Obit Imaging Tools

W. D. Cotton, May 29, 2025

Abstract

This memo describes a package of tools mostly related to Obit imaging of MeerKAT data but many can be adapted to other applications

Index Terms

interferometry, polarization, imaging, calibration

I. Introduction

THE general process of imaging radio interferometry data can be quite complex with a large number of options. In any particular case the number of useful options can be substantially reduced. This memo describes a set of tools to simplify processing interferometry datasets; these are particularly aimed at MeerKAT data but have broader applications.

II. MK_Tools.py

One package of tools is in $OBIT/share/scripts/MK_Tools. These consist of a number of python functions that run in the ObitTalk environment. These functions can accessed by copying this module to your cwd and in ObitTalk:

```
>>> from MK_Tools import *
```

- **SetMFImage:** This function defines the Obit task interface object and sets most of the parameters. These are generally appropriate for MeerKAT data with optional call parameters for those most likely to be changed.

```
SetMFImage(uv, src, gainUse=0, PDVer=-1, outDisk=1, outSeq=1, Niter=1000, \
           minFlux=5e-05, ref=0, nThreads=16)
    Setup Parameters for MFImage

    Generalized defaults for MeerKAT
    Parameters may need adjusting before execution
    * uv       = Python Obit UV object
    * src      = Source to be imaged
    * gainUse  = if >= 0 apply gain cal with gainUse (0->hi)
    * PDVer    = if >  0 apply poln cal using PDVer
    * outDisk  = AIPS disk number for images
    * outSeq   = AIPS sequence number for images
    * Niter    = Maximum number of iterations
    * minFlux  = Minimum flux density for first CLEAN
    * ref      = Reference antenna
    * nThreads = number of threads to use
    returns MFImage task object
```

- **ImageA2F:** This function writes an AIPS image to a FITS file giving in a name derived from the AIPS name: <AIPS name>_<AIPS class>_<AIPS seq>_.fits

```
ImageA2F(img, err, dir='./')
    Write an AIPS Image to a disk file

    * img      = Obit/python AIPS image
    * err      = Obit error/message stack
    * dir      = Directory name for FITS image
    returns obit Image object to FITS file
```

- **RMFitQU:** Function to do a Faraday analysis deriving the peak rotation measure in the Faraday spectrum, EVPA at $\lambda \sim 0$ and peak unwrapped polarized intensity. It can use cubes produced by MFImage or Imager. An example of peak RM and polarized intensity are shown in Figure 1.

National Radio Astronomy Observatory, 520 Edgemont Rd., Charlottesville, VA, 22903 USA email: bcotton@nrao.edu

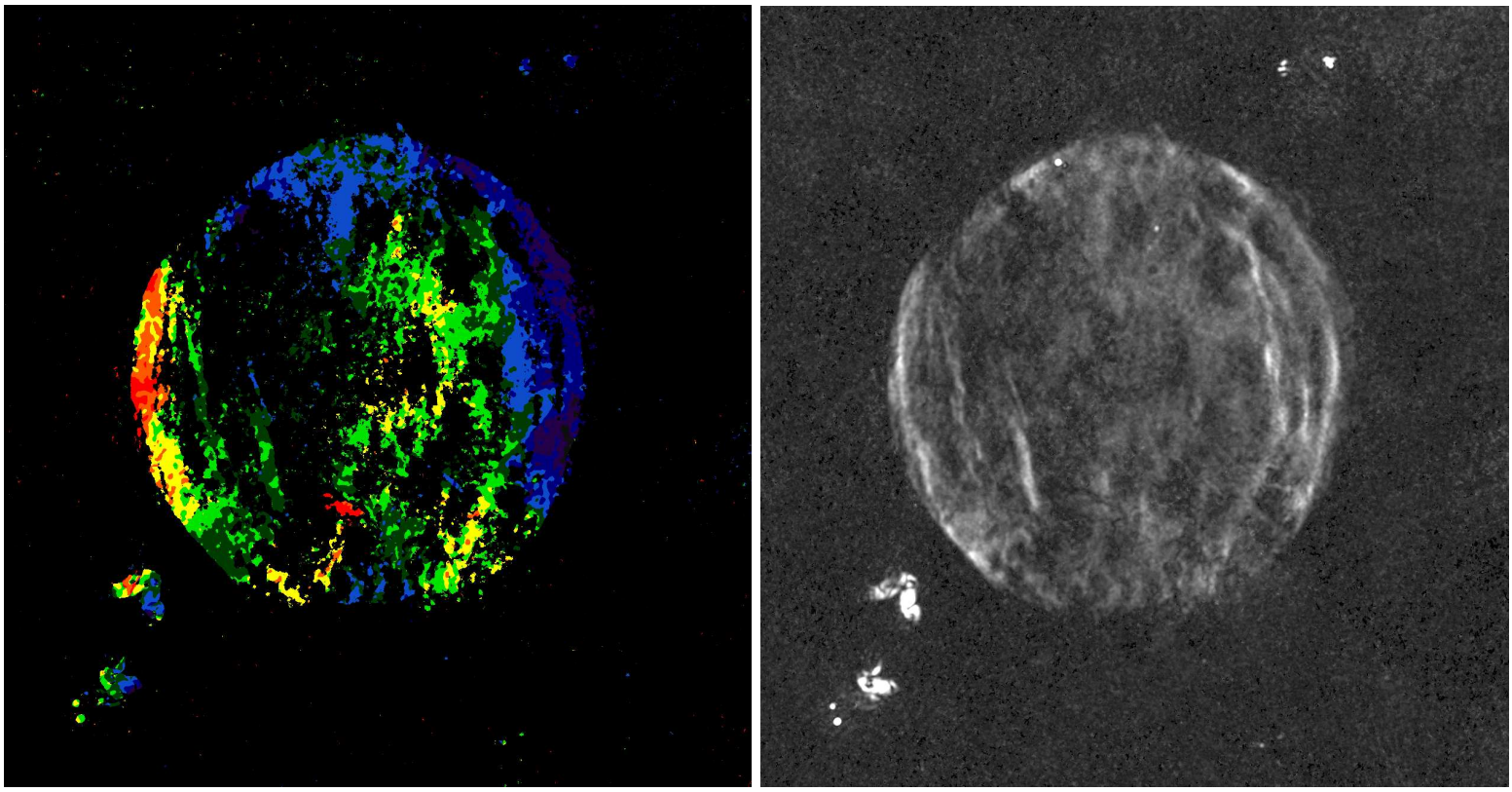

Fig. 1. Supernova remnant G4.8+6.2 observed with MeerKAT at L Band[1] and with Faraday analysis using RMFitQU. There is an extended background AGN in the lower left. **Left:** Peak rotation measure, red=50 rad m^{-2}, purple=-35 rad m^{-2}. **Right:** Peak unwrapped polarized intensity.

```
RMFitQU(name, imQ, imU, err, doRMSyn=True, nThreads=16, maxRM=100.0)
    Faraday synthesis of a Q/U image cube pair

    See help(RMFit.Cube) for details
    * name     = root of output file name
    * imQ      = Obit/python Q cube
    * imU      = Obit/python U cube
    * err      = Obit error/message stack
    * doRMSyn  = If True use peak of RM spectrum,
                 else least squares
    * nThreads = Number of threads to use
    * maxRM    = search +/- maxRM rad/m^2
    returns RM Obit Image object
```

- **MakeMask:** generates a CLEAN mask file for use in MFImage or Imager.

```
MakeMask(img, err, name=None, sigma=10.0, nThreads=10, maxWid=60.0, minSNR=5.0)
    Make a CLEAN mask from an image

    Generates a <?>.mask file in cwd which can be used as
    CLEANFile in MFImage or Imager.  Also, a text *.cat file
    Uses Obit/FndSou for source finding.
    NB: Any previous MF or VL tables deleted
    * img      = Obit/python Stokes I image, AIPS or FITS
    * err      = Obit error/message stack
    * name     = if given, the root mask name, <name>.mask
                 if none AIPS or FITS file name
    * sigma    = multiple of RMS to search for sources.
    * nThreads = Number of threads to use
    * maxWid   = Maximum component size in asec
```

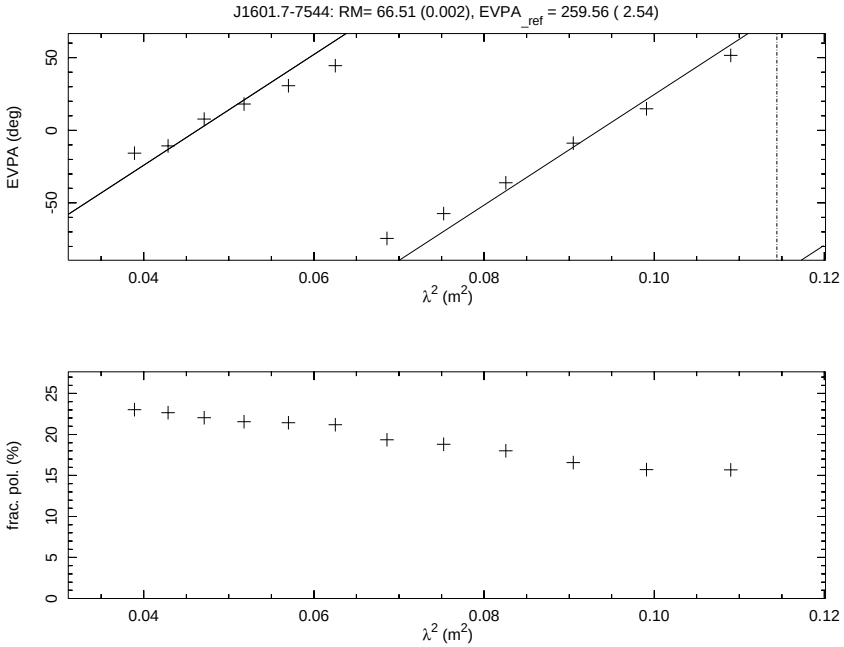

Fig. 2. Rotation measure plot of a source in a field observed with MeerKAT at L band plotted by PlotRM.

```
    * minSNR    = Minimum SNR of component
```

- **PlotRM:** Plots the EVPA v. λ^2 with a fitted RM and the fractional linear polarization v. λ^2 in a set of I,Q,U cubes at a given position. Fitted parameters are given in the title. Error bars are plotted by may be smaller than the symbol. An example plot is shown in figure 2.

```
PlotRM(src, icube, qcube, ucube, pos, plotfile, err, label=None, nThreads=1)
    Make rotation measure plot, EVPA v lambda^2

    Make RM and fractional poln. plots at position pos
    * src     = Source name
    * icube   = Obit IPol image cube
    * qcube   = Obit QPol image cube
    * ucube   = Obit UPol image cube
    * pos     = (RA, dec) in deg. Uses nearest pixel.
    * label   = label for plot, defaults to src
    * plotfile= name of postscript plot file
    * err     = Python Obit Error/message stack
    * nthreads= the number of threads to be used in calculations
```

- **PlotSpec:** Plots the spectrum with a fit at the specified position in a spectral cube. Uses PLPlot which does a piss–poor job of labeling log–log plots. Spectral fit values are given in the title. The flux density given is that of the fit corresponding to the first frequency. Error bars are plotted by may be smaller than the symbol. An example plot is shown in figure 3.

```
PlotSpec(src, icube, pos, plotfile, err, label=None, nThreads=1)
    Plot the spectrum at a location in an image

    * src     = Source name
    * icube   = Obit IPol image cube
    * pos     = (RA, dec) in deg.  interpolates
    * label   = label for plot, defaults to src
    * plotfile= name of postscript plot file
    * err     = Python Obit Error/message stack
    * nthreads= the number of threads to be used in calculations
```

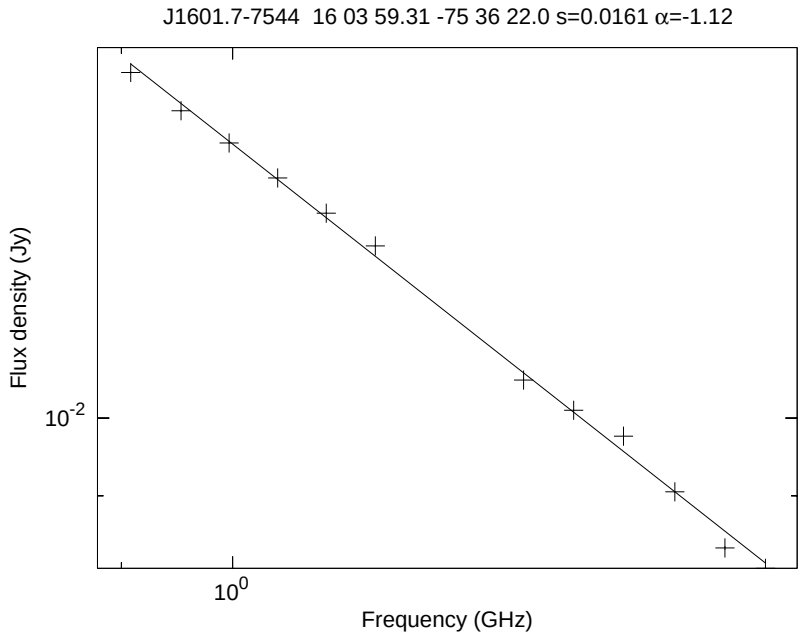

Fig. 3. Spectrum of a source in a field observed by MeerKAT at L band plotted using PlotSpec.

III. CLEANWINDOWEDIT.PY

In addition to the utility to generate a CLEAN mask from an image described in Section II, there is an interactive CLEAN window editor package using ObitView in $OBIT/share/scripts/CleanWindowEdit.py. To access this package, copy the file to your cwd and in ObitTalk:

```
>>> from CleanWindowEdit import *
```

This package works by generating a window list, a list of round boxes (no rectangles or unboxes), from interactive use of ObitView. The window list can then be converted into a CLEAN mask file suitable for MFImage or Imager use as CLEANFile. Window lists can be stored as pickle files for subsequent use.

Functions:

- **GetWindowList:** This function displays the image on ObitView and brings up the editing control. Select the "edit" button and follow the instructions given in the text window. Note: while the editor will generate rectangular and unboxes, they will not end up in the mask file. Large (>49 pixels) boxes will probably get truncated either when they are injested by MFImage/Imager or because they get too close to the edge of a facet. A set of smaller, overlapping boxes will reduce this problem. Editing can continue until the "continue"/"OK" buttons are hit. Thereafter, reEditWindow can restart the editing and produce a new window list.

```
GetWindowList(inIm, err, disp=<C ODisplay instance> ObitView)
    Create window on an image via editing on display disp

    Returns the window list:
    list of [id, type(0=rect, 1=round), parameters[blc,trc] or [rad,cenx,ceny])
    * inIm  = image for window
    * err   = Obit message/error object
    * disp  = image display
```

- **reEditWindow:** Restart editing and return the amended window list.

```
reEditWindowList(inIm, winList, err)
    Convert a winList to a OWindow and edit

    returns new WindowList
    * inIm    = Image being used
    * winList = WindowList to edit
    * err     = Obit message/error object
```

- **SaveWindowList:** Saves the current window list to a pickle file.

```
SaveWindowList(winList, pfile)
    Save WindowList to pickle jar

    * winList = OWindow to save
    * pfile   = root name of pickle file- ".pickle" added
```

- **FetchWindowList:** Retrieves a window list from a pickle file.

```
FetchWindowList(pfile)
    Get WindowList from pickle jar

    returns WindowList
    * pfile   = root name of pickle file- ".pickle" added
```

- **WindowList2Mask:** Convert the window list to a text CLEAN mask file.

```
WindowList2Mask(inIm, winList, mfile, err)
    Write a CLEAN mask file from a WindowList

    * inIm    = Image being used
    * winList = WindowList to write as mask file
    * mfile   = name of CLEAN mask file
```

IV. MK_POLCAL_XY.PY

For data that has had parallel hand calibration done in a fashion to enable polarization calibration, delay and bandpass calibration done with an unpolarized calibrator and gain calibration done using Stokes I, all with the same reference antenna; the polarization calibration can be performed using $OBIT/share/scripts/MK_PolCal_XY.py To access this package, copy the file to your cwd and in ObitTalk:

```
>>> from MK_PolCal_XY import MKPolCalXY
```

This script, while explicitedly targeted toward MeerKAT L Band observations, can be adapted to other circumstances involving arrays using linearly polarized feeds. This script has the following functions

- **Clip** calibrator data with excessive amplitudes using AutoFlag.
- **Phase calibrate** calibrators with a point model using Calib/CLCal.
- **PCal** for instrumental calibration.
- **XYDly** for X-Y phase and delay calibration.
- **CLCal** to apply the XYDly solution to CL table.

This script recognizes a number of calibrators, thru several aliases, for which it knows the relevant calibration parameters. Unrecognized calibrators will be included in the estimation of the instrumental polarization parameters in PCal while fitting for any potential polarized component. This feature is useful for calibrators (i.e. gain calibrators) that are observed over a range of parallactic angle which allows separating instrumental and source polarization. Documentation for this script is:

```
MKPolCalXY(uv, cals, err, refAnt=59, nthreads=1, doCalib=True,
        doClip=False, doPCal=True, doXYDly=True, ChWid=17,
        timeAvg=2.0, chAvg=8, solInt=1440, fitType=0, debug=False,
        noScrat=[0, 0, 0])
    Polarization calibrate a MeerKAT dataset using PCal and XYDly

    A list of "standard" calibrators has known MeerKAT L band values.
    Log file is given in file 'PolnCal_'+uv.[AF]name.strip()+'.log'
    * uv       = Python Obit UV object, AIPS or FITS
    * cals     = list of calibrator source names, recognizes known calibrators:
                 unpolarized: 1934-638, J1939-6342, 0408-65,
                 Polarized: J0108+0134, 3C138, 3C286, J1130-4049, J2329-4730
                 Others will be included in PCal as sources with joint
                 solutions for source & instrumental polarization.
    * err      = Python Obit Error/message stack
    * refAnt   = the reference antenna used for parallel hand calibration
                 NB: this is NOT used as the reference antenna in PCal.
    * nthreads = the number of threads to be used in calculations
    * doClip   = If True, clip excessively high values in data.
```

```
* doCalib  = If True, do point source calibration for sources in
             cals writing a new CL table
* doPCal   = If True, run PCal
* doXYDly  = If True, run XYDly
* ChWid    = width of block of channels for PCal
* timeAvg  = Data Averaging time (min) for PCal, XYDly
* chAvg    = Number of channels to average in XYDly
* solInt   = Solution interval for XYDly
* fitType  = fitting type in XYDly, 0=>both, 1=>XY, 2=>YX
* debug    = if True, save input files
* noScrat  = List of AIPS disks to avoid for scratch files
```

REFERENCES

[1] W. D. Cotton, R. Kothes, F. Camilo, P. Chandra, S. Buchner, and M. Nyamai, "MeerKAT 1.3 GHz Observations of Supernova Remnants," *ApJS*, vol. 270, p. 21, 2024.