Polarisation in VLBI Workshop

Tools for advanced calibration and analysis

Abstract:

One of the main observables of the M2FINDERS project is the polarization and with this the magnetic field distribution along the jet in Active Galactic Nuclei (AGN). We invited Prof. Ivan Marti-Vidal, who is an expert in VLBI data reduction, to lead a workshop on polarization calibration and analysis, with the main focus on applying the PolSolve software developed by Prof. Marti-Vidal. In contrast to the traditional approach using the method LPCAL within the Astronomical Imaging Processing System (AIPS) PolSolve allows to use multiple sources to calibrate the polarization in one run. The workshop consisted of a lecture on the origin of polarized emission in AGN jets and how to calibrate and interpret interferometric polarization observations as well as hands-on sessions using the PolSolve software. The participants learned to calibrate the polarization of simulated VLBI observations as well as real GMVA data. An output of this workshop is a guide on polarization calibration with PolSolve, which can be used as a reference by current and future team members.

Table of contents

Required Software – Installation instruction (tested at MPIfR computers):

- PolSolve is currently in Python2, so you need CASA version < version 6 (should be something 5.7.xxx)

You can install PolSolve basically in three ways:

1. A local installation of AIPS and CASA including CASA-poltools
2. A docker machine (by Alejandro Mus)
3. A virtual machine (by Ivan Marti-Vidal)

- Poltools: https://code.launchpad.net/casa-poltools
- Download the scripts (see later)

1) Installing directly along a casa version

1.) Prepare CASA:

a. Open the file .bashrc

b. Add line: export PATH=${PATH}:/soft/astro/casa/casa-release-5.7.0-134.el7/bin

c. Close file and do ‘source .bashrc‘ in terminal, casa now can be started by running ‘casa‘ in terminal

2.) Install and compile PolSolve:

a. Download casa-polsolve from https://owncloud.mpifr-bonn.mpg.de/index.php/s/3d6ETx3i3mqPXXs

b. Unpack package

c. cd to polsolve directory

d. terminal: rm -f so.*

e. terminal: /soft/astro/casa/casa-release-5.7.0-134.el7/bin/python setup.py build_ext –inplace

3.)Load PolSolve into casa:

a. terminal: /soft/astro/casa/casa-release-5.7.0-134.el7/bin/buildmytasks

b. run casa one time, close casa

c. cd to directory ~/.casa

d. Edit (or create) the file: init.py, add the lines:
    i. import sys
    ii. sys.path.append(<path to casa-poltools>)
    iii. execfile(<path to casa-poltools>/mytasks.py)

4.) Test casa

2) Installing using the docker

Instructions: https://owncloud.mpifr-bonn.mpg.de/index.php/s/yk6CwDgYRCa6jHx

This is a short guide on how to install docker and how to download the docker image of casa/Pol-Solve.

Important links:
• https://docs.docker.com/engine/install/
• https://github.com/AlejandroMus/kracs

Once the installation is finished, you can follow the README in the github page. Please, try to install Docker following the instructions depending on your OS and pull the image (see last page) before the tutorial, since it is big and it take some time.

LINUX (Ubuntu):

Official documentation: https://docs.docker.com/engine/install/ubuntu/

To install docker in ubuntu, follow the steps by copy-paste the lines in a terminal:

```
sudo apt-get update

sudo apt-get install \
ca-certificates \
curl \
gnupg \
lsb-release

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg

echo \
 "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-
keyring.gpg] https://download.docker.com/linux/ubuntu \
 $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
sudo docker run hello-world

sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
docker run hello-world
```

If you have another Linux distribution, please follow the instructions here
https://docs.docker.com/engine/install/

MAC:

Installing docker in Mac is very easy. You can use the AppStore. I link the official documentation.

https://docs.docker.com/desktop/mac/install/

WINDOWS:

Installing docker in Windows it is a bit tricky. Please, install WSL and we will install it during the training session.
Recommended to use the VirtualBox

Pulling image with software

Once you have docker install and you could run successfully by typing in the console

docker run hello-world

you can type in the shell terminal

docker pull alejandromus/kracs:eht_m87_latest

Troubleshooting docker

I encountered some problems on MAC when installing everything as in the instructions you may have to run:

docker run -dit -P --name test -p 8888:8889 --env="QT_X11_NO_MITSHM=1" $DISENV -v /Users/baczko/Polarisation_WS:/host alejandromus/kracs:eht_m87_lates

docker start test

docker attach test

cd /host/

######## Below the way to start casa/polsolve and loading a script file
casa

tget polsolve

execfile('Pol_exercise_simulate.py')

2) Using a virtual machine provided by Ivan

Please download http://consigna.uv.es/g?recoge:ca:5460_5445_4252_9172
Passwort VLBIrocks

Online Material and Website:

Website: https://events.mpifr-bonn.mpg.de/indico/event/261/

Recordings: https://events.mpifr-bonn.mpg.de/indico/event/261/material/0/

Slides: https://www.uv.es/imarvi2/workarea/BONN-2022_POL.pdf

Data and scripts: https://owncloud.mpifr-bonn.mpg.de/index.php/s/WMYeabdbADH6Nss

Find more files and information at the deki page:

https://deki.mpifr-bonn.mpg.de/M2FINDERS/Workshops/Polarisation_Workshop

Outline of the workshop:

There exists a slack space for the workshop with some discussion on trouble shooting and calibration outcomes. If you are interested drop Anne-Kathrin Baczko or another member of M2FINDERS a mail to be added. There is a copy of the whole slack discussions and files in the online folder.

Tutorial

DAY1: Get simulated data and play with it (how does it look with or without dterm, polarized/not polarized etc.):

Summary of files and data sets required

- Scripts 1st day, get them to your local machine:
  wget https://www.uv.es/imarvi2/workarea/MPIfR_POL_WORKSHOP_2022_DAY1.tar.gz
  and tar xzf MPIfR….
  located also in the owncloud folder: original_files/ MPIfR_POL_WORKSHOP_2022_DAY1/
- Best make a folder, e.g. DAY1, where you copy the files to

Short overview scripts

- Pol_excercise_simulate.py: Generate simulated data
- Plot_traces.py: To plot closure traces: the observables that are derived from the visibility matrices
- Pol_excercise_solve_selfcal.py: To solve with PolSolve
- Pol_excercise_solve_similarity.py: Similar to LPCAL

Further you need the EHT configuration file:

- EHT station file: https://www.uv.es/imarvi2/workarea/EHT.cfg

```
# observatory=EHT
# COFA=-0.0,-0.0
# coordsys=XYZ
2225060.81    -5440059.60    -2481681.15    70.    AA
2225039.53    -5441197.63    -2479303.36    12.    AP
-1828796.20    -5054406.80    3427865.20    10.    AZ
-5464584.68    -2493001.17    2150653.98    15.    JC
-768715.632    -5988507.07    2063354.85    50.    LM
-5464555.49    -2492927.99    2150797.18    6.     SM
0.01           0.01           -6359609.70    10.    SP
5088967.75    -301681.186    3825012.21     30.    PV
```

- TRACK_C.listobs : This is just a copy-paste from a real 2017 EHT campaign. If this file is used for ‘nscan’ variable, this time-setup will be used
- Require in path (folder DAY1): Casa-poltools:

Tutorial: now the fun starts

Go to DAY1 and start casa: $ casa

In casa we will need 3 tasks to be called with tget task: polsolve, polsimulate, CCextract

- >: tget polsolve
  >: inp (will give you all the set of input parameters to polsolve)
- Call also the other 2 with tget

Start with Pol_execise_simulate.py (for an explanation of the script see recording 2 (tutorial1) around 0:25:00):

Important for changing how data are simulated: the configuration header at the beginning (CASE,VISNAME,NCHAN,DOCLEAN)

We will look at different cases (see top of script comments CASES: 0 to 7)

Now first run of Pol_exercise_simulate.py

- First test: choose the CASE , for the start CASE = 0
- In casa: >: execfile('Pol_exercise_simulate.py')
- Ignore 'severe errors', these are totally normal as long as the script finishes with a dataset
- Finished: You will obtain a measurement dataset (for casa) and a uvfits file (for AIPS)

Plotting things

- Task: plotms
  this will open a new window. In File select the measurement-set: select POLSIM_CASE-0.ms
  Go to 'Axes':
  Choose x-Axis: UVdist
  Data: Amp
  press Plot will result in some mess as everything is plotted, all correlation parameters in one plot
- It might be that you have to provide the filename already when calling plotms as plotms('filexxx.ms')
- Better go to 'Display": Colorize: 'Corr' , then plot again. Now there are two colors (RR,LL around one and RL,LR around 0). Dispersion is due to the thermal noise.
- Select on the bottom the button 'generate a region' and select a region in the plot (square with a plus), then press button 'Locate' (Lupe on the down) will give a list of all the information about these scans, e.g., select a region at around '1' you will get that all Corr=LL or RR
- Use the 'remove' button' (square with '-') to remove the region and you may test with another region
- Now lets plot the phases: RR/LL phases close to 0 and RL/LR are totally spread. This means that there is no signal in crosshands, as in phase thermal noise manifests as the phase being spread everywhere.

Now other case: e.g. CASE = 1

- Execute again (: >: execfile('Pol_exercise_simulate.py')
- Look again at plotms, what is different or the same

END OF RECORDING Day 1, Session 2

Day 2: using simulated and real data: a GMVA data set

START RECORDING Day 2, Session 1, Part 1

Back to the script from yesterday, but today with polarization **Pol_exercise_simulate.py**

Question: Why is dterm calibration needed?
Answer: When you split the signal in two polarization, as you have a finite bandwidth, splitting will not be perfect and some signal will be leaked from one polarization to the other. Meaning, in the output of the right circular polarization there will be the right circular polarization plus a delayed and de-amplified version of the left circular polarization. This means we need complex numbers to model the leakage (amplitude and delay).

Case 1

- We start with Case 1: unpolarized with dterm
- After script did run start again msplot: select file with ‘case1’; x axis=’uvdist’; data=’ampl’; colorize=’Corr’ (in tab display)
- Colors: blue/turquize (rr,ll), green/lila (rl,lr) -> there is signal on the crosshands, but we did not simulate polarized sources, so this is leakage
- Now plot data=’phase’ and plot:
  RR,LL: stable (effect of the dterms in parallel hands is low)
  RL,LR: All over the place
- Try plotting with x-axis=’Elevation’: seems as smooth function of the elevation, so what we see looks like instrumental polarization
- Keep in mind: casa is not designed for doing VLBI
- If we would run polsolve on this, polsolve would use this elevation dependence to figure out the dterms, if it works and we obtain the dtemrs and apply them the phases should all be flat
- Hard thing will be with extended sources to disentangle the source brightness distribution and the instrumental polarization

CASE 2: polarized point source without dterms

- Run script and plotms (select CASE2)
- Phase vs uvdist
- All phases are distributed around 0, LR/RL sounds more noisy
- You are welcome to play with the script
  e.g., change Line 89: currently Stokes V: 2%, Q= 10%, Stokes I = 1Jy
- Looking at amplitude vs uvdist: clearly not a Rayleigh-distribution anymore, one of the parallel hands is slightly higher, that’s because we have stokes V (if stokes V is positive RR should be higher than LL)
- Plot closure traces of all 3 cases (not affected by any Jones matrix, only on the brightness structure), these are robust against any calibration effects

Now in casa use function **Plot_traces.py** , at the beginning antennas= select antennas you want to plot the closure traces for, just keep the script as it is

- Execfile('Plot_traces.py'), which defines the function
- PlotTraces(vis=['POLSIM_CASE_0.ms', 'POLSIM_CASE_1.ms', 'POLSIM_CASE_2.ms',])
- Plots will be saved
- At the beginning: low elevations at ALMA, baselines to ALMA are the most sensitive, because of the polsim simulates more noise, as soon as the elevation becomes higher noise goes down. We don't see the large dterms in the closure traces. Discussion about how noise is simulated see recording from 0:30:00 onwards

Now CASE=4 (polarized double source without dterms)

- Execfile and msplot (settings as before)
- Amp vs. uvdist: beautiful structure in RR/LL. It is not easy to separate how much would be due to structure and instrumental effects. In a real observations we don't know. RL/LR all this structure is due to source structure

Lets clean this in casa

- Tget tclean (new, buggy version) or clean (old, more bug-corrected version) ; lets try tclean
- Inp to get the list of inputs (many)
    - vis = 'POLSIM_CASE-4.ms'
    - selectdata (allows to select several things, interesting for multi-source or multi-epoch data, we are not setting something here)
    - datacolumn: You cannot change the original data., it's the data as it comes from the correlator. The 'corrected' does not exist, yet: datacolumn = 'data'
    - If a Jones-Matrix is created casa creates a new column 'correctedcolumn' which si a copy of the data but visibilities are replaced by the calibrated visibilities
    - third column 'modelcolumn' which stores the model (needed for selfcalibration)
    - imagename='test' (name of directory that will be created in which images will be saved)
    - imsize = 1024
    - cell = [0.000001arcsec] (size of 1px, could also try mas, but that did not work at least in older casa versions)
    - stokes = 'IQUV' (allow to clean all 4 stokes parameters)
    - startmodel: one can use an image of an extended source as a start model
    - deconvolver='hogbom' for interferometers for sparse uv-coverage
    - weighting = 'uniform' (if set to 'briggs' a new 'robust' parameter comes up)

- niter = 1000
- usemask = ‘user (the user set it by hand, you can also use images)
- savemodel=’modelcolumn’ (don’t use virtual, does not work with polsolve)
- cyclefactor=2.5 allows to run major cycles if >1.0, there higher the more often major cycles will be done
- interactive=TRUE (to get the graphical interface)
- go

- Graphical interface opens to define a mask
  - When you clean in polarization: the things you can set in the upper green box is what clean will do. Always, before each new clean you have to set ‘All polarizations’, if left to ‘This Polarization’ only stokes I will be calibrated.
  - Approach: hybrid imaging on stokes I (including self-calibration of gains on stokes I), when we are happy with this the gains will not be touched anymore, and we clean the stokes I image and set a mask, which is spread to QUV, then we start the self-calibration of the dterms
  - Our simulated data is expected to be calibrated in gains
  - Zooming in: magnifying glass at the top icons. You can set the action there to a specific mouse button.
  - E.g. create a polygon and if you are not happy with your selection press ESC, if it already is there press ‘erase’
  - Zoom set to left mouse button. Click ‘left where you want, drag region and double click inside the region to zoom in and double click outside to zoom out.
  - You can switched between the stokes parameters with the shift on the right side of the image on the top, just move it
  - Setting region: make region and double click inside until happy
  - Hit button “green arrow”: it runs the clean and comes back to the interface, then just continue, tclean seems to come back at every major iteration. If you don’t want to come back after each major cycle press the other button (blue arrow)
- Run Imview: to see the results of the cleaning
  - Click button to open images. Clean has created several images
  - Test.image: final image
  - Test.mask: the mask
  - Test.psf: dirty beam
  - Test.residual: image minus the convolved model
  - Select test.image ant raster image (there is a window with data display options now, set in order to see the image better

Now simulate CASE 4, 5 & 6 starts at 1:39:40

- PlotTraces(vis=[case 4, case 5, case 6])
- Now there is a structure. This is related to the differential polarimetry of the double source: case 6 is the rotated case (all stokes components are rotated in the same way in the Poincare sphere), hence the signal is exactly the same
- How to use this information to learn about the polarization source structure, inverting this into an image is still an open question

Test: set exactly the same QU for both sources: Line 101:

- For Case 6 set I=[1.0, 0.75]; Qfrac = [0.1, 0.1]; Ufrac = [0.0, 0.0]; Vfrac = [0.01,0.01]
- Simulate again (this is now the new case 6) and PlotTraces
- Somehow nothing changes

Try Qfrac=[0.0,0.8] rest same

- The slope of the phase and the hight of the amplitude seems to be sensitive to where the polarization is located

Now PolSolve, Case 5 using similarity approach

We will use the similarity approach, same strategy as LPCAL in AIPS

We need **'Pol_exercise_solve_selfcal.py'** and **'Pol_exercise_solve_similarity.py'**

At 1:54:53 until 2:06:00 (local time 11:56) the recording seems to stop!!! Nothing changes and then we are in 'Pol_exercise_solve_similarity.py' , resumes at 2:07 (local time 12:08)

Switch to Recording Day 2 Session 1 Part 2 starting at local time 11:58

We start with Pol_exercise_solve_similarity.py

Parameters:

- You only need to edit the first few lines (weighting, robust, MSNAME) at the beginning
- Have a look at the recording Day 1 session 2 Part 2 to have the script explained by Ivan. Below are a few notes on the script taken from the recording
    - weighting (set to whatever you want: uniform, natural…)
    - MSNAME: name of the measurement set, here '5' meaning double source with dterms
    - Line 30: what the script does : clean in interactive mode, we are going to solve for the fractional polarization, we are dividing the source into sub components and polsolve will solve for polarization and dterms, and we are cleaning anyway the four stokes parameters (just for fun, we don't need them in this approach).
    - This interactive clean will produce a mask for us, we will clean manually (major and minor sycles)

- Task ‘CCextract’ (part of the poltools) will use the model clean (please read help file of ccextract to see what options there are, e.g., for regions: help CCextract or inp in CASA , we will use the manual option(a nice interface will appear)
- ALLCCs gets all the subcomponent files
- CLEAN_models=ALLCCs if you want to use the similarity approach
- If you know the source polarization you can set it in Pfrac and PolSolve, at the moment its just set to 0 and to solve for it

- After a question some explanation on how to make the code more general (starts at 9:45)
  - If you know the mounts you can have a dictionary with antenna names and mounts and have a code to retrieve the antenna names from the measurementset and fill in the mounts list (3-4 lines of code) using the CASA TB-Tool

Now run it with the similarity approach (starts at 12:44):

- Execfile(‘Pol_exercise_solve_similarity’)
  - Clean graphical interface should appear.
  - Zoom in and look at the different stokes: stokes U is shifted wrt I, in constract to before there are now Dterms
  - Click again ‘All Polarizations’
  - Generate clean regions
  - Hit the blue arrow (between the green circular and the red cross)
  - After clean comes window ‘ccextract’ up
  - Green/red points are the clean components, color denotes the intensity; press ESC; to draw polygon around the clean components that should be in sub component 1 (right cloud), and the same for the left
  - Then a new plot with the clean image in grey scale and the sub component components in different colors (red for subc1 and green for subc2) and we have solved for the polarization for both individual sub components.
  - Polsolve finished in the background
  - Have a look at the term, what Polsolve did. E.g., There are 8 antennas: in principle one can set a dterm if you know it and only fit for the others
  - Other opened plot ‘SubComponentFitting’ correlation is perfect. In real life, with real data it’s not so easy

Start Day 2 Session 2

- We applied the similarity approach (very similar to LPCAL)
- Rerun CCextract to get used with the interface
  - Tget CCextract
  - Go

- Should run the same way as before
- Set Polygons to split into sub components (one polygon for the left and one for the right blob)

Now we switch to the selfcalibration approach start at 0:10:50: Pol_exercise_solve_selfcal.py

- Only need to change the first parameters (NITER, Weighting, robust, MSNAME)
- Tclean: same settings as before
- Selfcalibration: will run the clean the same way as before and polsolve in a different approach but with interactive=FALSE, so no graphical interface will be opened
- Polsolve after clean difference to before:
  - Clean_models not set to the output from CCextract, but to a CASA Modelimage. After finished Clean there are several images created: xx.model will be used here
  - PolSolve= [False], because we are not fitting the source polarization but use the output from clean
  - Target_fields='POLSIM' gives the name of the source for which you want to apply the dterms at each iteration. If that is not set it will not solve for dterms, only clean
  - In this example we are using only one source, but in principle you can solve for multiple calibrator sources to cover most of elevations
- Run it in CASA (execfile…) Now clean-window will appear: remember to press 'All Polarizations' and create boxes and run it. Here use only the middle arrow, but in reality you want to clean slowly using the circular arrow
  - Correlation Plot looks nice
- Once finished Tget polsolve to make some diagnostic plots
  - Plot_parang=True
  - Plot_residuals=True
  - Go and have a look at the plots
  - Feed angle (all mounts have been taken into account correctly): why are ALMA (AA) and APEX (AP) so different? Answ: different mounts
  - RL & LR Residual Plot: If there is a circle or arcs its an instrumental effect, if its something more stable its source intrinsic (model of the source not ideal). If residual everywhere so small we did a good job

BREAK back at 1:04

Where are the dterms stored?

- ITER_7.Dterms fodler are not supposed to be used (the CASA version that Ivan uses does not deal with Nasmyth mounts
- If you run PolSolve, there will be an ascii-file: POLSIM_CASE-5.ms.spw_0.PolSolve.CovMatrix has all spectral windows that are included in the fit: Full access to the error model of the fit.The order of the POST-FIT COVARIANCE MATRIX is

given in the same order as the list above (see recording 1:08:00 and following for more info on this)

Now we go to real data 1:16:30: GMVA observation

GMVA mounts: Pv an Yb ar Nasmyth

Real test data day2: https://owncloud.mpifr-bonn.mpg.de/index.php/s/3d6ETx3i3mqPXXs/download

Put it into a folder, say 'Day2', data is project C151B go to folder and start casa

- Tget listobs (like listr, prtan in AIPS)
- Vis = 'DAY2_DATA2.ms'
- Look at the CASA logger

Note: we are reading a dataset exported from aips. There is several information lost when loading it into CASA. E.g., Ivan had to manually resotre the scan-ids. Also Antenna names are lost (they are saved into stations)

- Write a small script to write the antenna mounts for the antennas written in the logger. However, all participating antennas are AltAzimut, so we don't have to do something here as the default mount is AltAz
- Note by Thomas, if you are not sure about the mounts. Apply them, correct with CLCOR 'Pang' in aips and look at the RR over LL phases.
- Think about which sources to use as calibrator to use: lets say BLLAC

We will build a script from scratch now:

Tips for casa:
- Use dictionaries as arguments for CASA tasks, e.g. for all the common arguments for clean

Import modules:

```
import pylab as pl
import numpy as np
import os,glob
```

Give list of calibrators

```
CalSources =['BLLAC']
VisName = 'DAY2_DATA2.ms'
```

Implement clean (tget tclean and inp to get the parameters we need/want to change)

```
tclean_args= { 'vis':VisName,
                    'niter':1000,
                    'gain':0.05,
                    'imsize':1024,
                    'datacolumn':'corrected',
                    'cell':'0.00001arcsec',
```

```
                'stokes':'IQUV',
                'deconvolver':'hogbom',
                'weighting':'uniform','cyclefactor': 2.5,
                'interactive':True,
                'usemask':'user',
                'savemodel':'modelcolumn'}
```

Polsolve arguments (fields are the fields to be used in the calibration, target_fields are the ones that should be calibrated)

```
polsolve_args =     {'vis':VisName,
                        'field:','.join(CalSources),
                        'CLEAN_models':'model_column',
                        'PolSolve':[False for fdl in CalSources],
                        'target_field':','.join(CalSources)}
```

Now lets run tclean

```
for calsour in CalSources:
```

to make it easier, run only tclean for the calibrators that have not been run, yet:

```
        if not os.path.exists('$s.mask'%calsour):
....
        tclean_args['field'] = calsour
```

… see final scripts …

We need to keep this first mask, so lets name it in a special way:

```
        os.system('cp -r %s_first_clean.mask %s.mask'%(calsour,calsour))
```

Now the automatic part with the selfcalibration, we finally start selfcal

At the top of the file we have to also set NITER= 1 #Number of selfcalibrations and it goes on

```
tclean_args['interactive']=False

for it in range(NITER):
```

Copy from above and change:

```
        for calsour in CalSources:
                …
                os.system('rm -rf %s_clean_ITER-%i.*'%(calsour,it))
                tclean_args['imagename'] = '%s_clean_ITER-%i'%(calsour,it)
                tclean_args['mask'] = '%s.mask'%calsour
                tclean(**tclean_args)
                …
```

Now lets run polsolve

```
        polsolve(**polsolve_args)
```

Keep CASA-dterms

```
        os.system('rm -rf DTERMS_ITER-%i.tab'%it)
        os.system('mv %s.spw_0.Dterms DTERMS_ITER-%i.tab'%(VisName,it))
```

Now lets run it in CASA

```
execfile('Polsolve_selfcal_DATA2.py')
```

- Seems px-size is too small
- Remember to click 'All Polarizations'

- Run it and clean (data is already nicely calibrated, hence its easy here) and if your happy stop (press the red buttom with the x)
- Polsolve should start automatically (now only one iteration, so that's not the end of the story)
- Look at output once finished.
  IMPORTANT: following line must exist at the end
  "Applying calibration to field if 0 (spw0)"
  This means its applying the calibration and written a correcteddata column
- Looks just fine, everything did run as expected

Add another line below the polsolve_args before running clean to ensure that each time we run the script the to make the correcteddata column will be a copy of the datacolumn. To always start from scratch when running the script:

clearcal(vis=VisName,addmodel=True)

Now set NITER=10 or 20 (whatever you like) and let it run

Instructions on using LPCAL from Carolina Casadio: Pol calibrations and imaging.pdf
On the MULTI file RLDLY was already run, hence one should only follow the steps from CCEDT (CCEDT, LPCAL). Then the single source uvfits file has to be transformed into a MULTI file again with task MULTI, in order to apply the AN table to data. Ultimately use SPLIT with DOPOL 1.

Final script shared by Ivan

```
import pylab as pl
import numpy as np
import os, glob



CalSources = ['BLLAC']
VisName = 'DAY2_DATA2.ms'
NITER = 10


tclean_args = {'vis':VisName,
          'niter':1000,
              'gain':0.05,
              'imsize':1024,
              'datacolumn':'corrected',
              'cell':'0.00001arcsec',
              'stokes':'IQUV',
              'deconvolver':'hogbom',
              'weighting':'uniform',
              'cyclefactor':2.5,
              'interactive':True,
              'usemask':'user',
              'savemodel':'modelcolumn'}
```

```
polsolve_args = {'vis':VisName,
                 'field':','.join(CalSources),
                         'CLEAN_models':'model_column',
                         'PolSolve':[False for fld in CalSources],
                         'target_field':','.join(CalSources)}



clearcal(vis=VisName,addmodel=True)


## Get antenna names
tb.open(os.path.join(VisName,'ANTENNA'))
ANTNAM = tb.getcol('NAME')
tb.close()


# Run tCLEAN in interactive mode:
for calsour in CalSources:
  if not os.path.exists('%s.mask'%calsour):
    os.system('rm -rf %s_first_clean.*'%calsour)
    tclean_args['field'] = calsour
    tclean_args['imagename'] = '%s_first_clean'%calsour
    tclean(**tclean_args)
    os.system('cp -r %s_first_clean.mask %s.mask'%(calsour,calsour))



tclean_args['interactive']=False


### WE FINALLY START WITH THE SELFCAL
for it in range(NITER):

  print '\n\n\n    ITERATION  %i \n\n\n'%it


  ## CLEAN ALL SOURCES
  for calsour in CalSources:
    os.system('rm -rf %s_clean_ITER-%i.*'%(calsour,it))
    tclean_args['field'] = calsour
    tclean_args['imagename'] = '%s_clean_ITER-%i'%(calsour,it)
    tclean_args['mask'] = '%s.mask'%calsour
    tclean(**tclean_args)


  polsolve(**polsolve_args)
  os.system('rm -rf DTERMS_ITER-%i.tab'%it)
  os.system('mv %s.spw_0.Dterms DTERMS_ITER-%i.tab'%(VisName,it))



## Plot Dterms:

  tb.open('DTERMS_ITER-%i.tab'%it)
  DTERMS = tb.getcol('CPARAM')
  DR = DTERMS[0,0,:]
```

```
DL = DTERMS[1,0,:]

subDR.cla()
subDL.cla()

subDL.plot(np.array([-0.2,0.2]),np.array([0.,0.]),':k')
subDR.plot(np.array([-0.2,0.2]),np.array([0.,0.]),':k')
subDL.plot(np.array([0.,0.]),np.array([-0.2,0.2]),':k')
subDR.plot(np.array([0.,0.]),np.array([-0.2,0.2]),':k')


for a in range(len(ANTNAM)):
  subDR.plot(DR[a].real, DR[a].imag,symb[a],label=ANTNAM[a],mew=2)
  subDL.plot(DL[a].real, DL[a].imag,symb[a],mew=2)

subDR.set_xlim((-0.2,0.2))
subDR.set_ylim((-0.2,0.2))
pl.legend(numpoints=1,ncol=1,bbox_to_anchor=(1.4, 1.0))
subDL.set_xlim((-0.2,0.2))
subDL.set_ylim((-0.2,0.2))

subDR.set_xlabel('DR Real')
subDR.set_ylabel('DR Imag')

subDL.set_xlabel('DL Real')
subDL.set_ylabel('DL Imag')

title.set_text('ITERATION %i'%it)

pl.savefig('Dterms_Iteration_%i.png'%it)
```

FINALIZES DAY2 RECORDINGS

Day 3- Session 1 (There is only one)

- Script made during Day2 with some additional plotting: Polsolve_selfcal_DATA2.py

Have a look at what we did yesterday:

in casa run *imview('BLLAC-clean_ITER-9.image')* (clean components convolved with beam)

Lets zoom in: this is BLLAC in stokes I. One window with the image another window for setting how the plot is made (set colorscale, the scaling etc.)

Questions: How to build an EVPA and polarization intensity image

1. CASA way
2. using script using matplotlib from Ivan

CASA way:

- Task: tget immath
  - imagename='BLLAC_clean_ITER-9.image'
  - mode='poli' (read the help file for the many options, first the polintensity, from the we set a threshold and then the polangle to only have polangles were snr is high)
  - outfile = 'final_polit_image' (to something you can remember)

- go
- It gets difficult to see as in the recording the speaker is now enlarged
- imview('final_polit_image')
- set threshold to 0.02

- Back to immath
  - mode='pola'
  - inp -> there is a new parameter
  - polithres = 0.02
  - outfile='final_polit_image_polang'
  - window opens (set how you want to display the map)
    - select ITER-9.image as 'contour map'
    - colorcode='hotmetal' 😊
    - in stokesI image: contour level: 0.05,0.1,0.2,0.4,0.5,…
  - select the anlge.image -> click on vector map
  - select in the window again how the EVPA are plotted
    - set x-direction and y-direction to 1 and other things until you see the arrows
    - For some more options on how to change the appearance of the plot see recording Day 3 session 1 from 0:20:00

Script: from 0:29:00 CASA_PolHelper.py

- execfile('CASA_PolHelper.py')
- plotImage('BLLAC_clean_ITER-9.image',mSigmaCut=4.0,dPol=2,zoom=10.0)

Plan: save images for each iteration and compare afterwards

- open Polsolve_selfcal_DATA2.py
  - add execfile('CASA_PolHelper.py') under inputs
  - in the clean loop after tclean(**tclean_args) add plotImage('%s_clean_ITER-%i.image'%i(calsour,it), mSigmaCut=4.0,dPol=2,zoom=10.0,figName='%s_fullpolImage_%i.png'% (calsour,it))

Final scripts used for the dterm calibration of GMVA data from Day3 (**Polsolve_selfcal_DATA2_DAY3.py**):

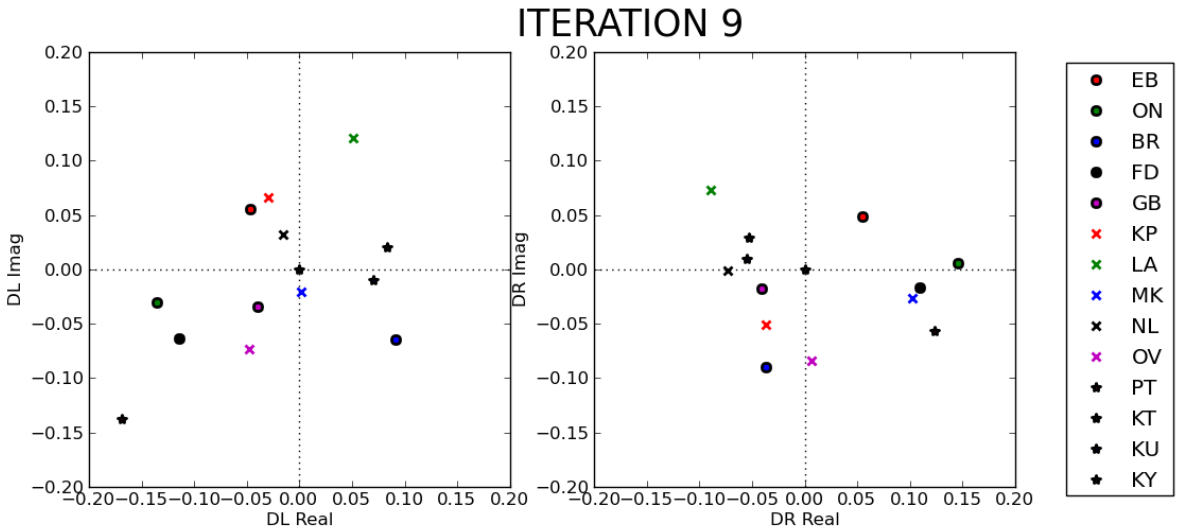
```
import pylab as pl
import numpy as np
import os, glob

execfile('CASA_PolHelper.py')
```

```
CalSources = ['BLLAC']
VisName = 'DAY2_DATA2.ms'
NITER = 10


tclean_args = {'vis':VisName,
           'niter':1000,
               'gain':0.05,
               'imsize':1024,
               'datacolumn':'corrected',
               'cell':'0.00001arcsec',
               'stokes':'IQUV',
               'deconvolver':'hogbom',
               'weighting':'uniform',
               'cyclefactor':2.5,
               'interactive':True,
               'usemask':'user',
               'savemodel':'modelcolumn'}


polsolve_args = {'vis':VisName,
             'field':','.join(CalSources),
                     'CLEAN_models':'model_column',
                     'PolSolve':[False for fld in CalSources],
                     'target_field':','.join(CalSources)}









fig = pl.figure(figsize=(12,5))
title = fig.suptitle('TEXT',fontsize=25)
subDL = fig.add_subplot(121)
subDR = fig.add_subplot(122)
fig.subplots_adjust(left=0.08,right=0.8)

symb = ['or','og','ob','ok','om','xr','xg','xb','xk','xm','*r','*g','*b','*g','*k','*m']









clearcal(vis=VisName,addmodel=True)


## Get antenna names
tb.open(os.path.join(VisName,'ANTENNA'))
ANTNAM = tb.getcol('NAME')
tb.close()


# Run tCLEAN in interactive mode:
for calsour in CalSources:
```

```
  if not os.path.exists('%s.mask'%calsour):
    os.system('rm -rf %s_first_clean.*'%calsour)
    tclean_args['field'] = calsour
    tclean_args['imagename'] = '%s_first_clean'%calsour
    tclean(**tclean_args)
    os.system('cp -r %s_first_clean.mask %s.mask'%(calsour,calsour))



tclean_args['interactive']=False


### WE FINALLY START WITH THE SELFCAL
for it in range(NITER):

  print '\n\n\n    ITERATION  %i \n\n\n'%it


  ## CLEAN ALL SOURCES
  for calsour in CalSources:
    os.system('rm -rf %s_clean_ITER-%i.*'%(calsour,it))
    tclean_args['field'] = calsour
    tclean_args['imagename'] = '%s_clean_ITER-%i'%(calsour,it)
    tclean_args['mask'] = '%s.mask'%calsour
    tclean(**tclean_args)
    plotImage('%s_clean_ITER-%i.image'%(calsour,it),mSigmaCut=4.0,dPol=2,zoom=10.0,fig-
Name='%s_fullpolImage_%i.png'%(calsour,it))



  polsolve(**polsolve_args)
  os.system('rm -rf DTERMS_ITER-%i.tab'%it)
  os.system('mv %s.spw_0.Dterms DTERMS_ITER-%i.tab'%(VisName,it))




## Plot Dterms:

 # fig = pl.figure()
  pl.sca(subDR)
  tb.open('DTERMS_ITER-%i.tab'%it)
  DTERMS = tb.getcol('CPARAM')
  DR = DTERMS[0,0,:]
  DL = DTERMS[1,0,:]

  subDR.cla()
  subDL.cla()

  subDL.plot(np.array([-0.2,0.2]),np.array([0.,0.]),':k')
  subDR.plot(np.array([-0.2,0.2]),np.array([0.,0.]),':k')
  subDL.plot(np.array([0.,0.]),np.array([-0.2,0.2]),':k')
  subDR.plot(np.array([0.,0.]),np.array([-0.2,0.2]),':k')


  for a in range(len(ANTNAM)):
    subDR.plot(DR[a].real, DR[a].imag,symb[a],label=ANTNAM[a],mew=2)
    subDL.plot(DL[a].real, DL[a].imag,symb[a],mew=2)
```

```
subDR.set_xlim((-0.2,0.2))
subDR.set_ylim((-0.2,0.2))
pl.legend(numpoints=1,ncol=1,bbox_to_anchor=(1.4, 1.0))
subDL.set_xlim((-0.2,0.2))
subDL.set_ylim((-0.2,0.2))

subDR.set_xlabel('DR Real')
subDR.set_ylabel('DR Imag')

subDL.set_xlabel('DL Real')
subDL.set_ylabel('DL Imag')

title.set_text('ITERATION %i'%it)

pl.savefig('Dterms_Iteration_%i.png'%it)
```

- See how the map changes with iteration, what structure does survive?
- How do the Dterms change?
- Example DTERM iteration after 9 iterations provided by Ivan Marti-Vidal:



Now: try yourself. Add more calibrators, play with the script, don't forget to press the 'All polarization' button before you draw any mask.

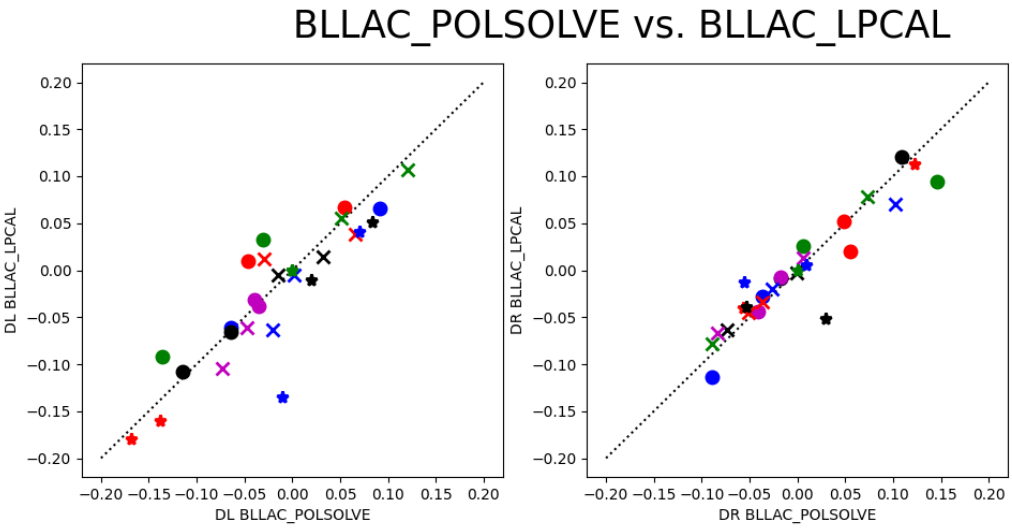
- **'May the force be with you ;-)**

Notes: for different tries make one directory for each run of Polsolve to not overwrite previous attempts

There is no recording for Day 3 Session 2 (afternoon)

Further scripts

- Scripts for DT comparison: DT_COMPARE.py
- Script for importing (sets pf) uvf files into a multisource measurement set: CASA_multi-source_importer.py
- Ivan did run LPCAL on BLLAC. And here is the Dterm correlation plot between PolSolve and LPCAL



Looks good. The original table from the paper seems to have some rotation applied as they did not correlate with the PolSolve Dterms.