AOFLAGGER

We got it working… sort of

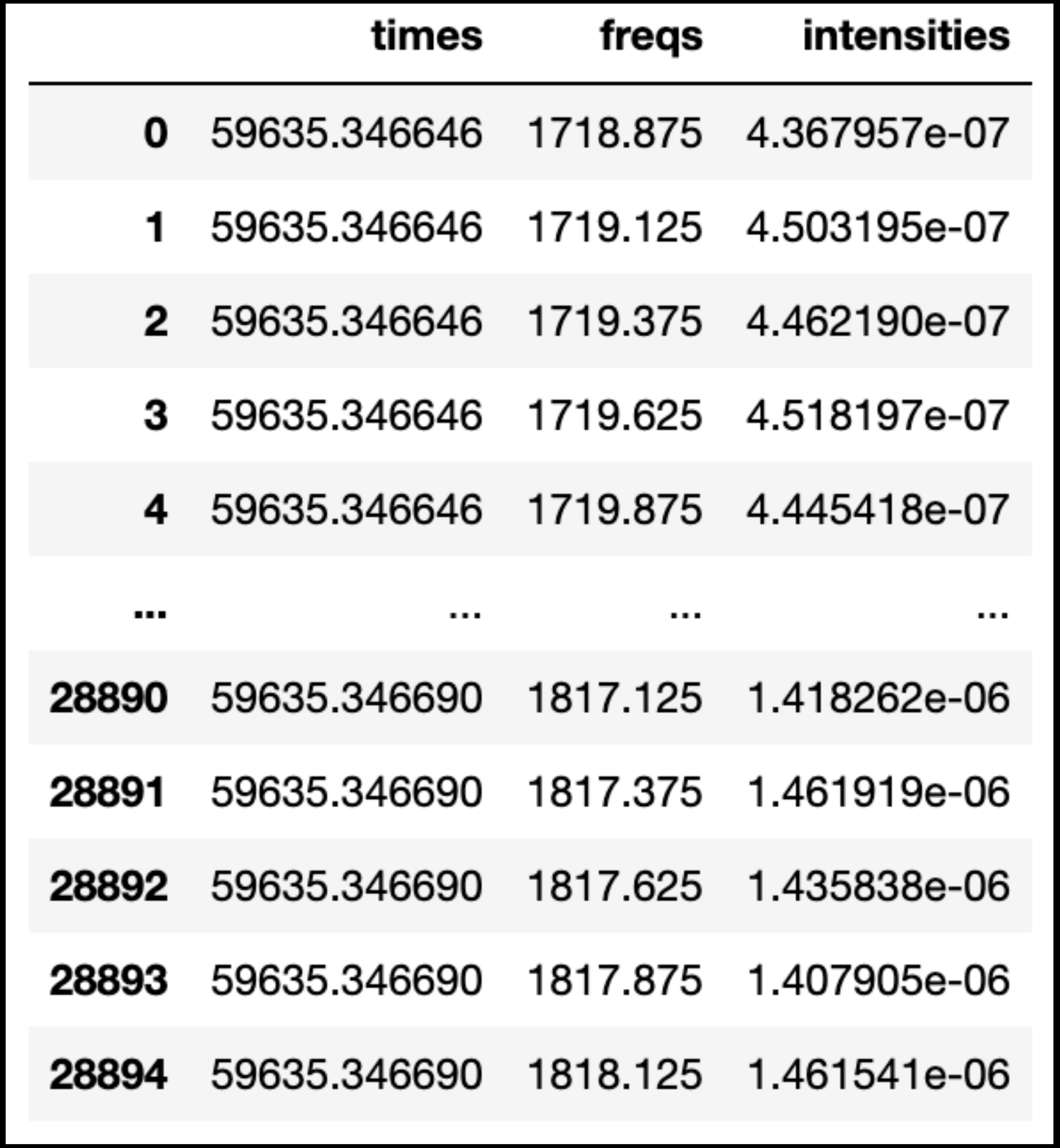
Sofia Sheikh - 04/26/2022

What is AOFLAGGER?

And why are we using it?

- AOFLAGGER is currently the most popular software tool for removing RFI from radio astronomy observations
- Consists of:
    - Algorithm in C (`SumThreshold`)
    - GUI run from terminal (`rfigui`)
    - Strategy files written in "Lua"
    - Python bindings / wrapper
- We do not want to use it for SETI purposes (too aggressive, it will remove our signals as well as the RFI), but we do want to use it to quantify the results from the RFI survey

Output we want:
Table of Flagged RFI from Survey



	times	freqs	intensities
0	59635.346646	1718.875	4.367957e-07
1	59635.346646	1719.125	4.503195e-07
2	59635.346646	1719.375	4.462190e-07
3	59635.346646	1719.625	4.518197e-07
4	59635.346646	1719.875	4.445418e-07
...	...	...	...
28890	59635.346690	1817.125	1.418262e-06
28891	59635.346690	1817.375	1.461919e-06
28892	59635.346690	1817.625	1.435838e-06
28893	59635.346690	1817.875	1.407905e-06
28894	59635.346690	1818.125	1.461541e-06

Expand to include day of week, time of day, direction

AOFLAGGER Workflow

It’s a whole software infrastructure

- The intended workflow of AOFLAGGER is:
  - Use `rfigui` to run a `SumThreshold`-based flagging method on a few example files from a given telescope, receiver, frequency range
  - Interactively tweak parameters to get the flagging right
  - Save that parameter-edited file as a strategy in the Lua language
  - Call that Lua strategy file in an AOFLAGGER Python/C/bash script to execute on large amounts of data
- Strategies exist for e.g., Arecibo, JVLA, Parkes…
- Right now, there’s no ATA strategy - need to use and edit default
- Goal: once we figure it out, have dev add ATA to the package permanently

Designing an optimal flagging algorithm in C

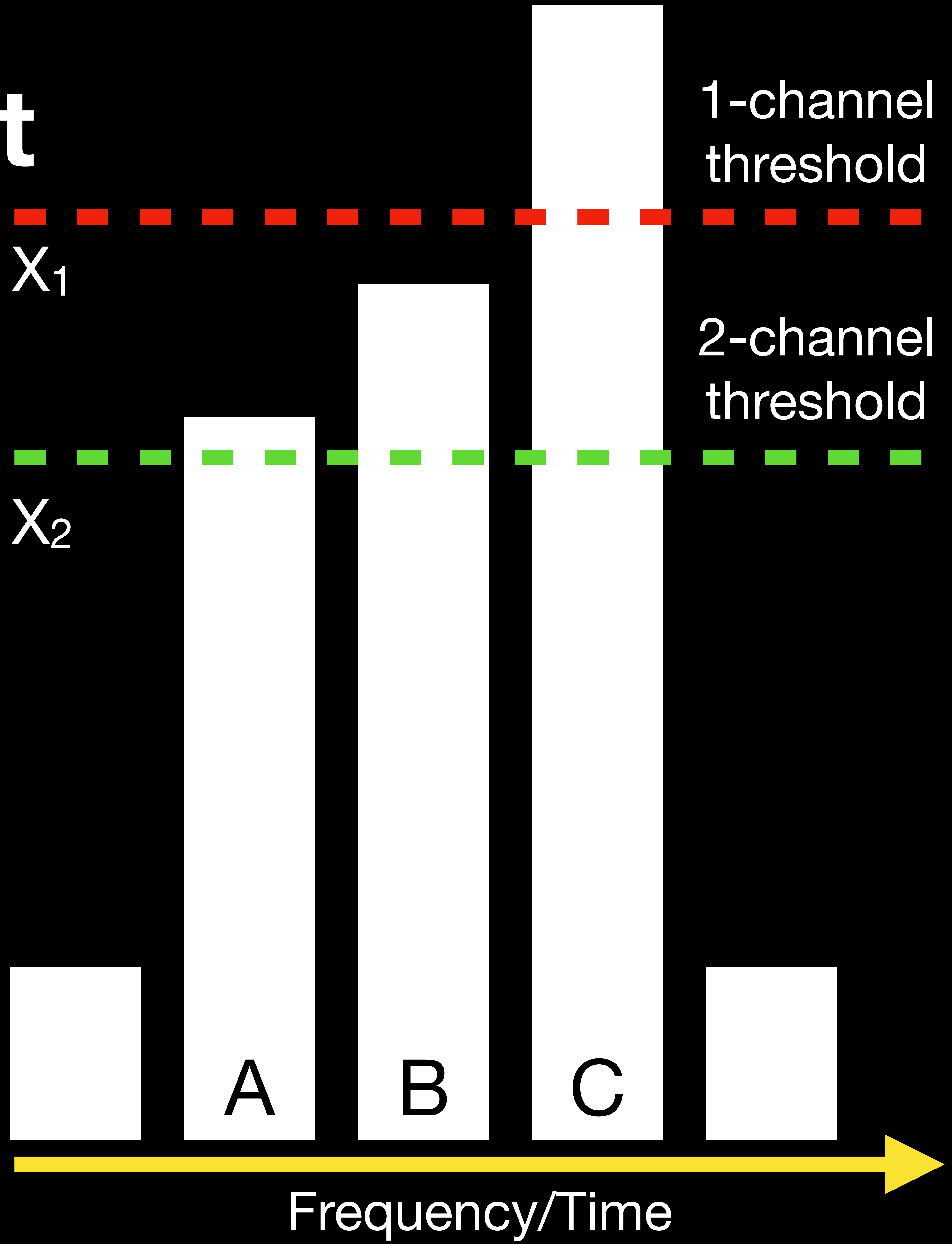
The making of `SumThreshold`

- Goal: detect RFI in time/frequency/antenna space and flag it, such that it is ignored in subsequent processing steps
- Assumption: most interferers are either concentrated in frequency or concentrated in time
  - This leads to RFI contamination of adjacent rows/columns - multiple samples connected in frequency/time
    - Premise: This feature can be leveraged in the flagging algorithm

`VarThreshold`: First attempt

The making of `SumThreshold`

- Introduce *combinatorial thresholding*
  - Use threshold X_1 for single channels
  - Use threshold $X_2 < X_1$ for combinations of 2 channels
  - Etc: for *i* adjacent channels, use threshold X_i
- If any threshold is passed (in time or frequency) the pixel is flagged



VarThreshold: Considerations

The making of SumThreshold

- How many i values (for X_i) should be tried, and up to what maximum?
- How to determine the threshold at each X_i

$$\chi_i = \frac{\chi_1}{\rho^{\log_2 i}}$$

  - Dev finds that $\rho = 1.5$ works well, empirically

- Accidental flagging of false-positives

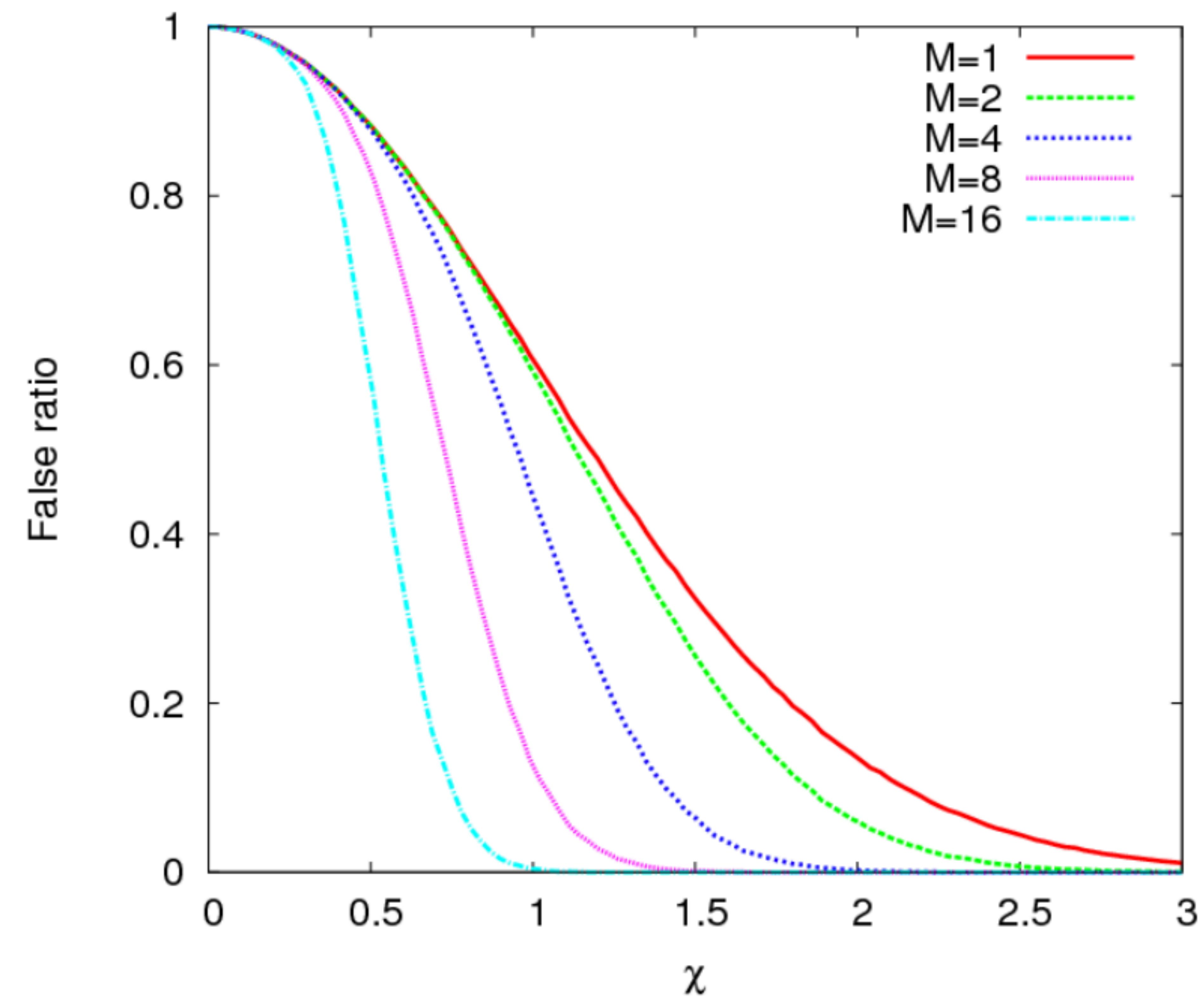


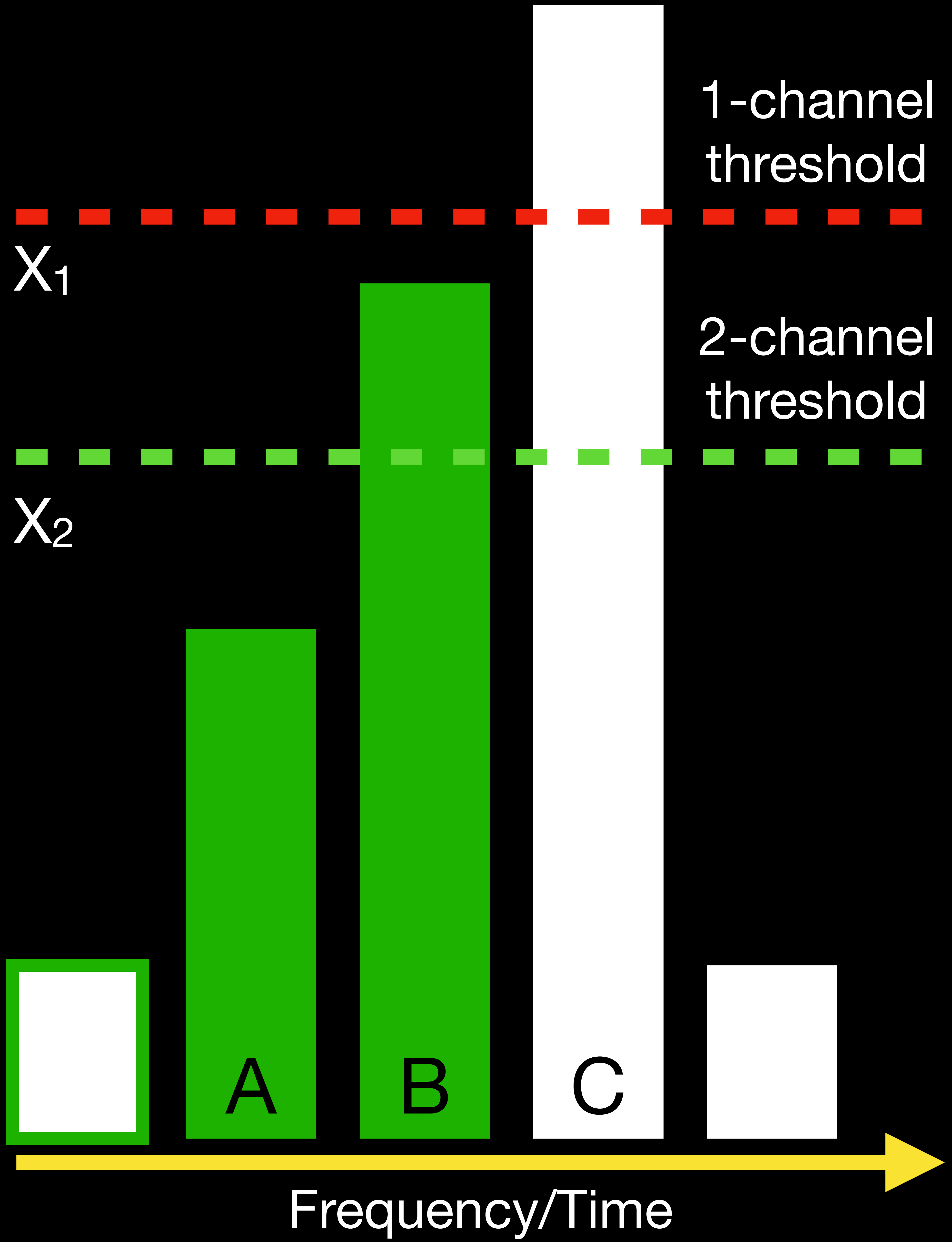
Figure 2. The false-positives of the **VarThreshold** method when flagging with a single combination $\mathcal{M} = \{M\}$ without surface fitting. Samples were selected from a Rayleigh distribution, which is the distribution of the visibility amplitudes. χ is relative to the mode of the distribution.

Offringa et al. 2010

SumThreshold

The heart of AOFLAGGER

- Use the same *combinatorial thresholding* idea
    - But now that threshold is applied to some *combined* property of the channels (mean or median of intensity etc.) so not every channel has to pass the intensity threshold, just the combined property
- Some protections against over-widening the flagging to include adjacent non-RFI bins
- Improved protection against false positives



`rfigui`

Using AOFLAGGER interactively in terminal

- AOFLAGGER consists of strategies that execute the SumThreshold method with different thresholds, iteration numbers, signal widths, etc.
- Can run and edit those strategies interactively in `rfigui`, run from terminal

`rfigui`: Example

rfigui

SumThreshold

- Strategy
  - Set no flags
  - For each polarisation
    - On amplitude
      - Iterate 2 times
        - SumThreshold
        - Combine flag results
          - Frequency selection
          - Time selection
        - Set contaminated = original
        - Change resolution
          - High-pass filter (Gaussian)
        - Visualize fit in UI
        - Visualize residual in UI
      - SumThreshold
      - Visualize residual in UI
  - Plot polarization flag counts
  - Apply flags to all polarisations
  - Morphological flagging
  - Time selection

[✓] In time direction (sensitive to spectral lines)

Time-direction base sensitivity: (low = sensitive)

1.0

[✓] In frequency direction (sensitive to temporal RFI)

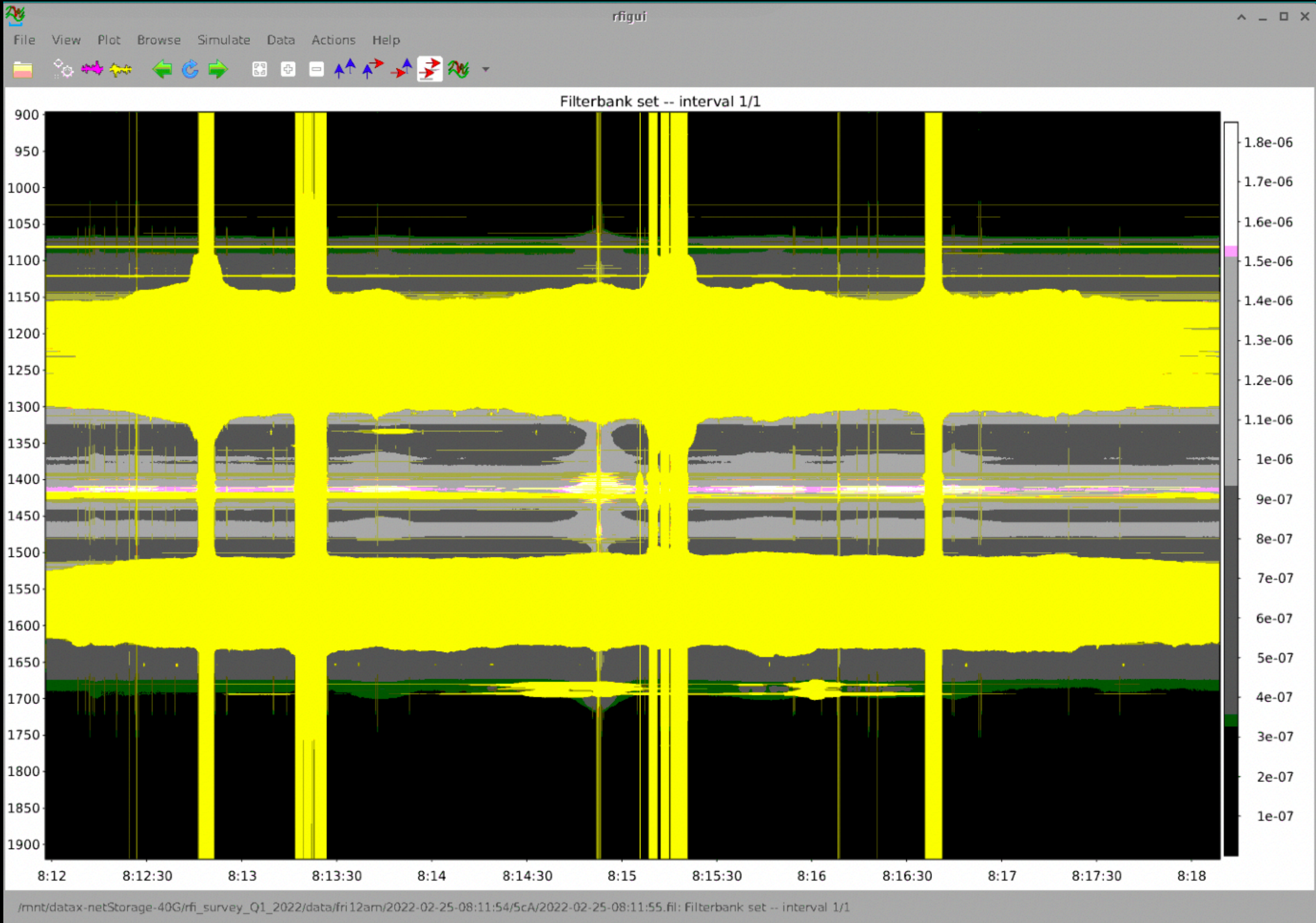
Frequency-direction base sensitivity: (low = sensitive)

1.0

[ ] Exclude original flags

`rfigui`: example with flagging

- Ignore the fact that the flagging is terrible, we'll talk about that soon



/mnt/datax-netStorage-40G/rfi_survey_Q1_2022/data/fri12am/2022-02-25-08:11:54/5cA/2022-02-25-08:11:55.fil: Filterbank set -- interval 1/1

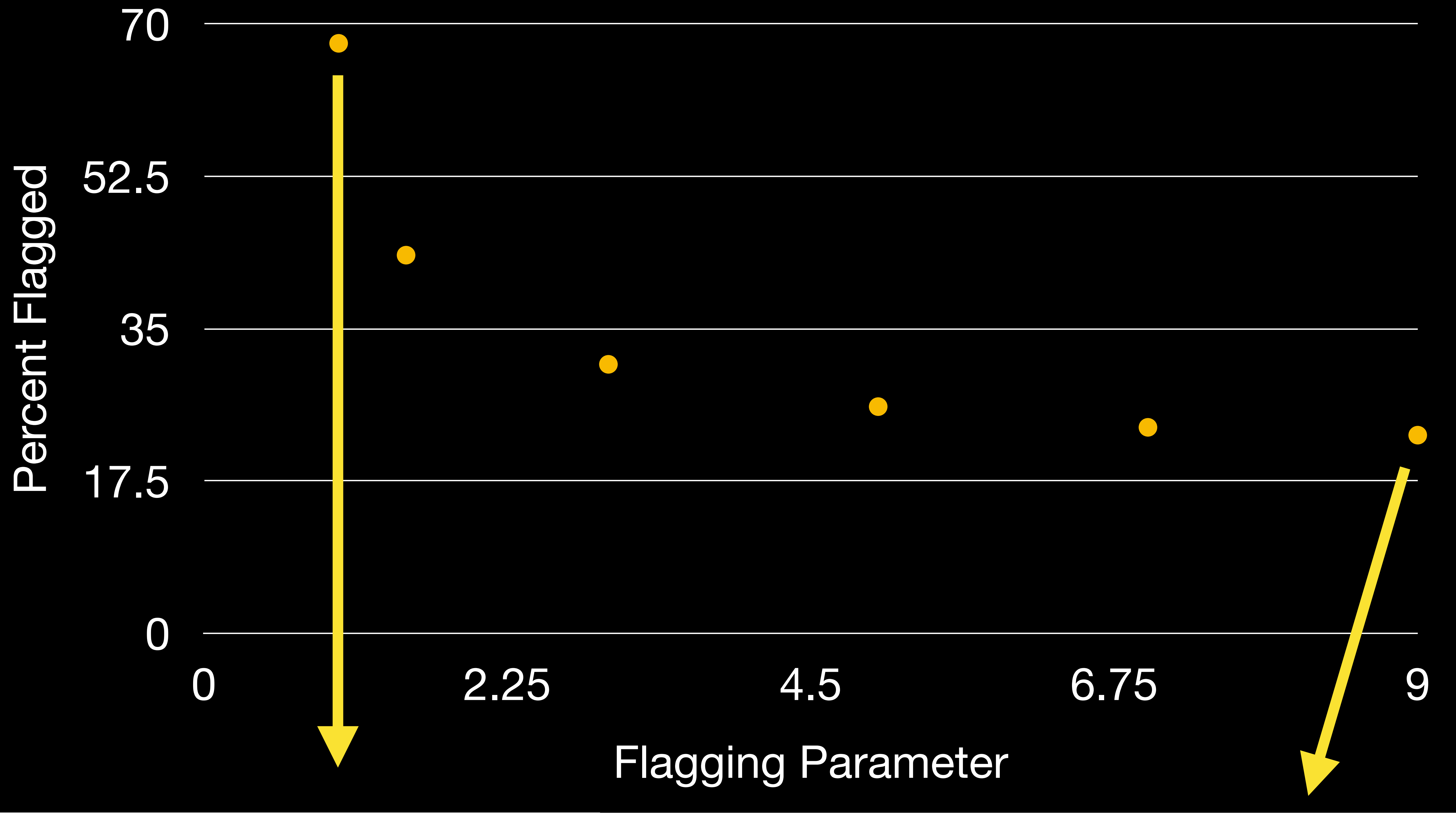
What is AOFLAGGER?

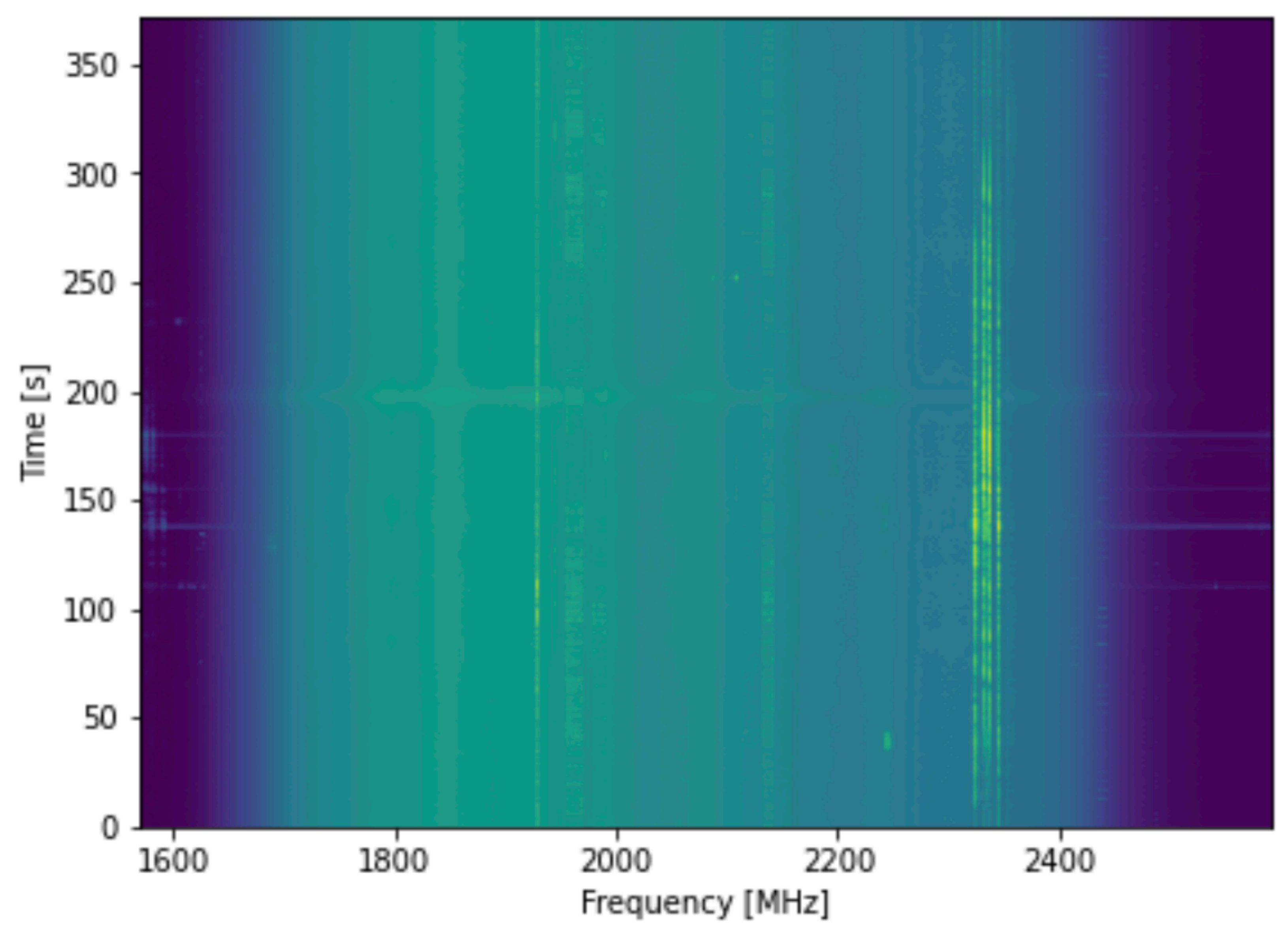
And why is it driving me batty

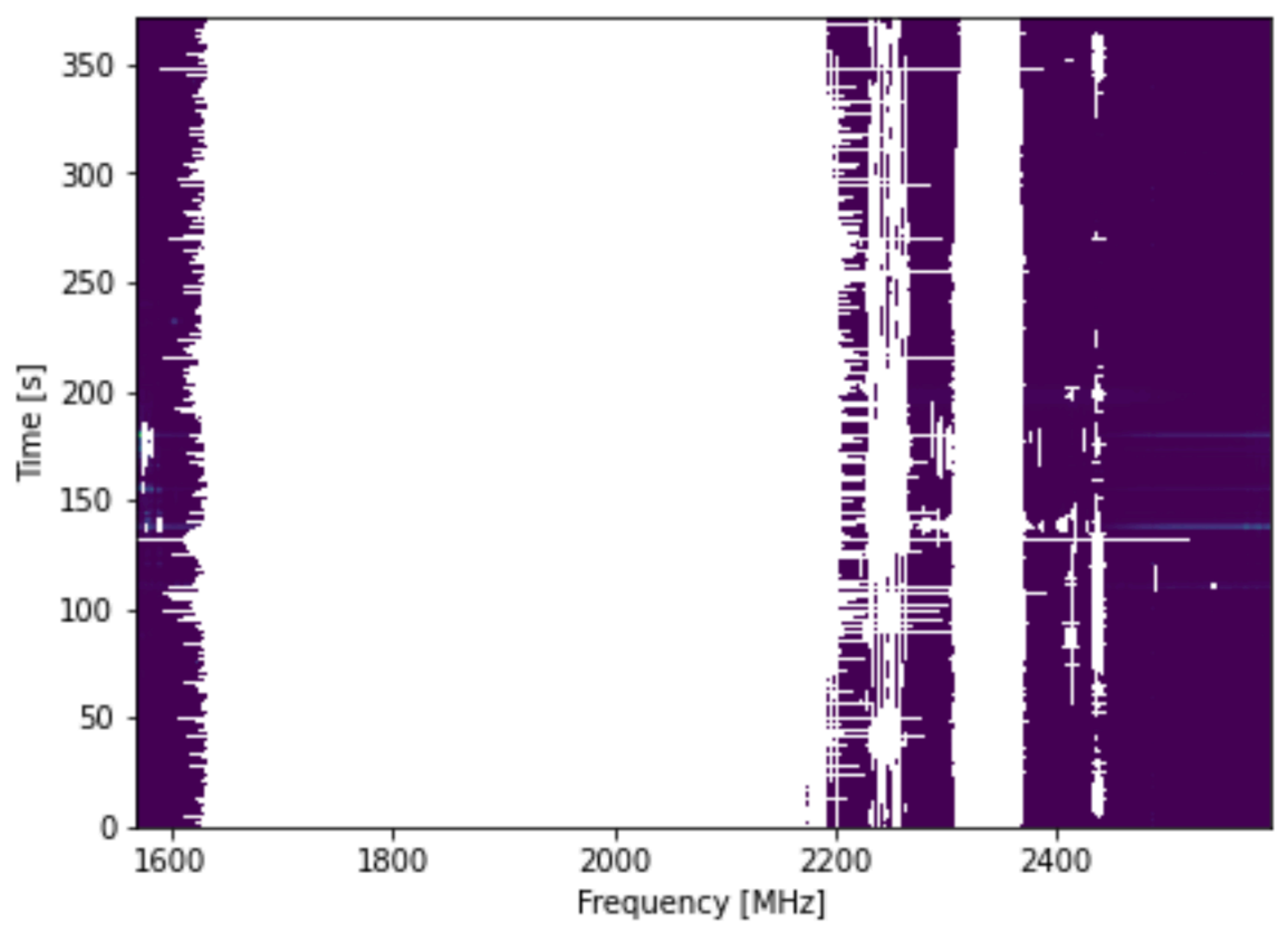
- Side note: it took multiple days of debugging for Wael and I to actually get it installed on a single (sonata) account
  - Probably would not have succeeded without Luigi
- Python utilities are… limited
- `rfigui`: ssh too slow, VNC is faster but colors are wacky
- Workflow: cannot get a `rfigui` output Lua strategy file to work with the Python LoadStrategy() Object (???)
- Default/recommended parameters are over-flagging like crazy - removing the bandpass beforehand (not doable in GUI) helps, but doesn't fully solve the issue

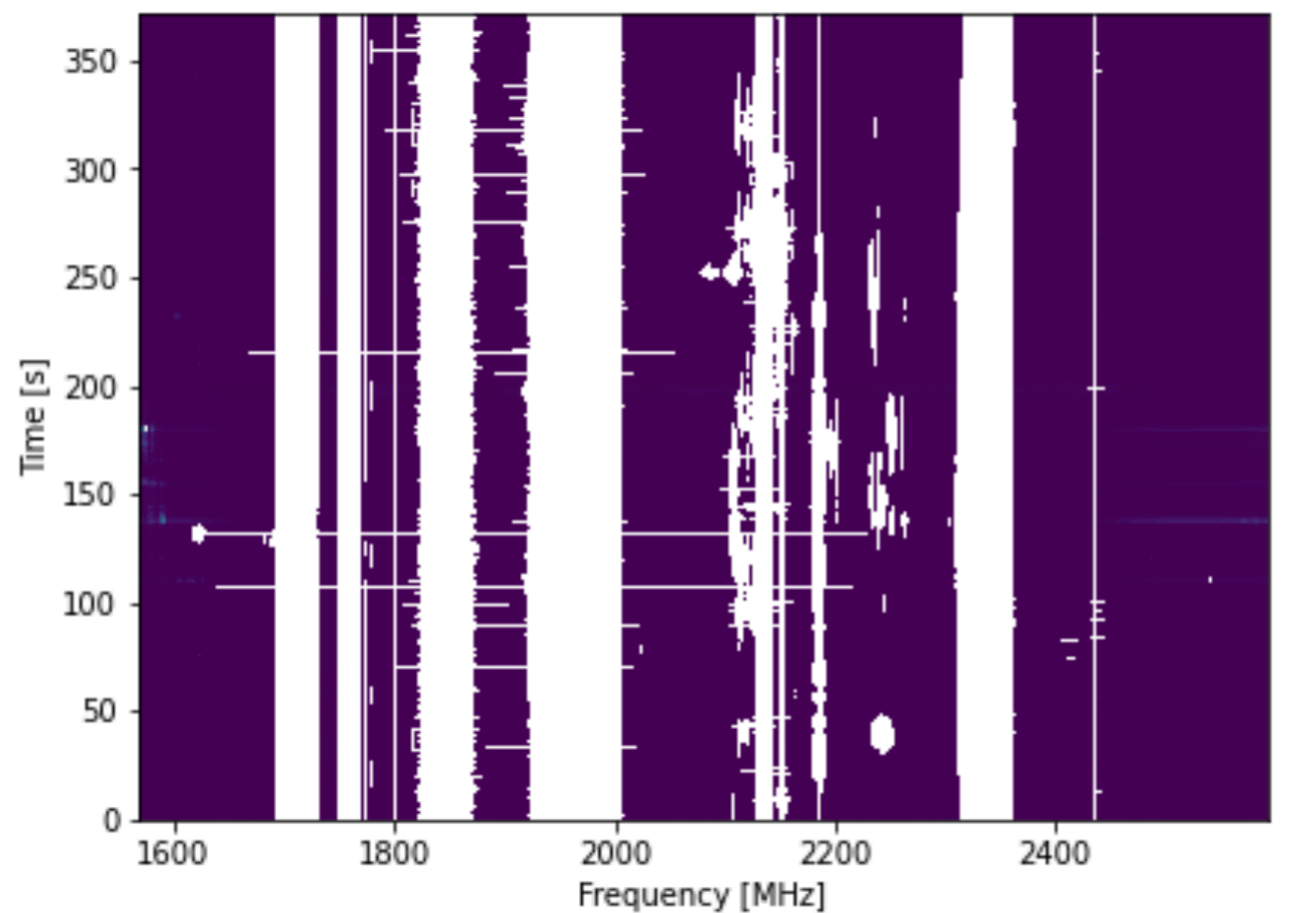
Current progress on ATA RFI Survey data

- Found a knob that I *can* turn from Python: `BaseThreshold` for flagging
- Turning this up helps a lot, but reaches diminishing returns









Next Steps with AOFLAGGER

- Figure out how to make `rfigui` output strategies compatible with the (limited) Python interface
- Do we need the Python interface? Major benefits:
  - Bandpass removal before input
  - Saving numpy flag array after output
- Make a Lua strategy that works well for each frequency band of ATA data (by GHz?)
- Execute the strategy on the whole survey
- Use results to create a `pandas` array of RFI