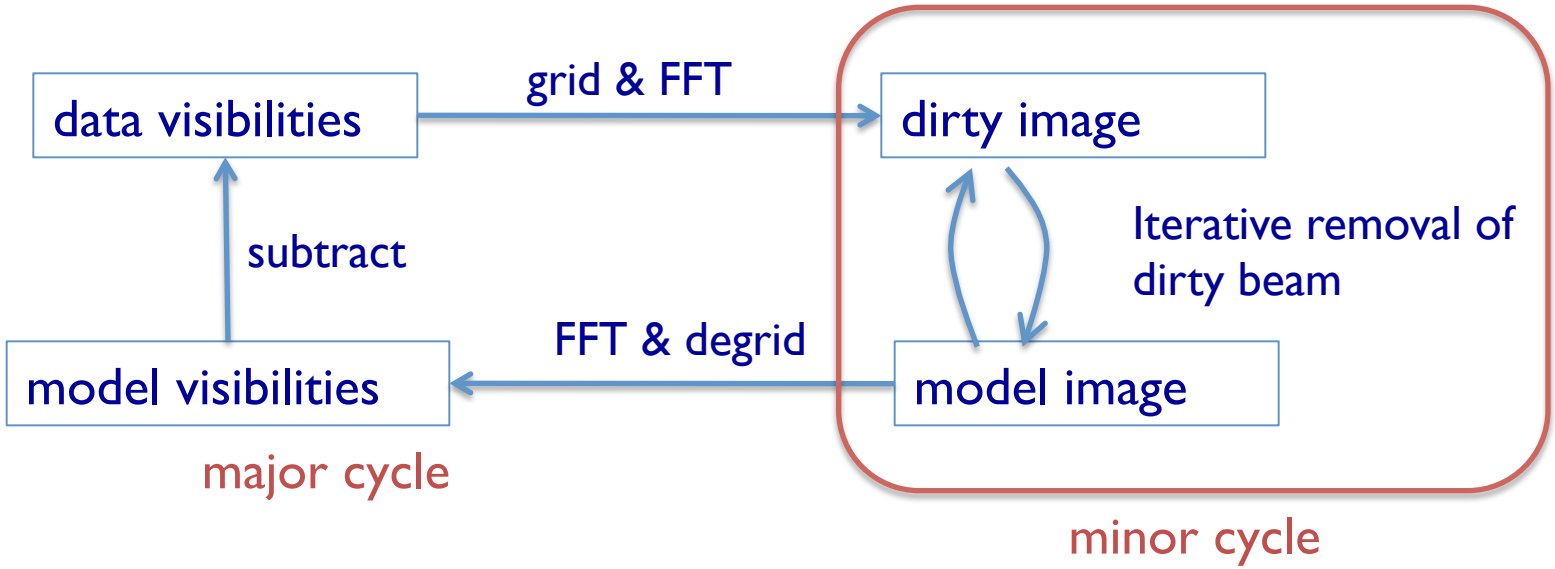
Deconvolution algorithms : Clark



subtracts truncated PSF in image domain

periodically subtracts from gridded data in uv domain

Deconvolution algorithms: Cotton-Schwab



Cotton-Schwab (csclean):

- subtracts truncated PSF in image domain
- major cycle subtracts from full visibilities
- significant I/O per major cycle

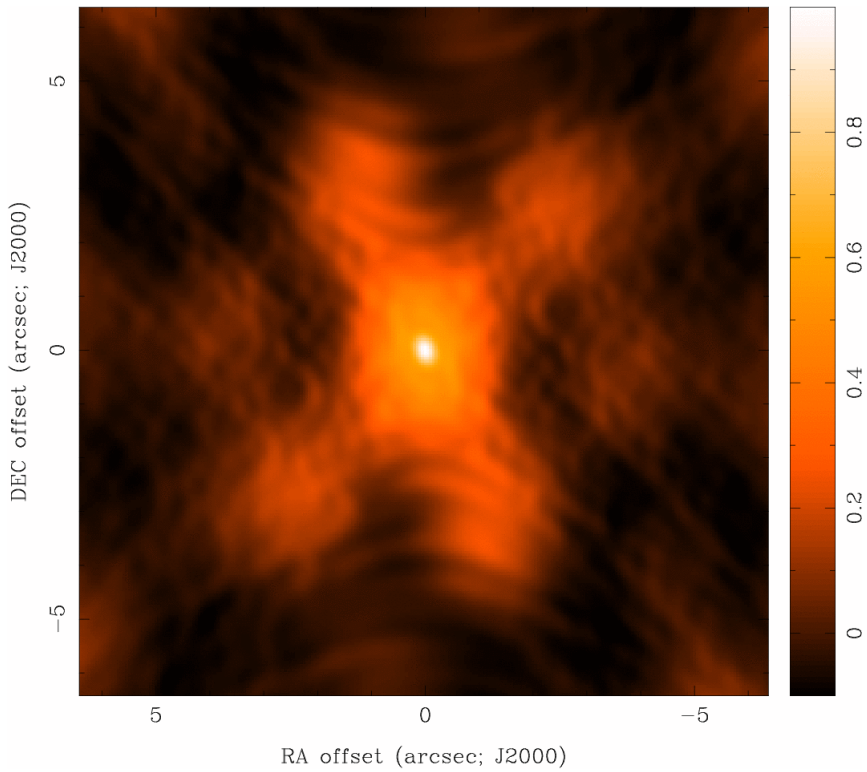
Dirty Beam Shape and Weighting

- Each visibility point is given a weight in the imaging step

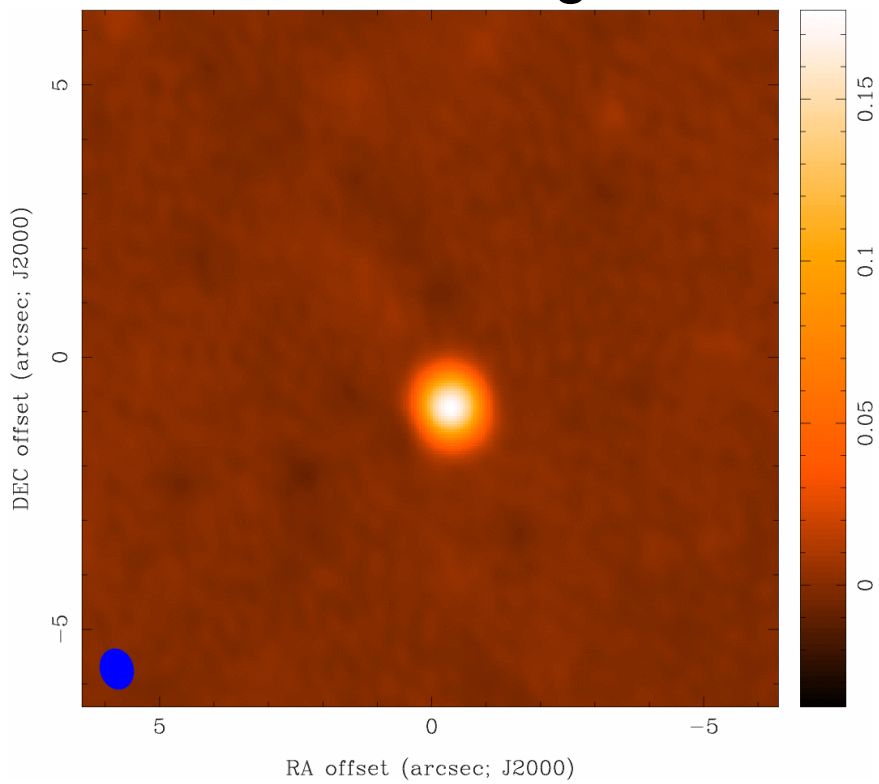
- Natural
  - Weights inversely proportional to noise variance
  - Best point-source sensitivity; poor beam characteristics
- Uniform
  - Weights inversely proportional to sampling density (longer baseline are given higher weight than in natural)
  - Best resolution; poorer noise characteristics
- Briggs (Robust)
  - A graduated scheme using the parameter *robust*
  - In CASA, set *robust* from -2 (~ uniform) to +2 (~ natural)
  - *robust* = 0 often a good choice

Imaging Results

Natural Weight Beam

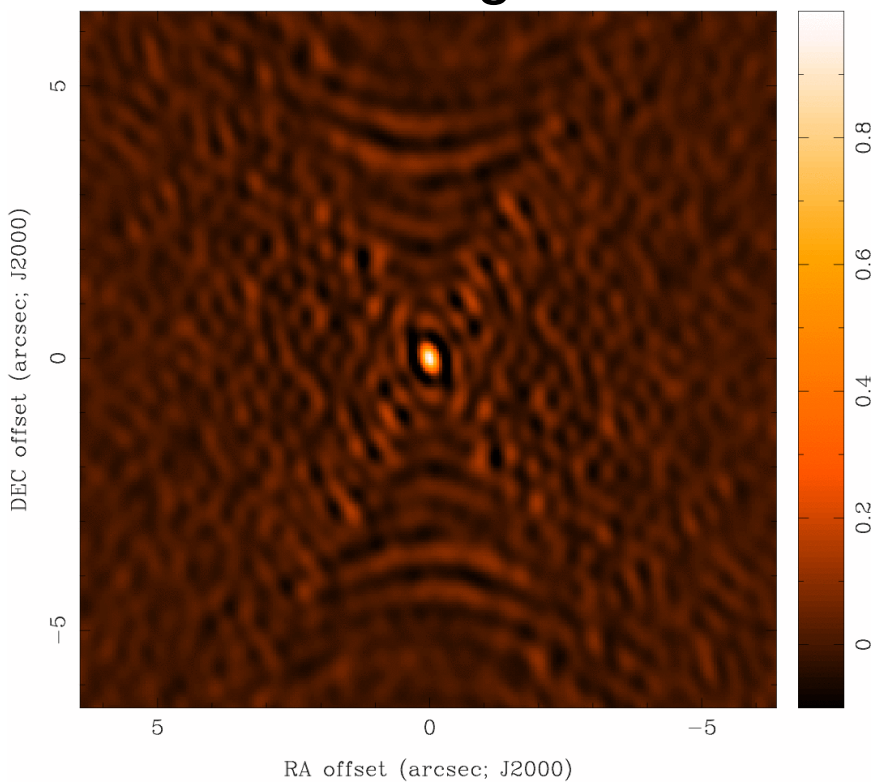


CLEAN image

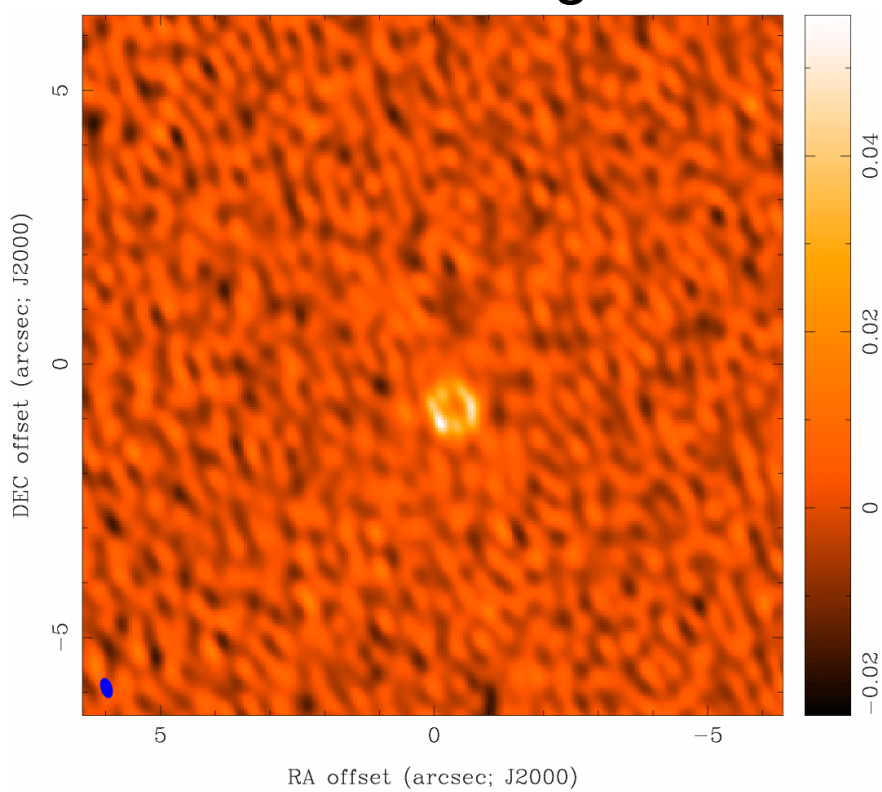


Imaging Results

Uniform Weight Beam

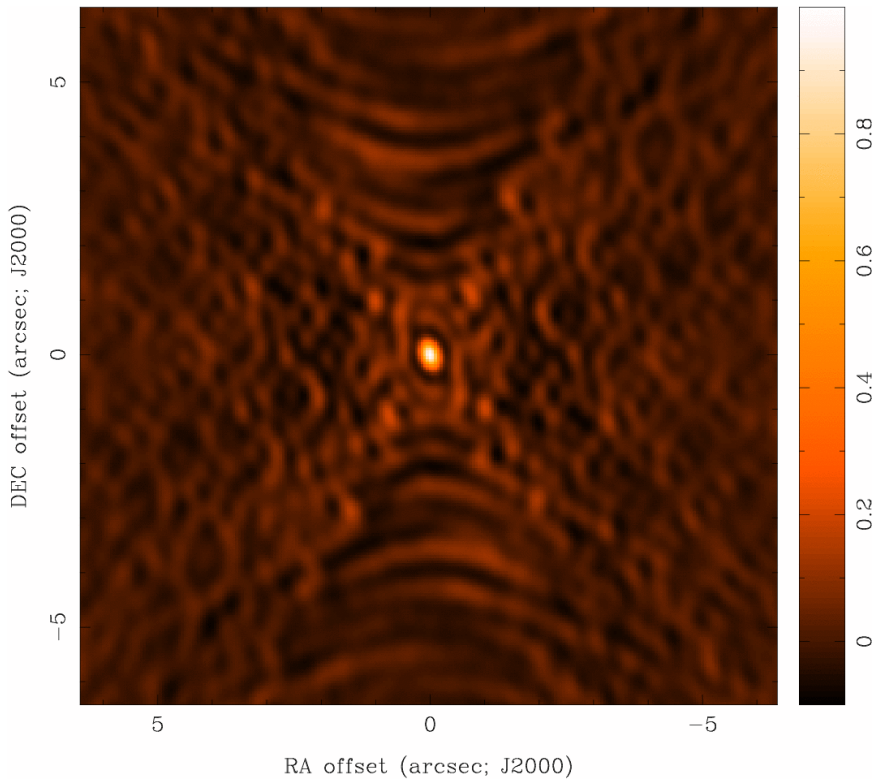


CLEAN image

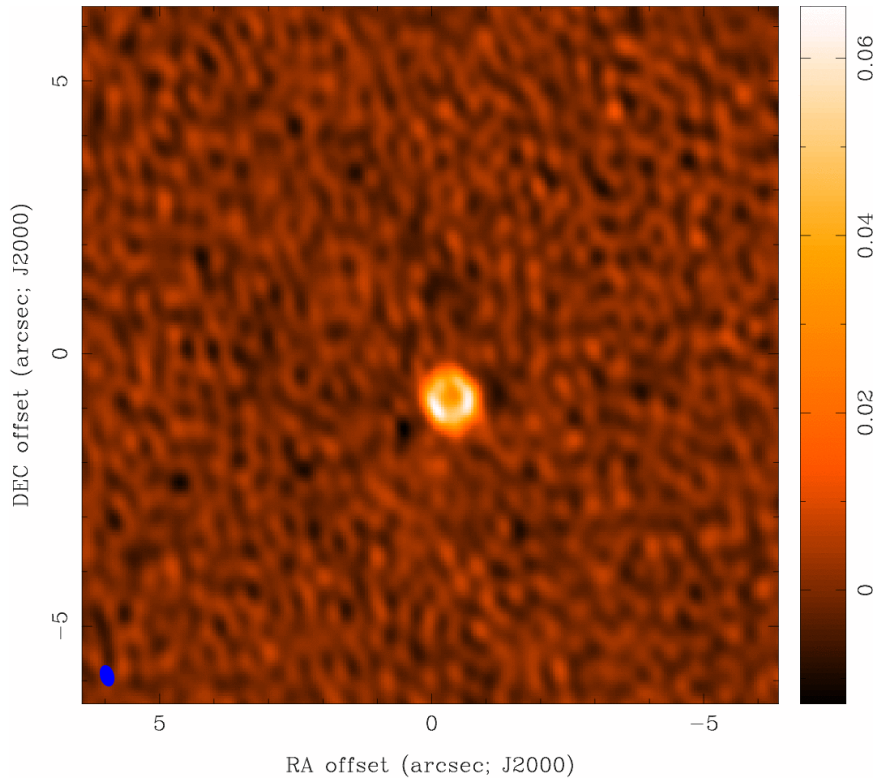


Imaging Results

Robust=0 Beam



CLEAN image



Clean in CASA:

```
CASA <13>: inp clean
---------> inp(clean)
#  clean :: Invert and deconvolve images with selected algorithm
vis                 =          ''        #  Name of input visibility file
imagename           =          ''        #  Pre-name of output images
outlierfile         =          ''        #  Text file with image names, sizes, centers for outliers
field               =          ''        #  Field Name or id
spw                 =          ''        #  Spectral windows e.g. '0~3', '' is all
selectdata          =       False        #  Other data selection parameters
mode                =       'mfs'        #  Spectral gridding type (mfs, channel, velocity, frequency)
     nterms         =           1        #  Number of Taylor coefficients to model the sky frequency dependence
     reffreq        =          ''        #  Reference frequency (nterms > 1),'' uses central data-frequency

gridmode            =          ''        #  Gridding kernel for FFT-based transforms, default='' None
niter               =         500        #  Maximum number of iterations
gain                =         0.1        #  Loop gain for cleaning
threshold           =    '0.0mJy'        #  Flux level to stop cleaning, must include units: '1.0mJy'
psfmode             =     'clark'        #  Method of PSF calculation to use during minor cycles
imagermode          =   'csclean'        #  Options: 'csclean' or 'mosaic', '', uses psfmode
     cyclefactor    =         1.5        #  Controls how often major cycles are done. (e.g. 5 for frequently)
     cyclespeedup   =          -1        #  Cycle threshold doubles in this number of iterations

multiscale          =          []        #  Deconvolution scales (pixels); [] = standard clean
interactive         =       False        #  Use interactive clean (with GUI viewer)
mask                =          []        #  Cleanbox(es), mask image(s), region(s), or a level
imsize              = [256, 256]         #  x and y image size in pixels.  Single value: same for both
cell                = ['1.0arcsec']      #  x and y cell size(s). Default unit arcsec.
phasecenter         =          ''        #  Image center: direction or field index
restfreq            =          ''        #  Rest frequency to assign to image (see help)
stokes              =         'I'        #  Stokes params to image (eg I,IV,IQ,IQUV)
weighting           =   'natural'        #  Weighting of uv (natural, uniform, briggs, ...)
uvtaper             =       False        #  Apply additional uv tapering of visibilities
modelimage          =          ''        #  Name of model image(s) to initialize cleaning
restoringbeam       =        ['']        #  Output Gaussian restoring beam for CLEAN image
pbcor               =       False        #  Output primary beam-corrected image
minpb               =         0.2        #  Minimum PB level to use
usescratch          =       False        #  True if to save model visibilities in MODEL_DATA column
allowchunk          =       False        #  Divide large image cubes into channel chunks for deconvolution
async               =       False        #  If true the taskname must be started using clean(...)
```

Basic Image Parameters: Pixel Size and Image Size

- pixel size
  - should satisfy $\Delta x < 1/(2\ u_{max})$ $\Delta y < 1/(2\ v_{max})$
  - in practice, 3 to 5 pixels across the main lobe of the dirty beam

- image size
  - Consider FWHM of primary beam (e.g. ~ 20" at Band 7)
  - Be aware that sensitivity is not uniform across the primary beam
  - Use mosaicing to image larger targets
  - Not restricted to powers of 2

* if there are bright sources in the sidelobes, they will be aliased into the image (need to make a larger image)

Maximum Angular Scale

Band	Frequency (GHz)	Primary beam (“)	Range of Scales (“) C32-1	C32-9
3	84-116	72 - 52	4.2 - 24.6	0.7 - 15.1
6	211-275	29 - 22	1.8 - 10.7	0.3 - 6.6
7	275-373	22 - 16	1.2 - 7.1	0.2 - 4.4
9	602-720	10 – 8.5	0.6 - 3.6	0.1 - 2.2

- **Range** from synthesized beam to maximum angular scale (MAS)
- **Smooth** structures larger than MAS begin to be resolved out.
- All flux on scales larger than λ/B_{min} (~2 x MAS) completely resolved out.

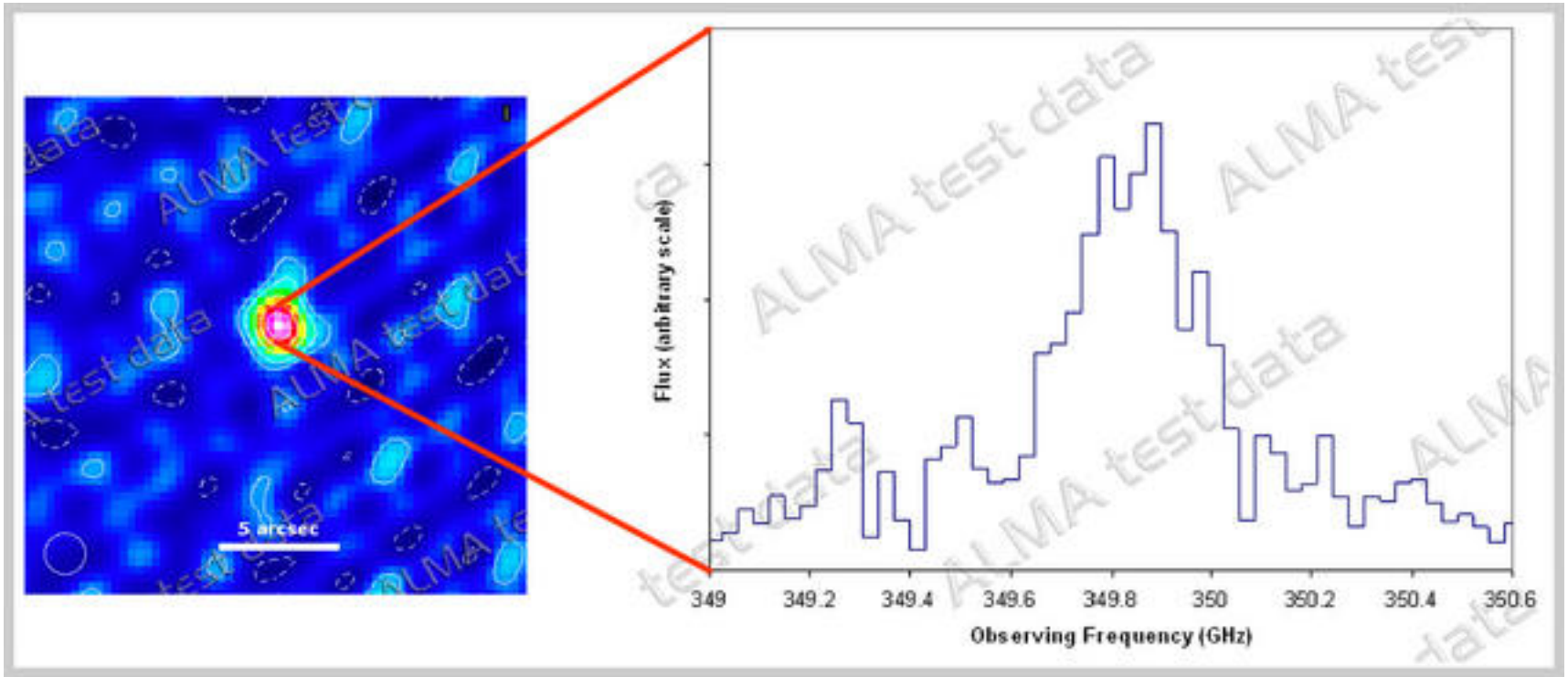
Output of *clean*

Minimally:

- my_image.flux — Relative sky sensitivity
- my_image.image — Cleaned and restored image
- my_image.mask — Clean "boxes"
- my_image.model — Clean components
- my_image.psf — Dirty beam
- my_image.residual — Residual

Some of the capabilities of *clean* …

Imaging spectral lines



Imaging spectral lines

```
#  clean :: Invert and deconvolve images with selected algorithm
vis                 = 'ngc3256_co.ms.contsub' #  Name of input visibility file
imagename           = 'ngc3256_co'      #  Pre-name of output images
outlierfile         =         ''        #  Text file with image names, sizes, centers for outliers
field               =         ''        #  Field Name or id
spw                 =  '0:38~87'        #  Spectral windows e.g. '0~3', '' is all
selectdata          =      False        #  Other data selection parameters
mode                =  'channel'        #  Spectral gridding type (mfs, channel, velocity, frequency)
     nchan          =         50        #  Number of channels (planes) in output image; -1 = all
     start          =         ''        #  Begin the output cube at the frequency of this channel in the MS
     width          =         ''        #  Width of output channel relative to MS channel (# to average)
     interpolation  =   'linear'         #  Spectral interpolation (nearest, linear, cubic). Use nearest for mode=channel
     chaniter       =      False        #  Clean each channel to completion (True), or all channels each cycle (False)
     outframe       =         ''        #  velocity frame of output image
```

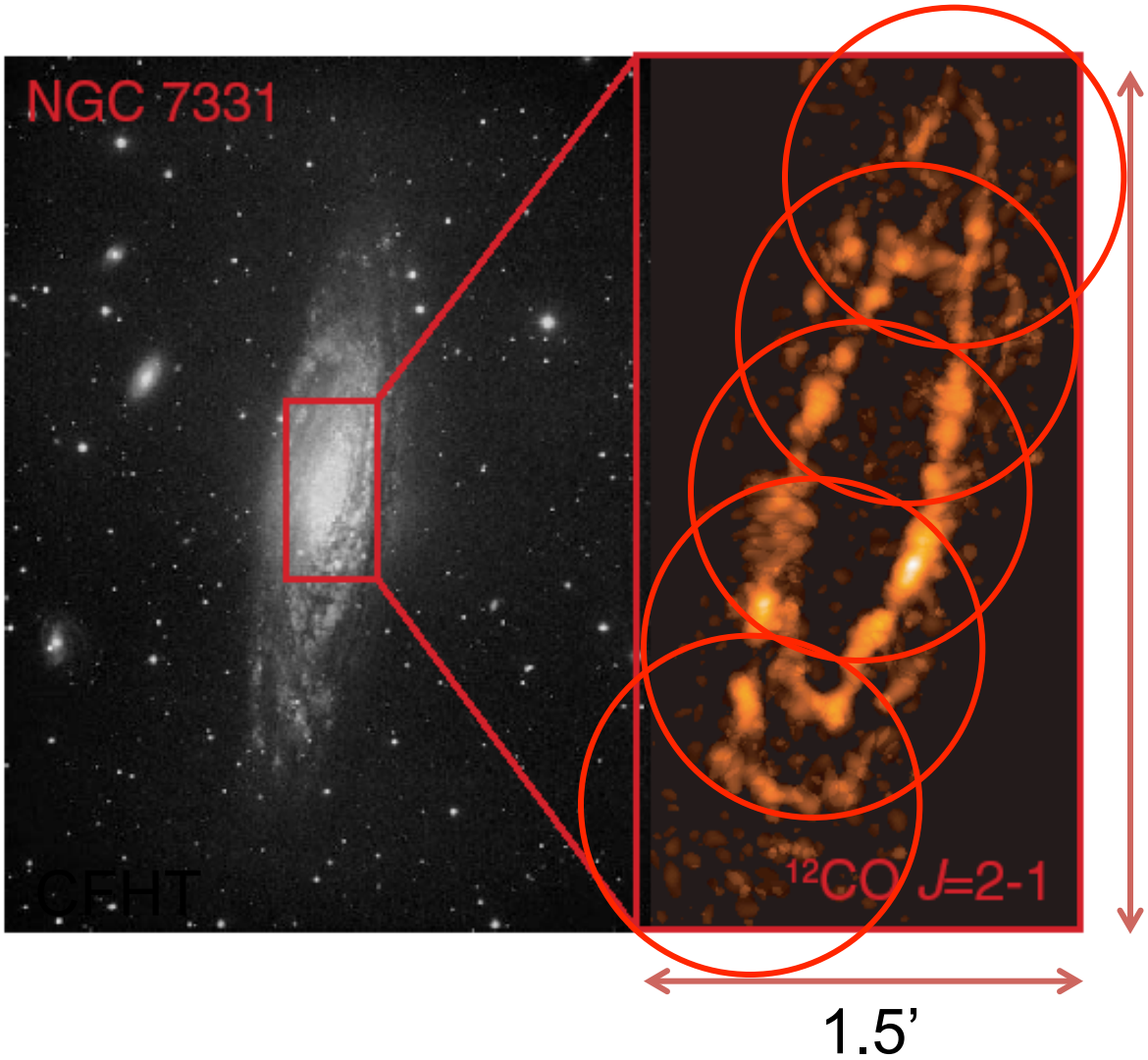
```
restfreq            = '115.271201800GHz' #  Rest frequency to assign to image (see help)
```

Imaging spectral lines: continuum subtraction

- Generally would like to subtract continuum emission (we will see how to identify line-free channels in hands-on session)
- Use `uvcontsub` to do the subtraction in uv plane.

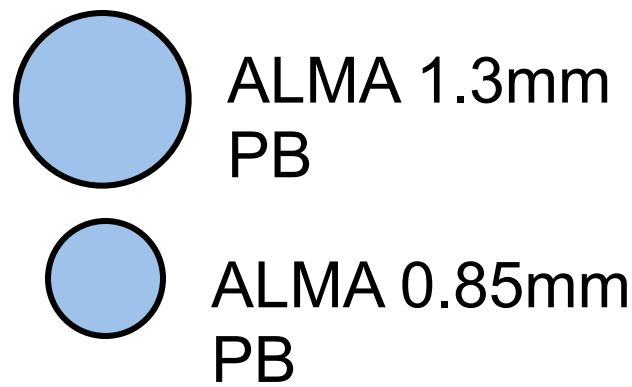
```
CASA <11>: inp
---------> inp()
#  uvcontsub :: Continuum fitting and subtraction in the uv plane
vis                 = 'ngc3256_co.ms'   #  Name of input MS.  Output goes to vis + ".contsub"
field               =         ''        #  Select field(s) using id(s) or name(s)
fitspw              = '0:20~53;71~120'  #  Spectral window:channel selection for fitting the continuum
combine             =         ''        #  Data axes to combine for the continuum estimation (none, or spw and/or scan)
solint              =      'int'        #  Continuum fit timescale (int recommended!)
fitorder            =          0        #  Polynomial order for the fits
spw                 =         ''        #  Spectral window selection for output
want_cont           =      False        #  Create vis + ".cont" to hold the continuum estimate.
async               =      False        #  If true the taskname must be started using uvcontsub(...)
```

Mosaics



Example: SMA 1.3 mm observations: 5 pointings

- Primary beam ~1'
- Resolution ~3"



In Cycle 1, the number of pointings <= 150.

Petitpas et al.

Imaging mosaics

```
imagermode      =   'mosaic'     #  Options: 'csclean' or 'mosaic', '', uses psfmode
   mosweight    =     False      #  Individually weight the fields of the mosaic
   ftmachine    =      'ft'      #  Gridding method for the image
   scaletype    =    'SAULT'     #  Controls scaling of pixels in the image plane. default='SAULT'; example:
                                  #   scaletype='PBCOR' Options: 'PBCOR','SAULT'
   cyclefactor  =       1.5      #  Controls how often major cycles are done. (e.g. 5 for frequently)
   cyclespeedup =        -1      #  Cycle threshold doubles in this number of iterations
   flatnoise    =      True      #  Controls whether searching for clean components is done in a constant noise
                                  #   residual image (True) or in an optimal signal-to-noise residual image
                                  #   (False)
```

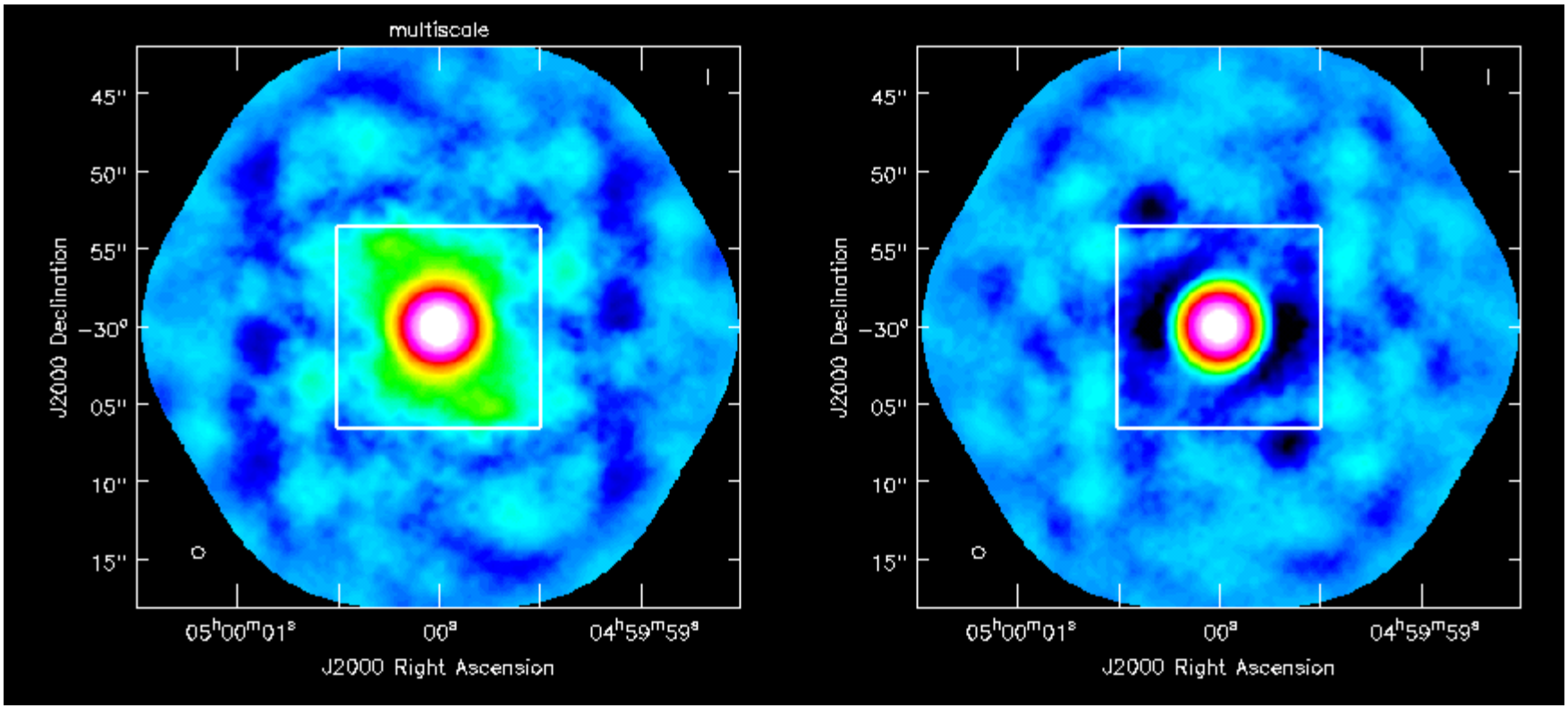
ftmachine = “ft” : shift and add in image plane

ftmachine = “mosaic” : add in uv plane and invert together

multiscale clean

multiscale

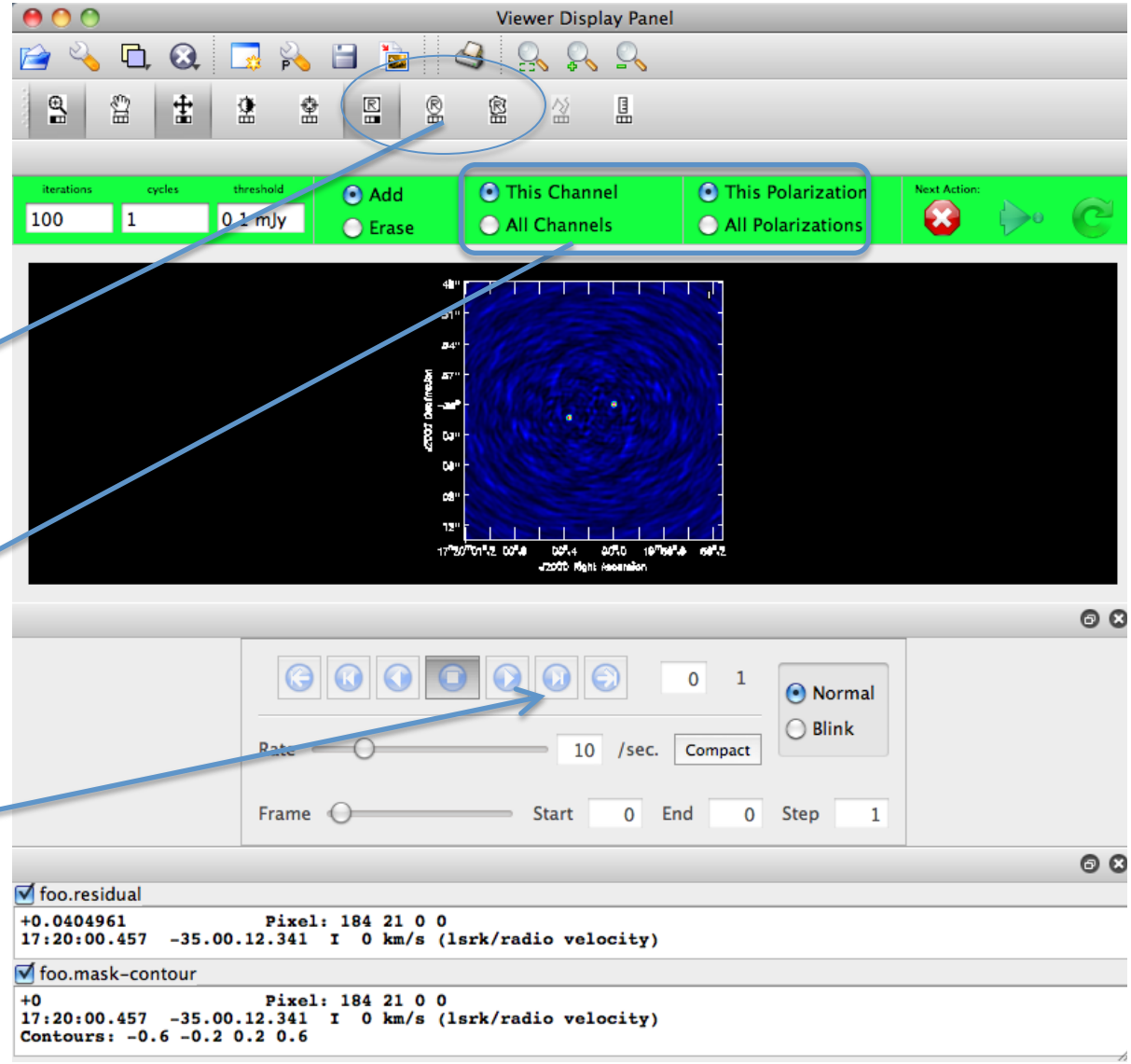
“classic” scale




```
multiscale          = [0, 5, 15]        # Deconvolution scales (pixels); [] = standard clean
```

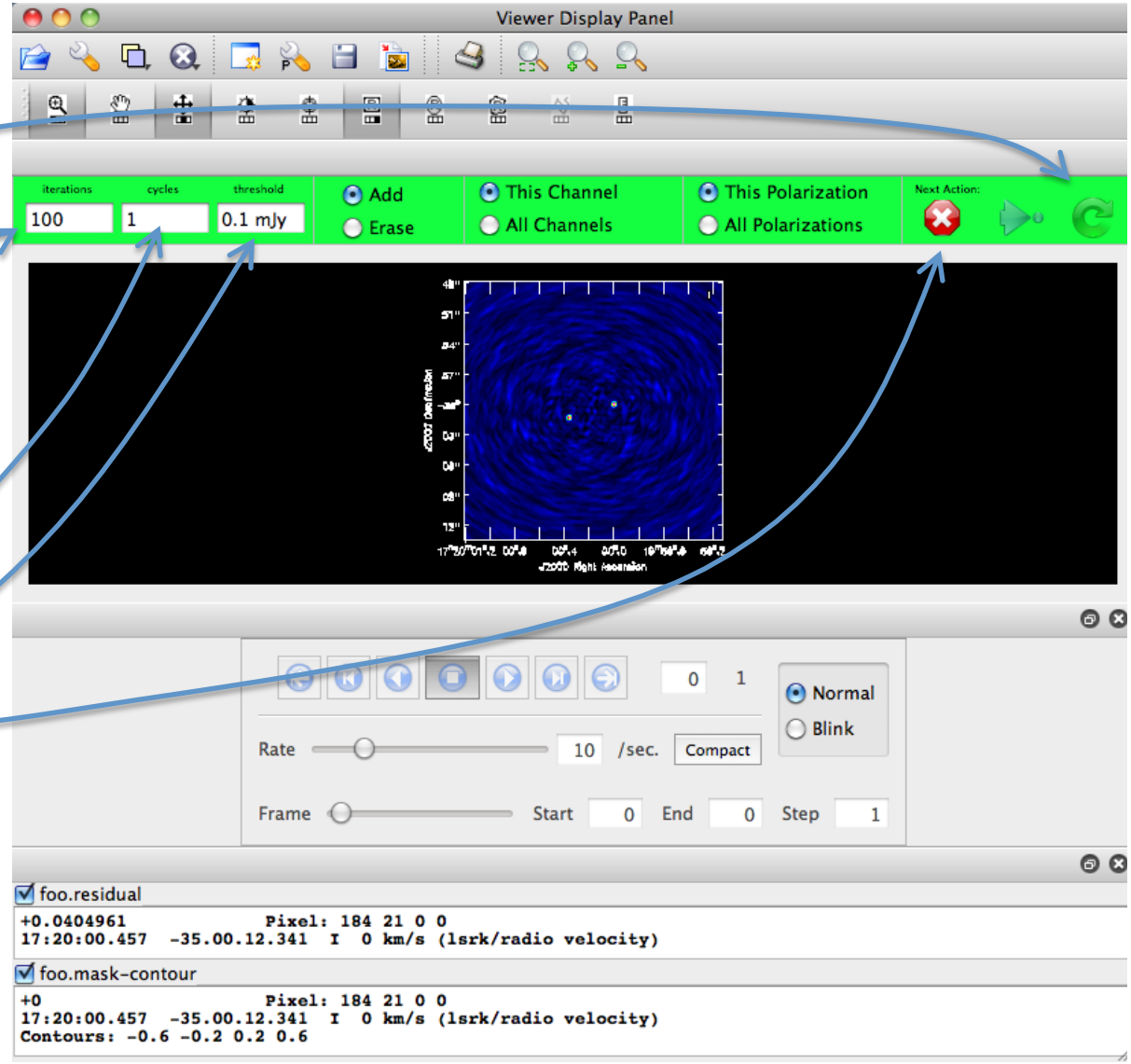
Interactive Clean

- residual image in viewer
- define a mask with R-click on shape type
- define the same mask for all channels
- or iterate through the channels with the tape deck and define separate masks



Interactive Clean

- perform N iterations
- and return – every time the residual is displayed is a major cycle
- continue until #cycles or threshold reached, or user stop



Combining with single-dish or other interferometric maps

If you have only images:

feather

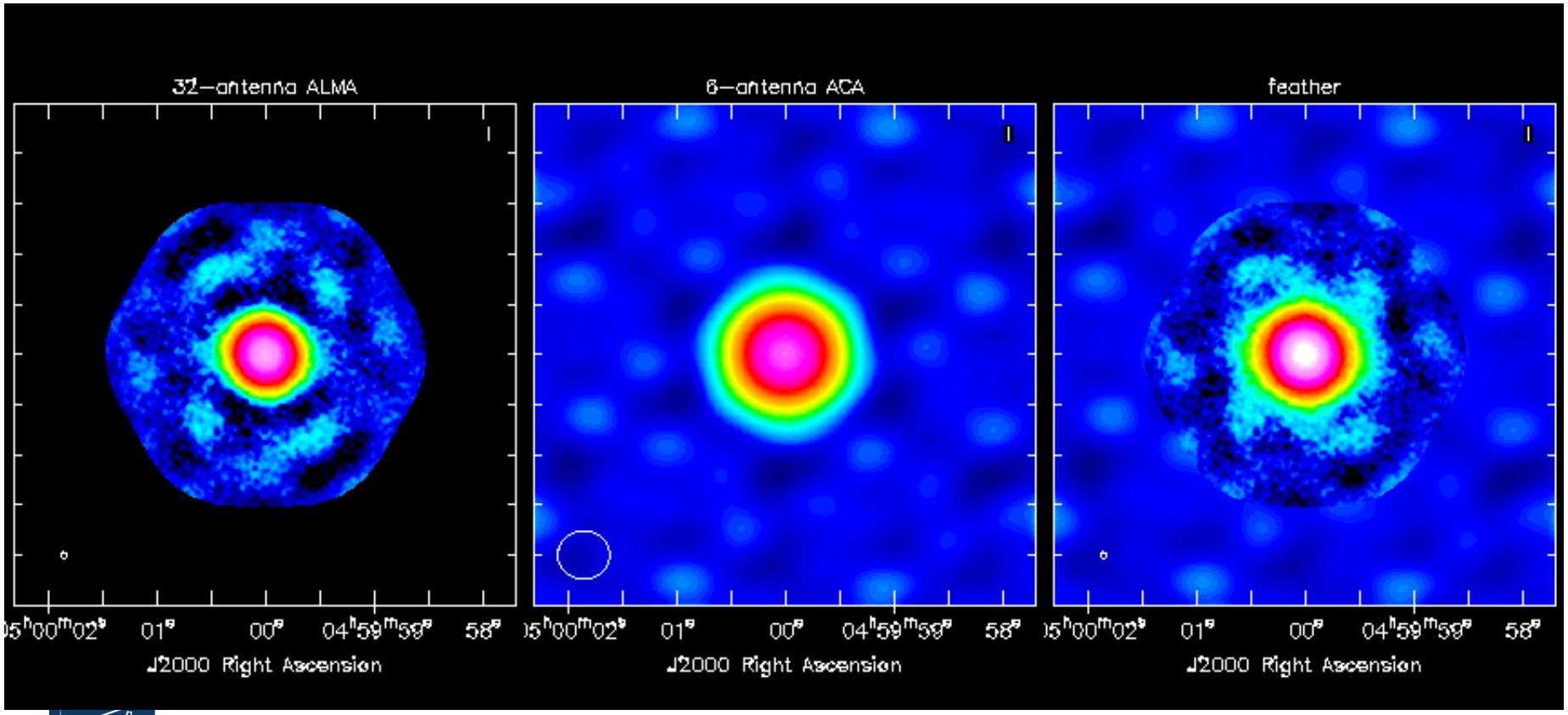
If you have an image and a MS:

use clean with the image as "modelimage"

or feather

Combining with other data: feather

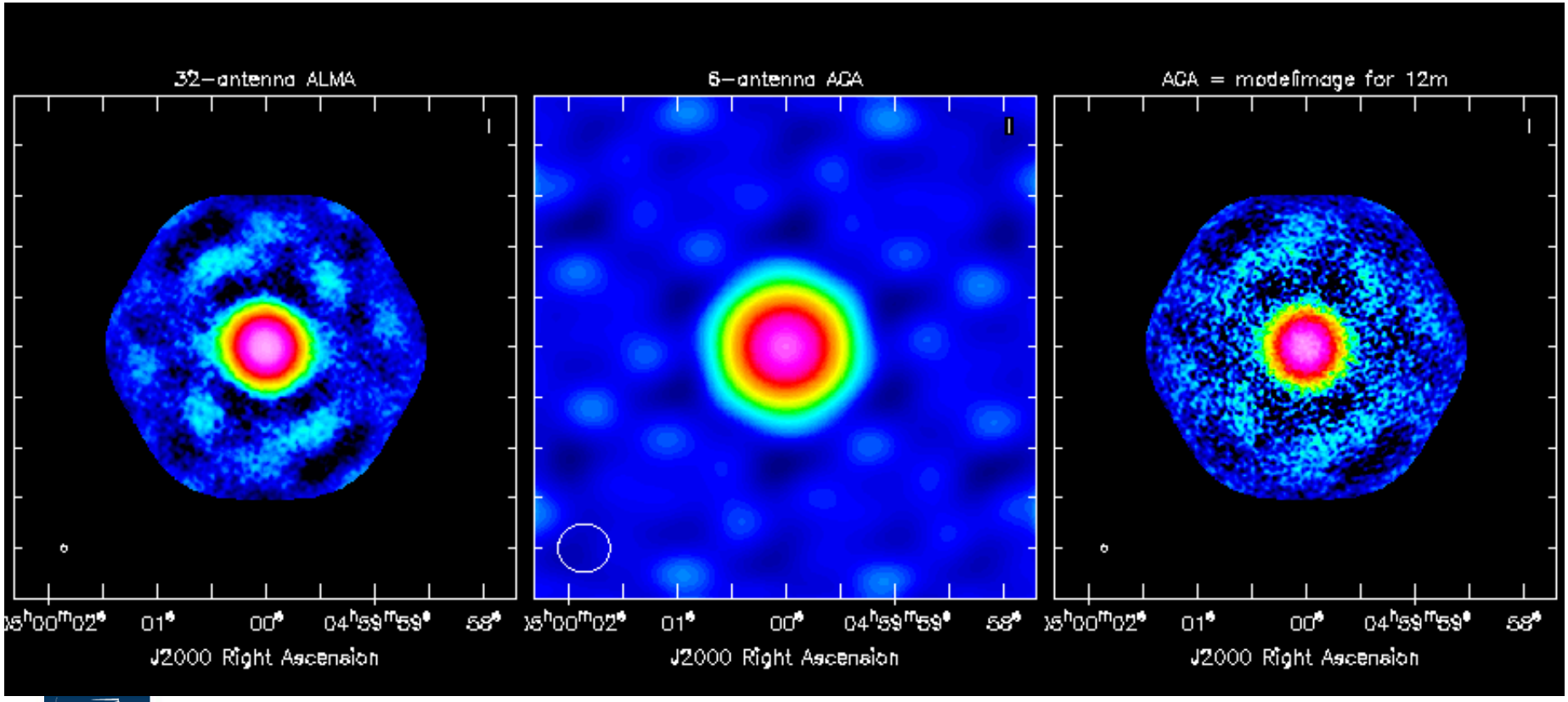
```
#  feather :: Combine two images using their Fourier transforms
imagename          =        ''        #  Name of output feathered image
highres            =        ''        #  Name of high resolution (interferometer) image
lowres             =        ''        #  Name of low resolution (single dish) image
async              =     False        #  If true the taskname must be started using feather(...)
```



Combining with other data: modelimage

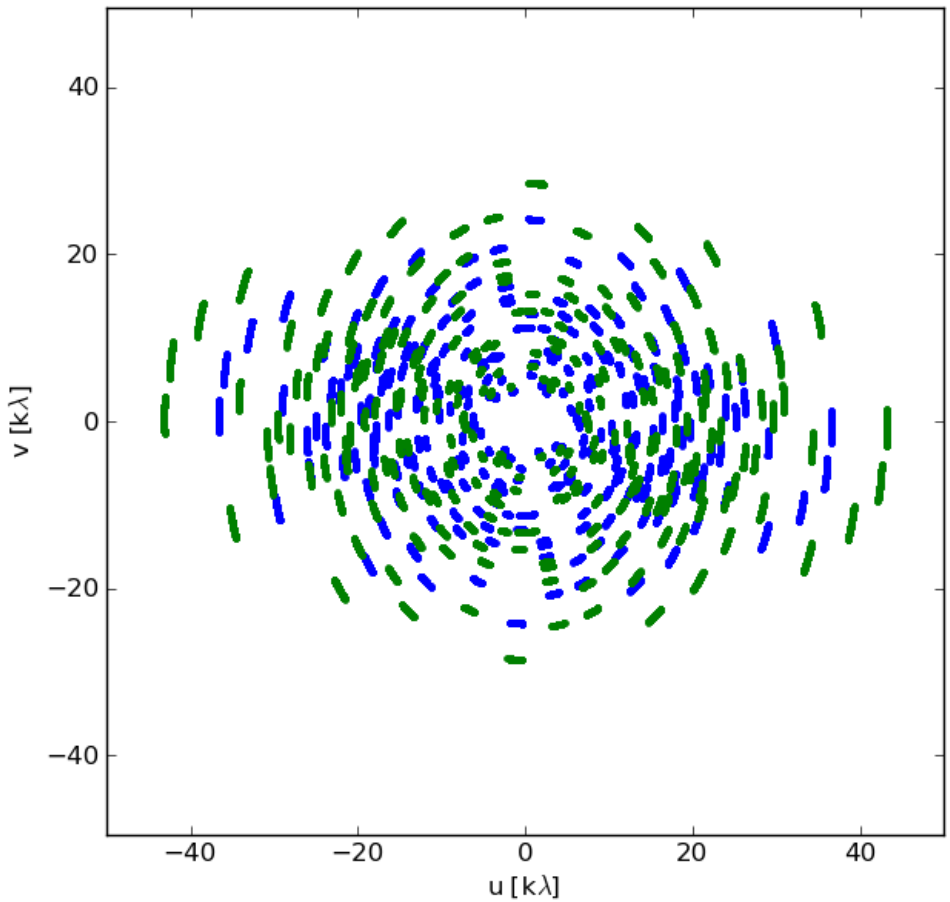
```
--------> inp(clean)
#  clean :: Invert and deconvolve images with selected algorithm

modelimage          =          ''          #  Name of model image(s) to initialize cleaning
```



Multi-Frequency Synthesis (mfs)

```
mode        =    'mfs'     #  Spectral gridding type (mfs, channel, velocity, frequency)
   nterms   =        1     #  Number of Taylor coefficients to model the sky frequency dependence
   reffreq  =       ''     #  Reference frequency (nterms > 1),'' uses central data-frequency
```



higher order mfs

- MFS (mode mfs)
  - nterm=2 compute spectral index, 3 for curvature etc.
  - needed for bandwidths ~5% or more (S/N dependent)
  - tt0 average intensity, tt1 alpha*tt0, alpha images output
  - takes at least nterms longer (image size dependent)

Miscellaneous

- clean will restart from existing files
  - will first recompute residuals from model
  - be sure this is what you want
- mask image in particular can be reused but be careful of imsize
- total cleaned flux not reported until end
- *don't do ^C while imaging – can do bad things to your MS*

OK, Let's give it a try!

The imaging script we will follow is here:

~/distrib/imaging/scripts/basic_imaging/Basic_imaging.py

You can get more information and background on the data and methods here:

http://casaguides.nrao.edu/index.php?title=WorkshopImaging