CASA VLBI

WORKSHOP 2020

2-6 NOVEMBER 2020

LECTURE 9: SELF-CALIBRATION

Javier Moldón (IAA-CSIC)

CASA Common Astronomy Software Applications

JIVE Joint Institute for VLBI ERIC

RadioNet

JUMPING JIVE Joint Institute for VLBI ERIC

About this talk

With inspiration and resources from [ERIS2015], [ERIS2019], [casaguides], [DARA], [NRAO synthesis workshop], [ALMA-INAF]

Before this talk you should know...

- How interferometry works, basic principles.
- What are visibilities and their relation with emission in the sky
- The CASA environment, tools and syntax
- How to perform general calibration and imaging

You will learn...

- Why a data model is needed to calibrate a source
- The steps of a self-calibration loop
- General implementation, practicalities and decisions
- How to implement it using CASA tasks

Motivation

The quality and reliability of your science output depends on doing self-calibration properly. Make sure you understand it completely

Self-calibration

- Calibration is transferred from calibrators to a target source, which has limitations
- The sensitivity that you achieve is often limited by residual calibration errors
- Many objects have enough Signal-to-Noise (S/N) that they can be used to calibrate themselves to obtain a better image

Why it works?

Self-calibration is just calibration

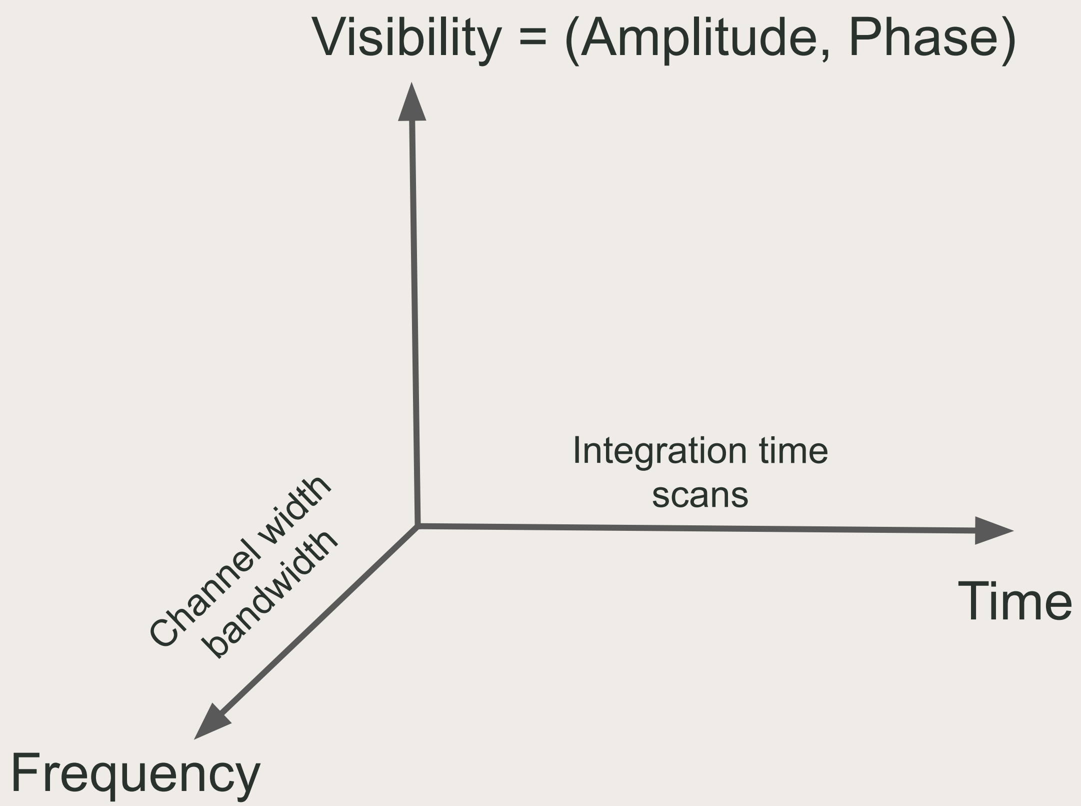
Redundancy

- For N antennas we measure N(N-1)/2 visibilities and after the calibration only N amplitude gains (N-1 phase gains) describe the complete calibration of the data
- It works because the number of baselines is much larger than the number of antennas so that **even an approximate source image** can improve calibration
- Errors in the model or low SNR can propagate into your self-calibration solutions, and you can diverge from the correct model ⚠

Data structure in CASA

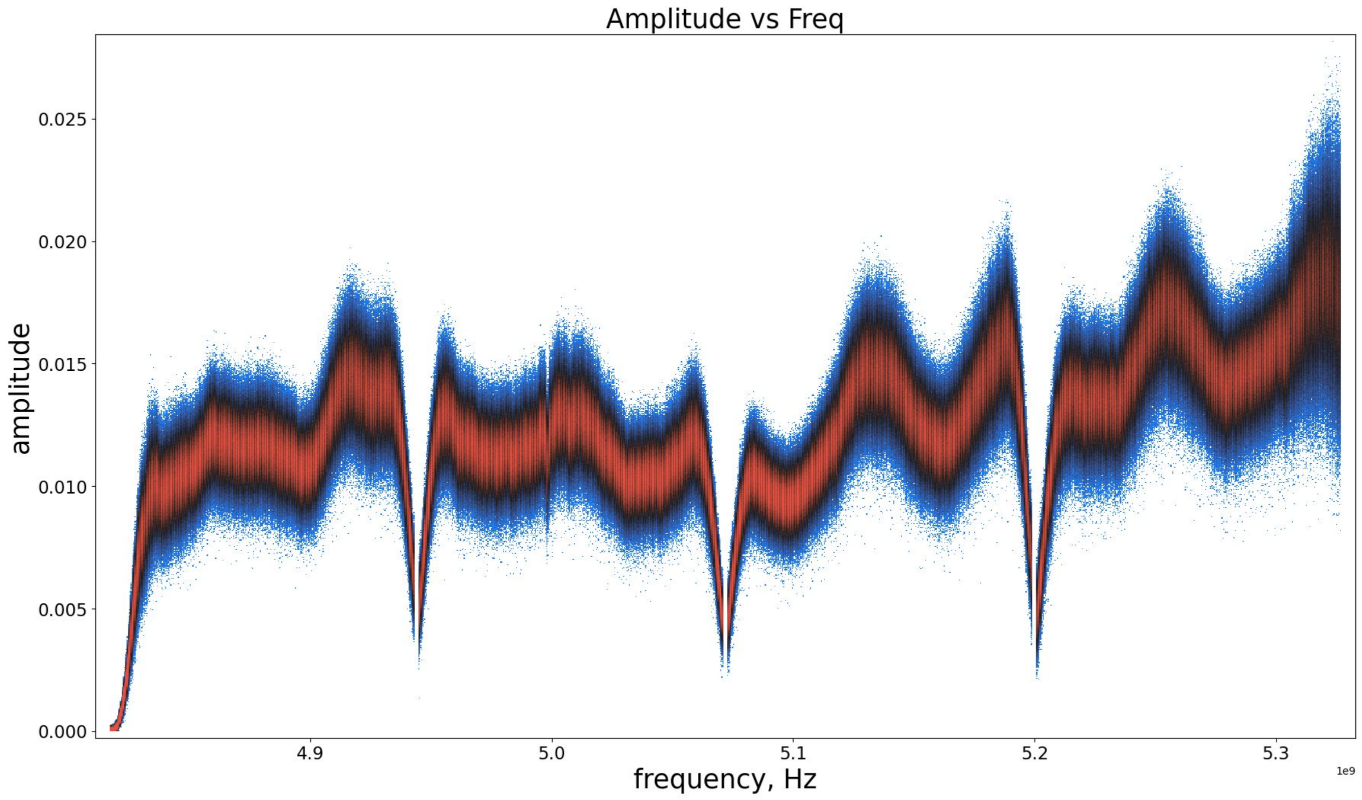
Measurement unit: visibility

Complex number that changes with frequency and time



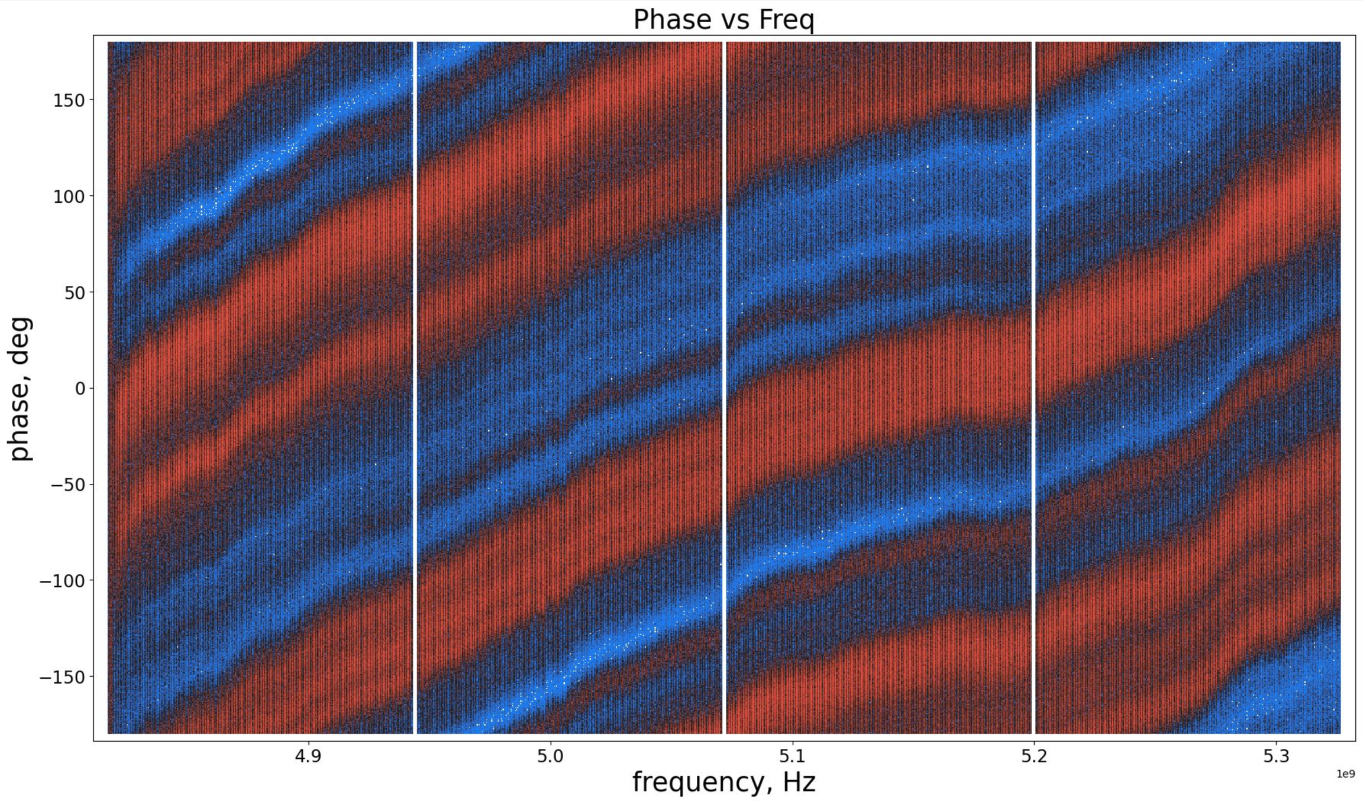
Bandpass: Amplitude vs freq

Shape is instrumental, no variation with time (unless observing setup is changed)

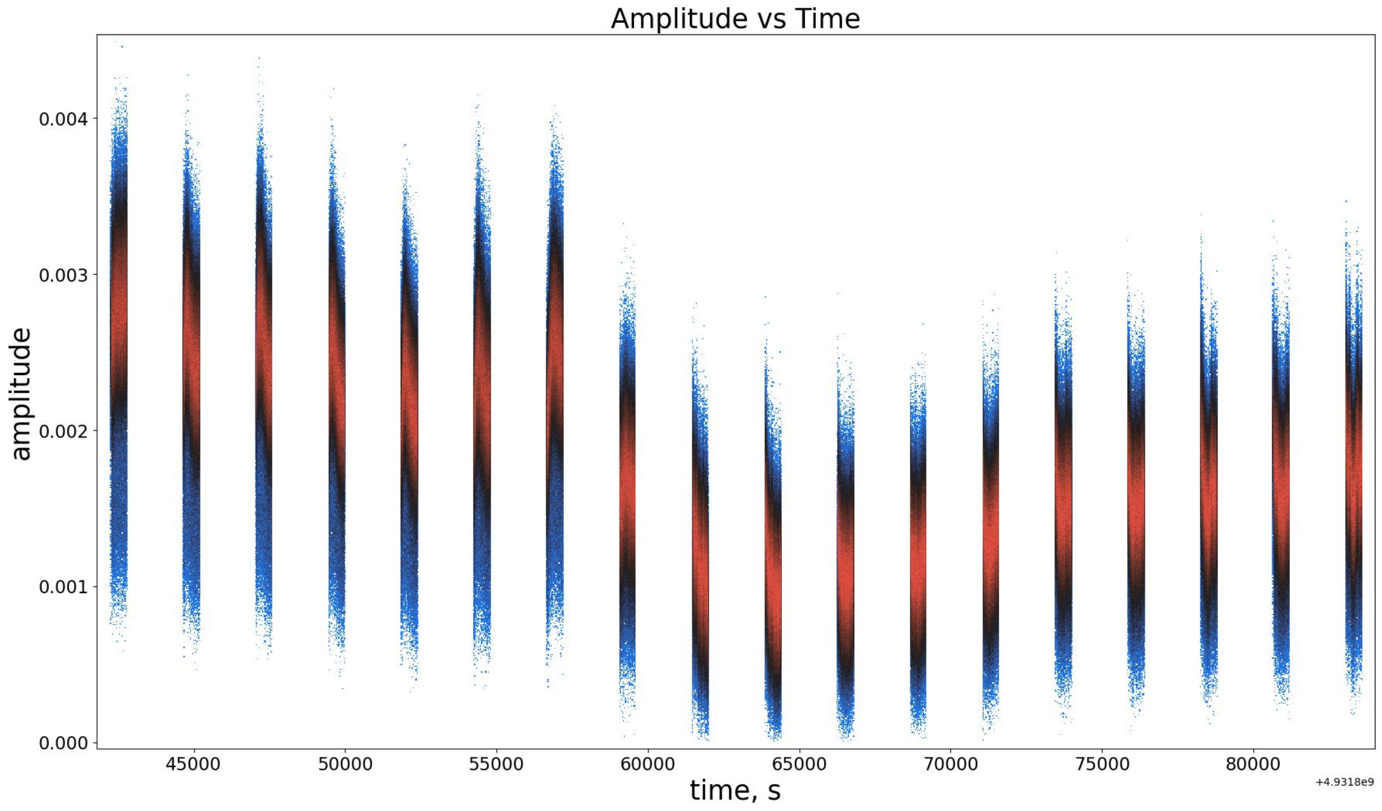


Phase delay (Phase vs Freq)

Instrumental, atmospheric, and source structure. In general, changes in minutes or more



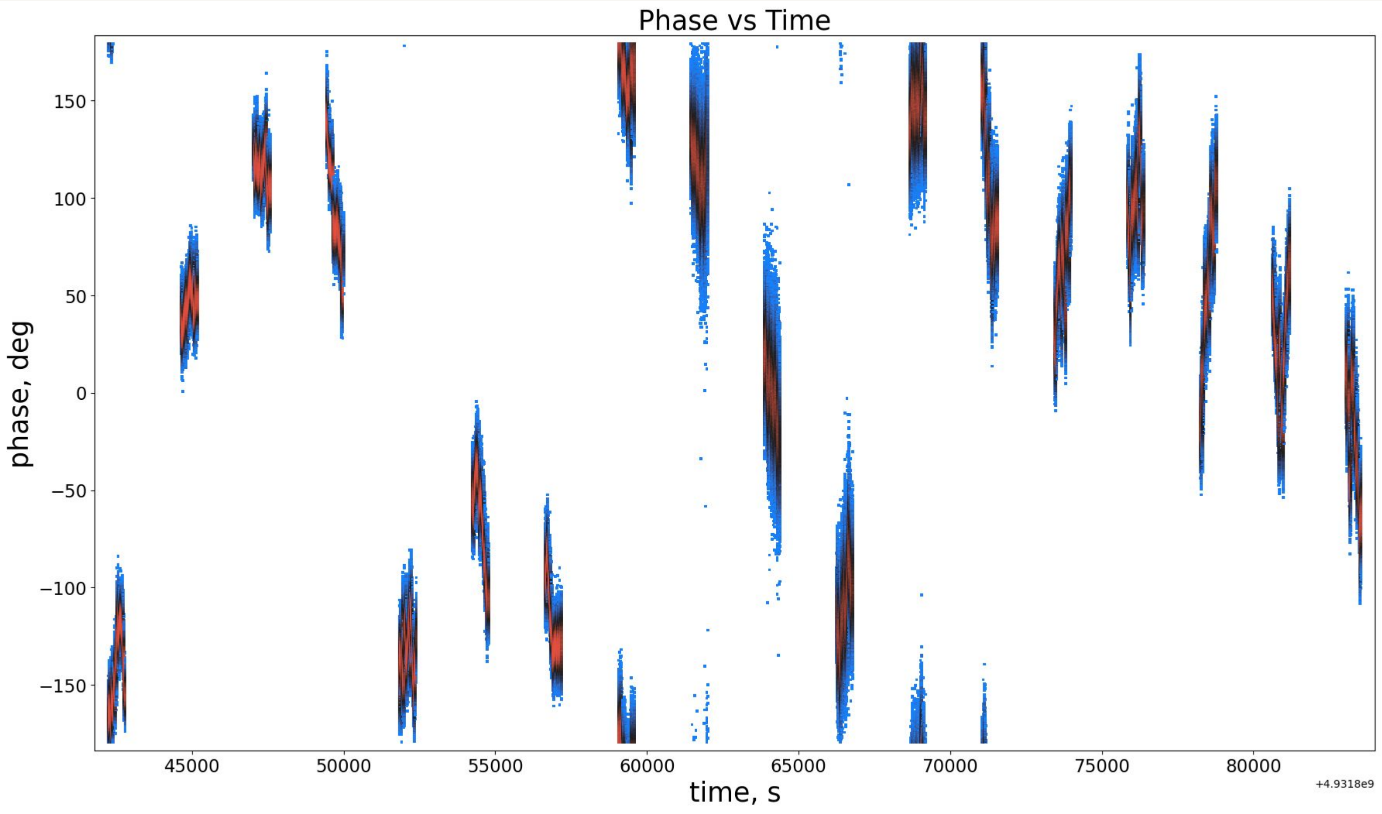
Amplitude vs time



Long-term variations (~hours) but also short-term variability

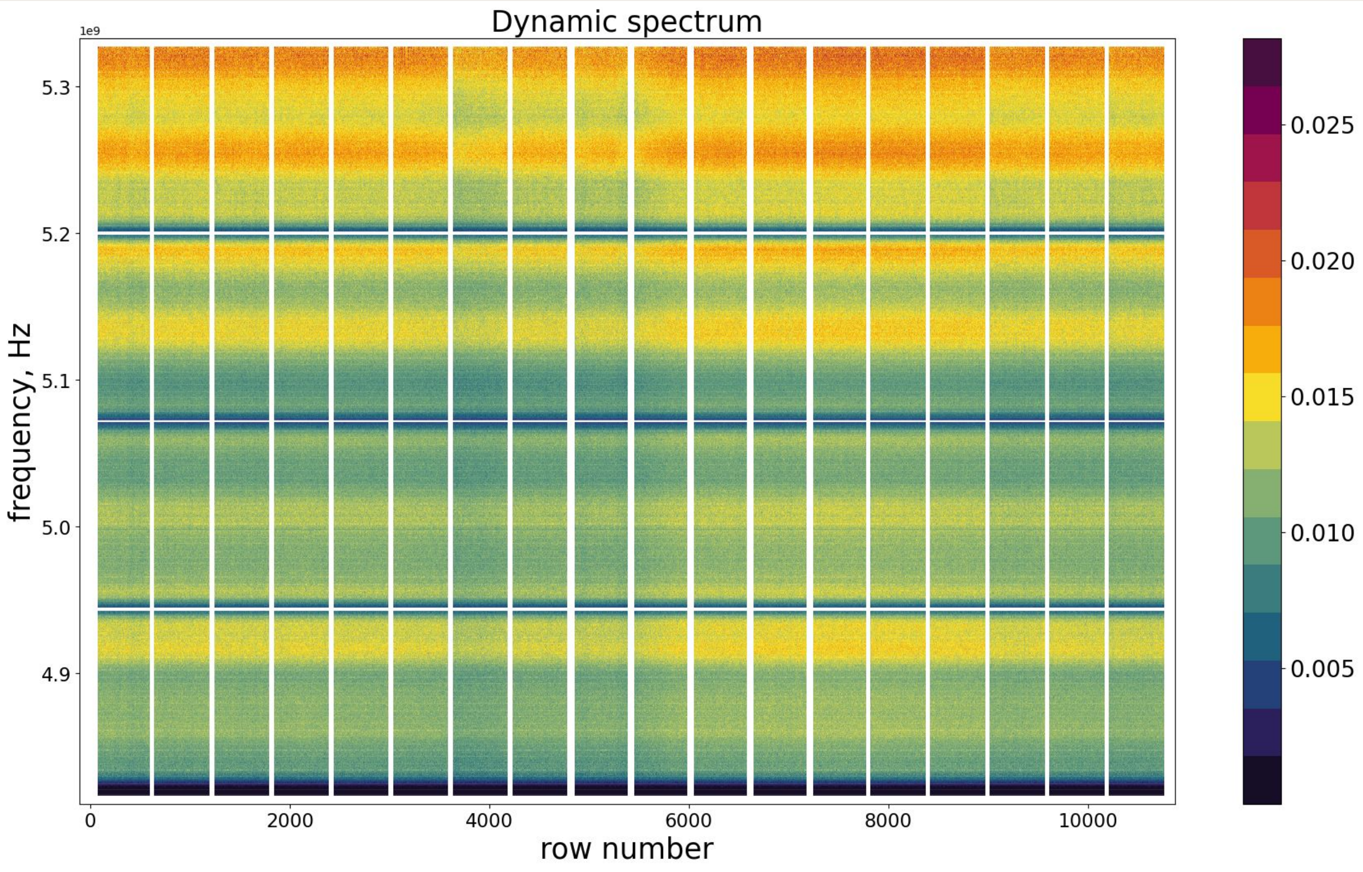
Phase vs time

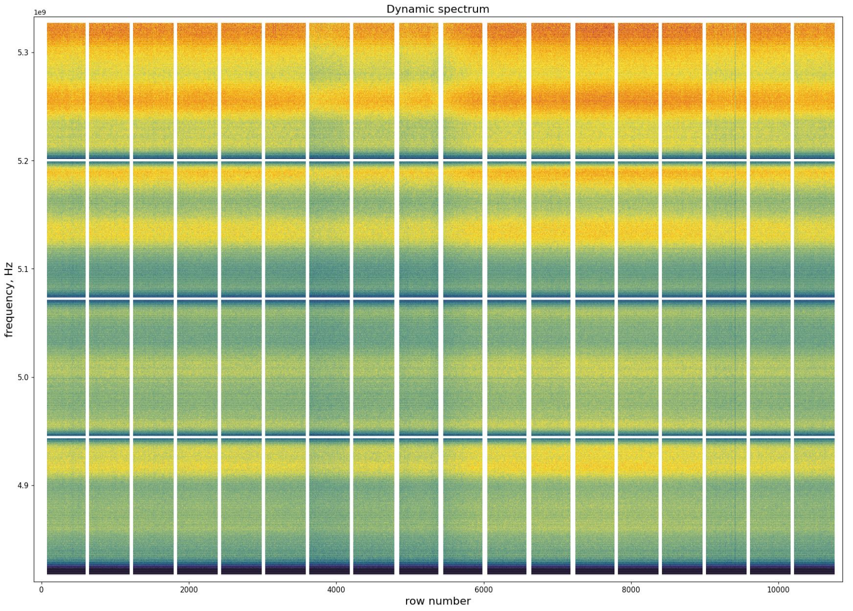
High variability both on short and long time scales. Intra-scan variability not corrected by phase referencing

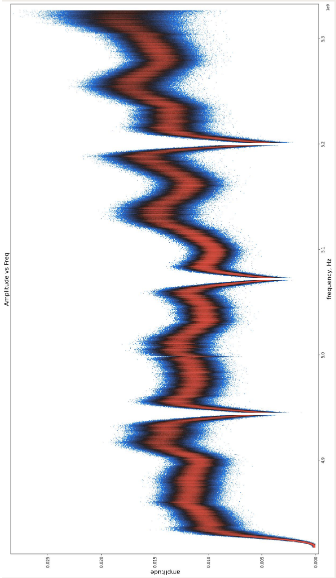


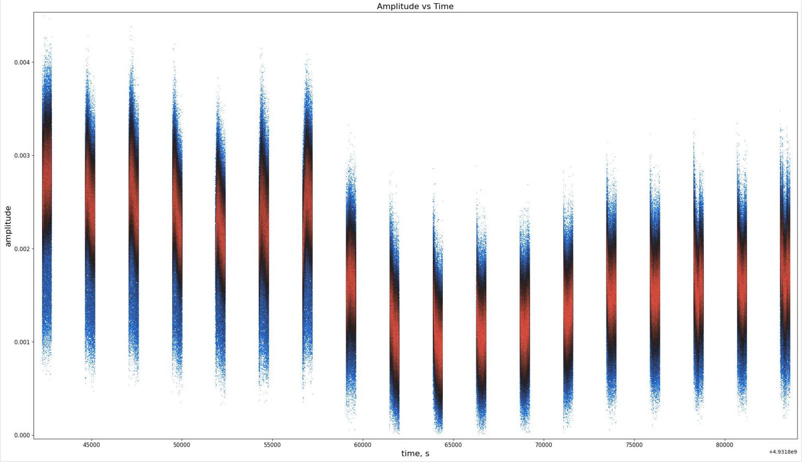
Dynamic spectrum (amp)

Vertical slices follow the bandpass shape



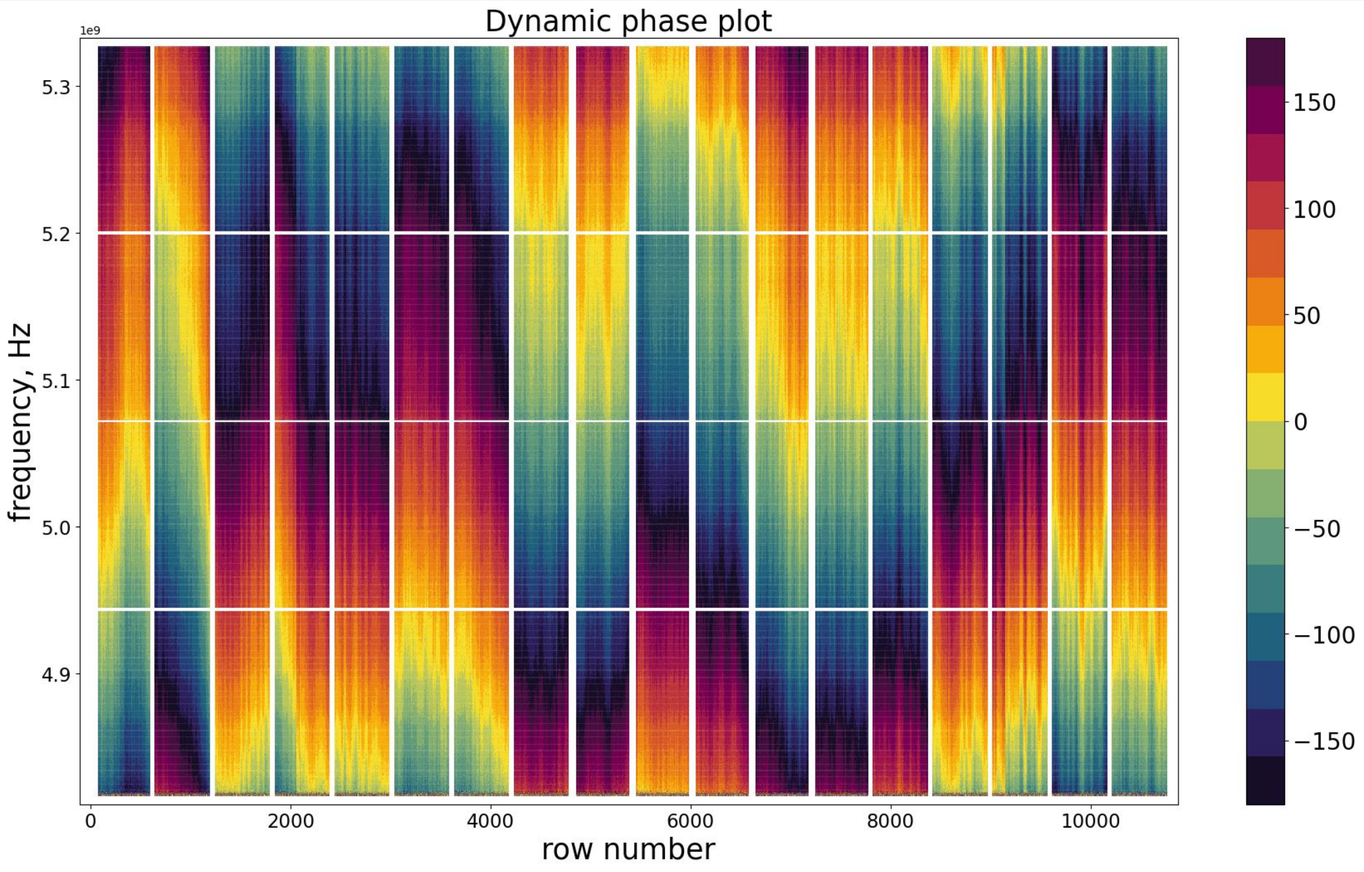

Dynamic spectrum
1e9
5.3
5.2
5.1
5.0
4.9
frequency, Hz
0
2000
4000
6000
8000
10000
row number

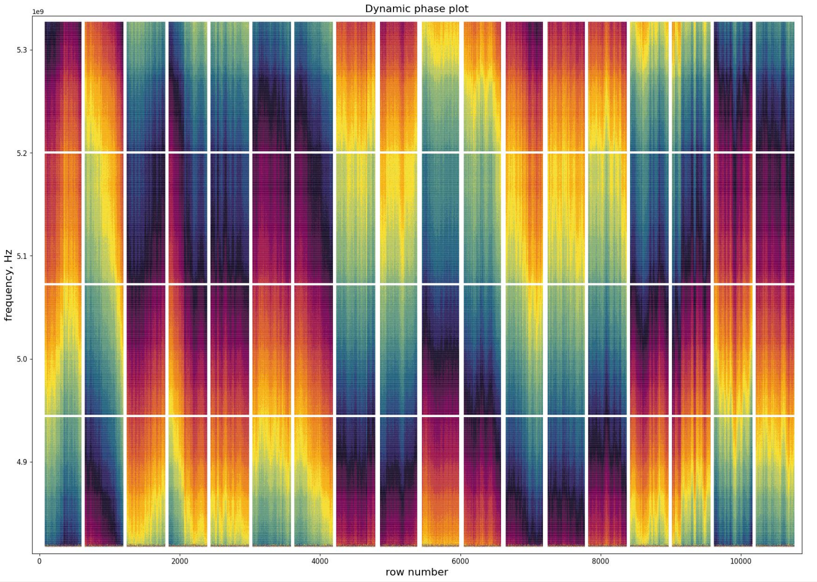

Amplitude vs Freq
frequency, Hz
amplitude

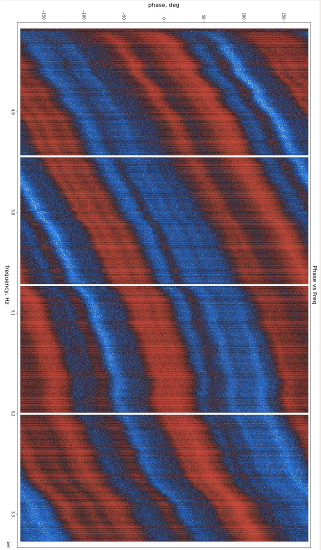

Amplitude vs Time
0.004
0.003
0.002
0.001
0.000
amplitude
45000
50000
55000
60000
65000
70000
75000
80000
+4.9318e9
time, s

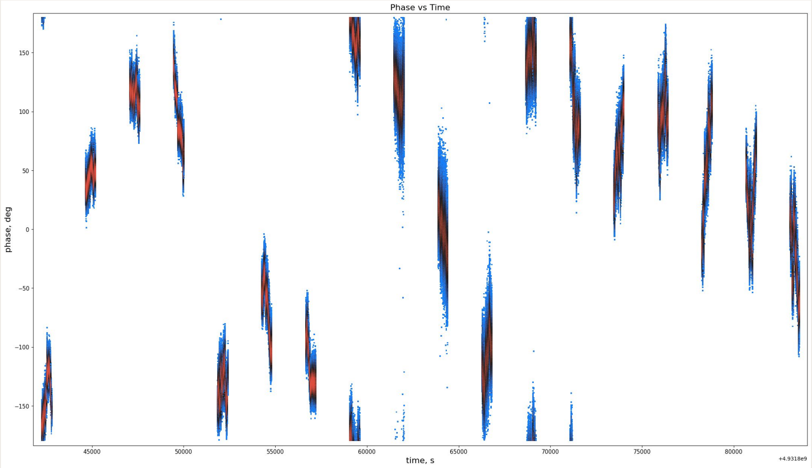
Dynamic spectrum (phase)

Vertical slices follow the delay slope




Dynamic phase plot
1e9
5.3
5.2
5.1
5.0
4.9
frequency, Hz
0
2000
4000
6000
8000
10000
row number


phase, deg
frequency, Hz
Phase vs Freq


Phase vs Time
150
100
50
0
−50
−100
−150
phase, deg
45000
50000
55000
60000
65000
70000
75000
80000
+4.9318e9
time, s

Visibility structure in CASA

From CASA:

`plotms`

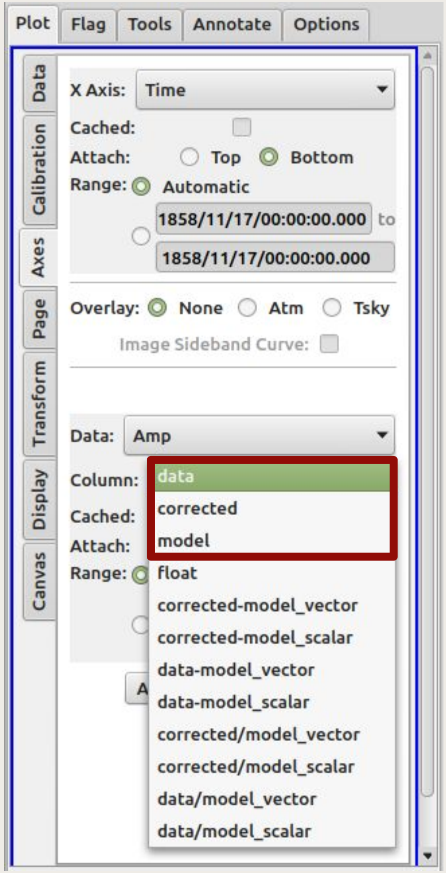
`browsetable`

Terminal standalone

`casabrowser`

`casaplotms`

casaplotms



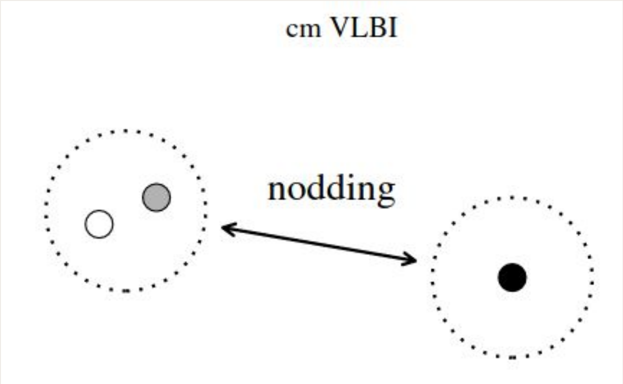
casabrowser

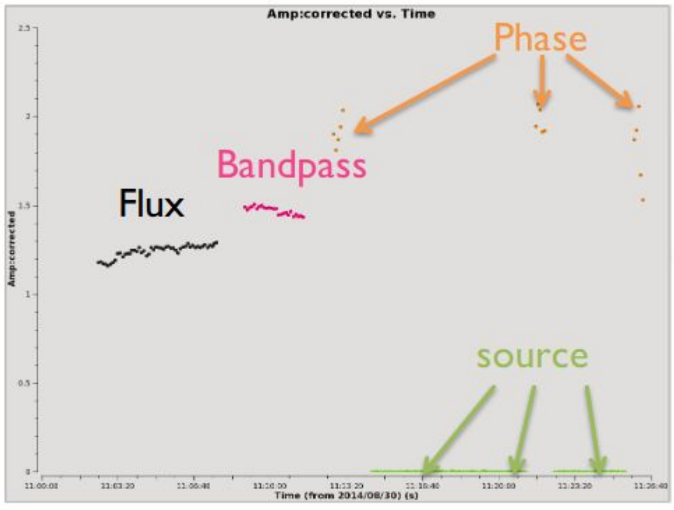
DATA	CORRECTED_DATA	MODEL_DATA
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex

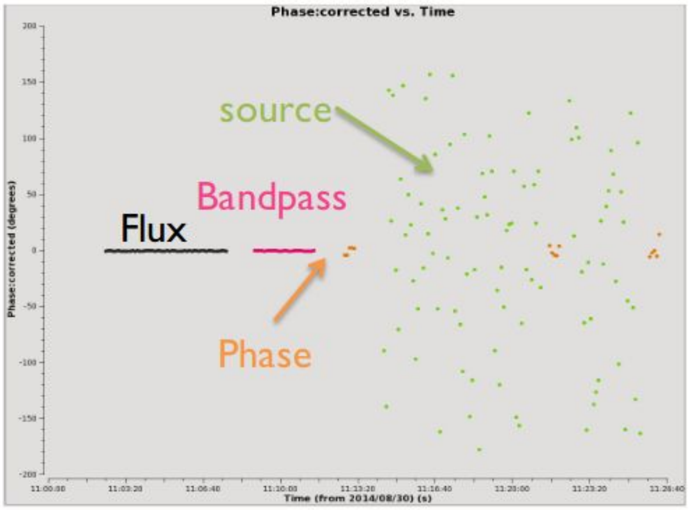
Why phase referencing is not enough?

Calibration

Phase-referencing to transfer solutions from nearby calibrator





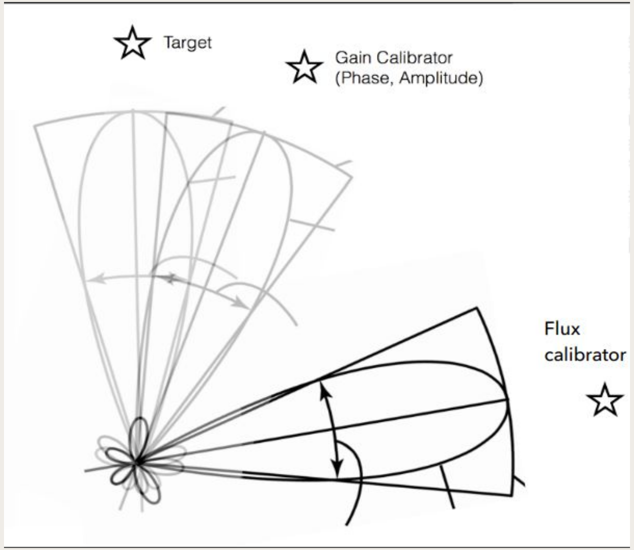


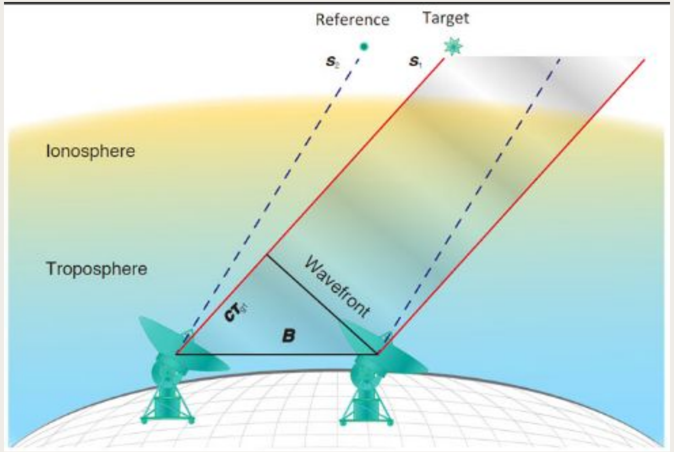
Residual errors

After phase-referencing there will be residual phase and amplitude errors

Atmospheric differences

The atmosphere is similar, not identical, above the target and above the phase-reference





Fast variability

Although the phase-ref cycle time should be longer than the coherence time, in VLBI phases still change very quickly

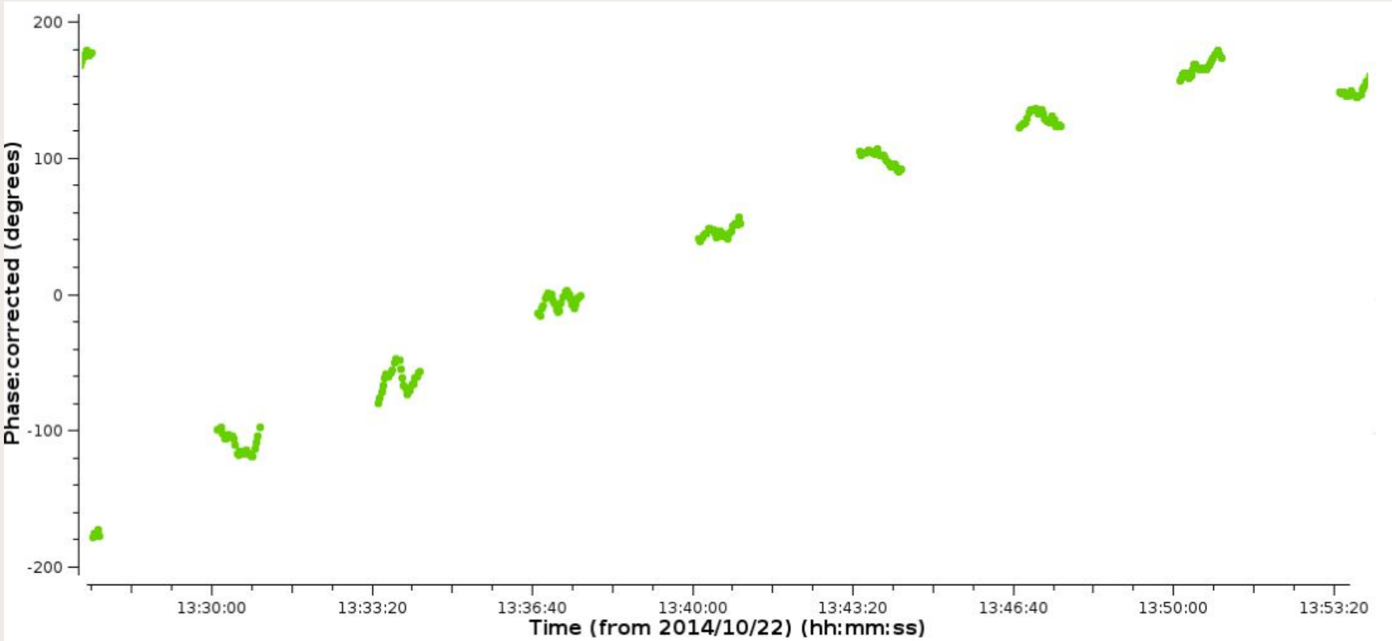
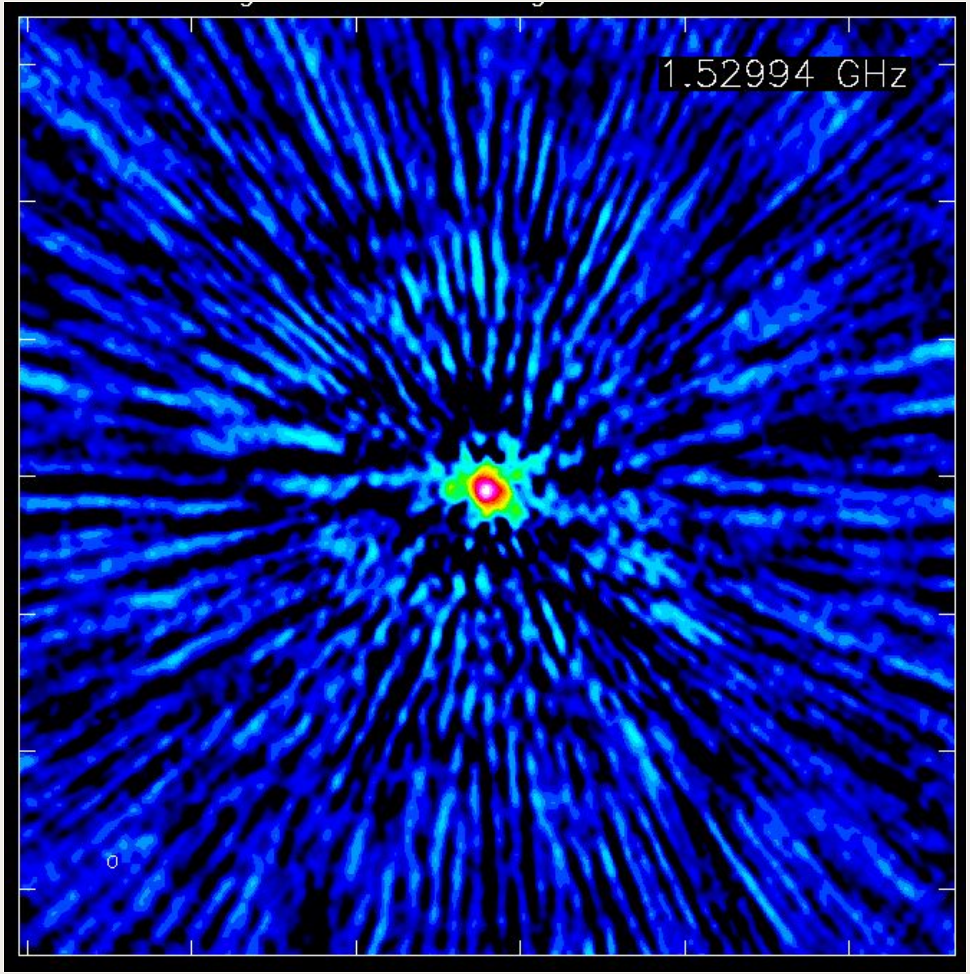


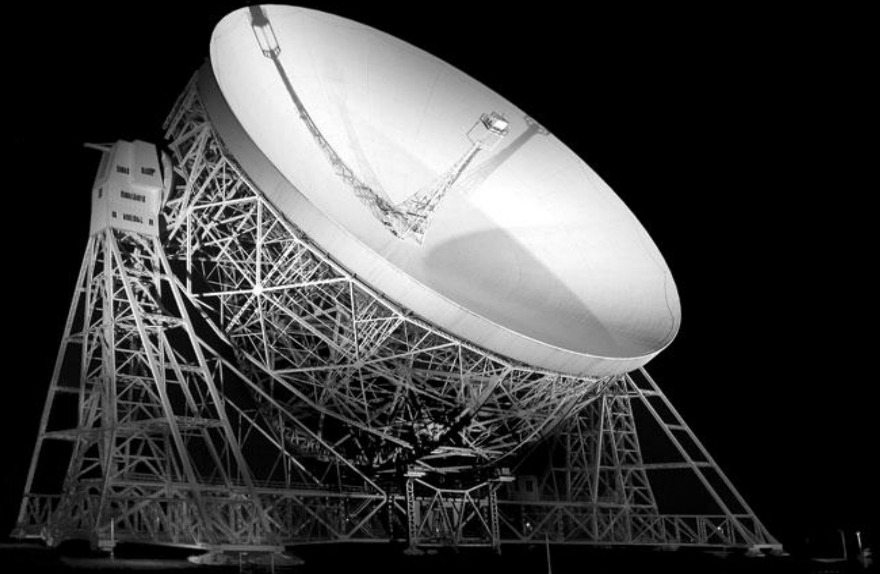
Image with residual errors

Severe phase residuals after phase-referenced calibration



Self-calibration is just calibration

MODEL!

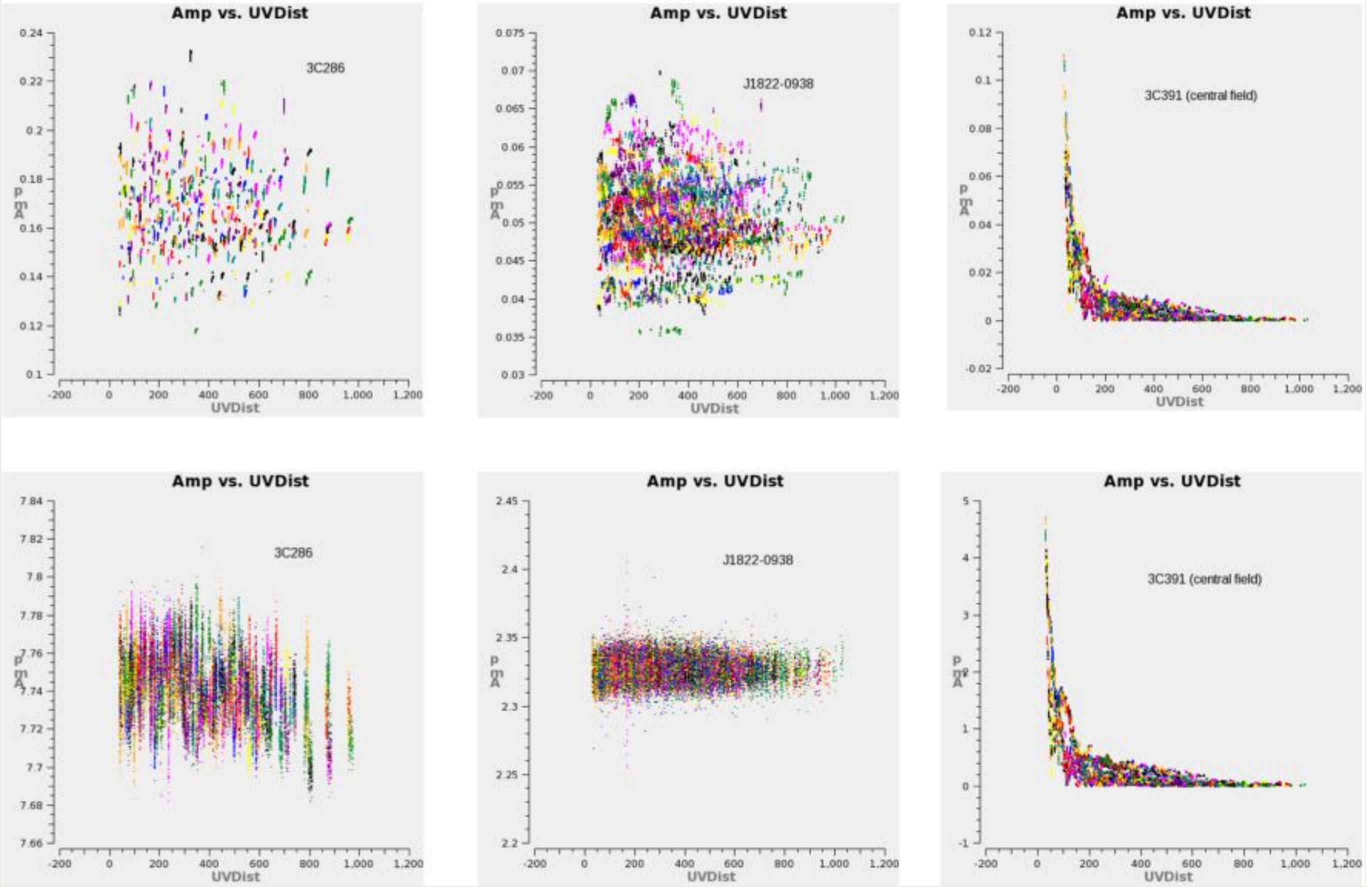


Amplitude vs uvdist

Top: before correction
Bottom: after

Target may have very complex structure

Flux scale calibrator	Phase calibrator	Target

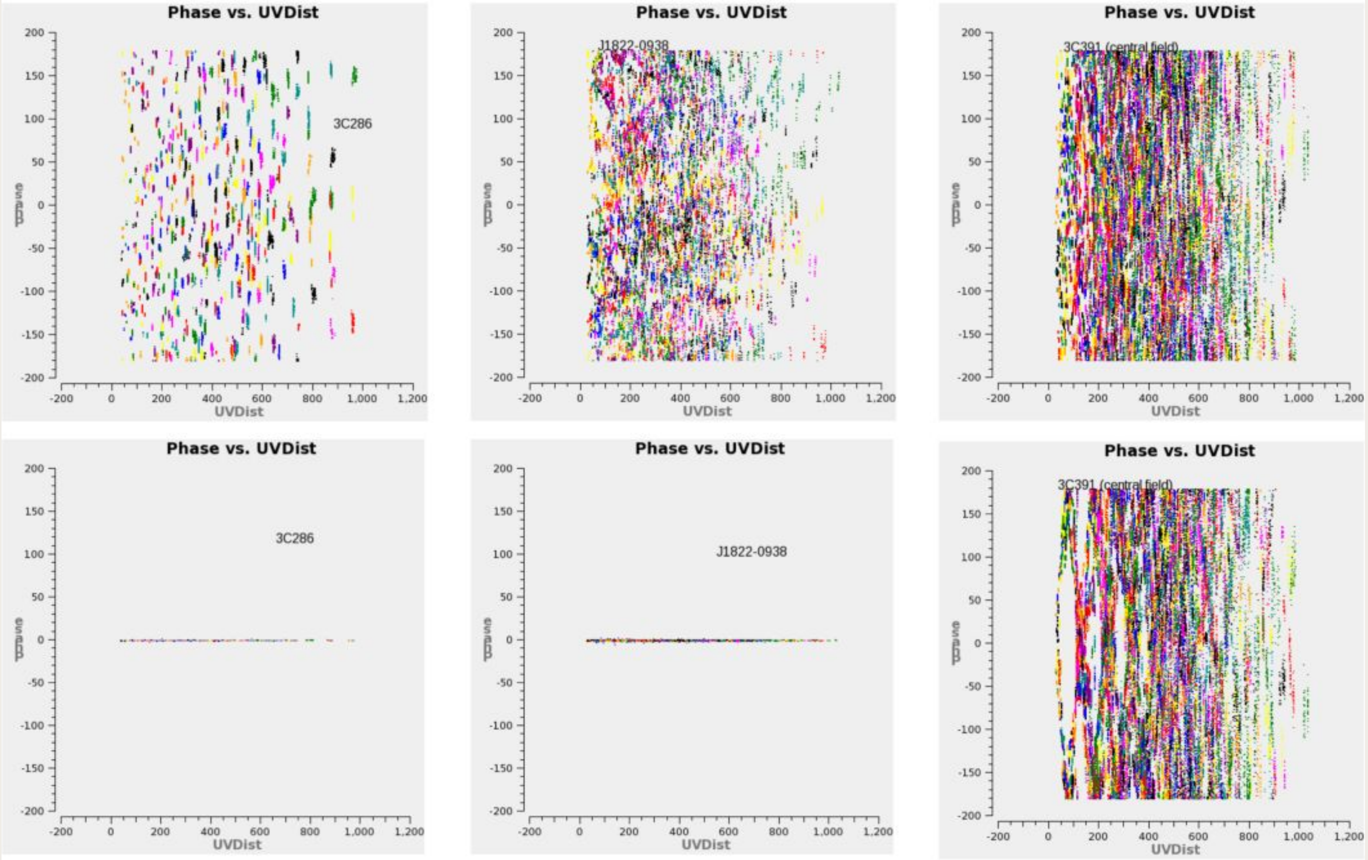


Phase vs uvdist

Top: before correction
Bottom: after

Target phases are not wrong. They follow complex structure

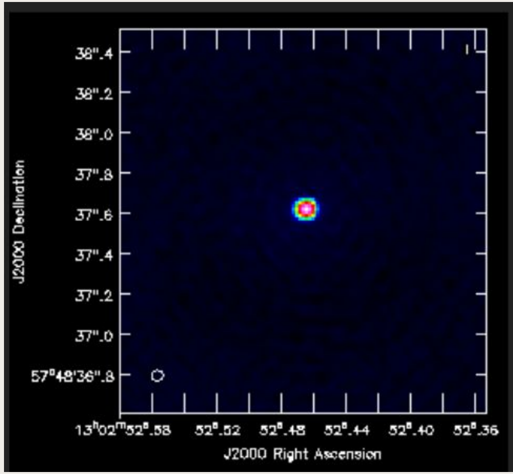
Flux scale calibrator	Phase calibrator	Target

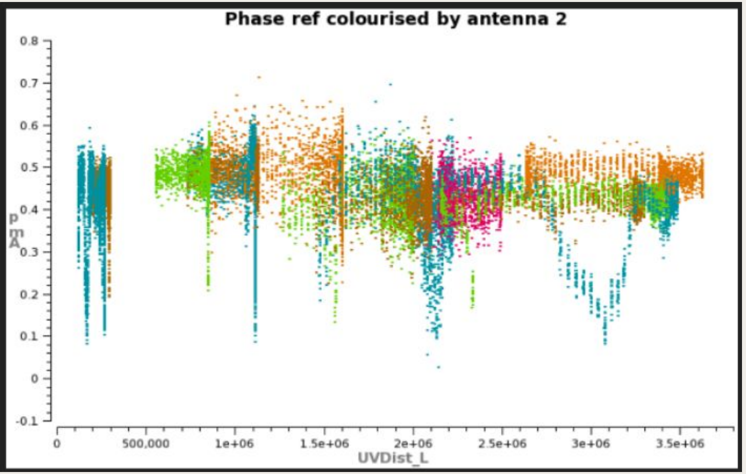


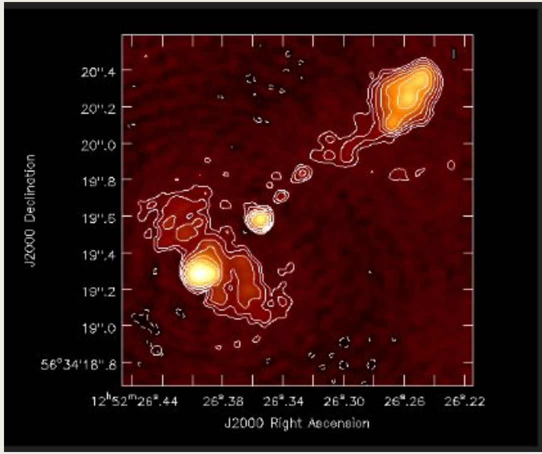
Impact of the model

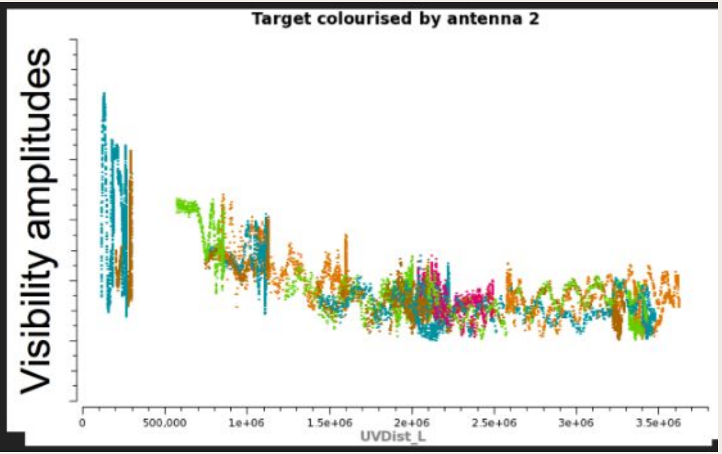
Point-like source model:

```
flux
spectral index
X
Y
```



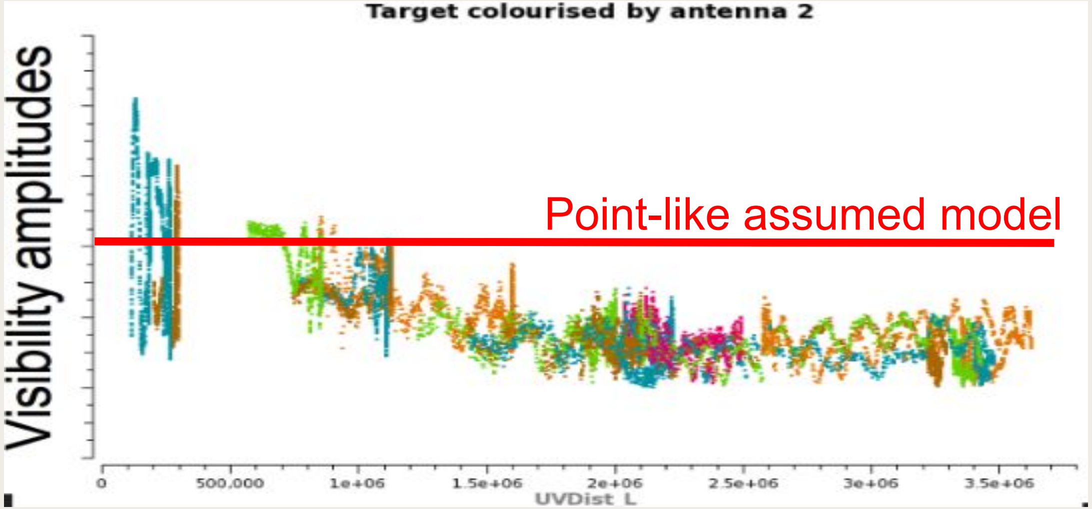


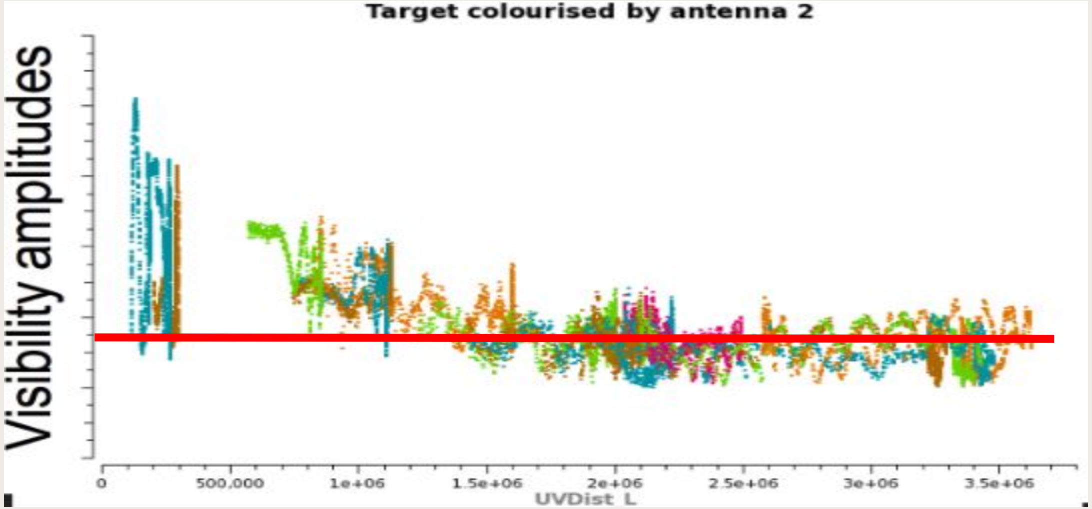




If we don't use a model

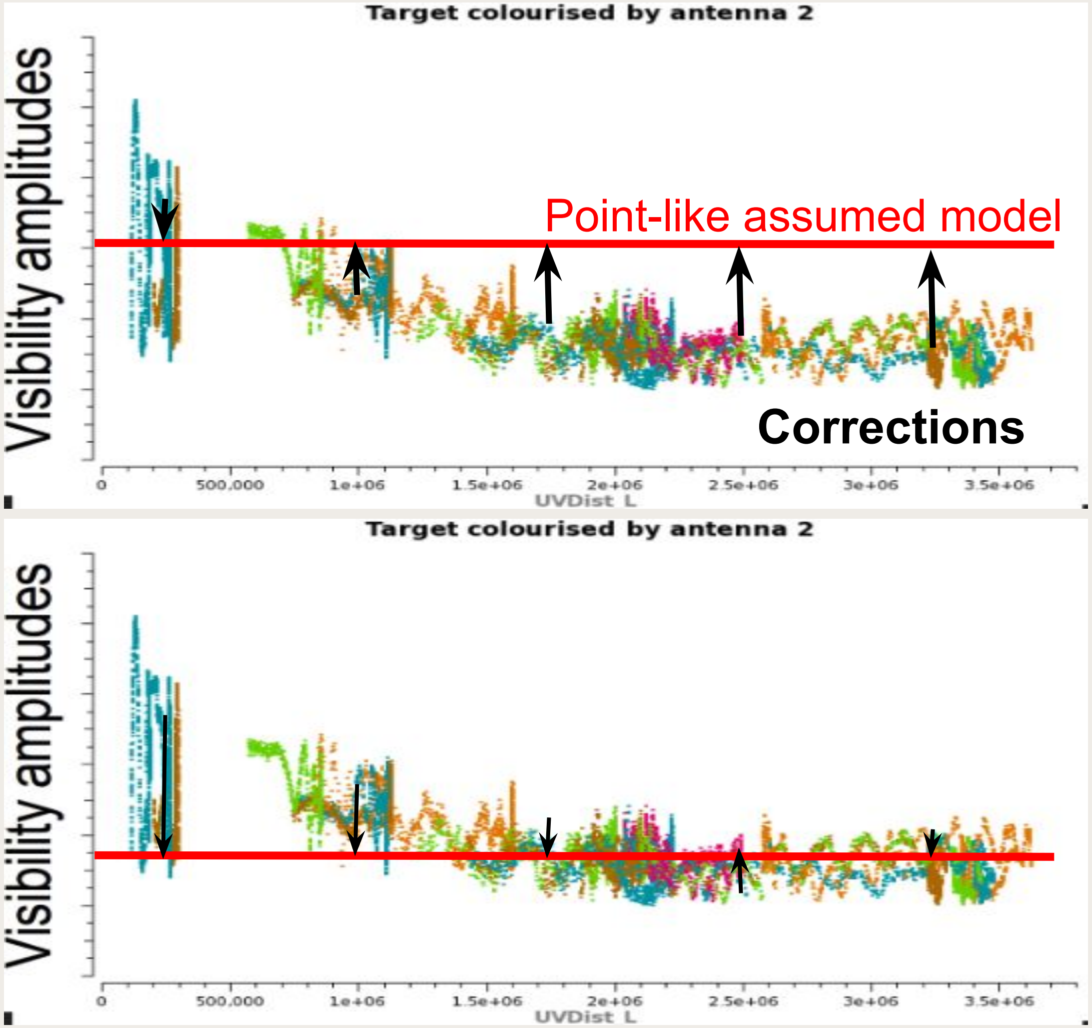
Assuming a point-like source will corrupt the antenna gains because there will always be baselines with the wrong flux density

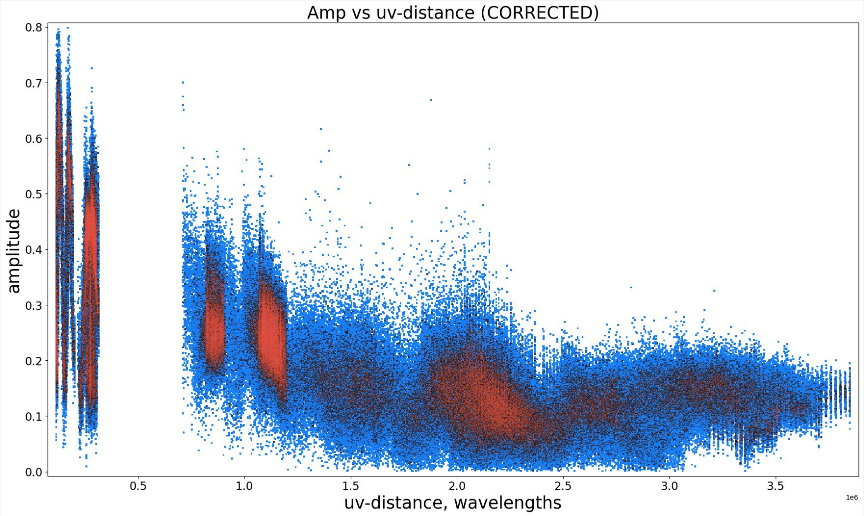


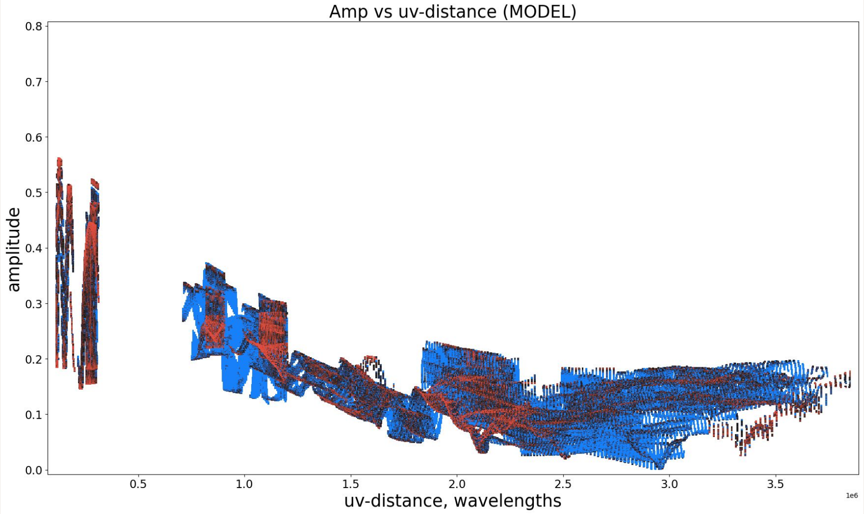


If we don’t use a model

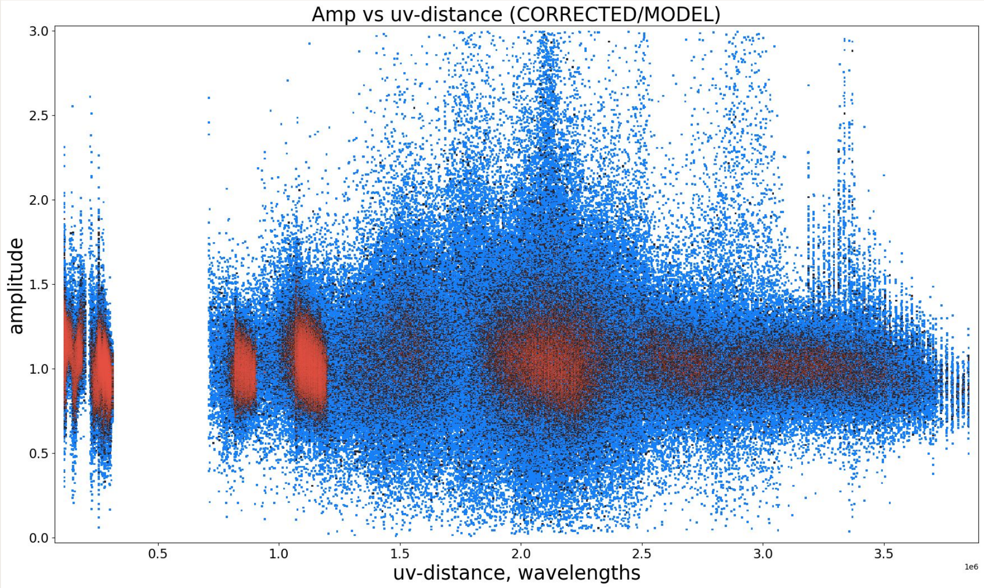
Assuming a point-like source will corrupt the antenna gains because there will always be baselines with the wrong flux density







By dividing the data by the model we get “point-like” visibilities

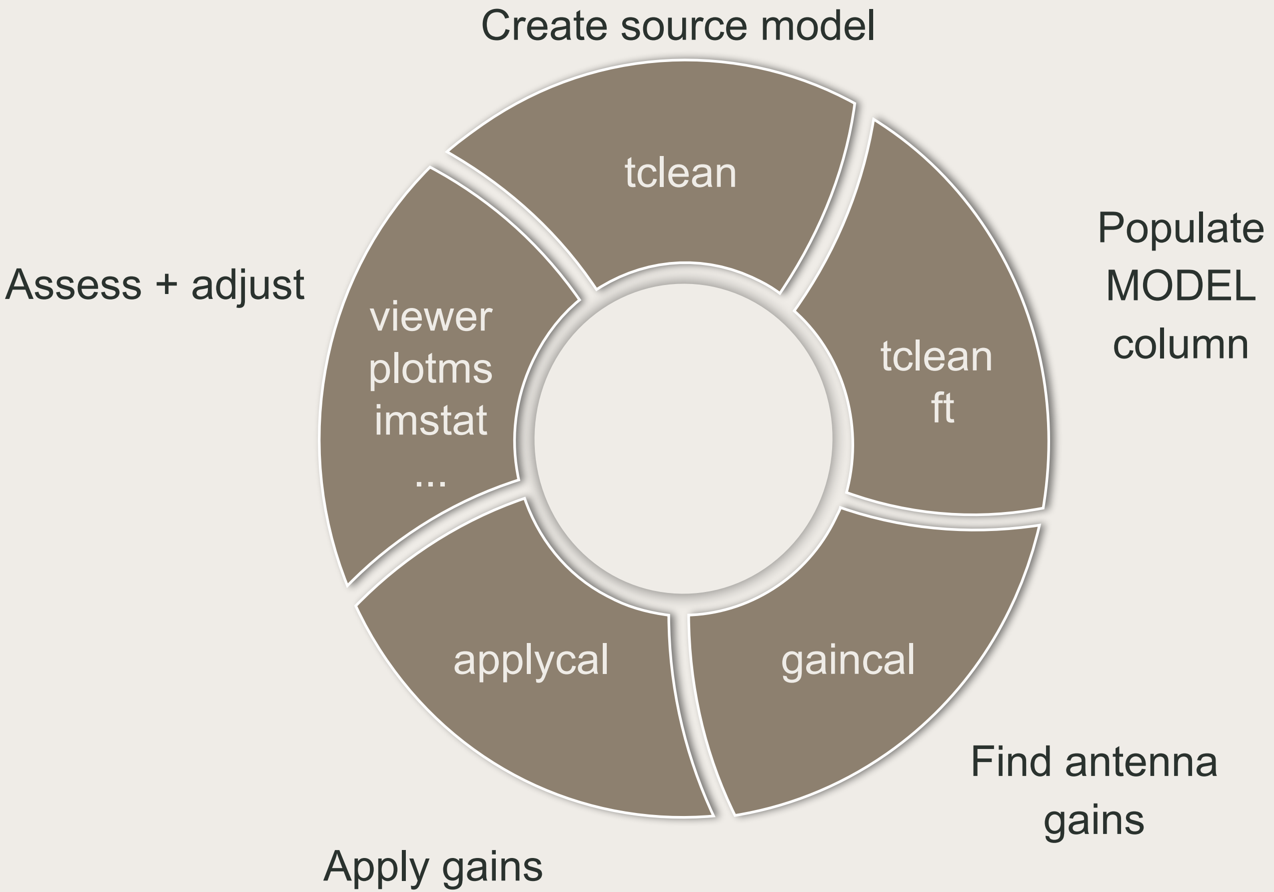


Model will not be perfect at first. So we need to iterate.

Self-calibration procedure

Self-cal loop

Self-calibration is an iterative process where we find antenna corrections, produce improved model and refine the cycle



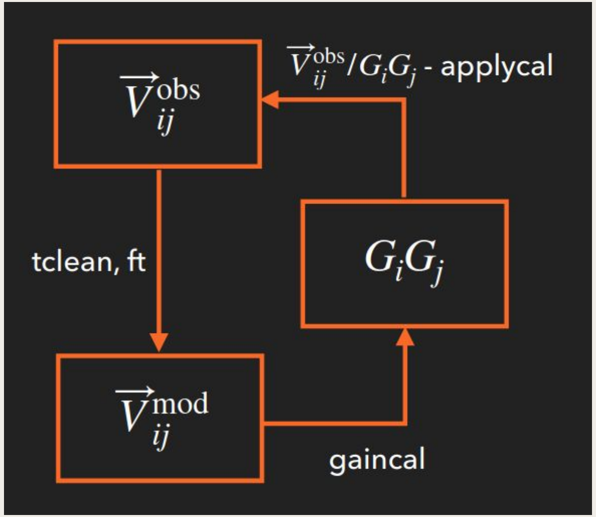
What do we self-calibrate?

Our aim is to calibrate the residual complex gains per antenna

We assume

- effects other than residual complex gain errors are already calibrated
- complex gains are antenna-based

$$\vec{V}_{ij}^{\mathrm{obs}} = G_i G_j \vec{V}_{ij}^{\mathrm{true}}$$



CASA pieces

Measurement Set. Columns

DATA | MODEL | CORRECTED

CASA tasks

tclean | gaincal | applycal

Calibration table Gain solutions

table1 | table2

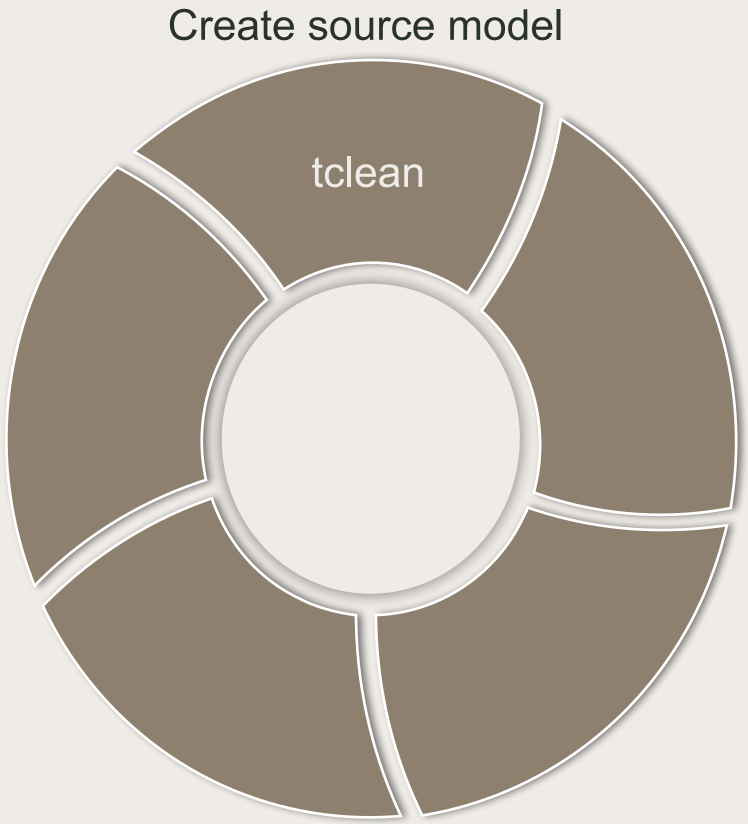
Images

img1

(image, residual, model, …)

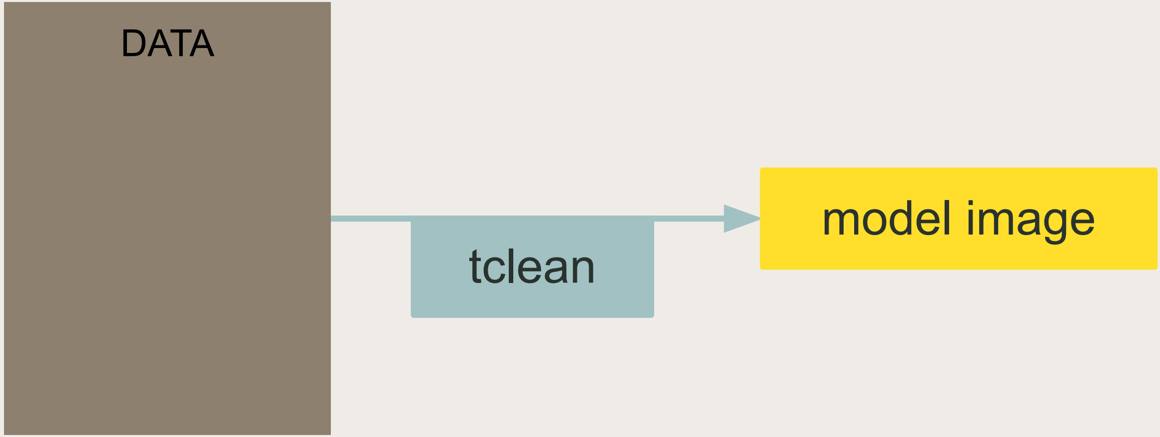
Self-cal loop

Start with an image as a starting model. If not possible, you may try a point-like source



Create source model

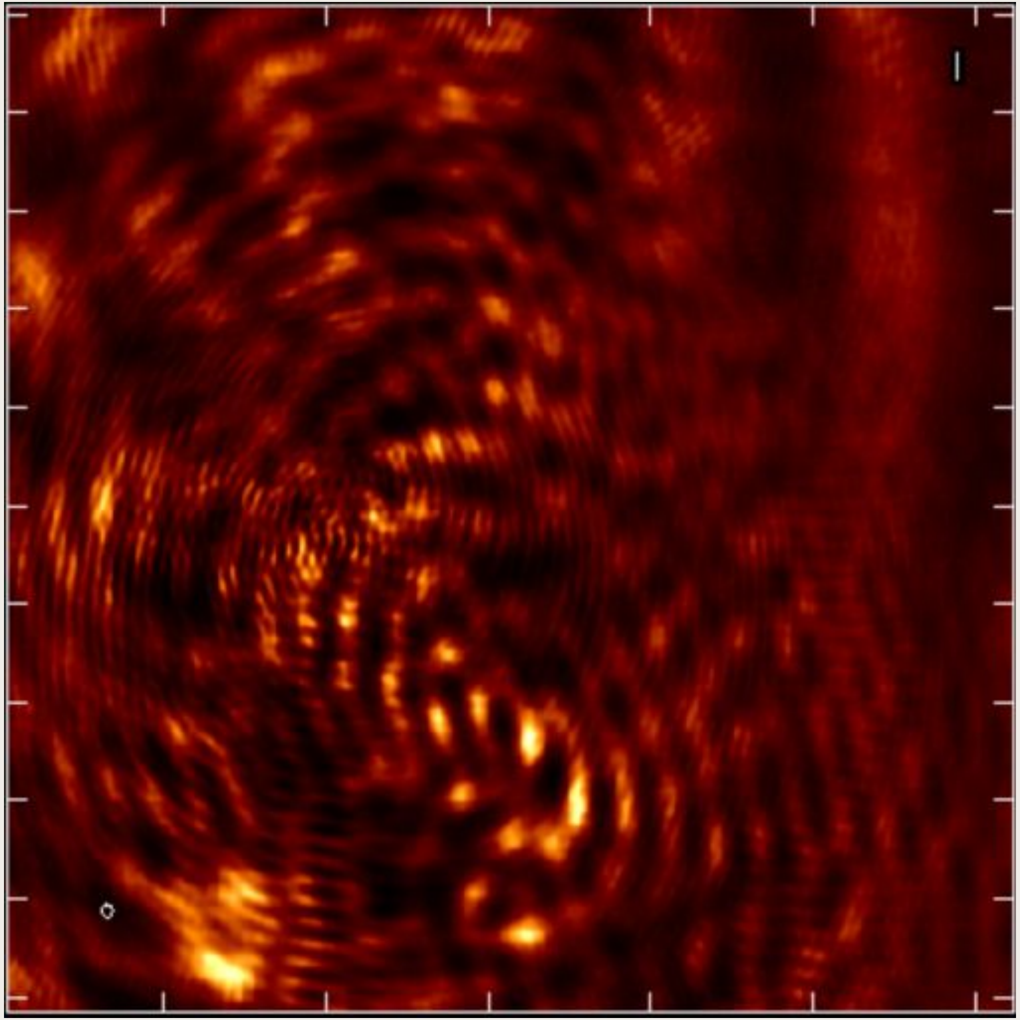
`tclean` produces a clean image and writes the Fourier transformed of the model into the visibilities



Dirty map (raw data)

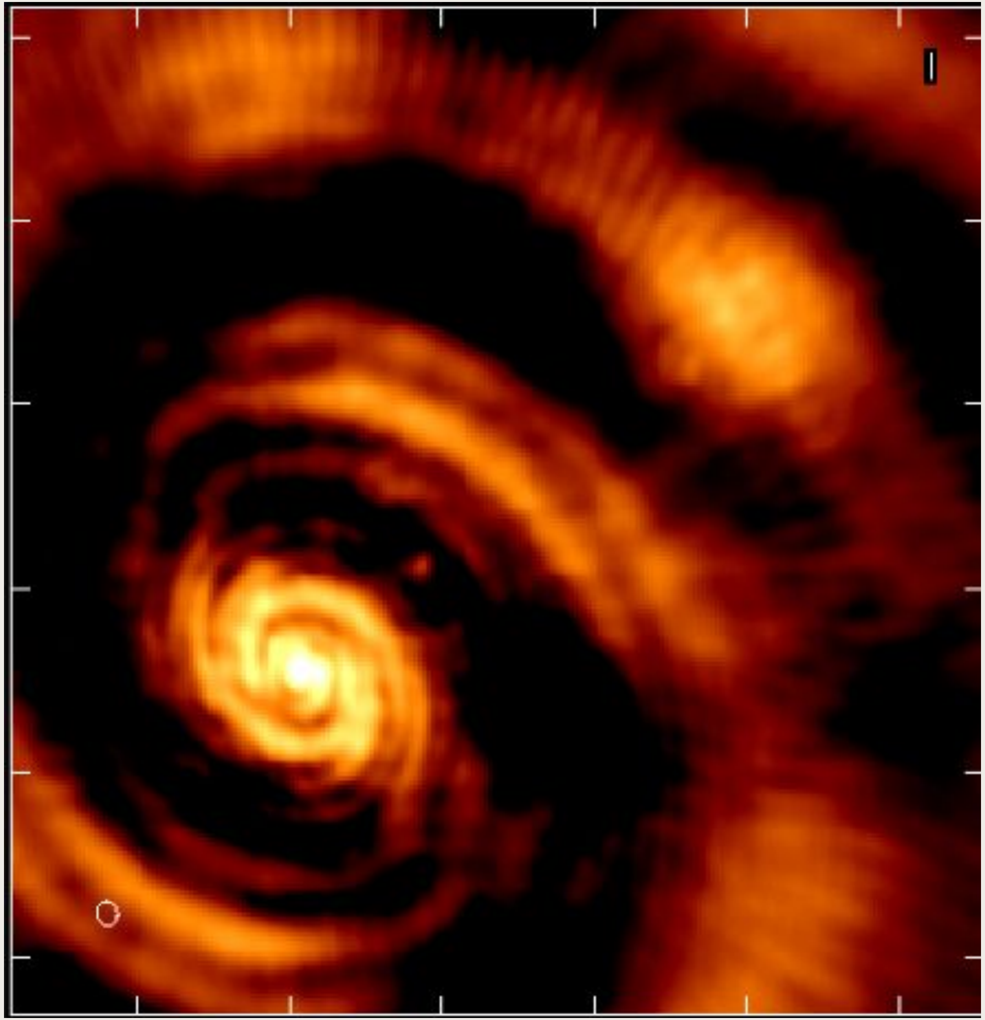
Target without phase referencing.

No phase calibration



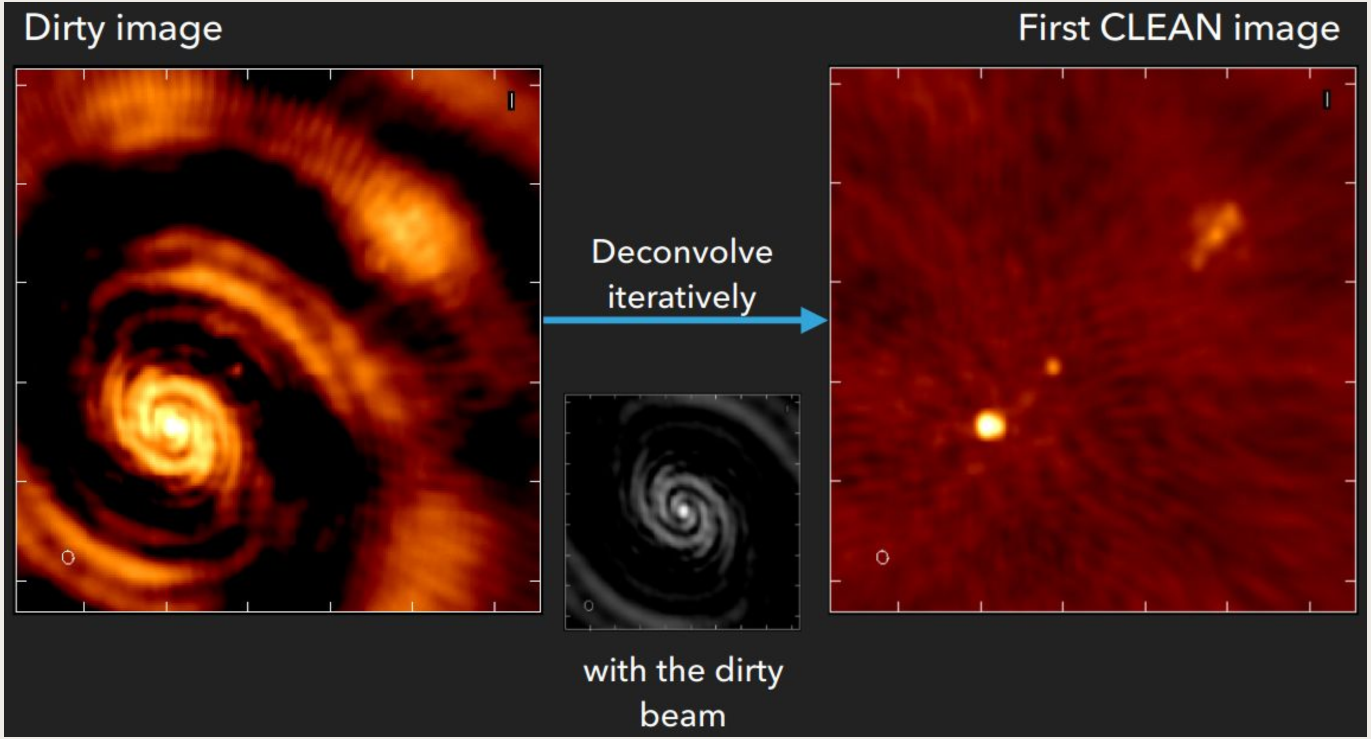
Dirty map (calibrated data)

Target with phase referencing



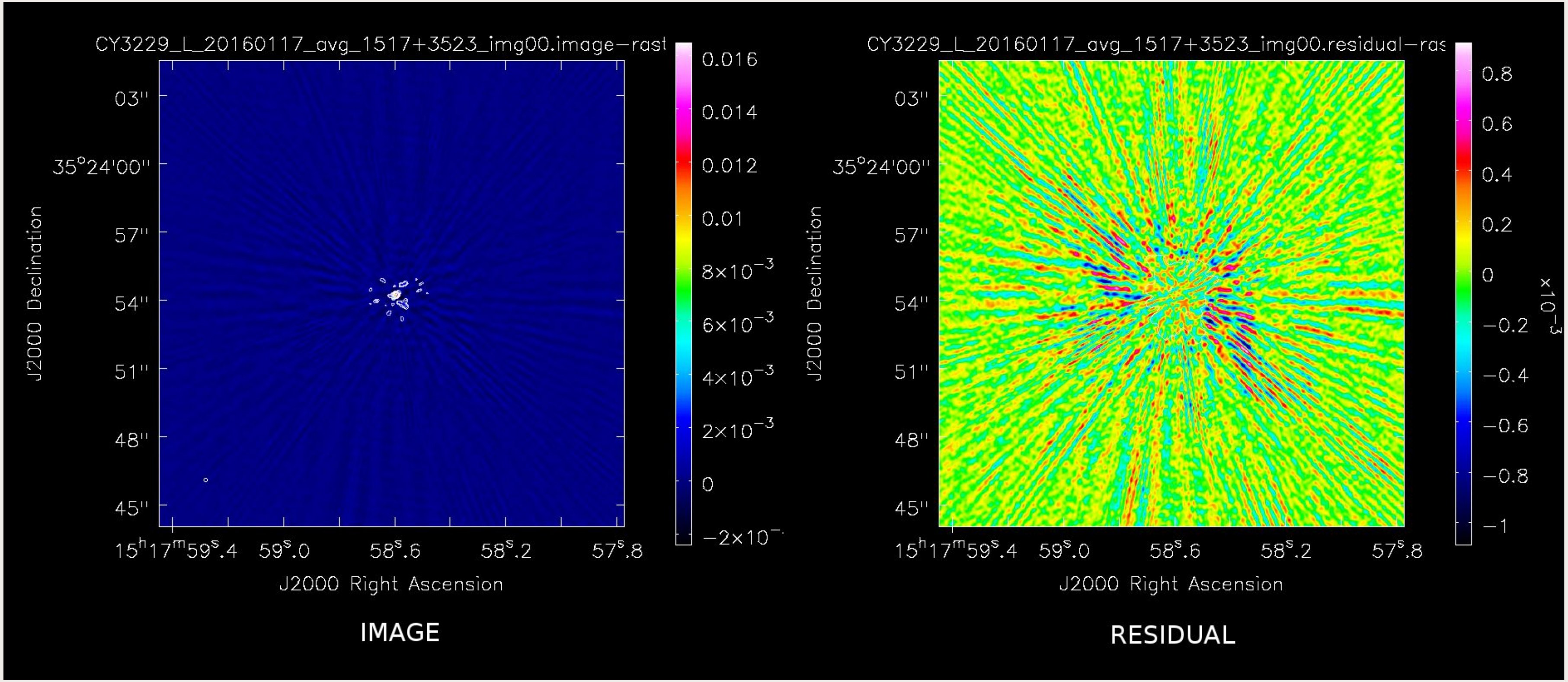
Phase-referencing CLEAN image

Non-Gaussian noise. Dynamic range is limited by residual errors



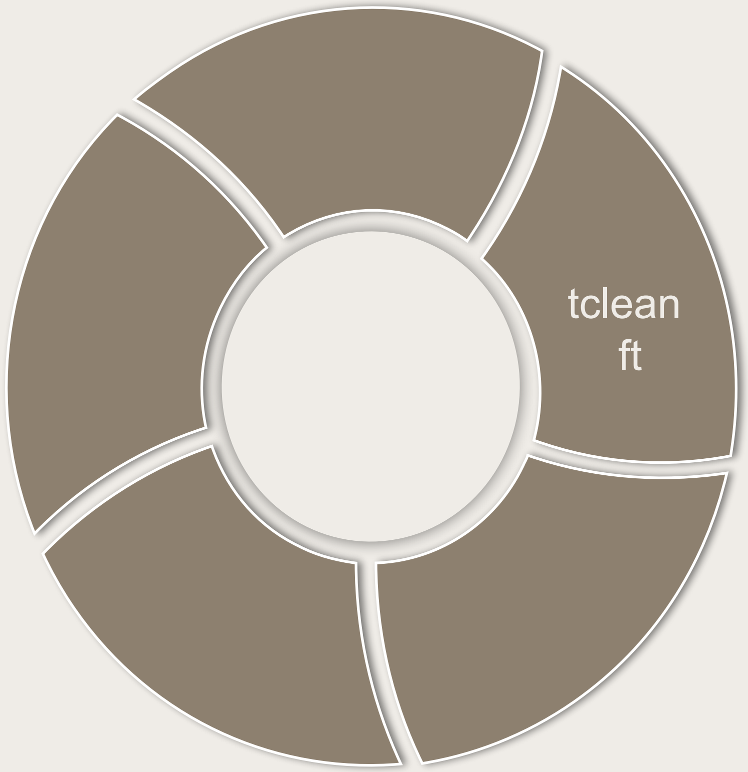
First clean image and residual

- Background contains random noise + residual calibration errors



MODEL

The model image needs to be included in the MS **per visibility**



Populate MODEL column

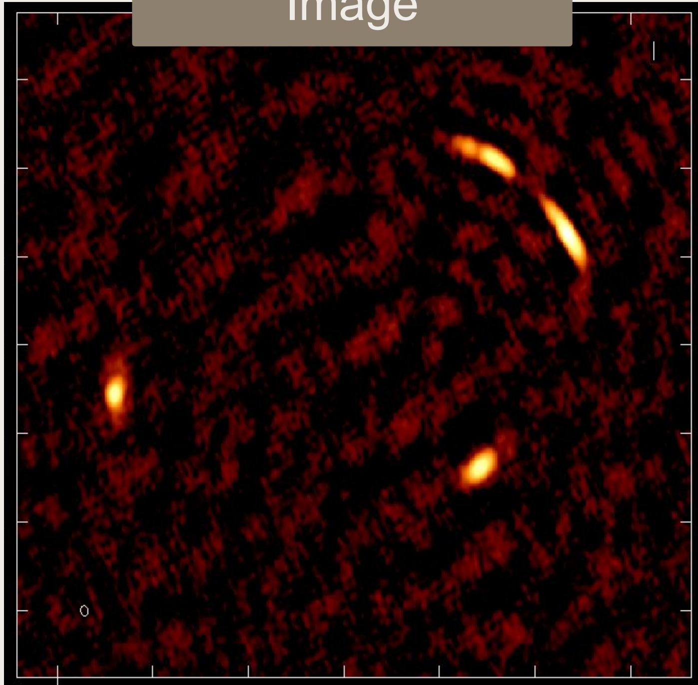
What are models in CASA?

CASA does not use Clean Components, but full 2D images and can include spectral component

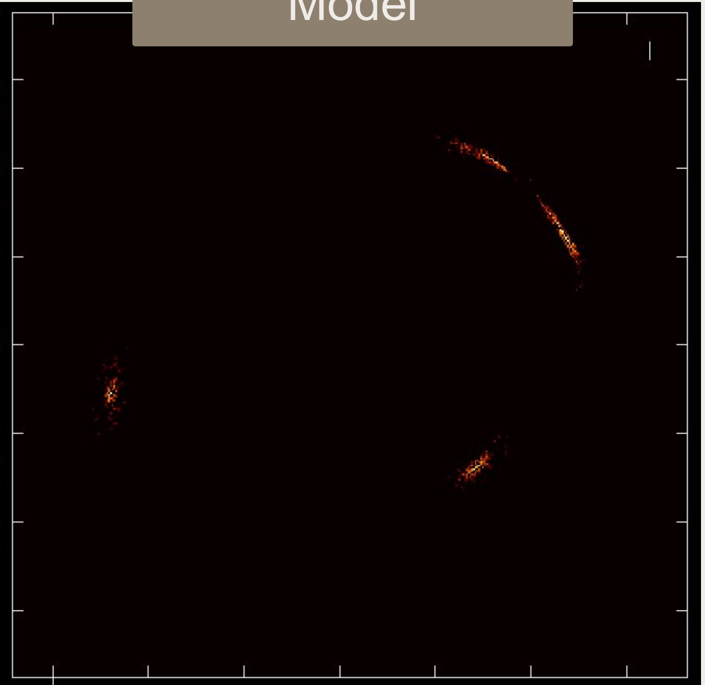
Model in CASA is an image

```
CY10217_L_005_20201012_avg_1828-0912_img00.psf
CY10217_L_005_20201012_avg_1828-0912_img00.sumwt
CY10217_L_005_20201012_avg_1828-0912_img00.pb
CY10217_L_005_20201012_avg_1828-0912_img00.residual
CY10217_L_005_20201012_avg_1828-0912_img00.mask
CY10217_L_005_20201012_avg_1828-0912_img00.model
CY10217_L_005_20201012_avg_1828-0912_img00.image
```

Image



Model



MODEL into the MS

`tclean` will automatically populate the MODEL column in the MS

If not, use `ft`

`tclean` will automatically fill the MODEL column

```
savemodel            = 'modelcolumn'      #  Options to save model visibilities
                                          #   (none, virtual, modelcolumn)
```

Make sure your log shows:

```
2020-11-02 16:41:05 INFO  …i2::runMajorCycle  Saving model column
```

Backup plan

If the model was not written in the MODEL column, do it manually with ft (not needed in most recent CASA versions)

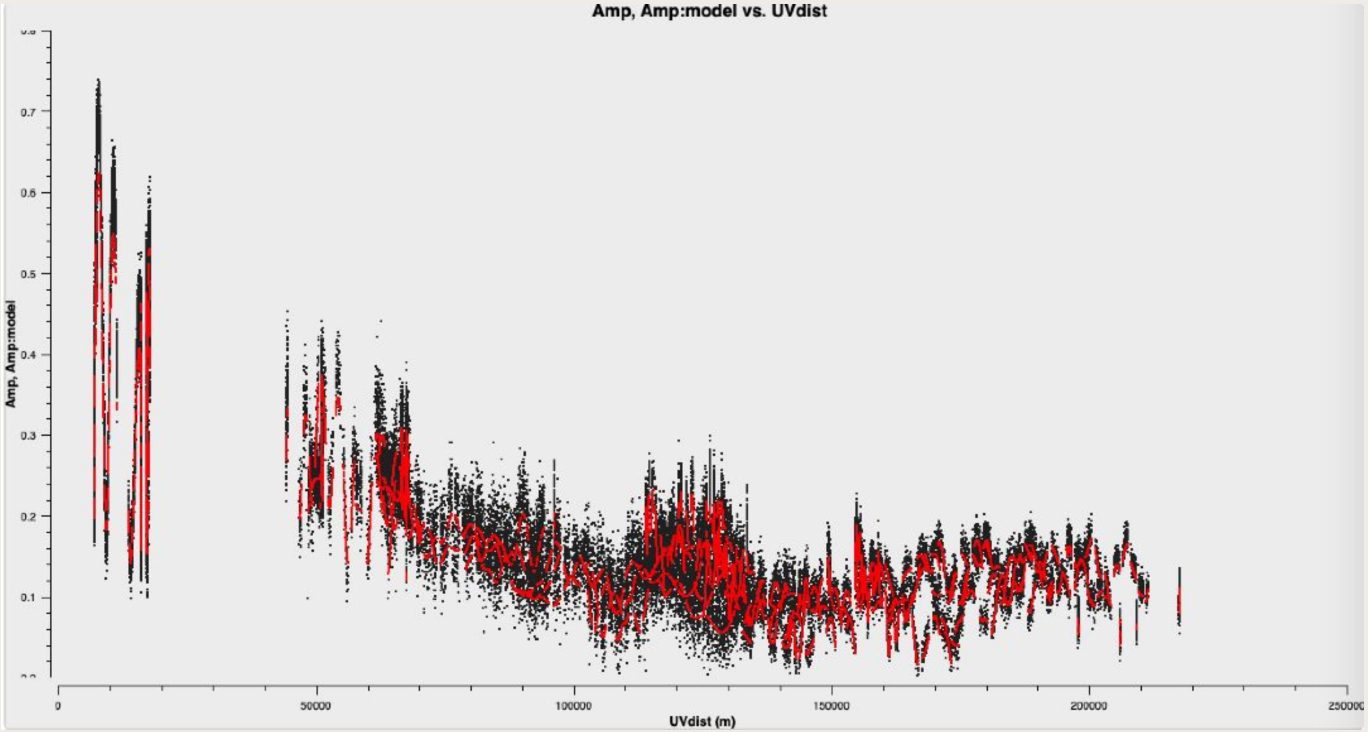
```
#  ft :: Insert a source model as a visibility set
vis                 =    'my.ms'        #  Name of input visibility file
field               = 'sourcename'      #  Select field using field id(s) or
                                        #   field name(s)
spw                 =         ''        #  Select spectral window/channels
model               = 'img1.model'      #  Name of input model image(s)
nterms              =          1        #  Number of terms used to model the sky
                                        #   frequency dependence
complist            =         ''        #  Name of component list
incremental         =      False        #  Add to the existing model visibility?
usescratch          =       True        #  If True, predicted  visibility  is
                                        #   stored in MODEL_DATA column
```

MODEL into the MS

Always make sure that your MODEL column is right:

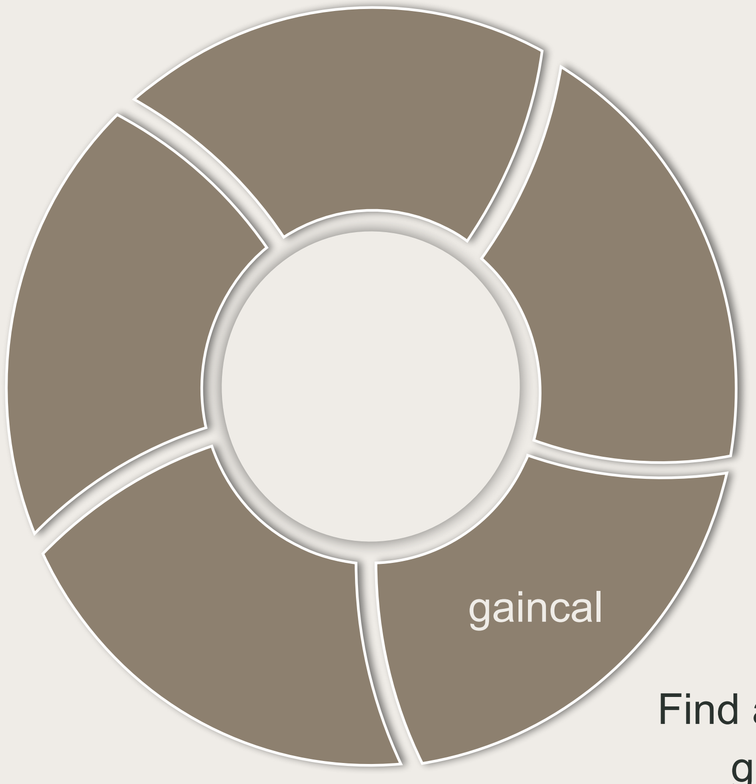
not empty, and not all 0 or 1

DATA	CORRECTED_DATA	MODEL_DATA
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex
[4, 128] Complex	[4, 128] Complex	[4, 128] Complex



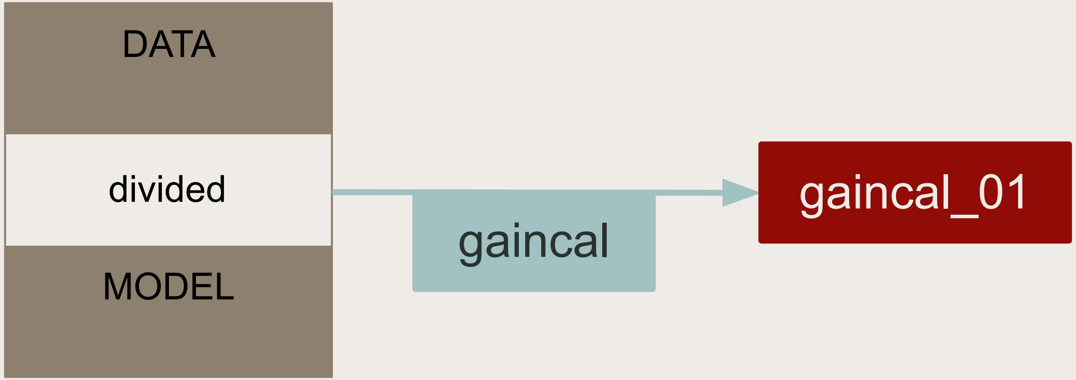
Self-cal loop

Calibration step, where gains are actually calculated



Find antenna gains

gaincal

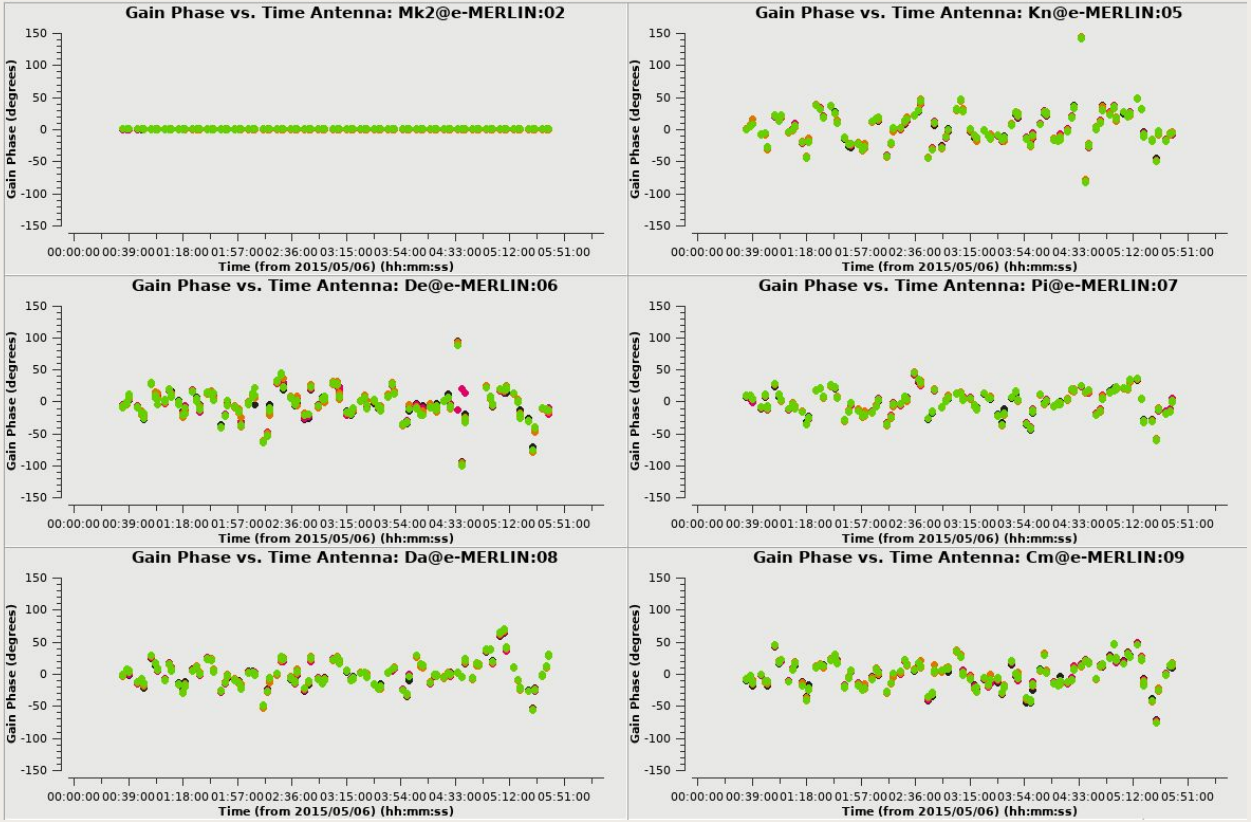



```
#  gaincal :: Determine temporal gains from calibrator observations
vis                 =        'my.ms'       #  Name of input visibility file
caltable            = 'gaincal_01'         #  Name of output gain calibration table
field               =             ''        #  Select field using field id(s) or field name(s)
solint              =          '32s'        #  Solution interval
combine             =             ''        #  Data axes which to combine for solve (obs, scan, spw, and/or field)
refant              =          'Mk2'        #  Reference antenna name(s)
refantmode          =         'flex'        #  Reference antenna mode
minblperant         =              3        #  Minimum baselines _per antenna_ required for solve
minsnr              =              3        #  Reject solutions below this SNR
solnorm             =          False        #  Normalize (squared) solution amplitudes (G, T only)
gaintype            =            'G'        #  Type of gain solution (G,T,GSPLINE,K,KCROSS)
calmode             =           'ap'        #  Type of solution" ('ap', 'p', 'a')
append              =          False        #  Append solutions to the (existing) table
```

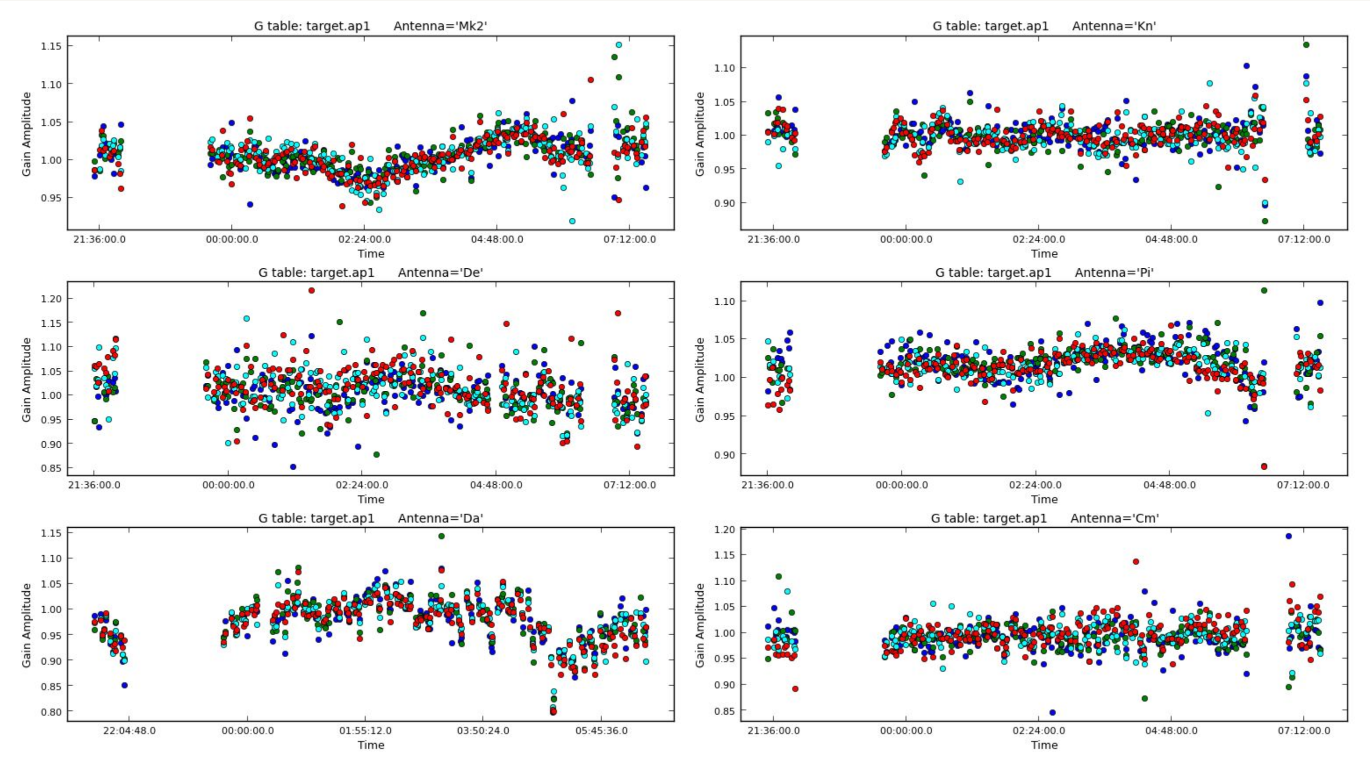
Gaincal: Phase solutions

We are correcting residual phases, so corrections should not be large, although 180deg wraps could happen

```
INFO    calibrater::solve       Finished solving.
INFO    gaincal::::     Calibration solve statistics per spw:  (expected/attempted/succeeded):
INFO    gaincal::::       Spw 0: 632/629/629
INFO    gaincal::::       Spw 1: 632/630/630
INFO    gaincal::::       Spw 2: 632/630/630
INFO    gaincal::::       Spw 3: 632/629/629
```



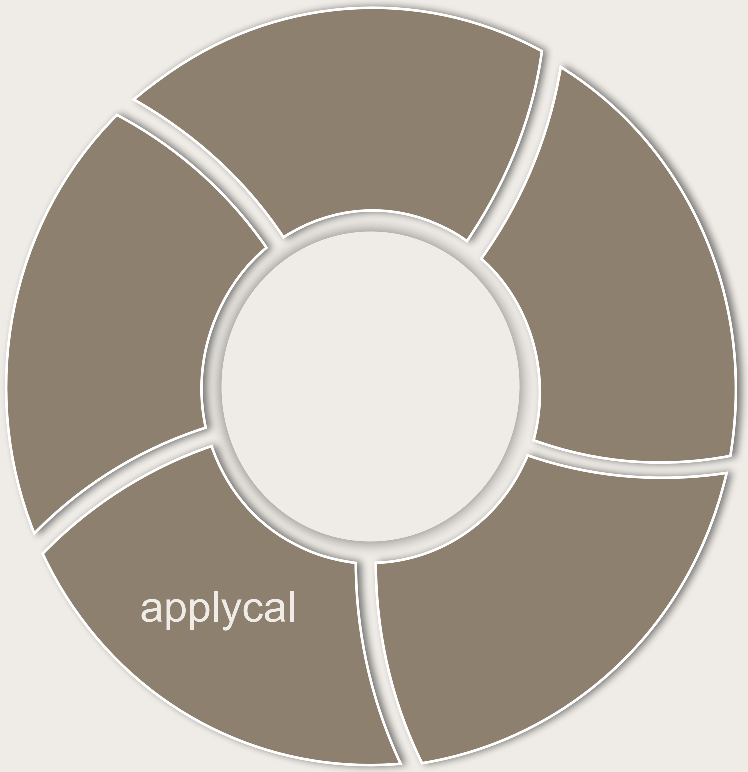
Example of amplitude solutions



No closure-phase restriction means amplitudes can be unbound. Look for large outliers

Self-cal loop

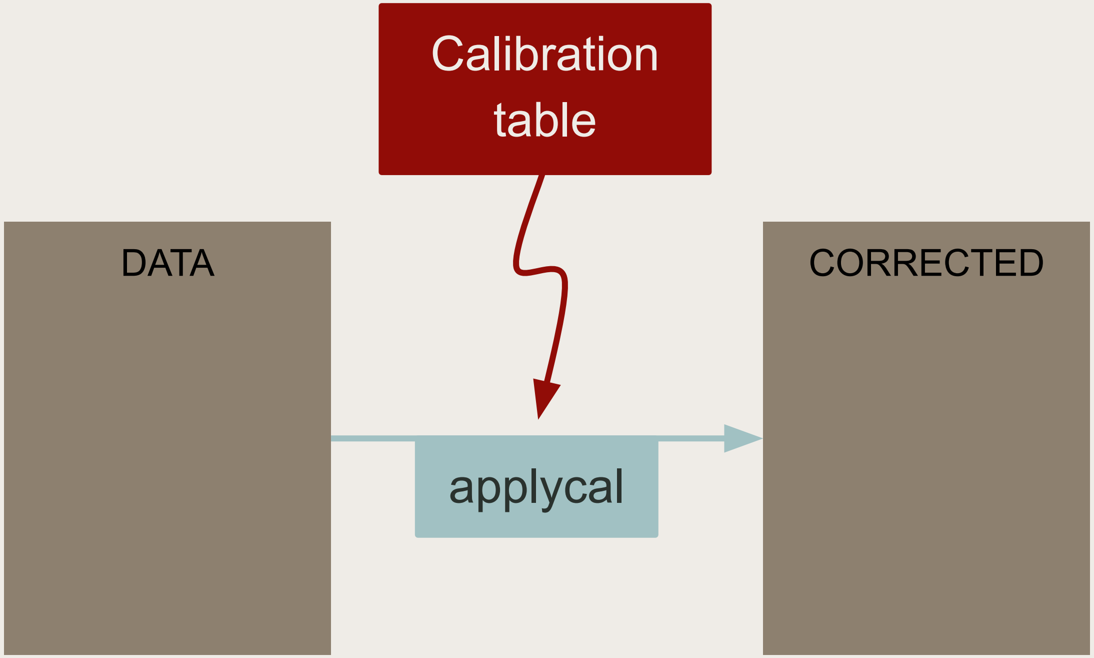
Apply the gain solutions to the data



Apply gains

`applycal` step

Here we are modifying the CORRECTED column from the MS



`applycal` step

You can apply one or multiple tables at the same time: `gaintable` as python list

Example applycal parameters for a single table

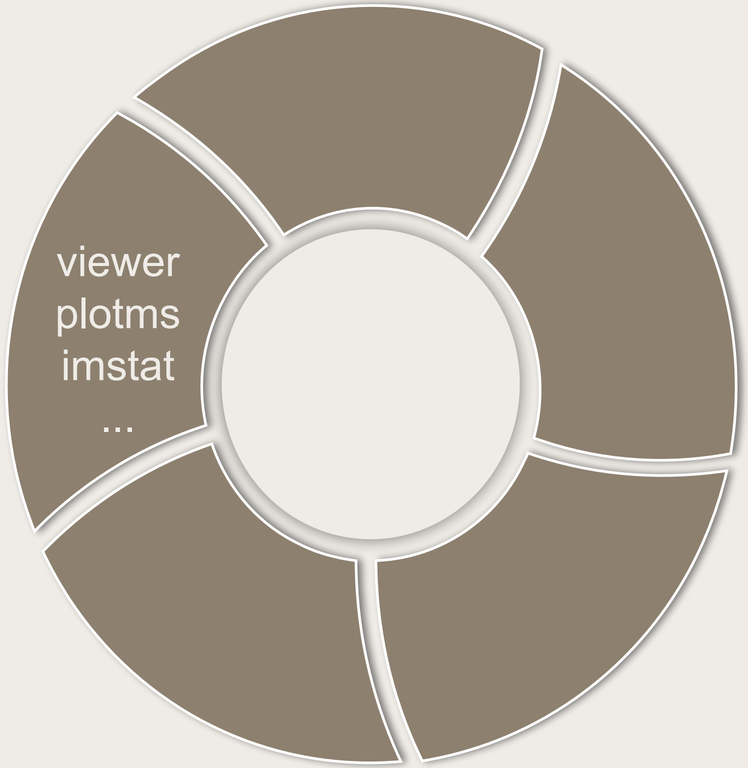
```
#  applycal :: Apply calibrations solutions(s) to data
vis                 =     'my.ms'     #  Name of input visibility file
field               =          ''     #  Select field using field id(s) or field name(s)
docallib            =       False     #  Use callib or traditional cal apply parameters
     gaintable      = 'gaincal_01'    #  Gain calibration table(s) to apply on the fly
     gainfield      =          []     #  Select a subset of calibrators from gaintable(s)
     interp         =          []     #  Interpolation parameters for each gaintable, as a list
     spwmap         =          []     #  Spectral windows combinations to form for gaintables(s)
     calwt          =      [True]     #  Calibrate data weights per gaintable.
applymode           =          ''     #  Calibration mode: ""="calflag","calflagstrict","trial",
                                      # "flagonly","flagonlystrict", or "calonly"
flagbackup          =        True     #  Automatically back up the state of flags before the run?
```

Applycal will flag data! check applymode

```
G Jones: In: 2056097 / 16147968   (12.7328528271%) --> Out: 2076913 / 16147968   (12.8617606872%)
```

Self-cal loop

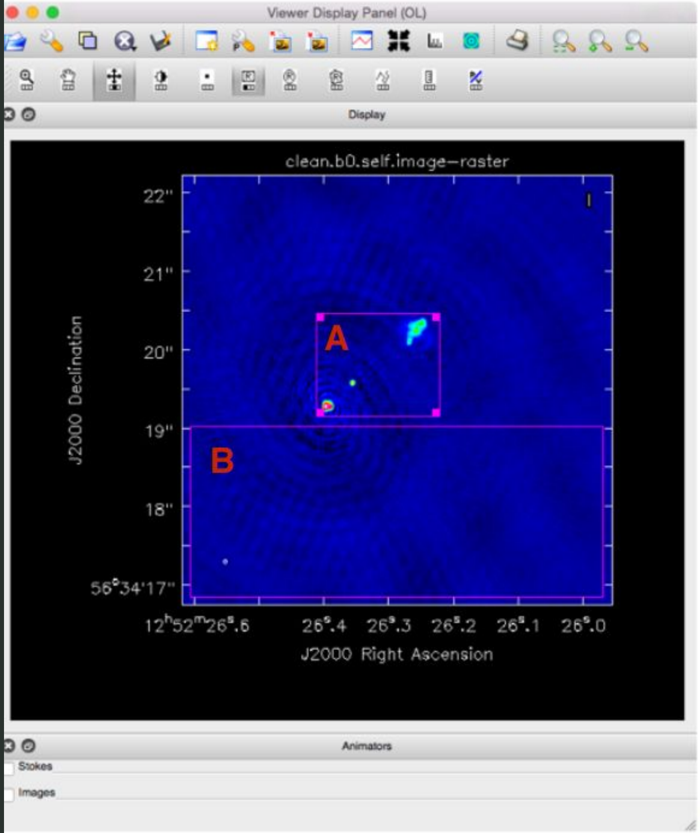
Assess + adjust



Always review all the outputs (data, tables, images)

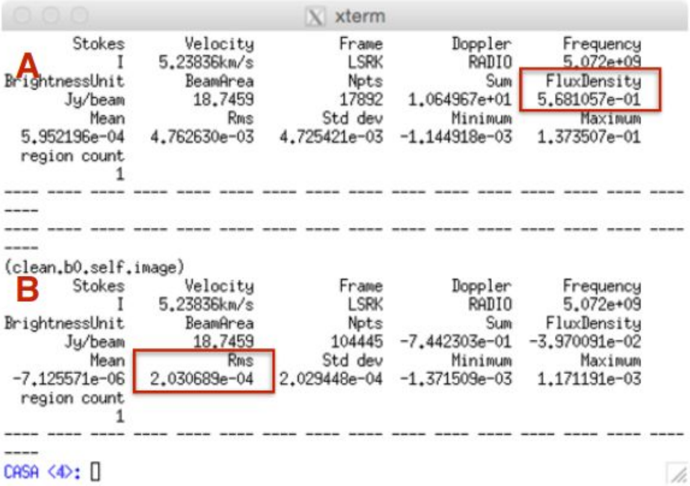
Inspect your image

Also inspect residual image: lowest negative, artifacts, etc



Double click inside the regions to get the statistics.

We find that self-calibration has lowered the noise, and increased the removed flux of the sources




```
        Stokes       Velocity        Frame       Doppler      Frequency
             I   5.23836km/s          LSRK         RADIO      5.072e+09
BrightnessUnit      BeamArea          Npts           Sum    FluxDensity
       Jy/beam       18.7459         17892  1.064967e+01   5.681057e-01
          Mean           Rms       Std dev       Minimum        Maximum
  5.952196e-04  4.762630e-03  4.725421e-03 -1.144918e-03   1.373507e-01
  region count
             1
---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
----
---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
----
(clean.b0.self.image)
        Stokes       Velocity        Frame       Doppler      Frequency
             I   5.23836km/s          LSRK         RADIO      5.072e+09
BrightnessUnit      BeamArea          Npts           Sum    FluxDensity
       Jy/beam       18.7459        104445 -7.442303e-01  -3.970091e-02
          Mean           Rms       Std dev       Minimum        Maximum
 -7.125571e-06  2.030689e-04  2.029448e-04 -1.371509e-03   1.171191e-03
  region count
             1
---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
----
CASA <4>:
```

Post-calibration flagging

Inspect calibration tables and corrected data for spurius values

Manual flagging. Always use external file

```
mode                =      'list'          #  Flagging mode (list/manual/clip/quack
                                           #   /shadow/elevation/tfcrop/rflag/antin
                                           #   t/extent/unflag/summary)
     inpfile        = 'manual.flags'       #  Input ASCII file, list of files or
                                           #   Python list of strings with flag
                                           #   commands.
```

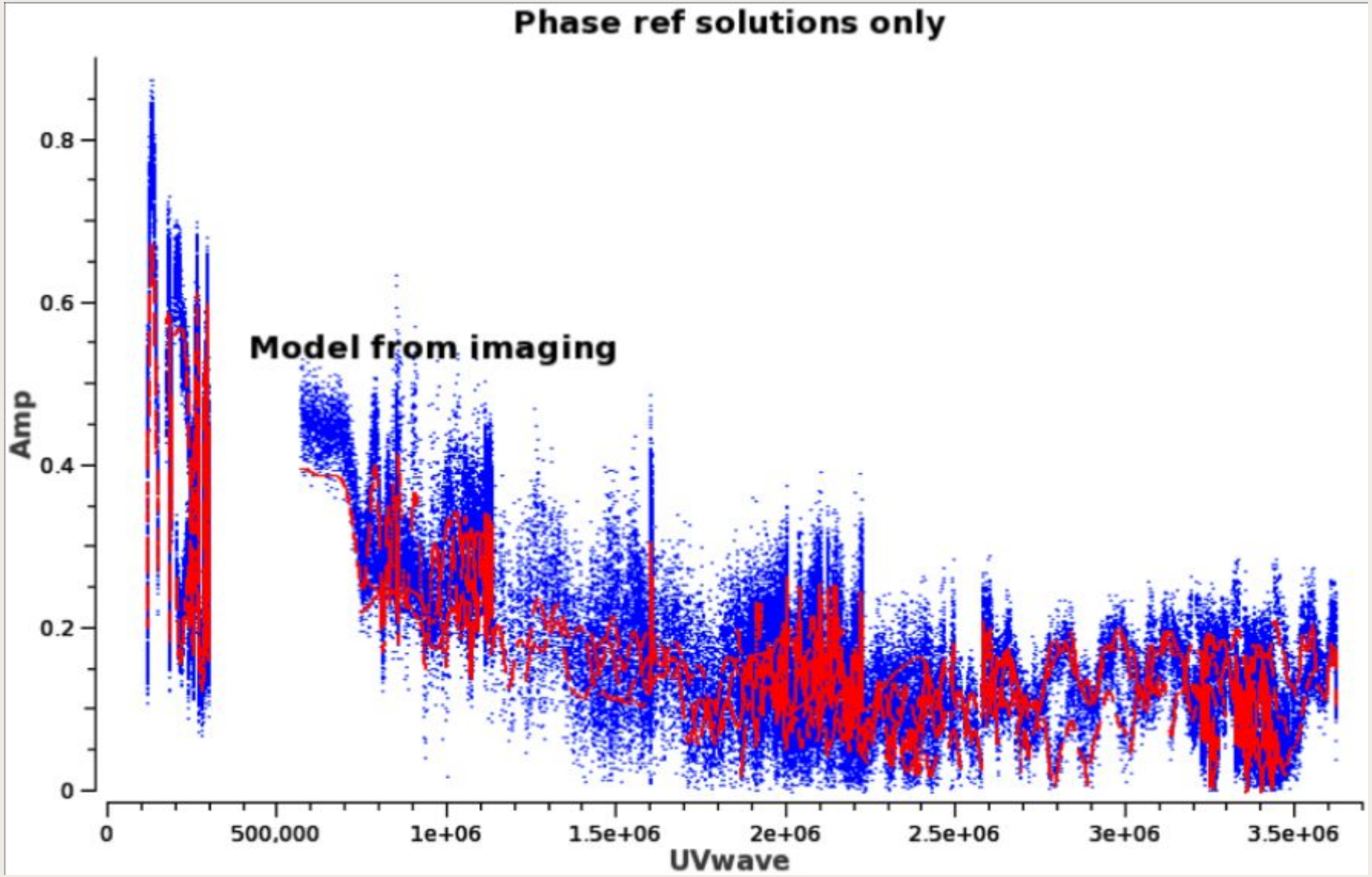
```
# File: manual.flags
mode='manual' timerange='2020/10/04/12:35:00~2020/10/04/12:40:30' antenna='Cm'
mode='manual' timerange='2020/10/04/12:57:30~2020/10/04/13:04:00' antenna='Cm'
mode='manual' timerange='2020/10/04/12:33:00~2020/10/04/12:36:00' field='1407+2827'
mode='manual' timerange='2020/10/04/12:46:41~2020/10/04/12:46:42' antenna='Da'
# I can add comments to describe my decisions
mode='quack' field='1258-2219,1309-2322' quackinterval=24.
```

Automatic flagdata: mode TFCROP/RFLAG

```
mode                =   "tfcrop"
datacolumn          =   "residual"
timecutoff          =   4.5
freqcutoff          =   4.5
timefit             =   "line"
freqfit             =   "poly"
maxnpieces          =   7
flagdimension       =   "freqtime"
```

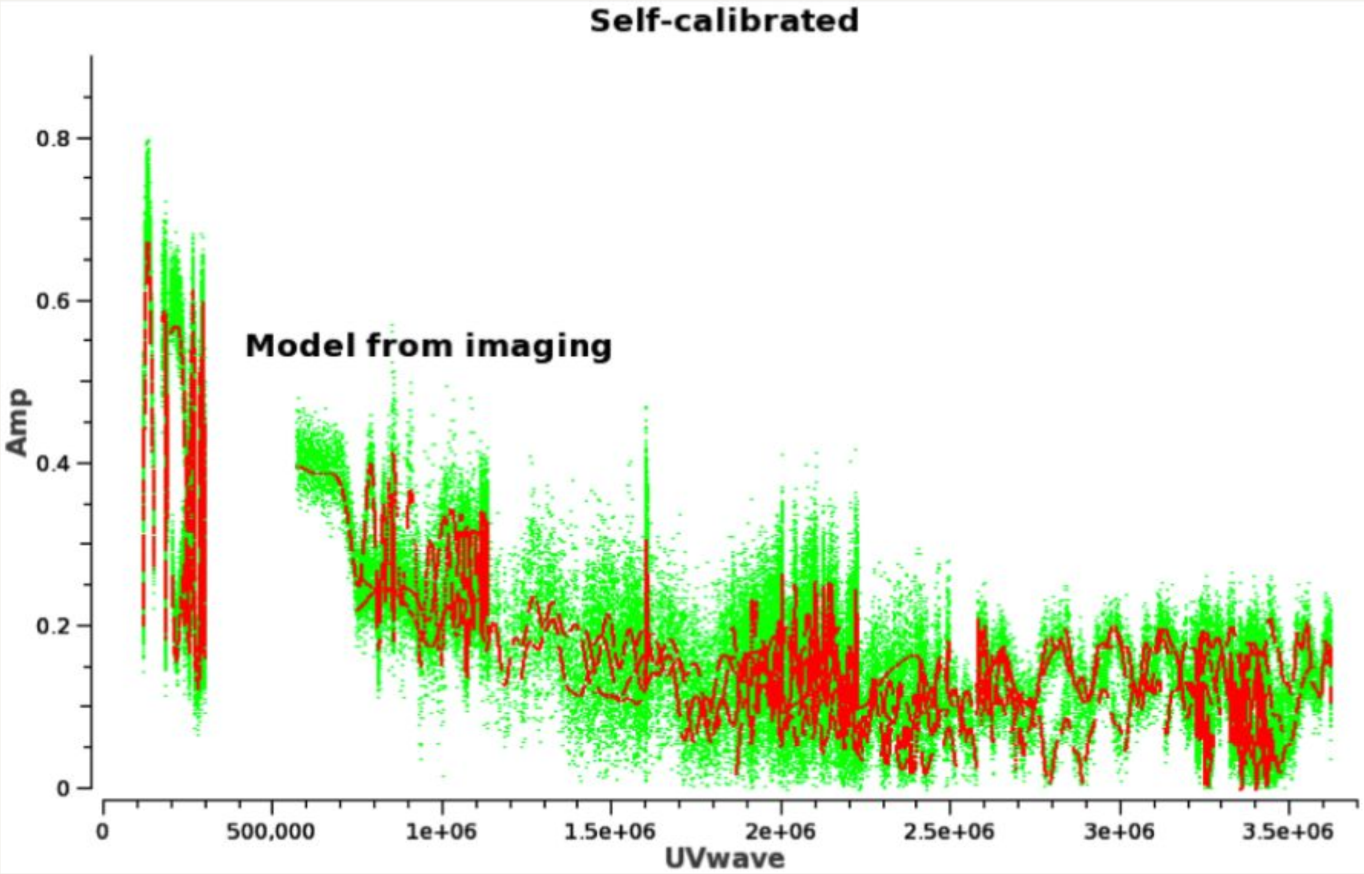
Inspect the data

Look for any spurious deviations, or for missing flux on the short baselines



Inspect the data

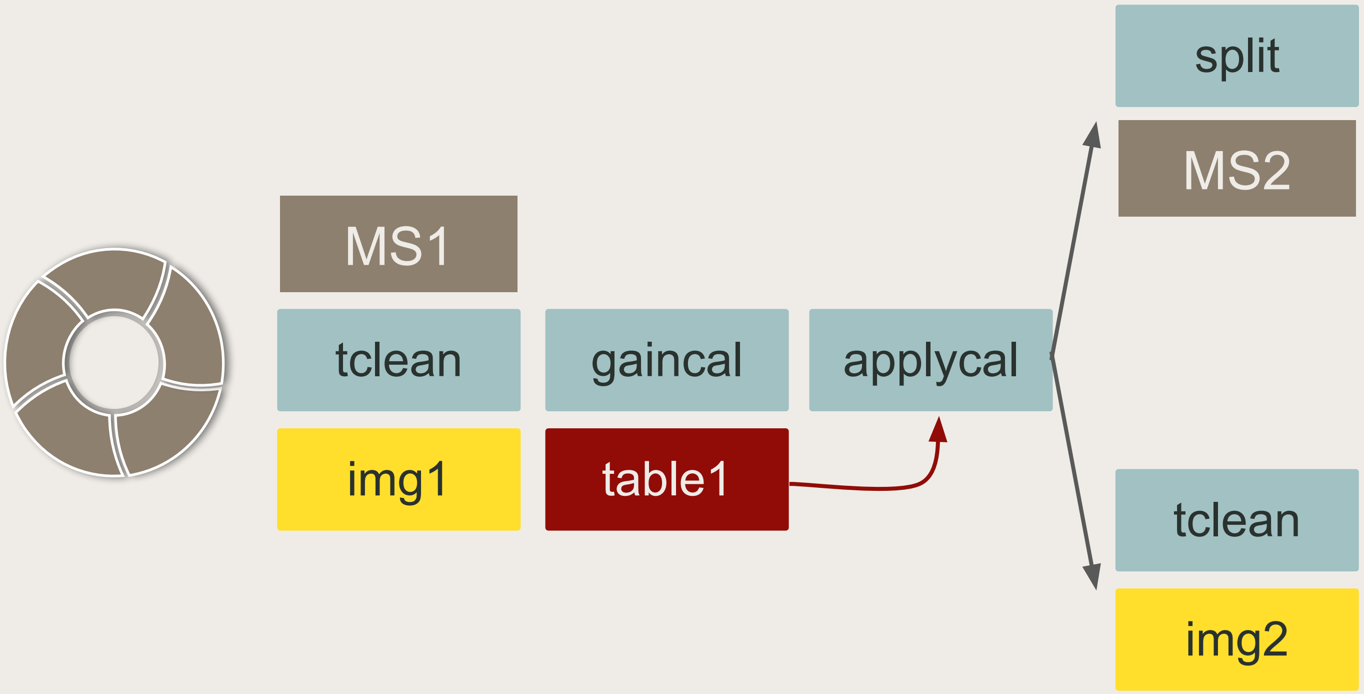
Look for any spurious deviations, or for missing flux on the short baselines



Let's iterate some self-cal loops

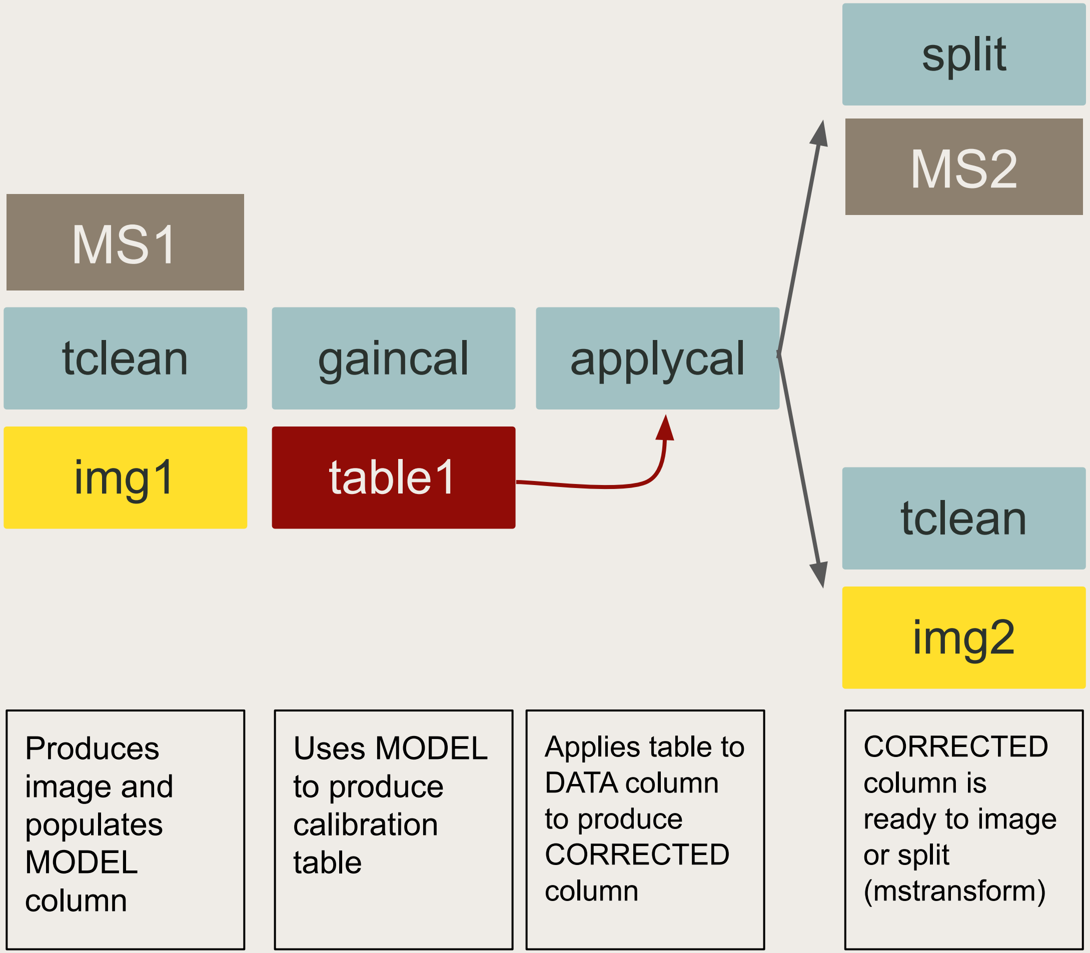
Complete selfcal cycle

Once corrected, you can choose to split (mstransform) to fix the corrections, or to image the corrected column



Complete selfcal cycle

Once corrected, you can choose to split (mstransform) to fix the corrections, or to image the corrected column



Cycles

This is only an example to show some gaincal parameters. All would use `gaintype='G'`

1. **Phase**

   solint = ‘64s’

2. **Phase**

   solint = ‘32s’

3. **Amplitude & Phase**

   solint = ‘3600s’, combine=’scan’

Cycles

This is only an example

4

Phase

solint = ‘8s’, combine=’spw’

5

Phase

gaintype=’T’, solint = ‘int’, combine=’spw’

6

Amplitude & Phase

solint = ‘inf’, combine = ’’

Three options to implement the loops

Option 1

Fixing the corrections with mstransform at each iteration
(too much wasted disk space)

Option 2

Work always with original data. Only update the model
(unsuitable for amplitude calibration)

Option 3

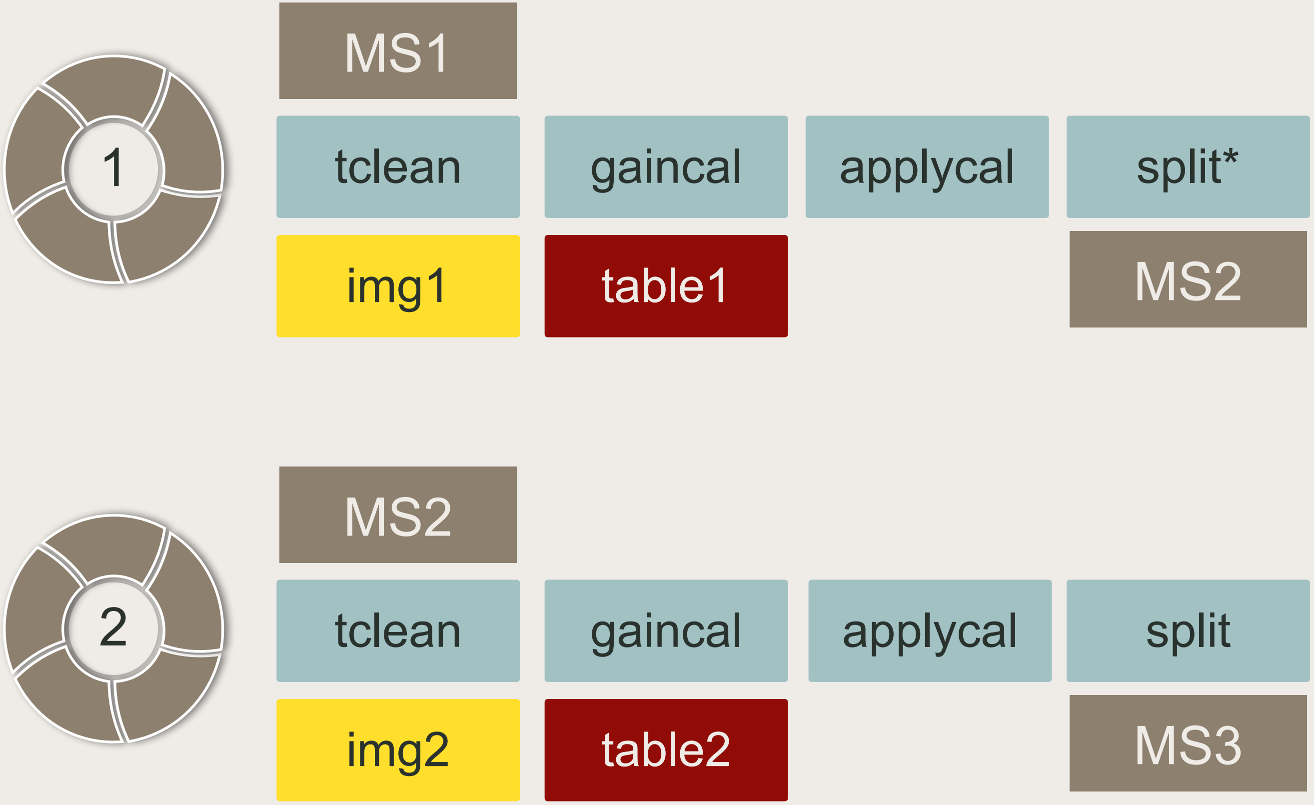
Incrementally adding more tables
(the one I recommend)

Fixing the corrections at each iteration

Table2 will have just incremental corrections, so you can check second order effects

*split = mstransform

OPTION 1

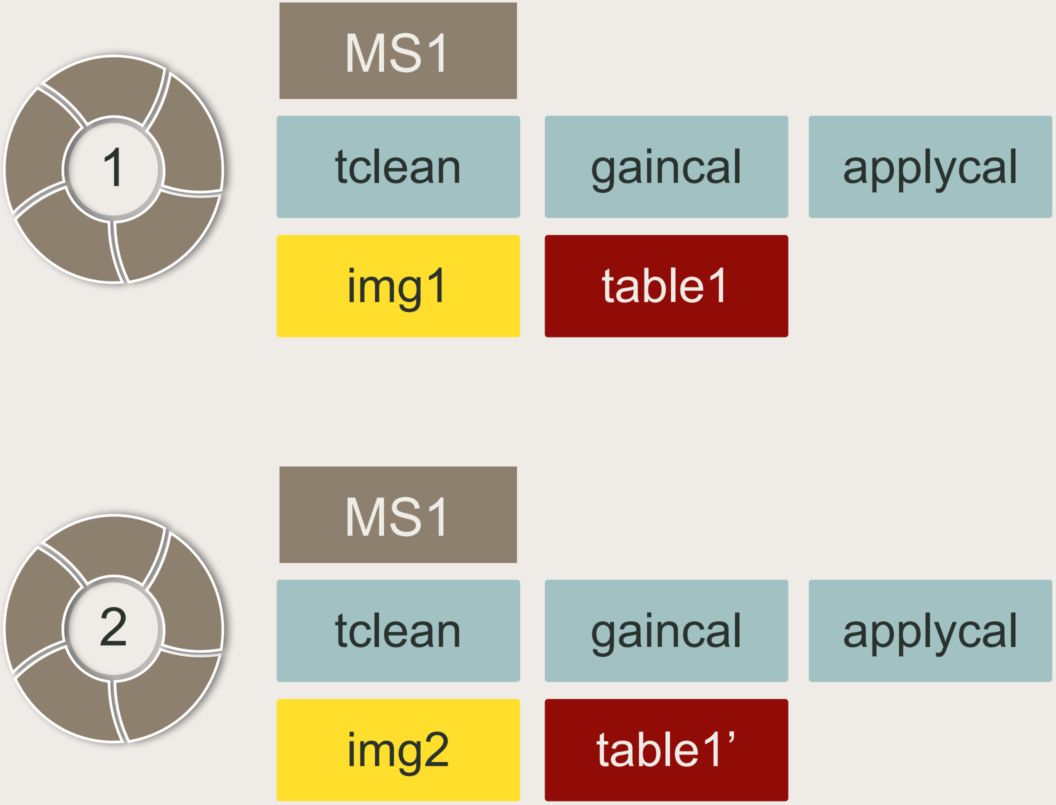


Only update the model. All corrections in one table

Table1’ contains corrections from scratch.

Not recommended in general due to amp decoherence.

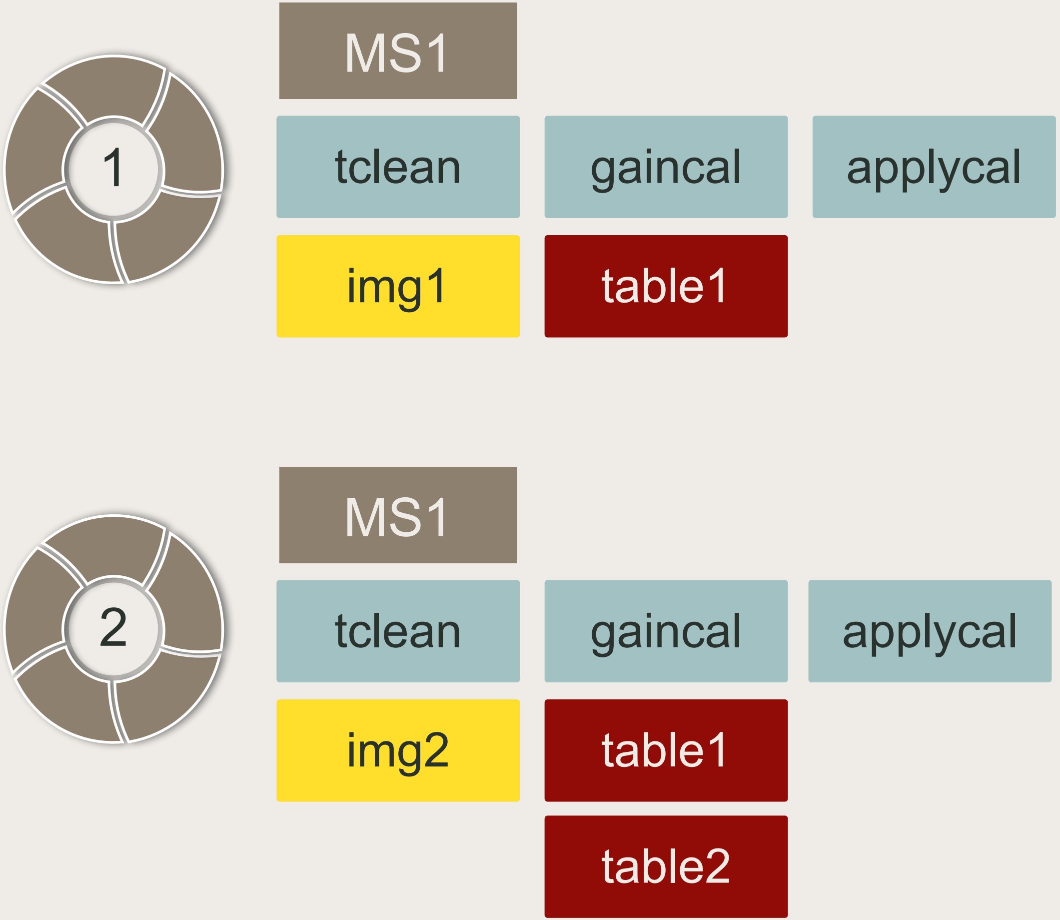
OPTION 2



OPTION 3

Incrementally adding more tables

Applycal will use all the previous tables combined. Table2 will have just incremental corrections, so you can check second order effects



OPTION 3

gaincal | applycal

1

```
#  gaincal :: Determine temporal gains
vis                    =       'my.ms'
caltable               = 'gaincal_01'
docallib               =        False
     gaintable         =           ''
```

```
#  applycal :: Apply calibrations solutions
vis                    =        'my.ms'
docallib               =         False
     gaintable         = 'gaincal_01'
```

2

```
#  gaincal :: Determine temporal gains
vis                    =       'my.ms'
caltable               = 'gaincal_02'
docallib               =        False
     gaintable         = 'gaincal_01'
```

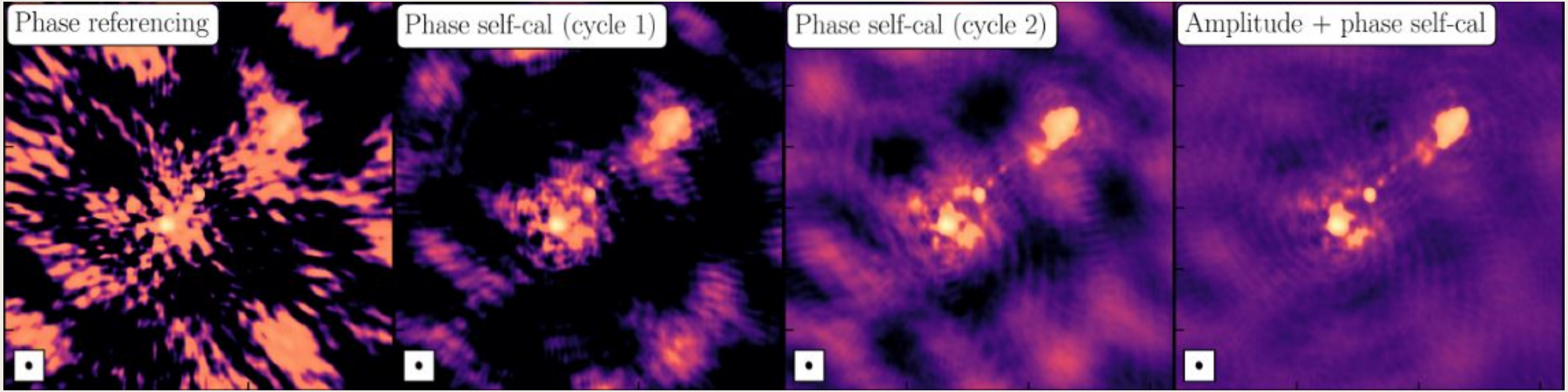
```
#  applycal :: Apply calibrations solutions
vis                    =        'my.ms'
docallib               =         False
     gaintable         = ['gaincal_01',
                          'gaincal_02']
```

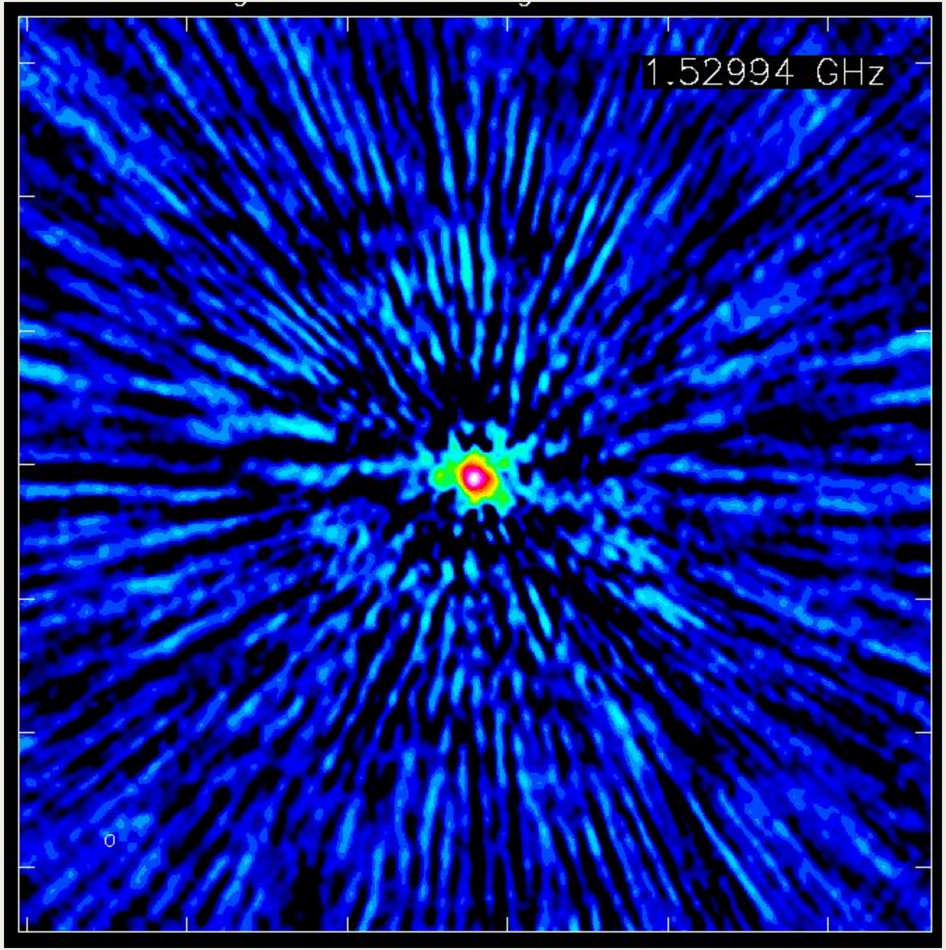
3

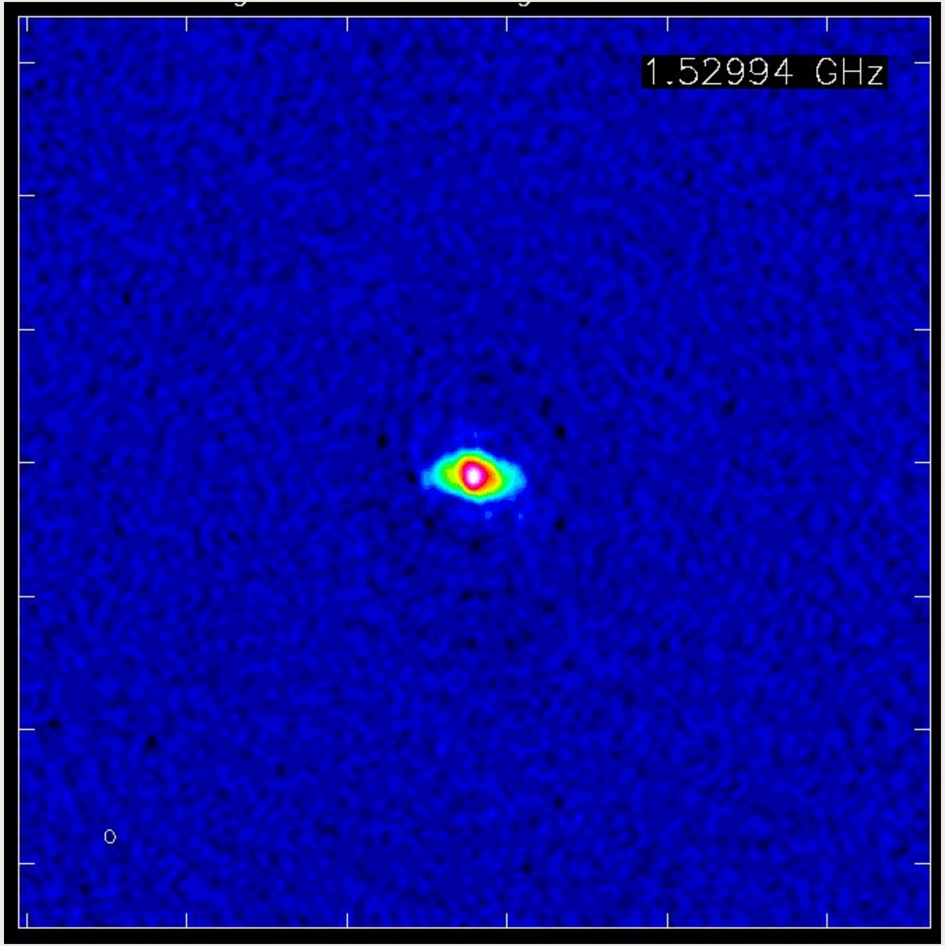
```
#  gaincal :: Determine temporal gains
vis                    =       'my.ms'
caltable               = 'gaincal_03'
docallib               =        False
     gaintable         = ['gaincal_01',
                          'gaincal_02']
```

```
#  applycal :: Apply calibrations solutions
vis                    =        'my.ms'
docallib               =         False
gaintable              = ['gaincal_01',
                          'gaincal_02',
                          'gaincal_03']
```

Self-calibration improvements


Phase referencing
Phase self-cal (cycle 1)
Phase self-cal (cycle 2)
Amplitude + phase self-cal


1.52994 GHz
0


1.52994 GHz
0

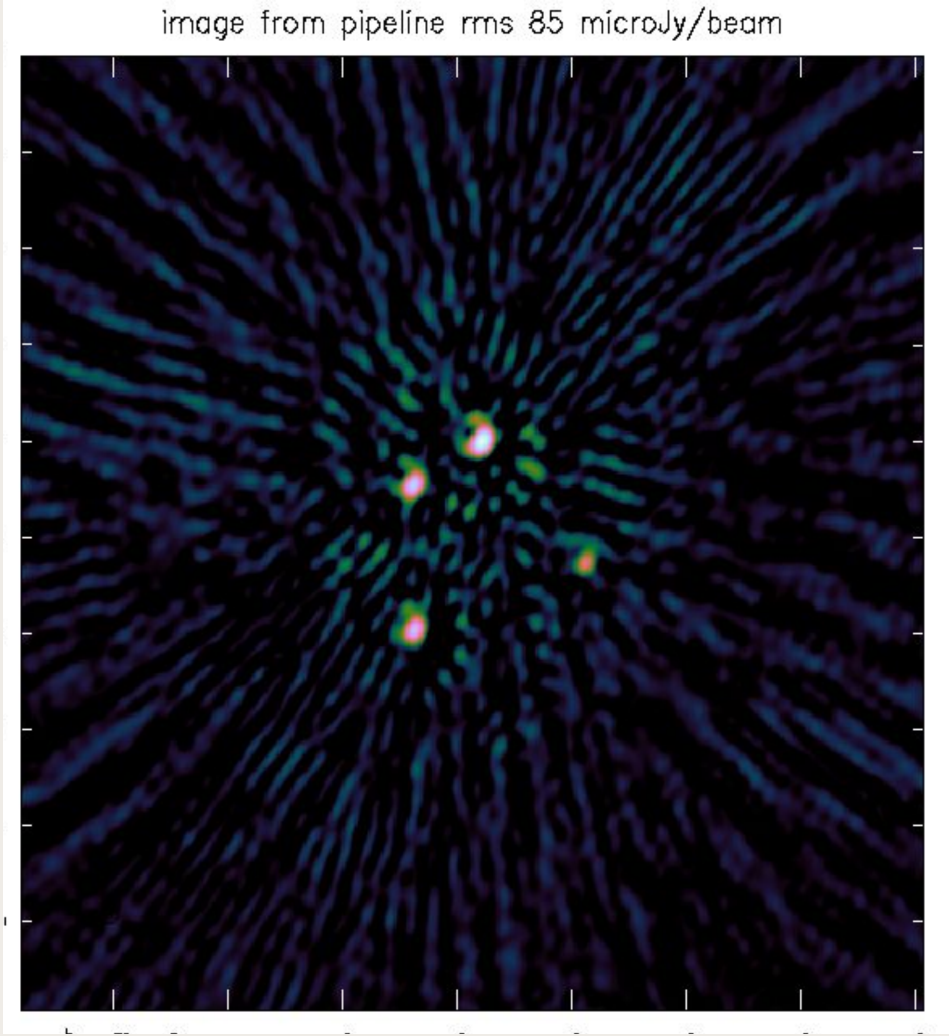
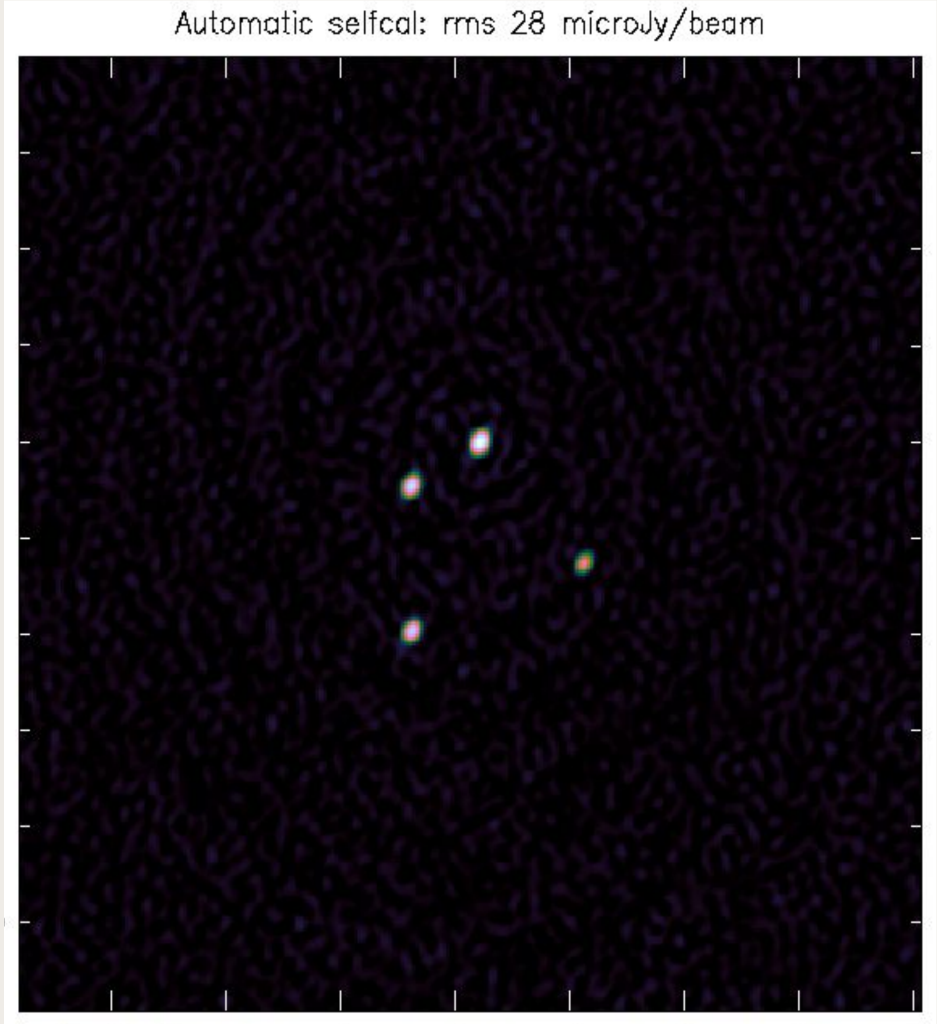
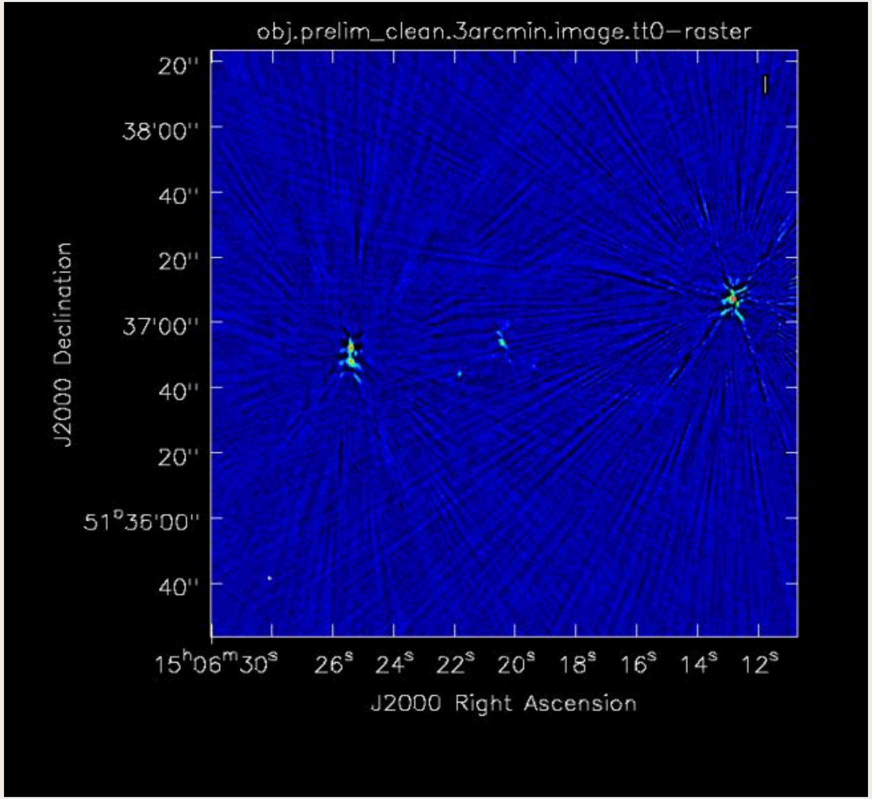
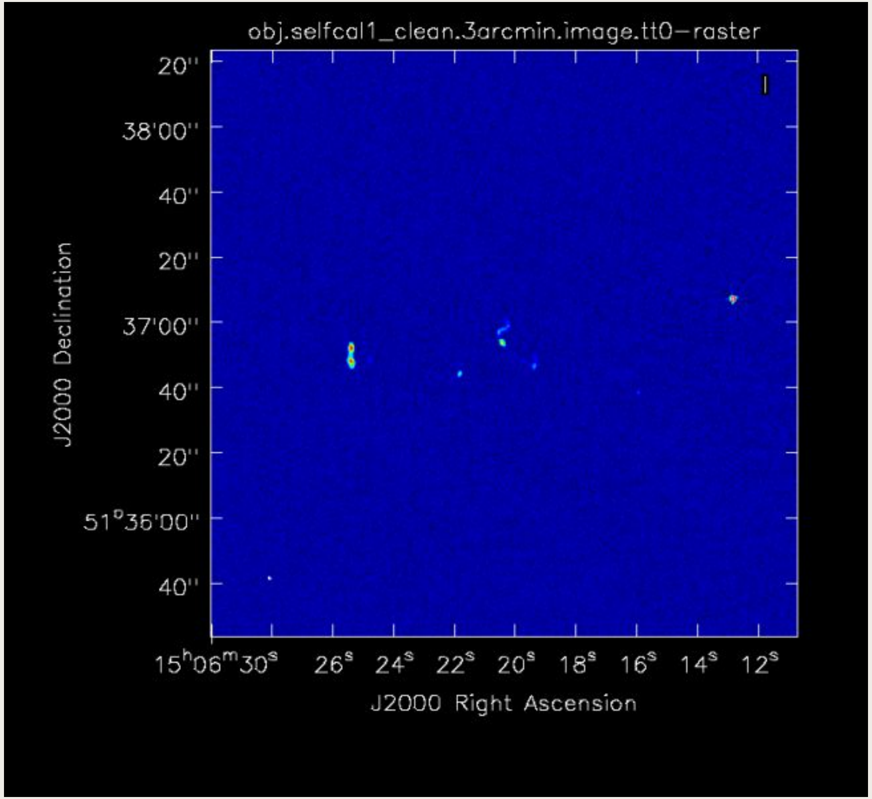


image from pipeline rms 85 microJy/beam


Automatic selfcal: rms 28 microJy/beam

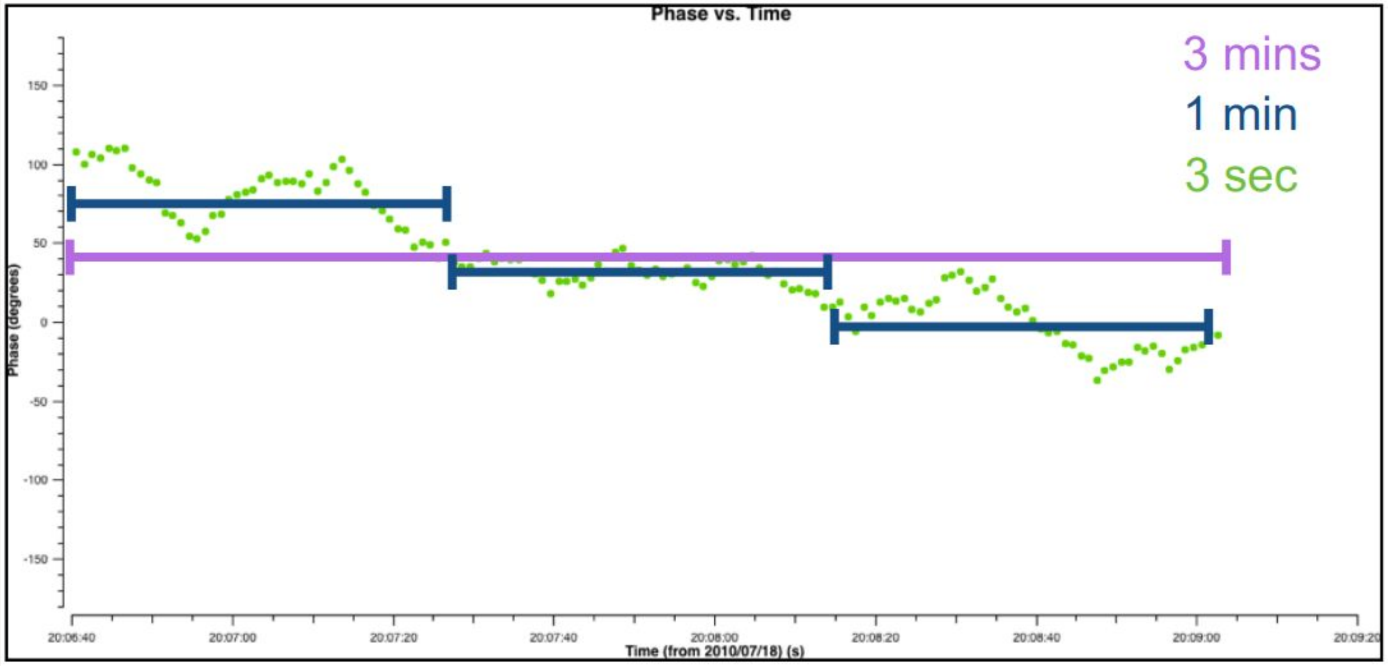

obj.prelim_clean.3arcmin.image.tt0−raster
J2000 Declination
J2000 Right Ascension
20''
38'00''
40''
20''
37'00''
40''
20''
51°36'00''
40''
15h06m30s
26s
24s
22s
20s
18s
16s
14s
12s


obj.selfcal1_clean.3arcmin.image.tt0−raster
J2000 Declination
J2000 Right Ascension
20''
38'00''
40''
20''
37'00''
40''
20''
51°36'00''
40''
15h06m30s
26s
24s
22s
20s
18s
16s
14s
12s

Choices

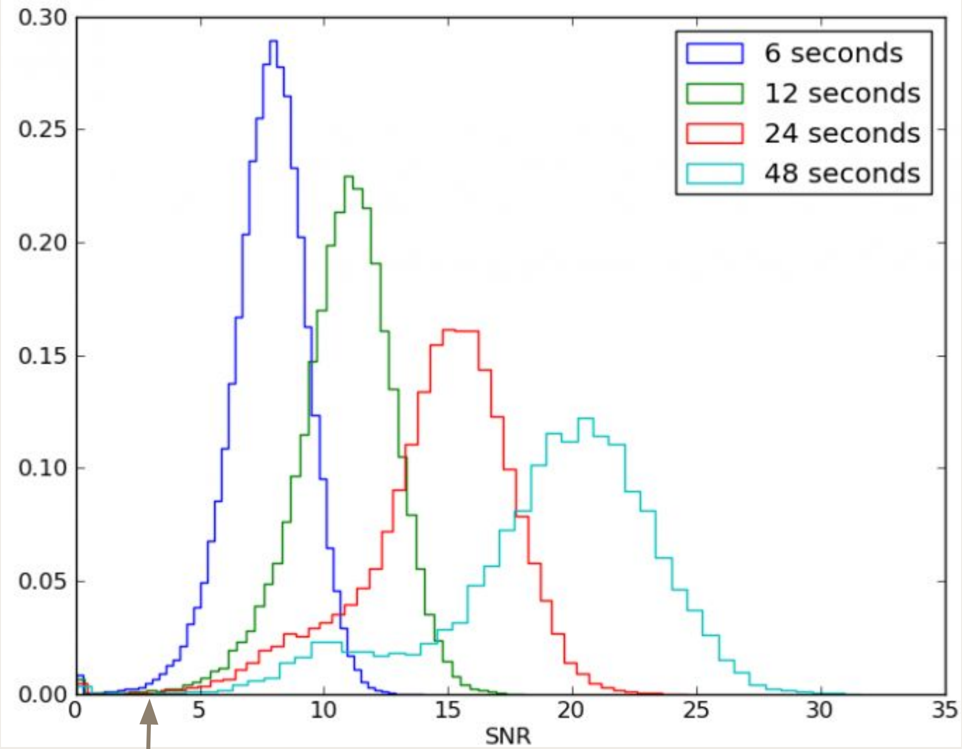
Solution interval

Shortest possible time-scale to track the gain variations, whist being long enough to have a sufficient S/N



Solution interval

It is a tradeoff between accurate change tracking, and having enough SNR to avoid flagging uncorrected data



`minsnr` parameter in gaincal cuts these solutions

How to decide the loops?

Progressively improve model and reduce solint

- Always start with phase-only self-calibration
- Don’t run amplitude self-cal until your phases are good. Otherwise amplitude corrections will compensate for decorrelation.
- Decrease the solution intervals progressively so you get an improved model without including artifacts
- At some point, you may need to combine spw or pols
- Decide the new loop based on model quality, SNR of solutions, stability of solutions.

When to stop?

Some guidelines

- Complex sources may require several cycles
- For many VLBI runs, try a few (p, p, a&p) cycles
- Go as sort `solint`s as allowed by your data
- Don’t rush the model construction

- Iterate until no further dynamic range improvement
- Reach the image that is good enough for your science

Other practical considerations

Combine to improve S/N

By combining more data we can reach sorter solution intervals when S/N is limited

combine = ‘scan’

Usually needed for initial amplitude self-cal steps

combine = ‘spw’

After phase offsets are removed. You will need to use spwmap properly. Example: spwmap = [0,0,0,0].
Use for amplitudes only if mtmfs model is present.

gaintype = ‘T’

Both parallel-hand correlations are combined together.
Improves S/N by a factor $\sqrt{2}$

Callib file in applycal

It is really useful for scripting as you encapsulate the additive calibration in an external file

Traditional applycal

```
docallib      =  False
  gaintable   =  ['./calibration/gcal.loop_01','./calibration/gcal.loop_02',
                   './calibration/gcal.loop_03','./calibration/gcal.loop_04']
  gainfield   =  []
  interp      =  ['linear','linear','linear','linear']
  spwmap      =  [[0,0],[0,0],[],[0,0]]
```

Using Cal library

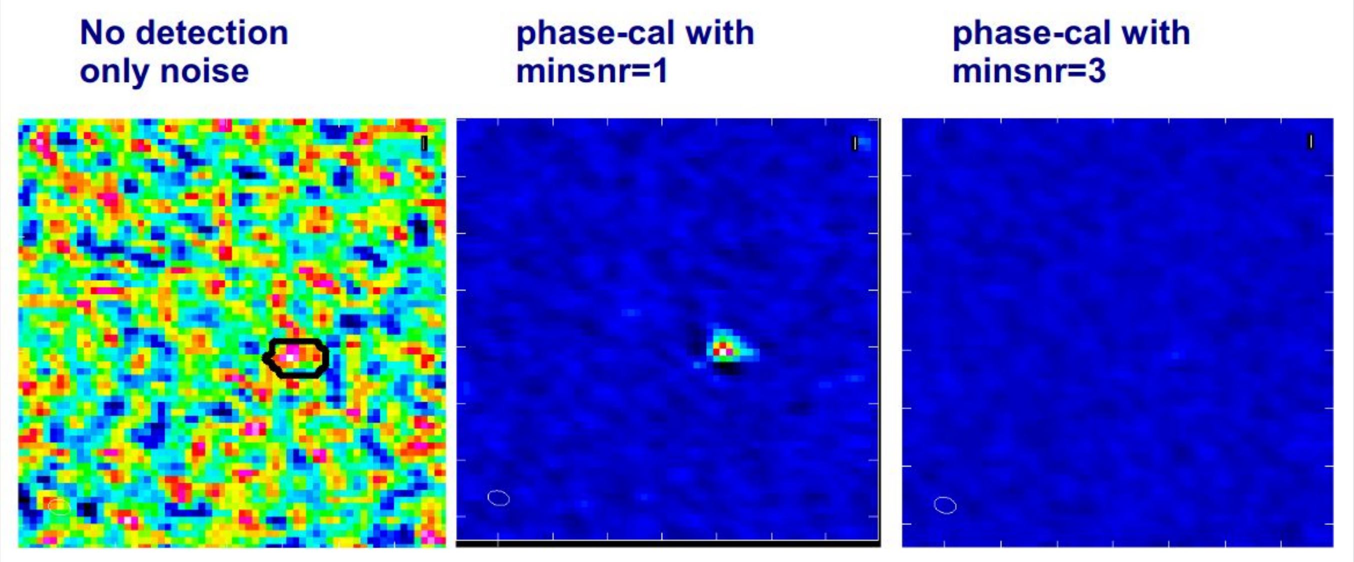
```
docallib      =  True            #  Use callib or traditional cal apply parameters
    callib    =    'callib.txt'  #  Cal Library filename
```

Example callib file

```
caltable='./calibration/gcal.loop_01' calwt=True tinterp='linear' spwmap=[0, 0]
caltable='./calibration/gcal.loop_02' calwt=True tinterp='linear' spwmap=[0, 0]
caltable='./calibration/gcal.loop_03' calwt=True tinterp='linear'
caltable='./calibration/gcal.loop_04' calwt=True tinterp='linear' spwmap=[0, 0]
```

Never include noise into your model

Be careful with the minimum S/N because fake sources can gather unreal signal

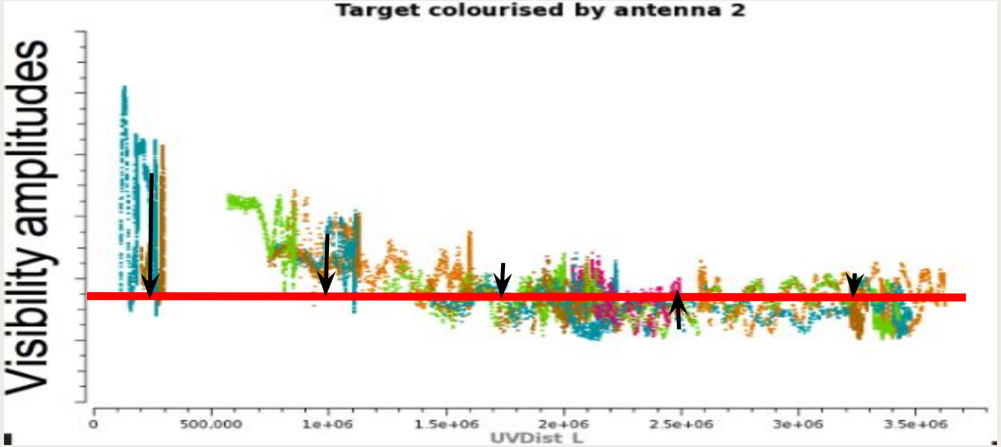


Amplitude normalization

Avoid altering the global flux scale

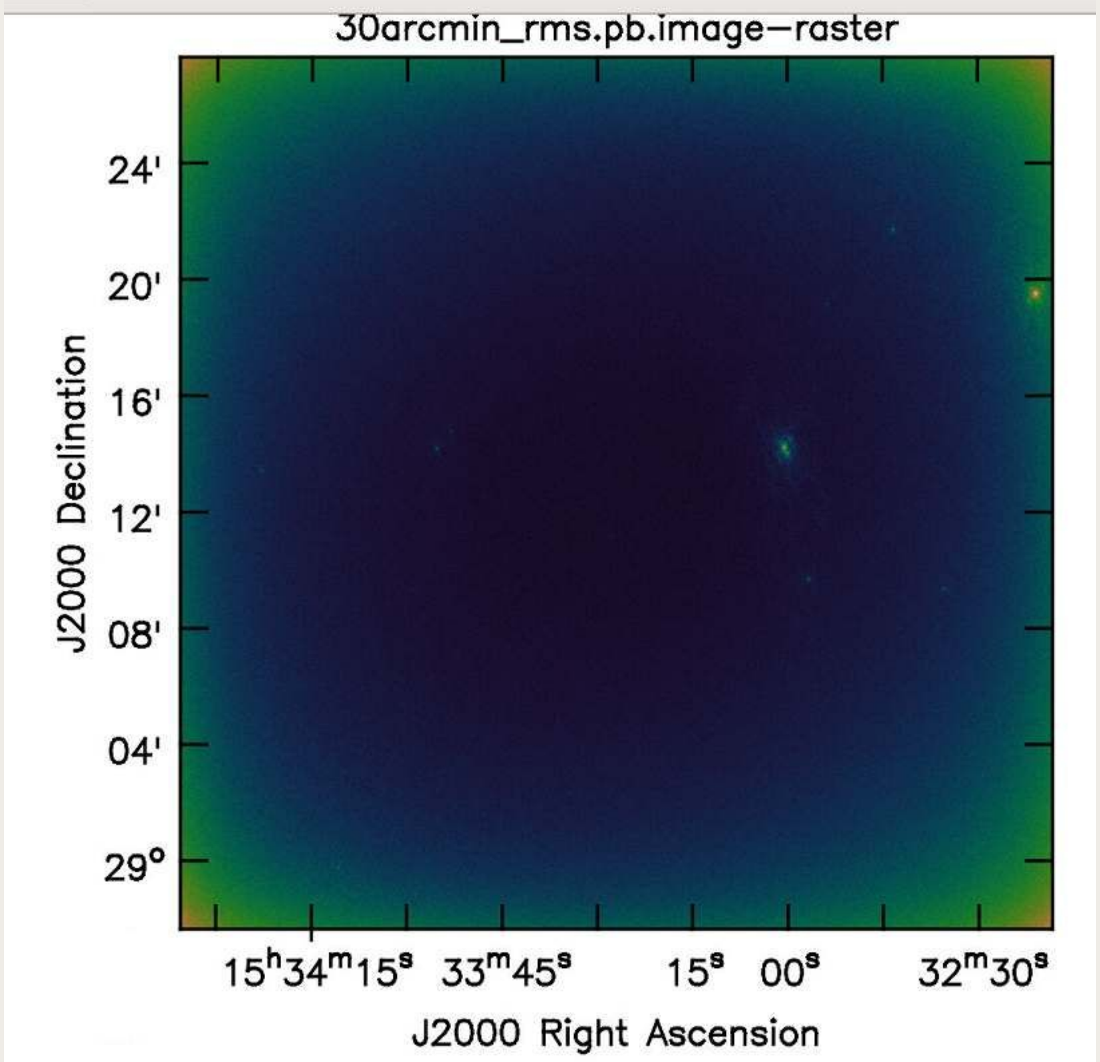
- Amplitude self-cal is meant to solve for time-dependent gain residuals, not the flux scale
- Models may underestimate total flux in the field
- To avoid drifting the flux towards an incomplete model, you can normalize the amplitude solutions:

```
solnorm = True
```



Other sources in the FoV

They can help with selfcal or as flux/astrometry check. May require multiple phase centres during correlation



Conclusions

To conclude

- Self-calibration is a powerful tool
- It is also dangerous
- But it is safe if you know the assumptions

- You need it to extract all the science from your data
- The quality and reliability of your science output depends on doing self-calibration properly

THANKS TO OUR SPONSORS:

CASA
VLBI

JUMPING JIVE
Joint Institute for VLBI
ERIC

RadioNet

THIS EVENT HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S HORIZON 2020 RESEARCH AND INNOVATION PROGRAMME UNDER GRANT AGREEMENTS 730562 (RADIONET) AND 7308844 (JUMPING JIVE)