RFI Identification and Flagging

Evangelia (Lilia) Tremou (NRAO)

etremou@nrao.edu

This Talk:

- CASA version 6.2

  On lustre:

  > casa -r 6.2.0-124

- Interactive steps can be done after the talk

  - tasks & parameters are given in the following slides

- Data access: https://astrocloud.nrao.edu/s/DcwqTKkKTEi98sk

  (drwRFI.ms ~ 235MB)

What is RFI?

RFI - Radio Frequency Interference

- A disturbance caused by various sources (man-made) of radio waves emitting around our targeted frequencies that affect our data, introducing noise.

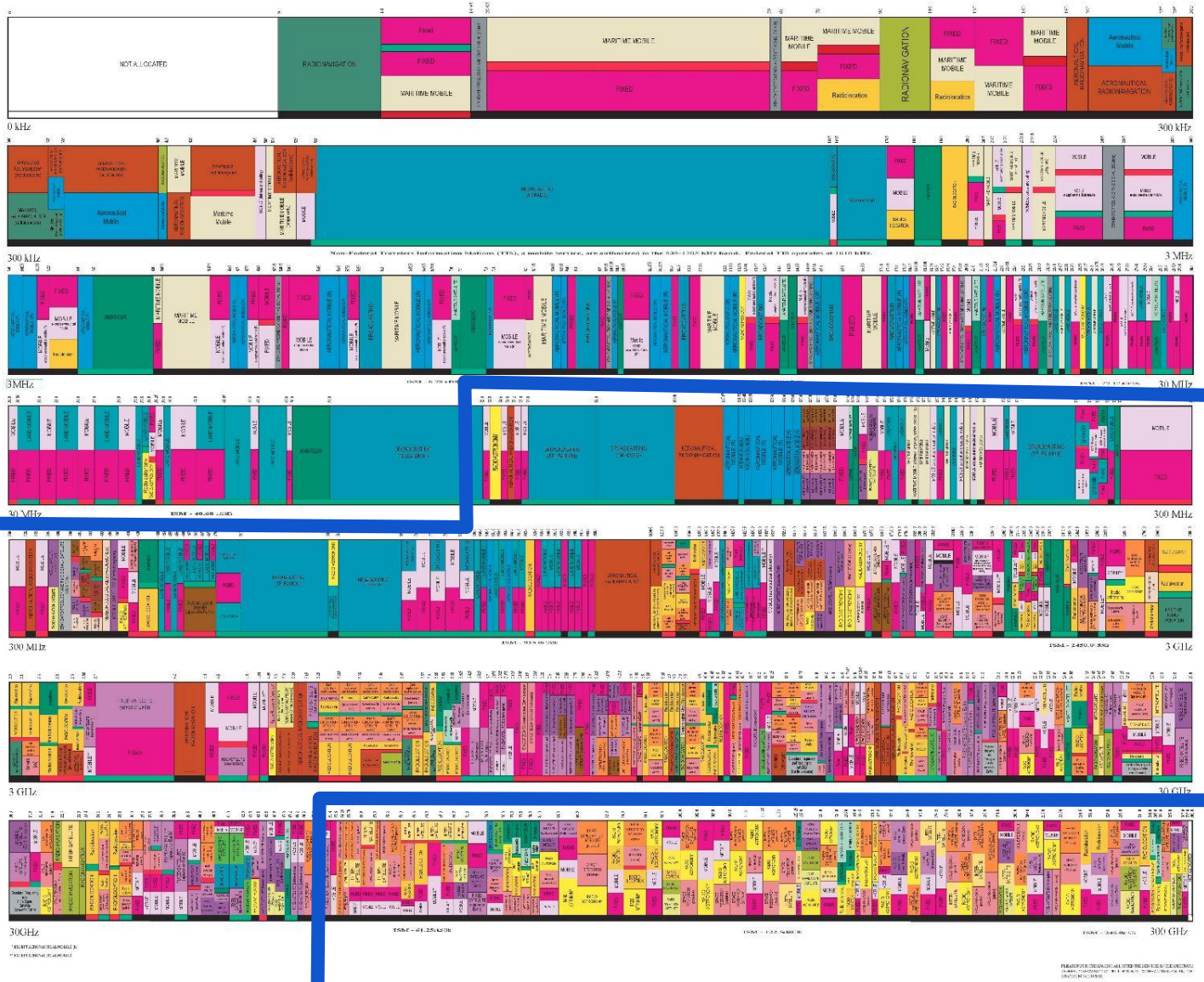
- Often RFI is stronger than the science data, and can limit science (parts of the spectrum are unusable)

- RFI increases the detected power and the estimated antenna temperature → degradation of the retrieved astronomical signal.

- Multiple approaches to perform RFI mitigation (i.e. detection algorithms, filters)

UNITED STATES FREQUENCY ALLOCATIONS

THE RADIO SPECTRUM

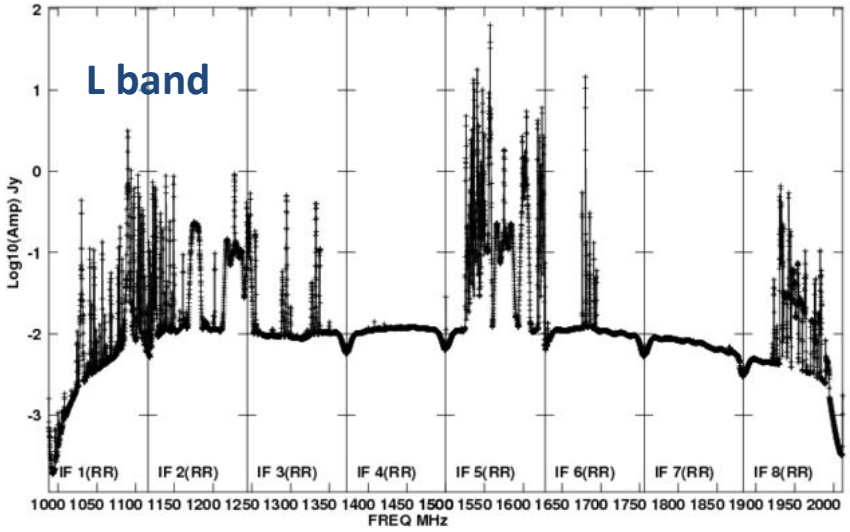


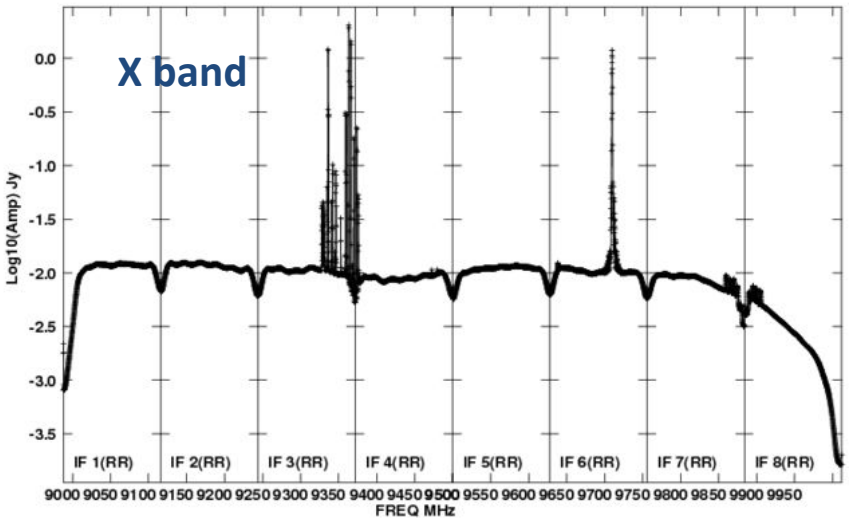
VLA frequency range

RFI at the VLA

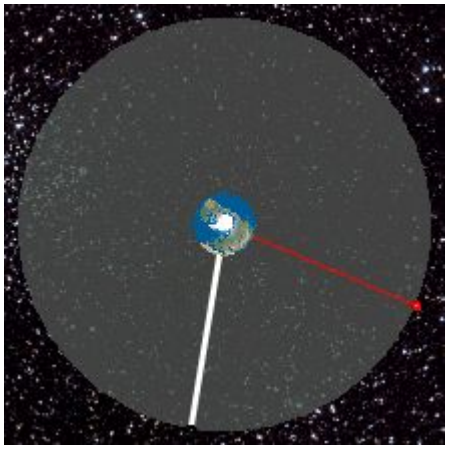
https://science.nrao.edu/facilities/vla/docs/manuals/obsguide/rfi

- Biggest issue at lower frequency bands: 4, P, L, S, C, and D-configuration.
- But it does not mean it doesn't exist at higher frequency bands.
- Can be internal or external. Internally-generated RFI is minimised





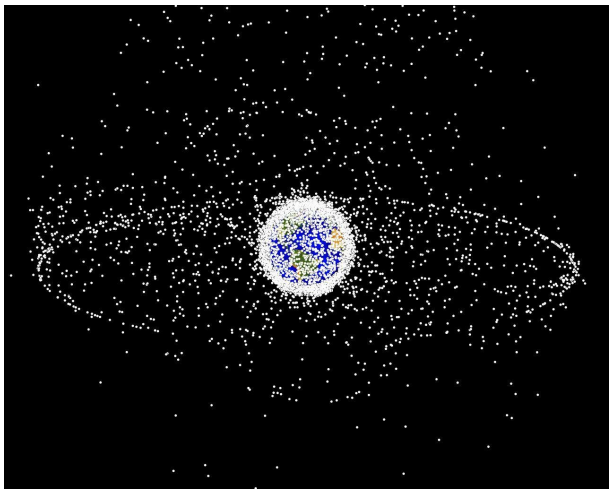
Satellite Transmissions & Clarke Belt



- Earth is heavily surrounded by the satellites (2000+)
- Hundreds along the Clarke belt

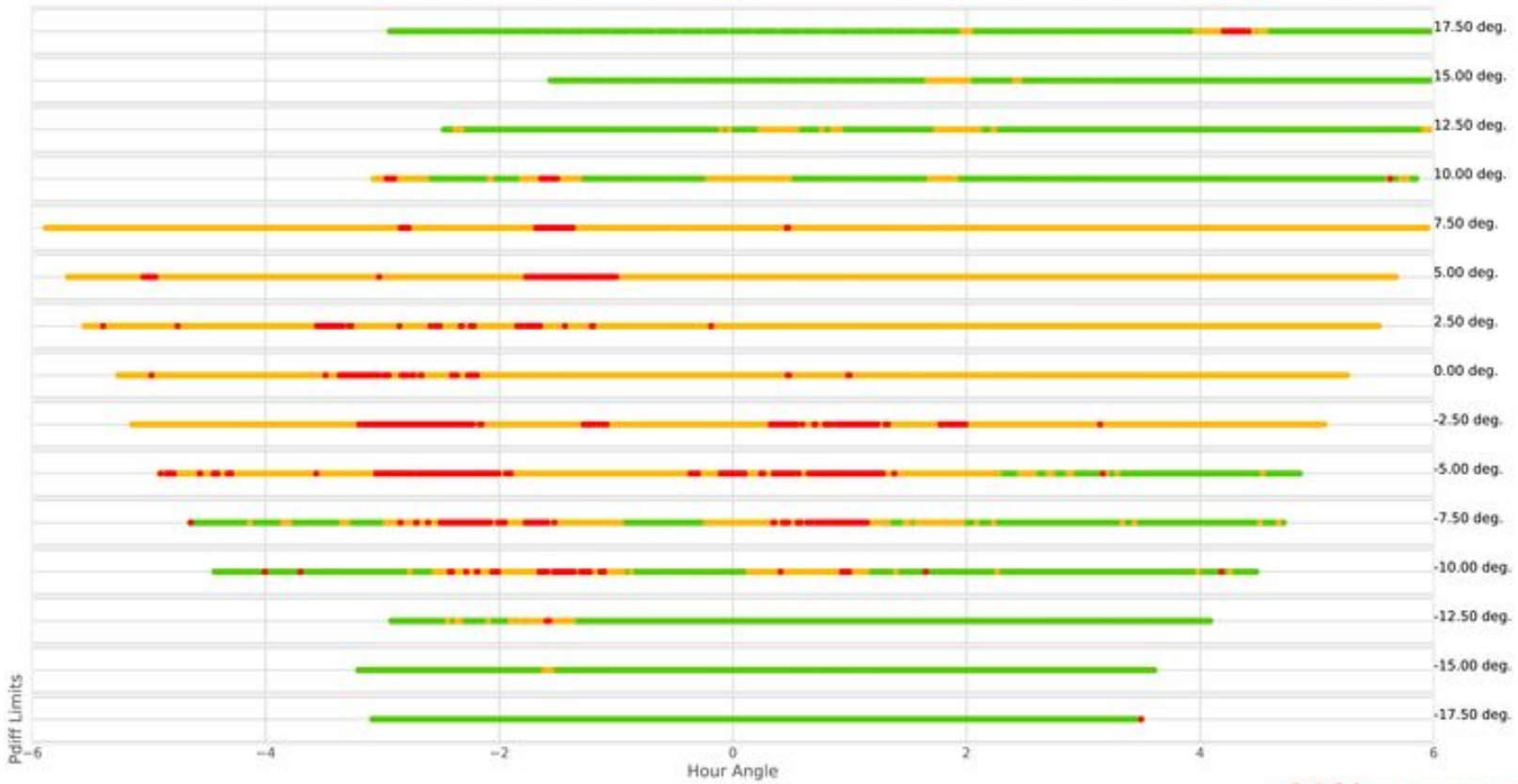
How does it affect VLA data?

- significant degradation of data can occur if VLA antenna observes within 10° of the satellite
- @VLA Clarke Belt at declinations –15° to +5°
- mainly S, C, Ku, K and Ka bands
- in C, X and Ku bands satellites can also saturate the 3-bit samplers (special 8/3-bit set up required)



Satellite Transmissions & Clarke Belt

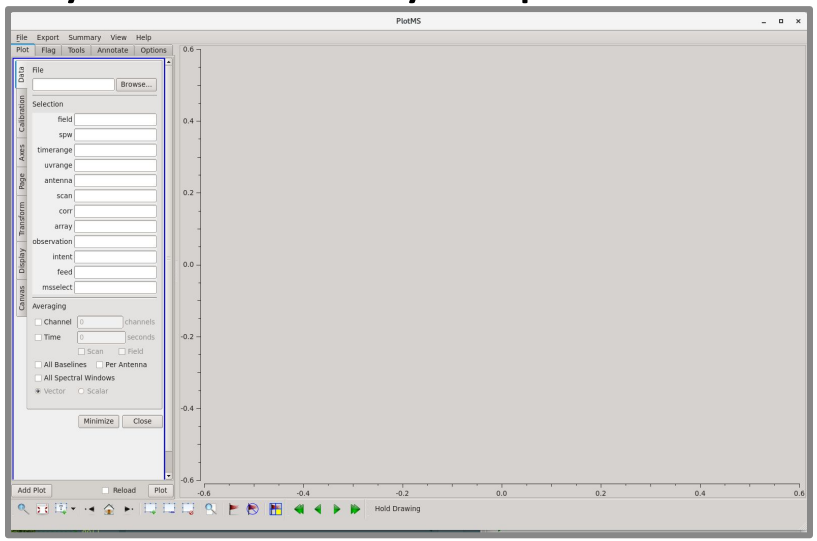
S band (2 - 4 GHz) survey of satellite interference at VLA (conducted in 2016/2017)



>20% compression
5–20% compression
<5% compression

Finding RFI in your data

CASA task that allow you to visually inspect the data: **plotms()**



- Leaving off strong RFI in your images will have issues such as “ripples”, high noise, etc.

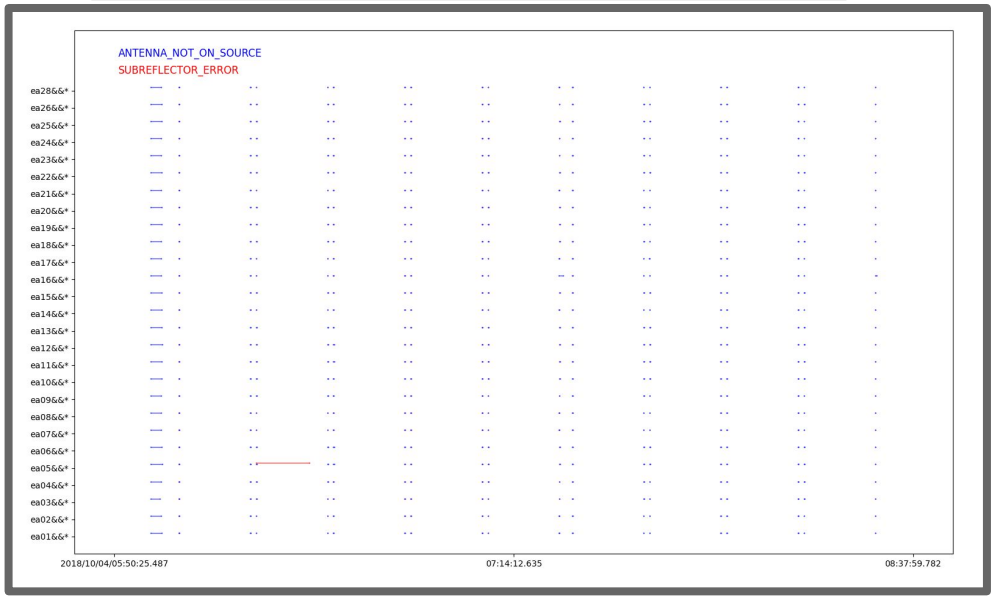
- This is also how you can check for the remaining offending RFI in your data: **image**!

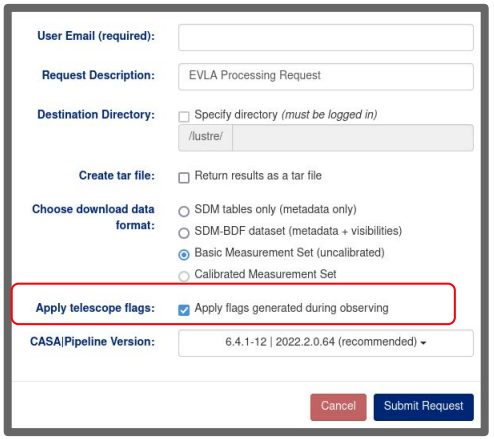
CASA Flagging tasks

flagcmd () → apply flags info on which stored in external file

- Example: Online Flags
  - issues recorded by the operators such as slew, subreflector, focus
  - During downloading data from NRAO archive → apply online flags

```
flagcmd(vis='drwRFI.ms', action='plot')
```





CASA Flagging tasks

flagdata () → recommended task to flag data

Deterministic

mode='manual' or mode='unflag'	# Use MS-selection syntax to pick subsets of flag/unflag
mode='quack'	# Flag data at the beginning or end of a scan (operate on selected data)
mode='elevation'	# Flag data between specified elevation limits (operate on selected data)
mode='shadow'	# Flag baselines with shadowed-antennas
mode='clip'	# Threshold-based flagging on data-expressions (ABS_RR, ABS_I, etc.)

Autoflag

mode='tfcrop'	# Find outliers on the 2D time-frequency plane
mode='rflag'	# Find outliers based on sliding-window RMS filters
mode='extend'	# Grow/extend flags around existing ones

Operational

mode='summary'	# Count existing flags and return a dictionary of counts per antenna, spw, etc.
mode='list'	# Supply a list of flag commands, built out of parameters of any other mode

Run-modes: "apply / calculate" + runtime "display" of data before and after flagging, and reports.

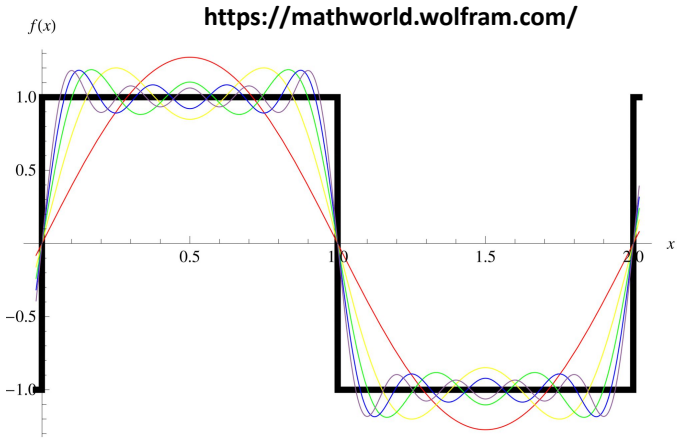
Hanning smoothing

Not for spectral line data!

Gibbs phenomenon

➔ Oscillatory behaviour of Fourier series at a discontinuity

➔ “Ringing” pattern spreading to channels neighbouring the strong narrow RFI spike

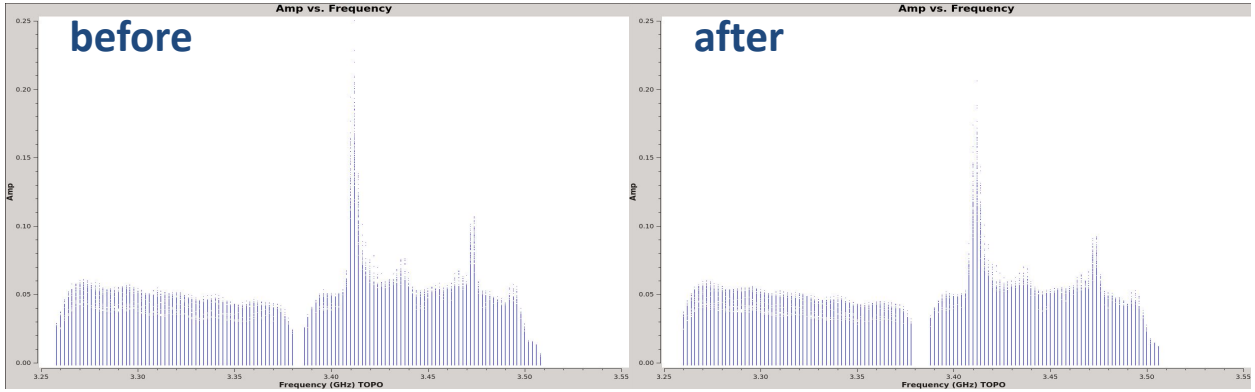


hanningsmooth ()

- removes amplitude spikes and reduce the ringing, reducing number of affected channels
- reduces the spectral resolution by a factor of ~2
- doubles the disk space requirement (new MS)

```
hanningsmooth(vis = 'drwRFI.ms', outputvis = 'drwRFI_hansm.ms')
```

Our data:

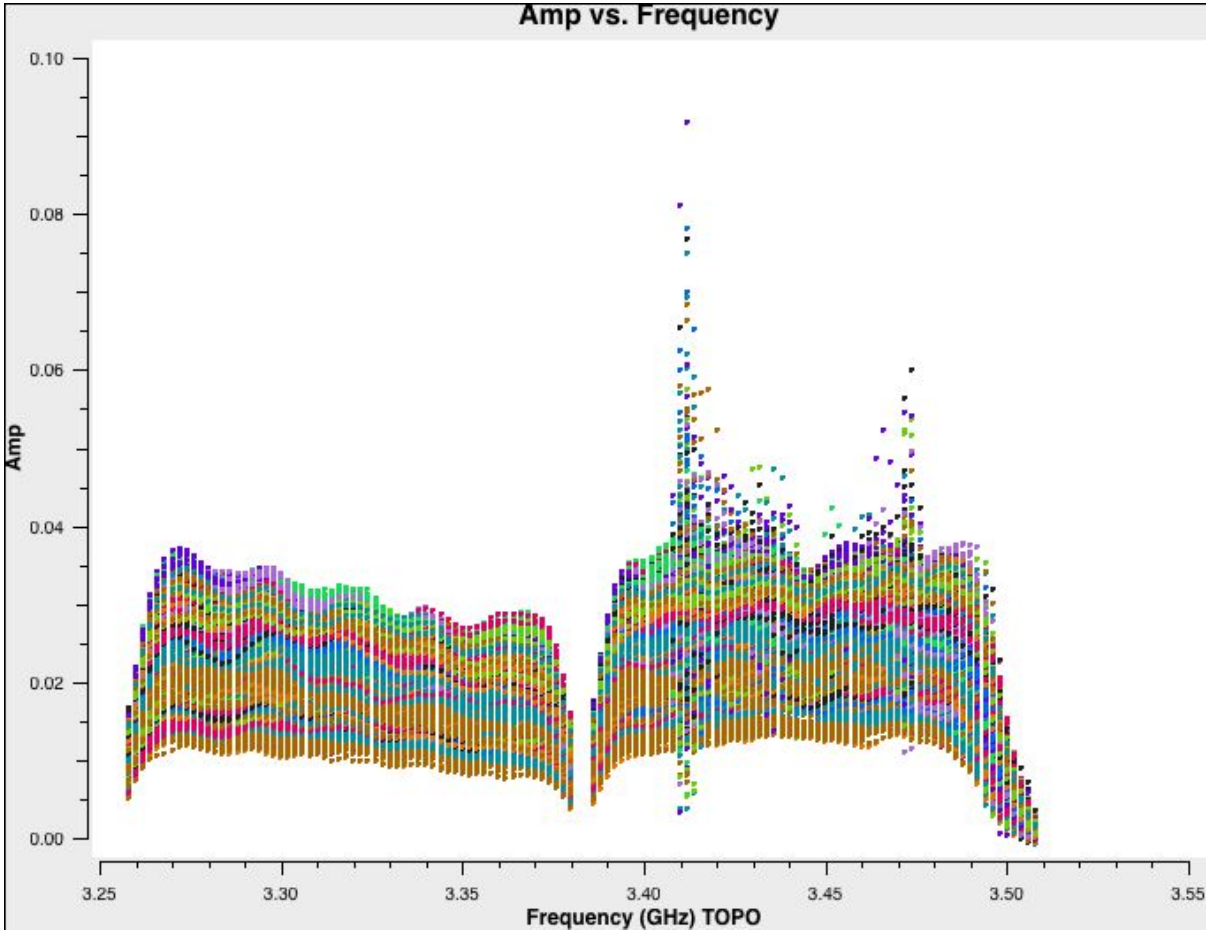


Manual flagging

Interactive

plotms () → to see where the RFI is in our current dataset

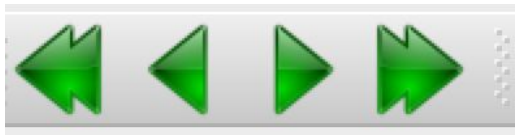
```
default plotms
vis = 'drwRFI.ms'
xaxis= 'frequency'
yaxis='amp'
avgtime = '1e4'
coloraxis = 'Antenna1'
customsymbol=True
symbolshape='circle'
symbolsize=2
inp
go
```



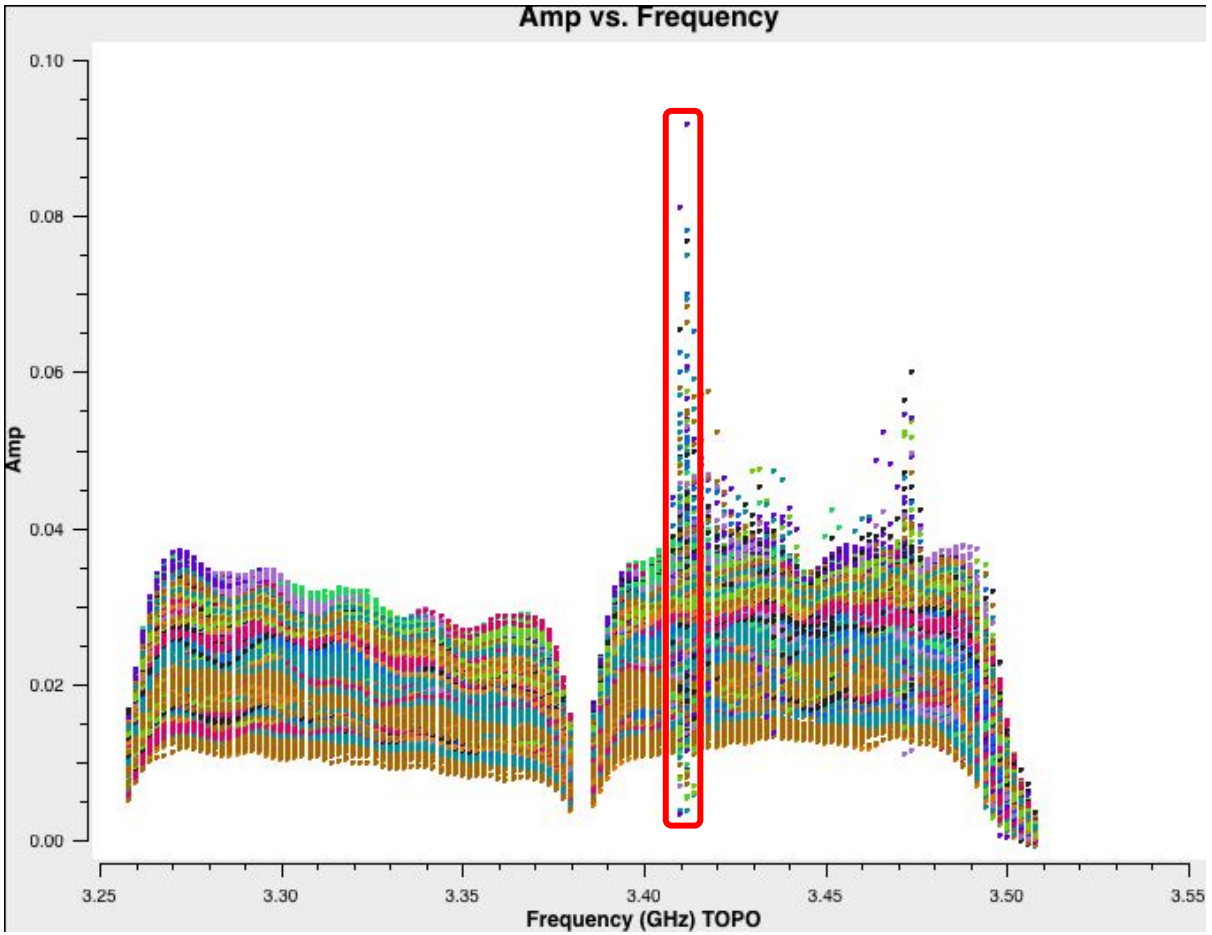
Manual flagging

plotms () → to see where the RFI is in our current dataset

- Inspecting all antennas (paging through) one by one with the parameter : **iteraxis = 'antenna'**



Spike: spw=1, channels=13~15



Manual flagging

Interactive

flagdata () task with mode = 'manual'

Once the RFI is located, the best is to use flagging tasks outside the **plotms ()**.
This way we can backup the flags, and revert steps if needed.

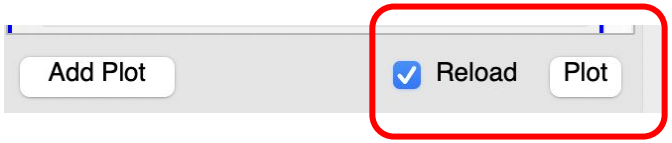
```
# In CASA
default flagdata
vis='drwRFI.ms'
mode='manual'
spw='1:13~15'
flagbackup=True          # required if to restore previous flagging versions
inp
go
```

Manual flagging

Interactive

To see the effect of our flagging get back to **plotms ()** and inspect the data again.

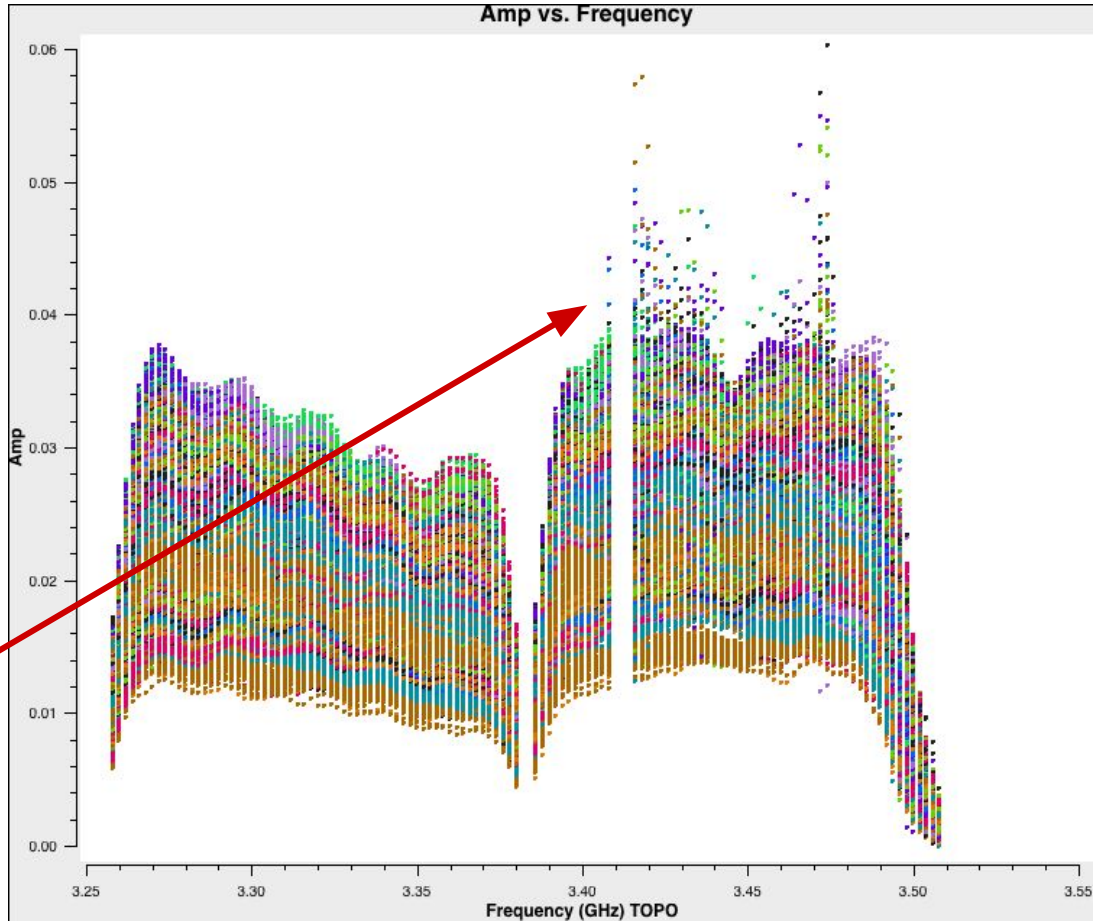
- If **plotms ()** is still open:



- If not:

```
tget plotms
go
```

Spike is gone!



Flagging manager

Interactive

flagdata ()

```
# In CASA
default flagdata
vis='drwRFI.ms'
mode='manual'
spw='1:13~15'
flagbackup=True        # required if to restore previous flagging versions
inp
go
```

```
# In CASA
default flagmanager
vis='drwRFI.ms'
mode='save'
versionname='after_manual'
comment='after manual flagging'
inp

go
```

flagmanager ()

Manual save of the flag state at any time

Flagging manager

Interactive

If you run both the **flagmanager()** task to save the flags, and the **flagdata()** task with **flagbackup=True**, you should have two files now within the MS that contain the flagging done so far.

```
CASA <35>: ls -lstr drwRFI.ms.flagversions/
total 8
0 drwxr-xr-x  8 etremou  staff  256 Oct  8 10:35 flags.flagdata_1/
8 -rw-r--r--  1 etremou  staff   88 Oct  8 10:58 FLAG_VERSION_LIST
0 drwxr-xr-x  8 etremou  staff  256 Oct  8 10:58 flags.after_manual/

CASA <36>:
```

flagmanager() task can be also used to access information in these files and revert flagging

Flagging manager

Restoring flagged data

➔ Flagging with **plotms()**: data can't be restored!

➔ Using **flagdata()** to restore the data with **mode = 'unflag'** :
It will unflag everything, not only the last execution!

➔ **flagmanager()** may be a better tool!

Flagging manager

Interactive

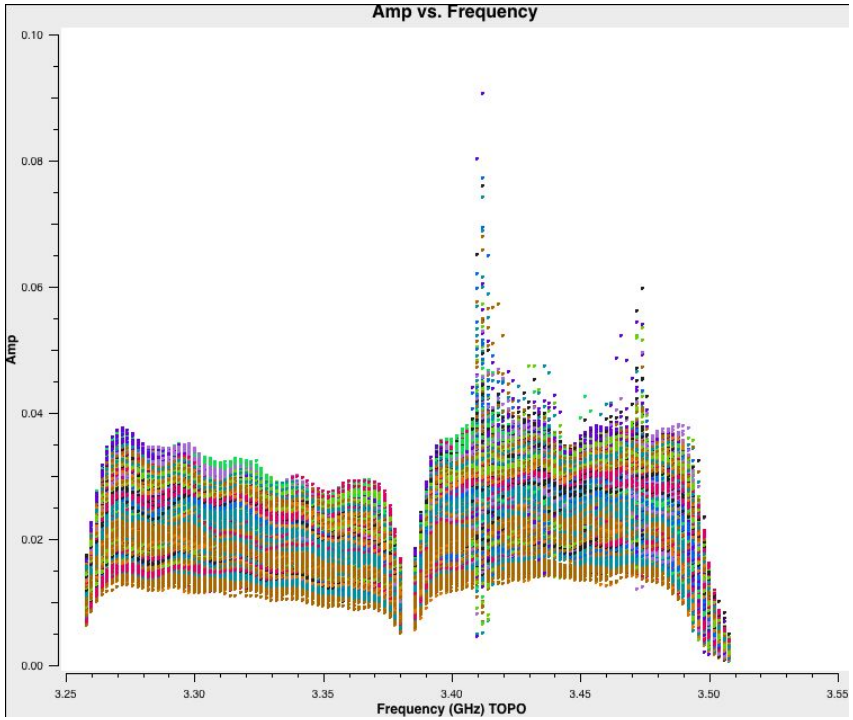
flagmanager () task with mode = 'restore'

```
# In CASA
default flagmanager
vis='drwRFI.ms'
mode='restore'
versionname='flagdata_1'
inp

go
```

Check the result:

```
tget plotms
go
```



Flagging manager

Interactive

flagmanager () task with mode = 'list'

```
# In CASA
default flagmanager
vis='drwRFI.ms'
mode='list'
inp

go
```

The previous flag tables won't be removed by the restore mode, mode='delete' will do that!

Check the result in CASA logger window:

```
2022-10-08 17:26:49  INFO          …ger::::casa  ##########################################
2022-10-08 17:26:49  INFO          …ger::::casa  ##### Begin Task: flagmanager        #####
2022-10-08 17:26:49  INFO          …ger::::casa  flagmanager( vis='drwRFI.ms/', mode='list', versionname='', oldname='', comment='', merge='repla
2022-10-08 17:26:49  INFO          …agger::open  Table type is Measurement Set
2022-10-08 17:26:50  INFO          …tflagger::
2022-10-08 17:26:50  INFO
2022-10-08 17:26:50  INFO          …tflagger::   main : working copy in main table
2022-10-08 17:26:50  INFO          …tflagger::   flagdata_1 : Flags autosave on 2022-10-08 10:35:15
2022-10-08 17:26:50  INFO          …tflagger::   after_manual : after manual flagging
2022-10-08 17:26:50  INFO          …ger::::casa  Task flagmanager complete. Start time: 2022-10-08 11:26:49.481759 End time: 2022-10-08 11:26:49.
2022-10-08 17:26:50  INFO          …ger::::casa  ##### End Task: flagmanager        #####
2022-10-08 17:26:50  INFO          …ger::::casa  ##########################################
```

Flagging statistics

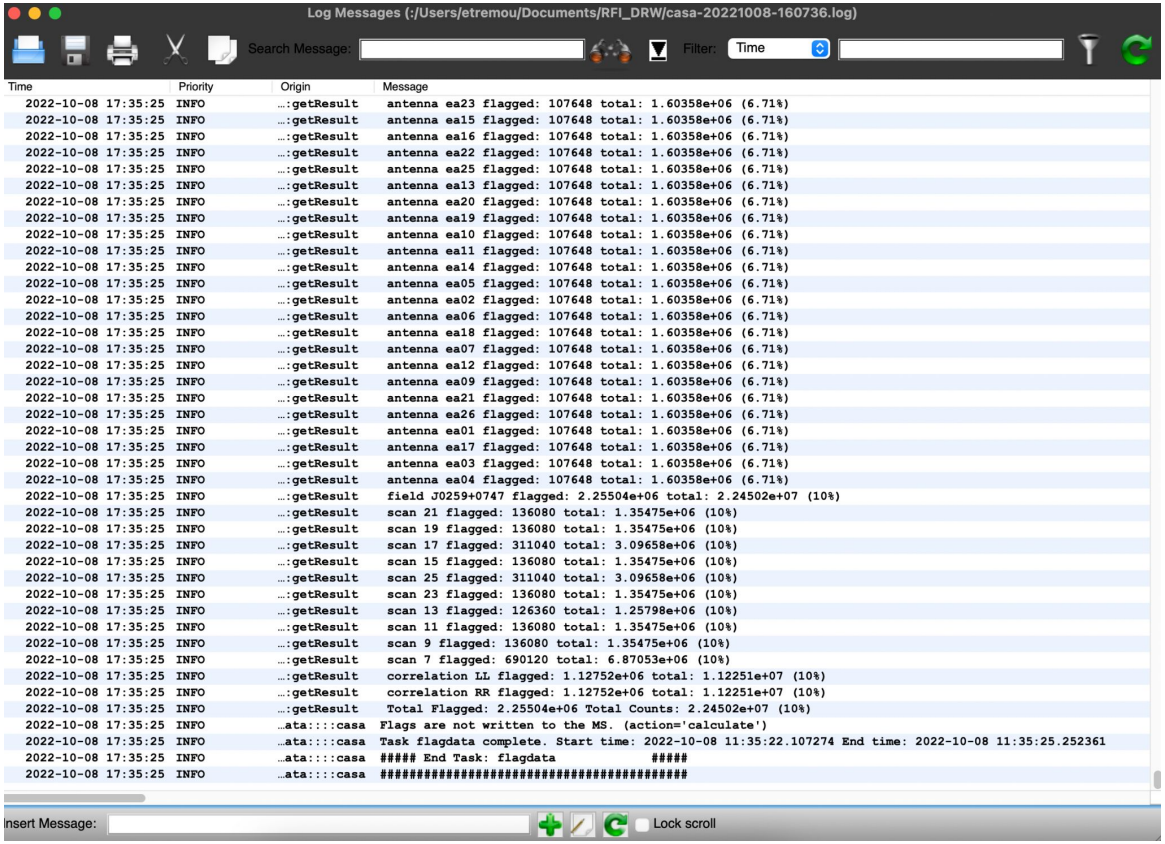
Interactive

flagdata () task with mode = 'summary'

```
# In CASA
default flagdata
vis='drwRFI.ms'
mode='summary'
inp

go
```

CASA logger output:



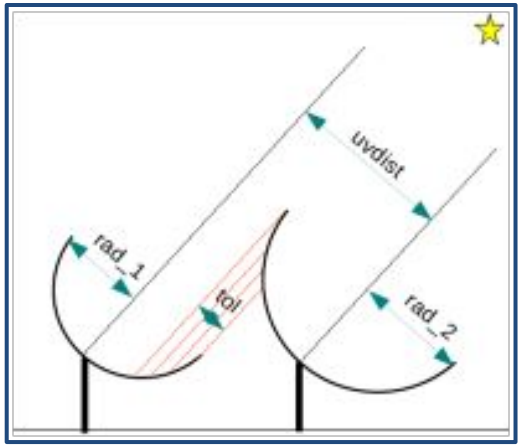
Deterministic flagging modes

flagdata () task with mode = 'quack'

- ➔ **quack** mode is used to remove data at scan boundaries
- ➔ antennas have just been slewing to new source, the slewing is flagged (online flags), but often they need “settling” time
- ➔ **quackinterval** and **quackmode** parameters available

flagdata () task with mode = 'shadow'

- ➔ **shadow** mode is used when one antenna blocks part of the aperture of a second antenna that is behind the first one.
- ➔ Compact configurations may be affected (e.g. D-configuration)



Auto-flagging: TFCrop

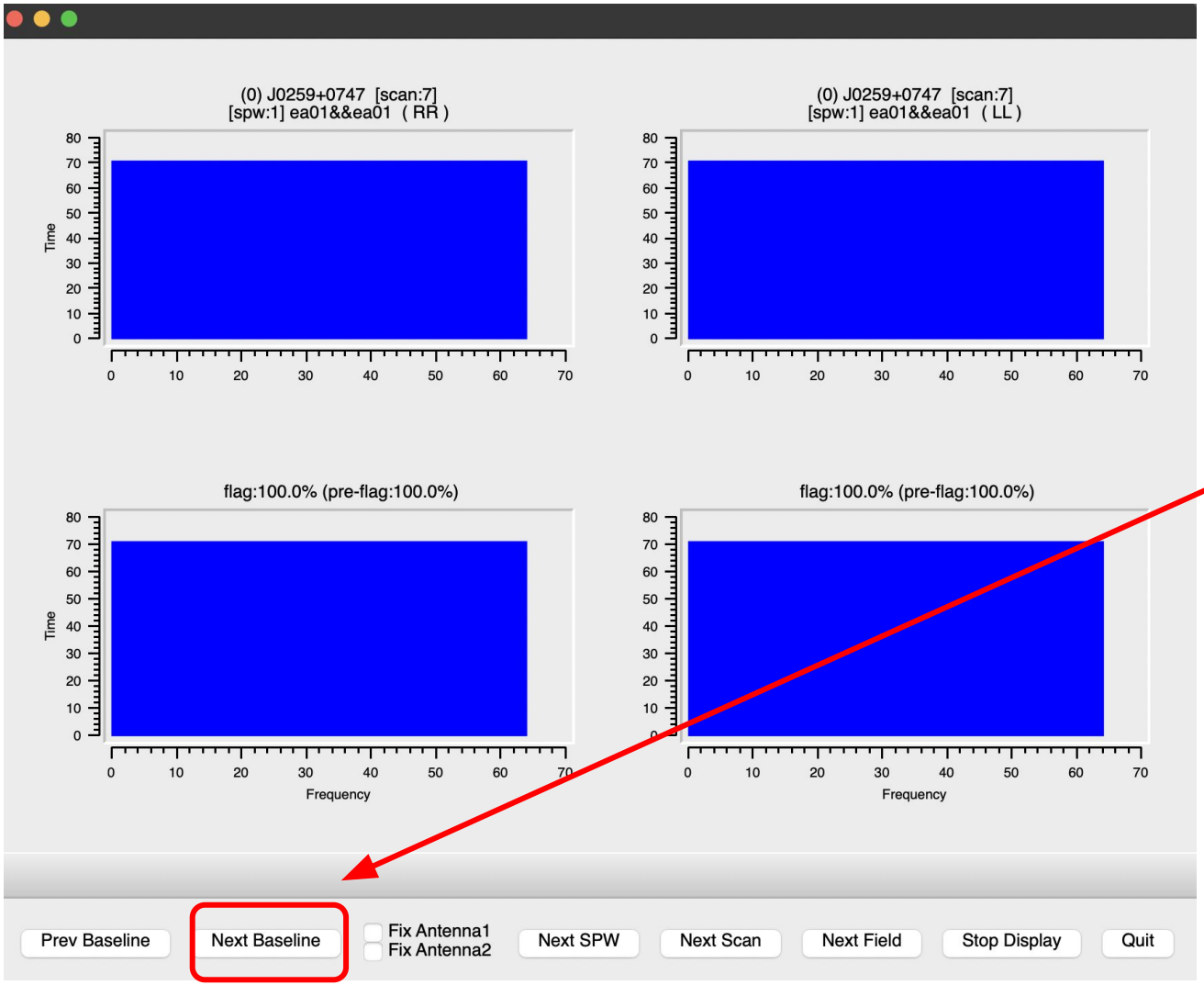
Interactive

- **TFCrop** is an auto-flag algorithm that detects outliers on the 2D time - frequency plane
- It can operate on un-calibrated data (non-bandpass corrected)
- Statistics based on each baseline, correlation
- Its default parameters are optimised for strong narrow band RFI

```
# In CASA
tget flagdata
mode='tfcrop'
spw='1'
timecutoff=3.0          # threshold in units of deviations from the fit
action = 'calculate'
display = 'both'
flagbackup=True         # required if to restore previous flagging versions
inp
go
```

Auto-flagging: TFCrop

Interactive



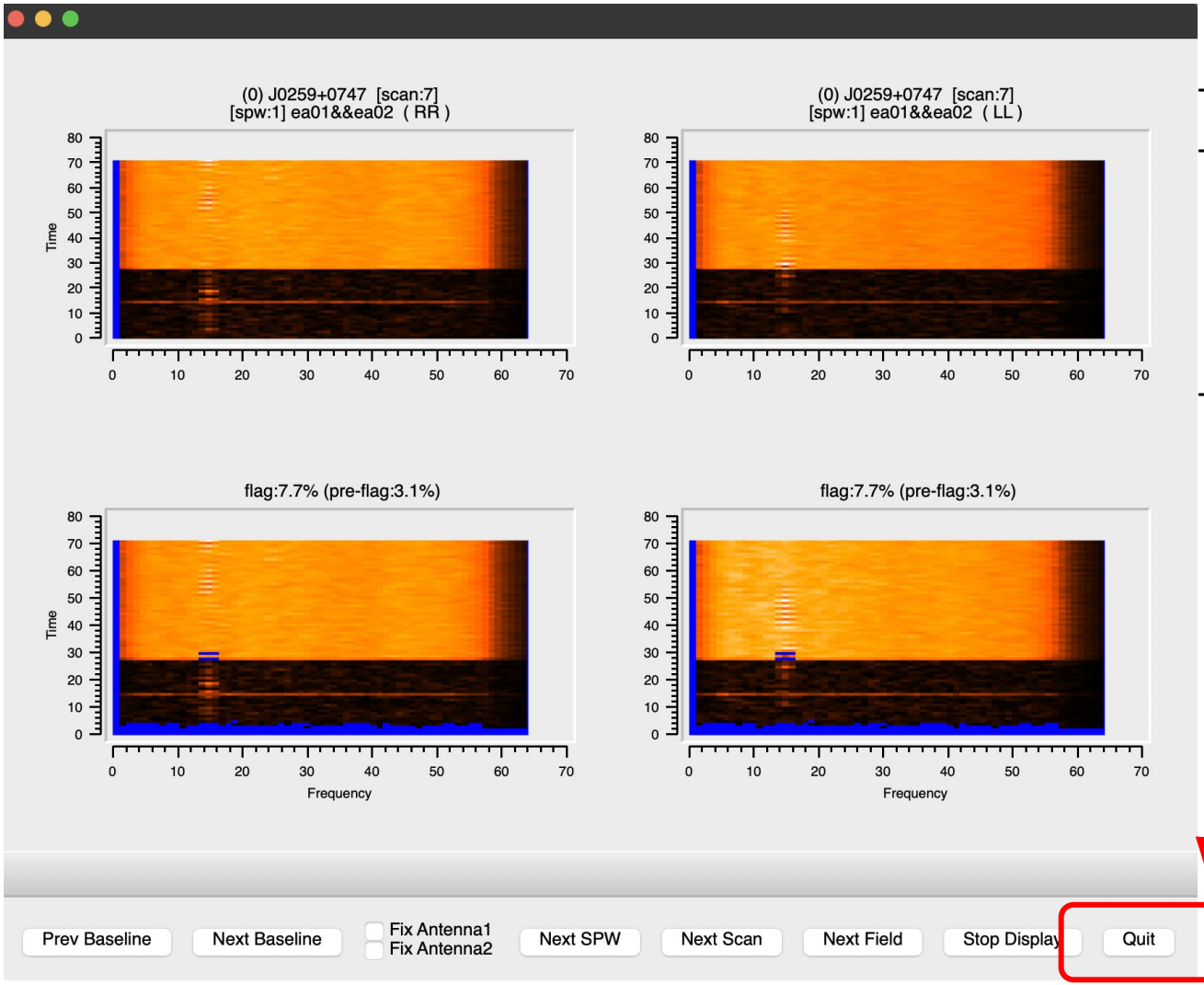
Blue shows the flagged data

Navigate through different baselines

Auto-flagging: TFCrop

Interactive

After TFCrop flagging *Current flagging status*



- *Quit* kills the task
- *Stop Display* will just turn off display and let flagdata() to finish.
- But we are in action ='calculate', either ways will work.

Auto-flagging: TFCrop

- Tuning parameters may give the best result:

timecutoff, freqcutoff	# threshold for finding outliers, in units of fit st.dev.
flagdimension = 'freqtime'	# direction(s) in which to calculate statistics
channelavg, timeavg	# pre-average the data
timefit = 'line'	# fitting function along time axis, line is default (ok: poly/line)
freqfit = 'poly'	# fitting function along freq axis, poly is default (ok: poly/line)
maxnpieces = n	# n order of polynomial in fitting functions above

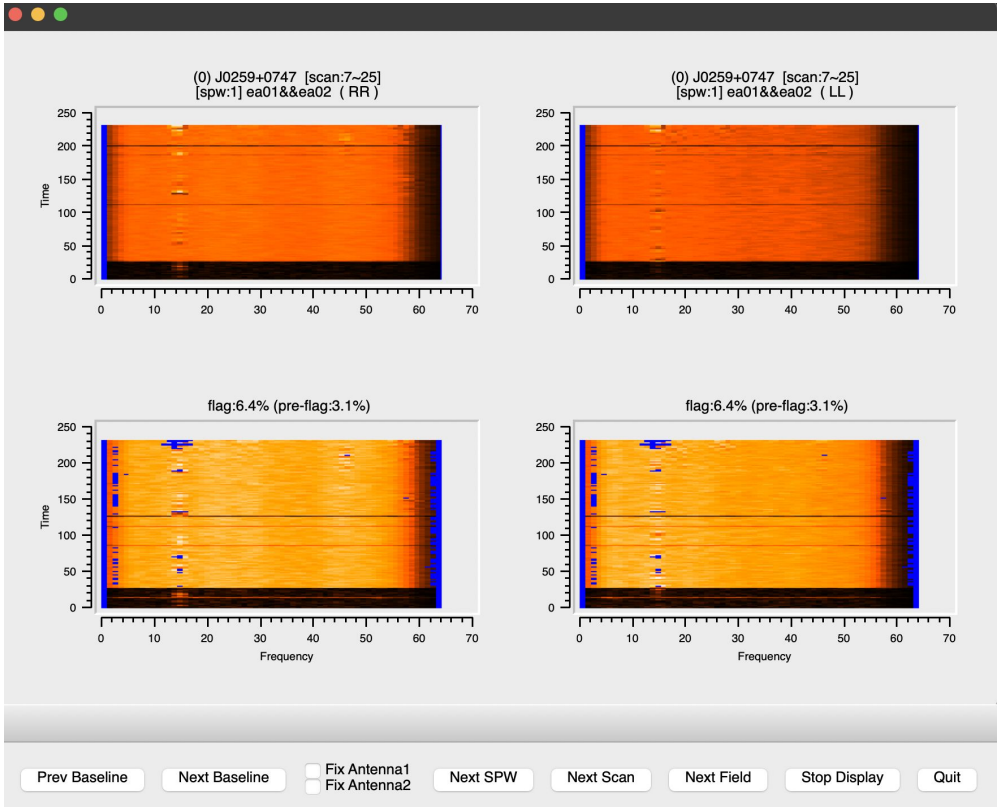
- Sometime you may also need to vary the parameters for e.g. spws or bands within the same data set.
- Each data set is different and may need different parameter set up for best results
  → make sure you inspect your data well
  → know what you are dealing with
  → choose the parameters accordingly.
- **tfcrop** can be run multiple times on the same data

Auto-flagging: TFCrop

Interactive

Let's try some tuning:

```
# In CASA
tget flagdata
maxnpieces=4
usewindowstats='std'
halfwin=2
combinescans=True
inp
go
```



Auto-flagging: TFCrop

Interactive

Once you are happy with the flagging, apply it:

```
# In CASA
tget flagdata
action='apply'
display=''
inp

go
```

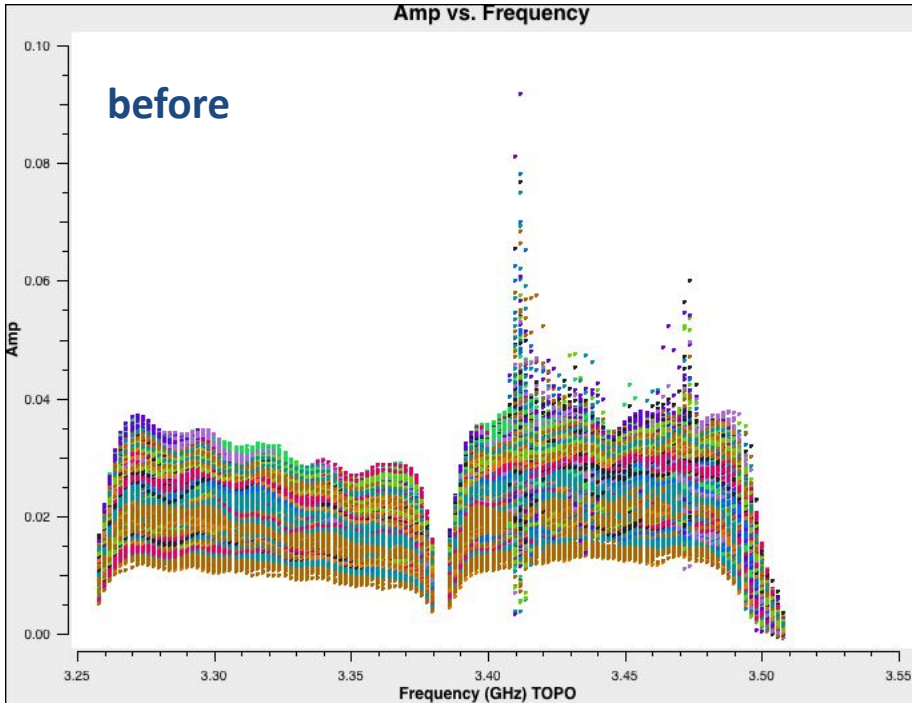
Auto-flagging: Inspect results

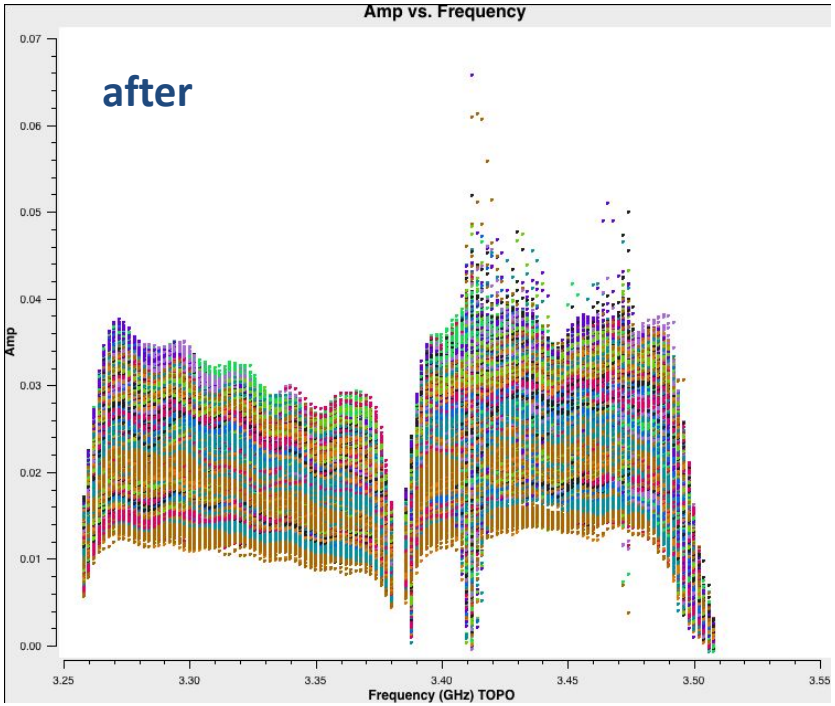
Interactive

plotms () for inspection:

```
# In CASA
tget plotms
inp

go
```





Extending the flags

- standalone mode in **flagdata()**
- as an extend flags parameter within **tfcrop** and **rflag** modes

➔ It will extend or grow flags accumulated in the MS file along time, frequency, polarisation, baseline etc

- e.g. if you applied flag only to RR product, you can extend that afterwards also to LL.

➔ Flag growing, example parameters:

growtime=80.0	# for each time flag entire timerange if >80% data flagged
growfreq=92.0	# for each channel flag selected chans if >92% data flagged
growaround	# flag a data point if >4 neighbouring points are flagged
flagneartime	# flag a data point before and after a flagged one

- It is recommended when executing auto-flagging modes.

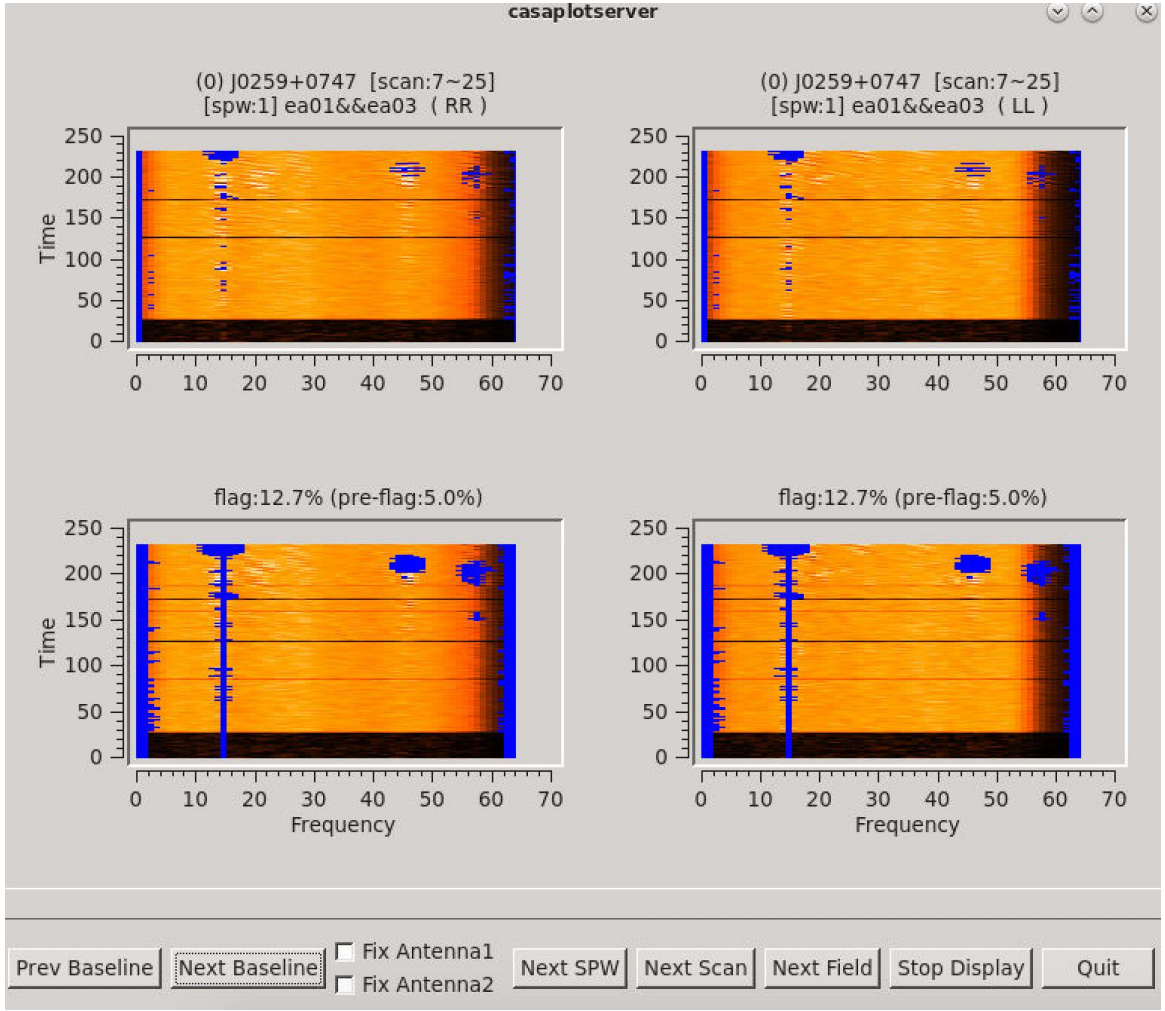
Extending the flags

Interactive

```
# In CASA
tget flagdata
mode='extend'
combinescans=True
growtime=30
growfreq = 30
growaround=True
flagneartime = True
flagnearfreq = True
action = 'calculate'
display = 'both'
inp

go
```

Extending the flags



Baseline: ea01&ea03

SPW: 2

RFI removed!

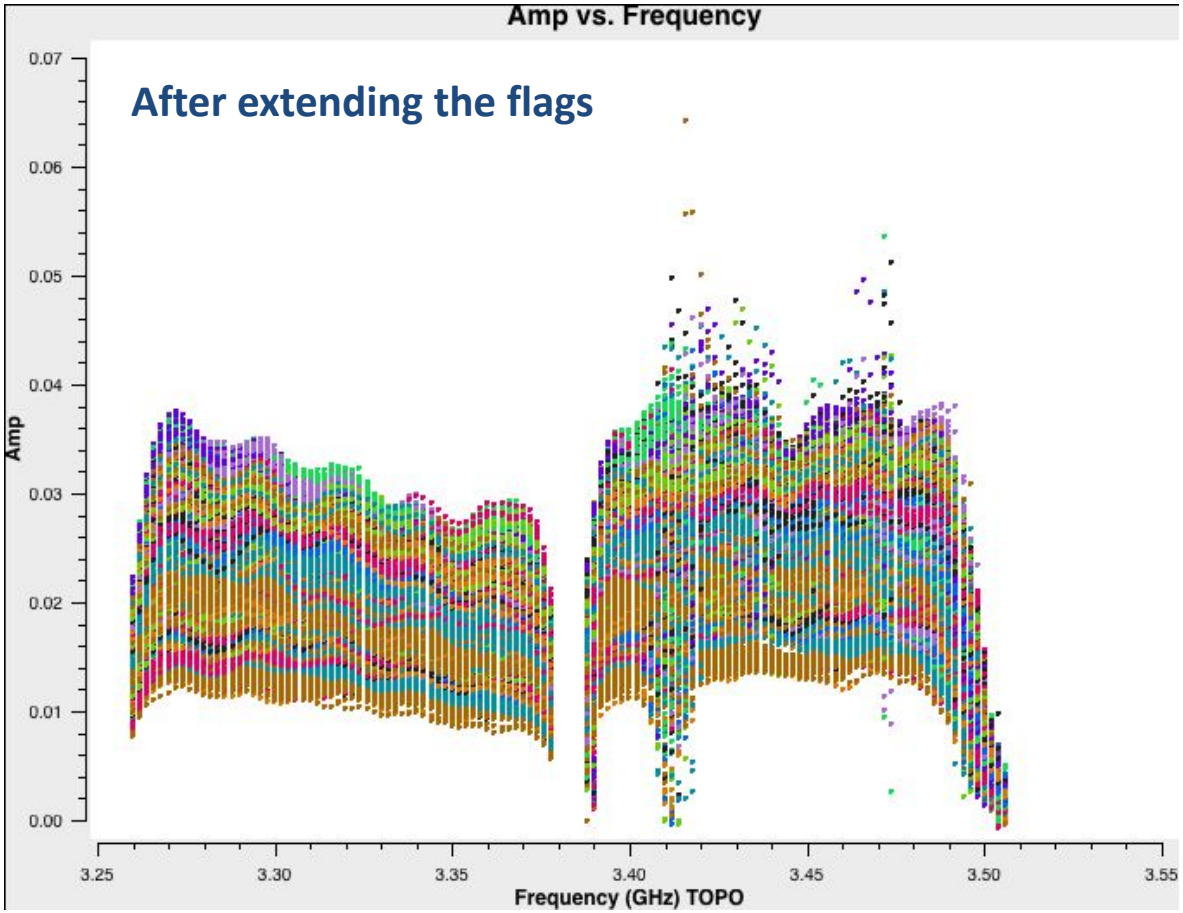
Let's apply them!

Extending the flags

Interactive

```
# In CASA
tget flagdata
action = 'apply'
display = ''
inp
go

tget plotms
inp
go
```



Auto-flagging: Rflag

- **Rflag** is an auto-flag algorithm that detects outliers based on local rms statistics
- It requires calibrated data
- It iterates through time chunks calculating local rms of *imag* and *real* visibilities within a sliding time window, and deriving a median rms across given time window
- It iterates through frequency chunks (channels) for each time chunk calculating rms of avg of *imag* and *real* visibilities
- The VLA calibration uses **Rflag**
- Tuning parameters may give the best result:

winsize	**#** number of timesteps in the sliding time window
timedevscale, freqdevscale	# st.dev. threshold for outlier flagging
channelavg, timeavg	# pre-average the data

TFCrop vs Rflag: which one to use when?

Worth executing both in the same data set

	TFCrop	Rflag
How does it work?	→ search for RFI spikes above smooth base, per baseline	→ use local vs global stats to find outliers
Strong, spiky RFI	Great!	Good, but continuous RFI in time/freq needs tuning
Noisy RFI	Good only for bright RFI, won't work well for low noisy RFI spikes	Great!
Broadband RFI	Not robust, but possible with some tuning	Good for noisy RFI. Continuous RFI needs tuning.
Extended emission	Great! [each baseline treated separately]	Not great, biased by high flux density on short baselines
Raw, uncalibrated data	**Yes**	**No**
Calibrated data	**Yes**	**Yes**

Auto-flagging: Spectral lines

- **TFCrop and Rflag** may remove precious spectral lines in auto-flagging.

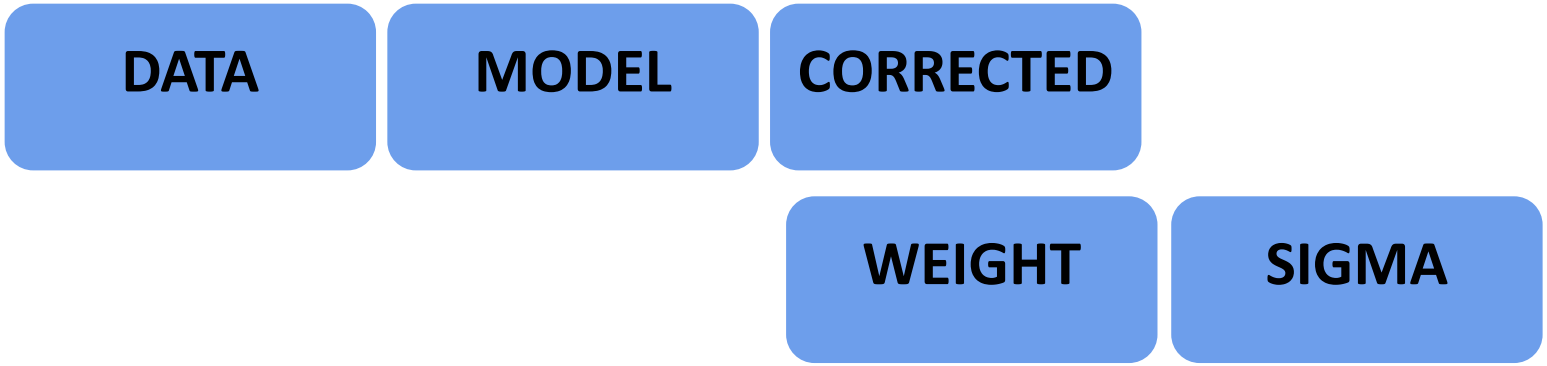
- If the spectral lines are not known, then auto-flagging is not recommended.

- Adjust the parameters in advance by excluding the location of spw/channels of the interesting spectral lines.

  Example:

  - you have 2 spectral windows (0,1), each with 64 channels
  - your line is in spw=1, channels=21~22
  - exclude that location with the following format of the spw parameter: **spw = '0, 1:0~20; 23~63'**

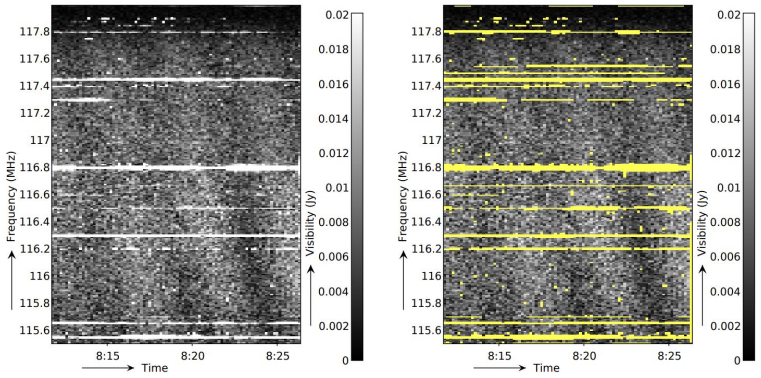
A note on statwt() task



- WEIGHT and SIGMA columns are set to some arbitrary values (e.g. 1).
- **statwt()** will empirically measure the visibility scatter (e.g. as a function of time, antenna, and/or baseline) and use it to set WEIGHT and SIGMA
- it may be beneficial sometimes to weight down any remaining RFI in your data with **statwt()** prior to imaging
- It requires calibrated visibilities

A note on other flagging algorithms

- **AOFlagger**, https://aoflagger.readthedocs.io/en/latest/
  - It can make use of Lua scripts to make flagging strategies flexible
  - Applicable to a wide set of telescopes especially in low frequency arrays (initially developed for LOFAR)
  - Accompanied with python scriptable plotting tools



Offringa et al. 2012

- **Tricolour** (**Hugo et al. 2021**), https://github.com/ratt-ru/tricolour
  - Optimised for channelized wideband data (e.g. MeerKAT)
  - Configurable, parallel and optimized (Dask and Numba)

Summary

- All data have a level of RFI and it will only get worse over time.
- Automatic Flagging options exist.
- They all need tuning. Usually, one setup per SPW or band
  - Look at small pieces of your data, and decide flagging strategy
  - Use plotms or viewer or flagdata (action='calculate', display='both') and try different flagging setups.
  - Defaults will not suffice for all cases, experiment with various parameters.
- All types of data can be flagged (e.g. visibilities, weights, calibration tables)
- Documentation:

- https://casadocs.readthedocs.io/en/stable/notebooks/data_examination.html

- https://colab.research.google.com/github/casangi/casadocs/blob/2316c9b/docs/notebooks/data_examination.ipynb

www.nrao.edu
science.nrao.edu
public.nrao.edu

The National Radio Astronomy Observatory is a facility of the National Science Foundation operated under cooperative agreement by Associated Universities, Inc.