SERPent

Luke Peck, Danielle Fenech, Jack Morford, Raman Prinja, Jeremy Yates

UCL

Scripted E-merlin RFI-mitigation PipelinE for iNTerferometry

Peck & Fenech 2013 Astronomy & Computing, 2, 54-66

Other than a highly contrived acronym...

Automated reduction package for flagging bad data from e-MERLIN.

Written in Python/ Parseltongue.

Consists of two files – a user input file and the main body of code.

Interacts with AIPS software used to reduce radio data from e-MERLIN.

Creates and appends an AIPS FG table to the input data.

Designed for COBRaS – a low source flux continuum project.

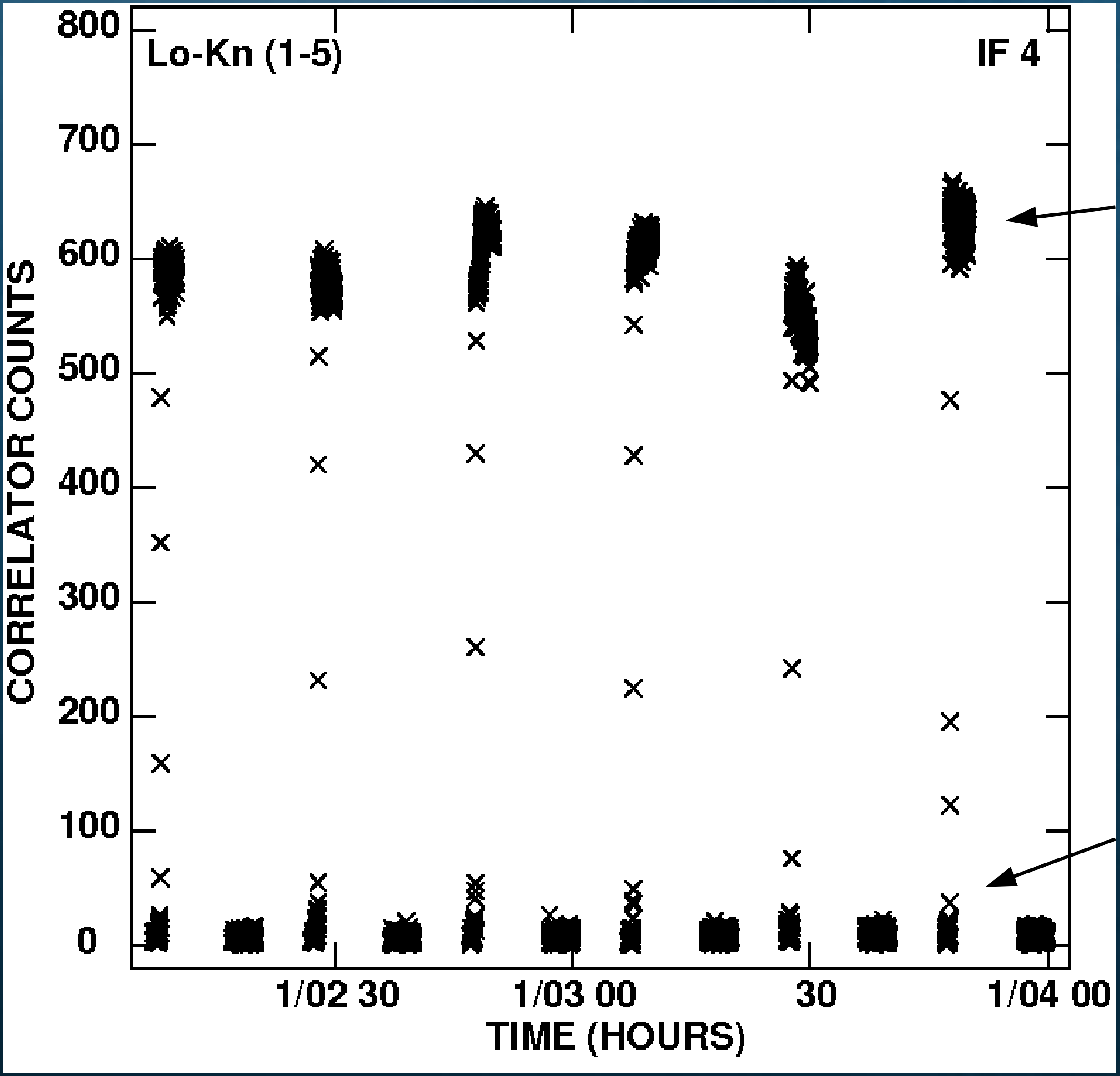
SERPent contains three main passages each designed to deal with specific problems in the datasets from e-MERLIN:

1. Lovell Stationary scans

2. Zero Level Dropouts (Telescope slew errors)

3. Radio Frequency Interference (RFI) mitigation.

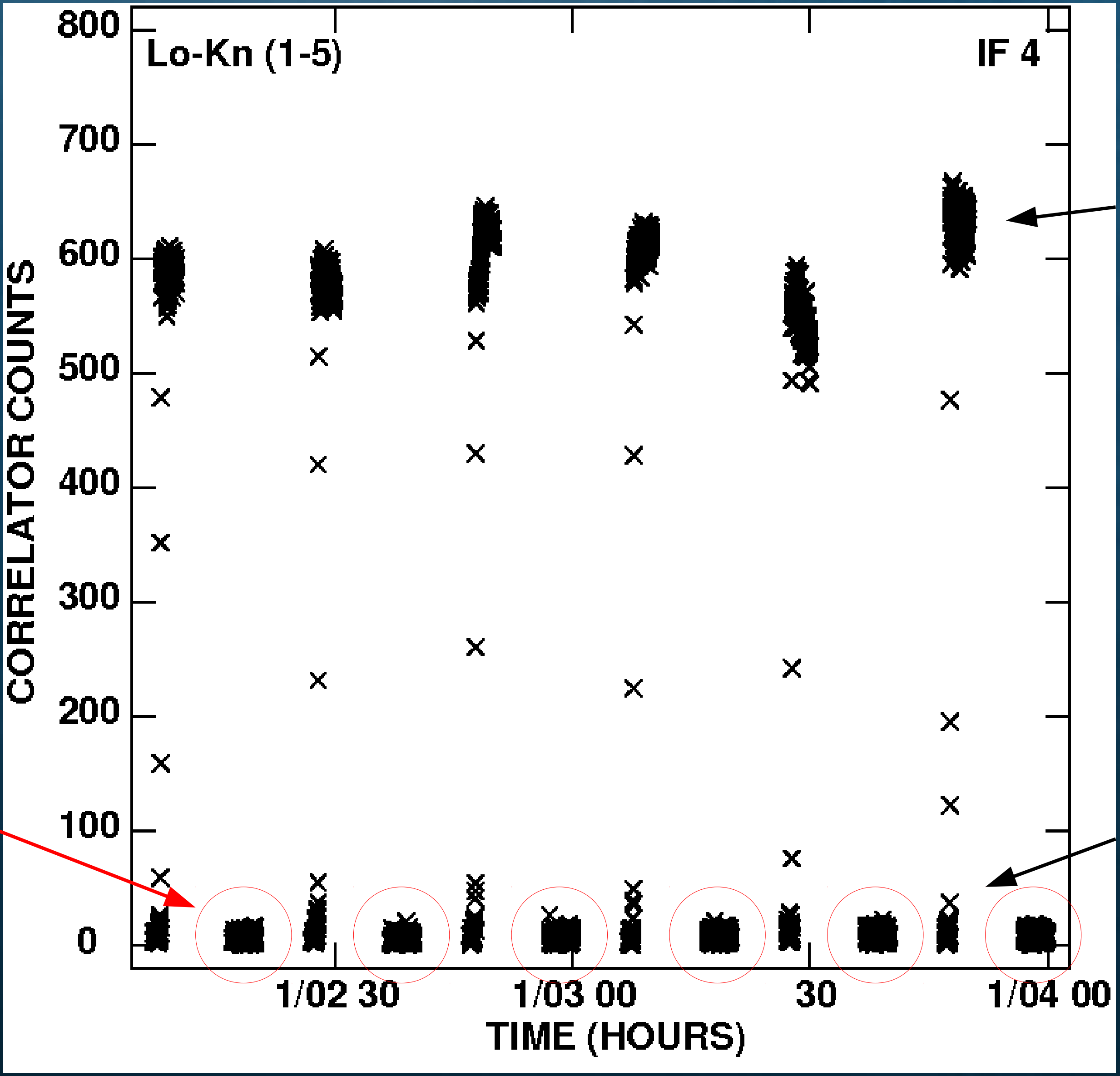
1. Lovell Stationary Scan Removal



Good visibilities

Bad visibilities

1. Lovell Stationary Scan Removal

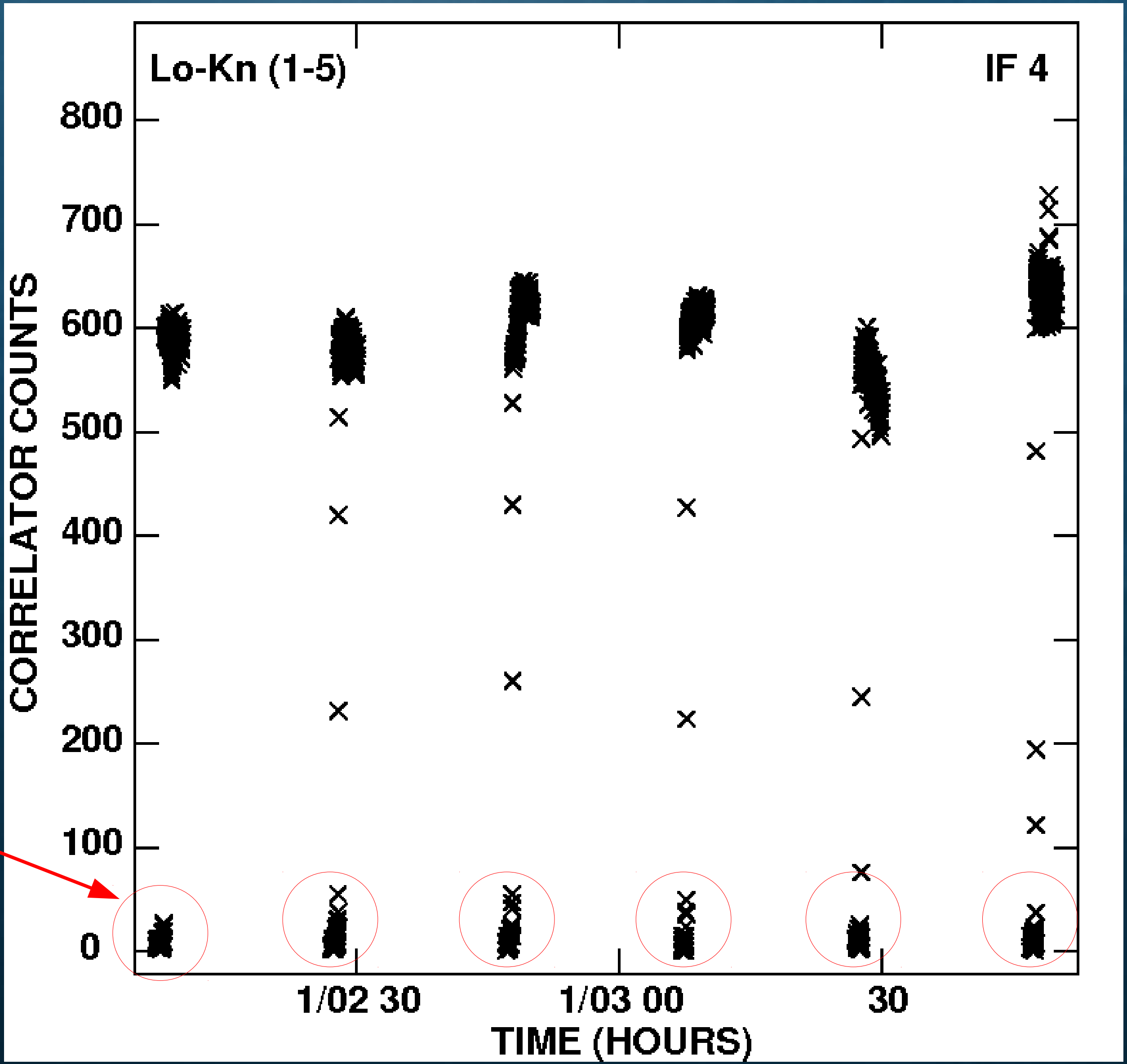


Good visibilities

Bad visibilities

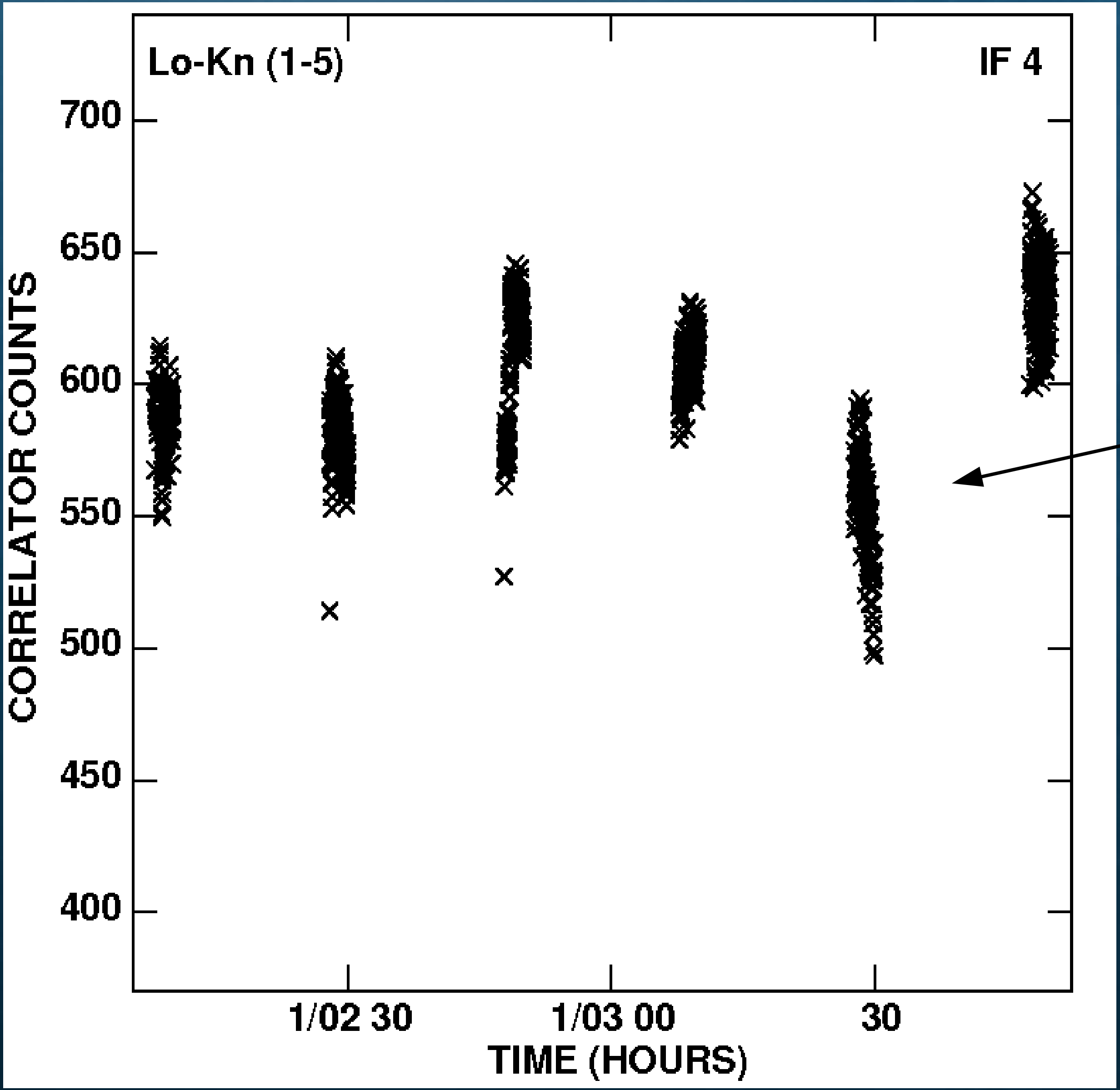
Lovell Stationary Scans

2. Zero Level Dropout (Telescope Slew Errors)



Zero Level Dropouts

After Steps 1 and 2



Only good visibilities remain

3. RFI-mitigation

RFI has a number of origins from CCTV, mobile phones, microwaves, aeroplanes etc...

RFI causes an increase in visibility amplitudes which if not removed causes problems during calibration and imaging.

Lots of RFI at L-band, too much to manually flag. Need automated procedures like SERPent.

SERPent uses a modified algorithm to that used by the AOflagger for LOFAR.

SumThreshold method (Offringa et al. 2010, MNRAS, 405, 155).

3. RFI-mitigation – SumThreshold Method

Data is split by baselines, IFs and stokes and arranged into 2D arrays with the amplitudes sorted in time and frequency.

A threshold is set by the equation:

$$\chi_1 = median + (aggressiveness \times mad)$$

A test window of a certain size scans across the rows and columns of the array and if the averaged contents of the window are greater than the designated threshold, the visibilities are flagged.

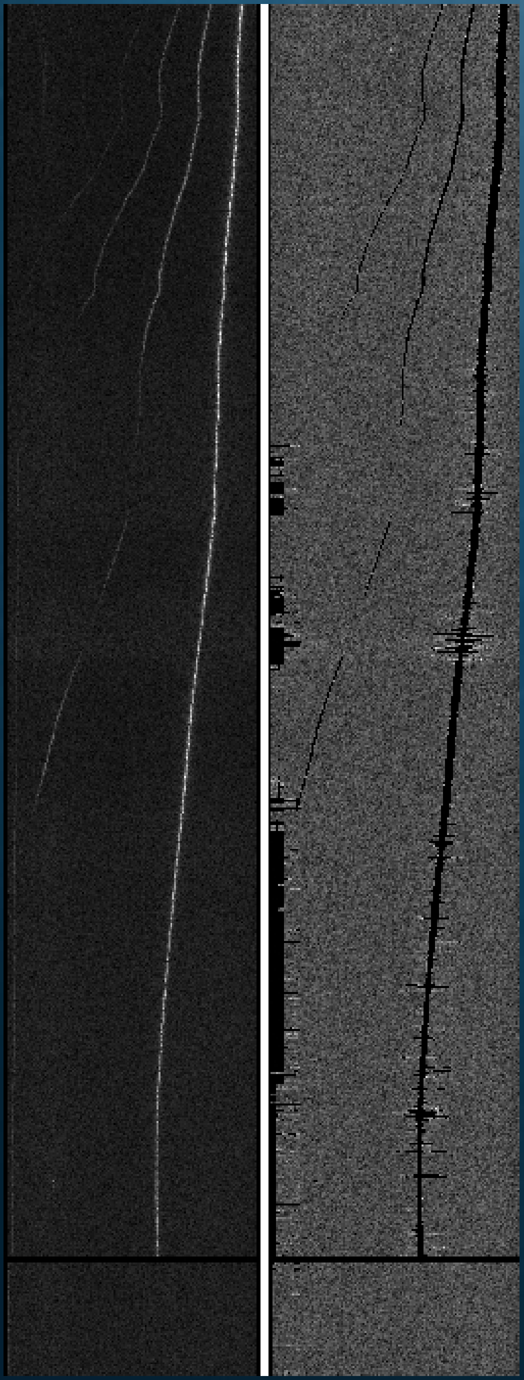
The test window grows in size via subset i = {1, 2, 4, 8, 16, 32, 64...}

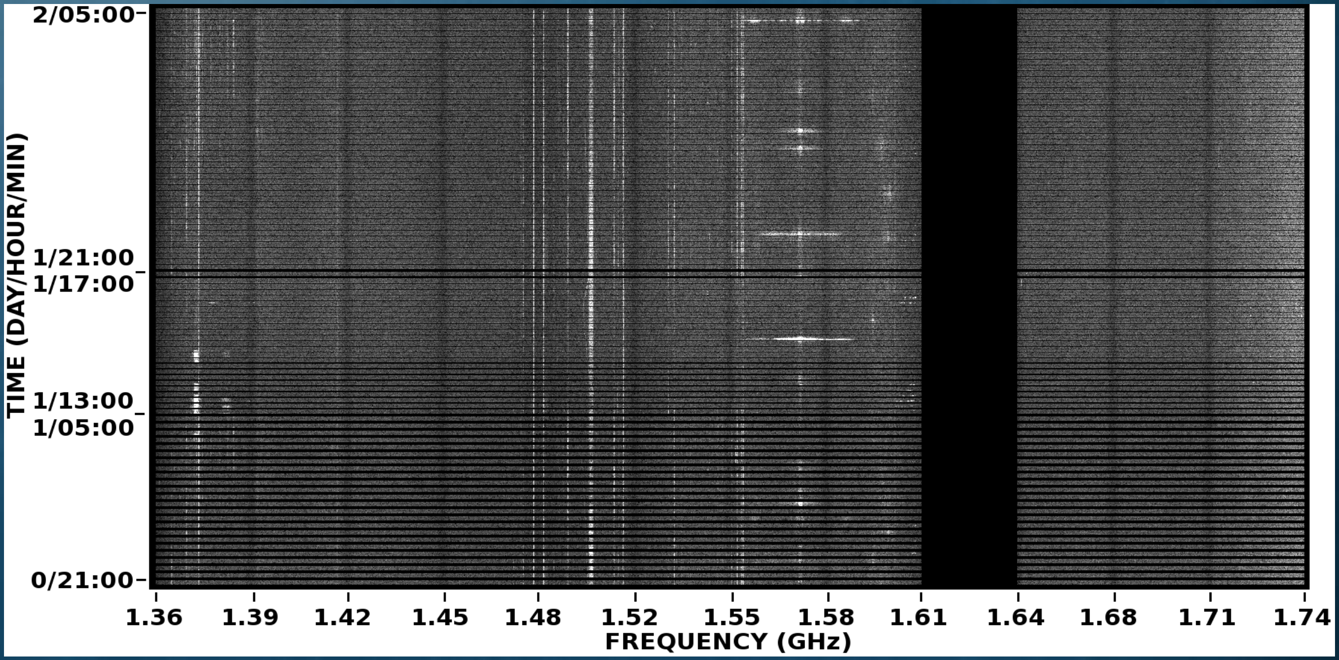
Each time the window increases in size, the threshold is lowered by:

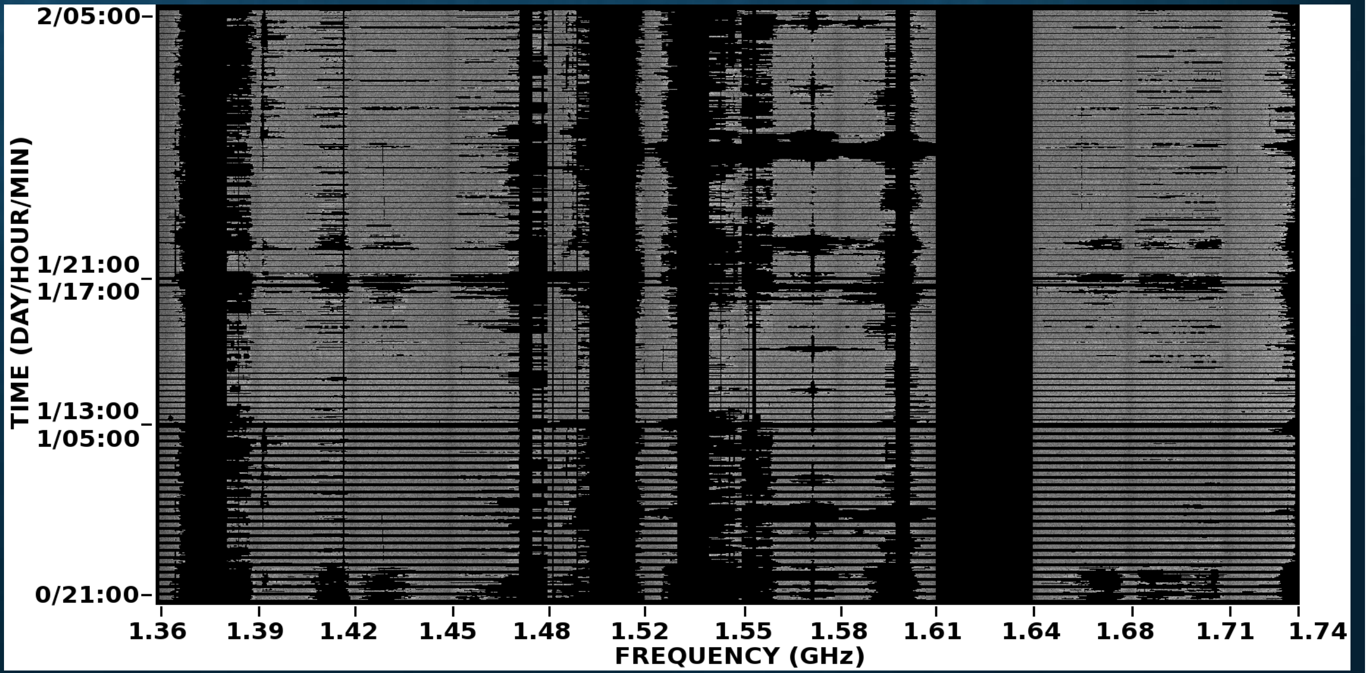
$$\chi_i = \frac{\chi_1}{\rho^{\log_2 i}}$$

SERPent

Luke Peck, UCL

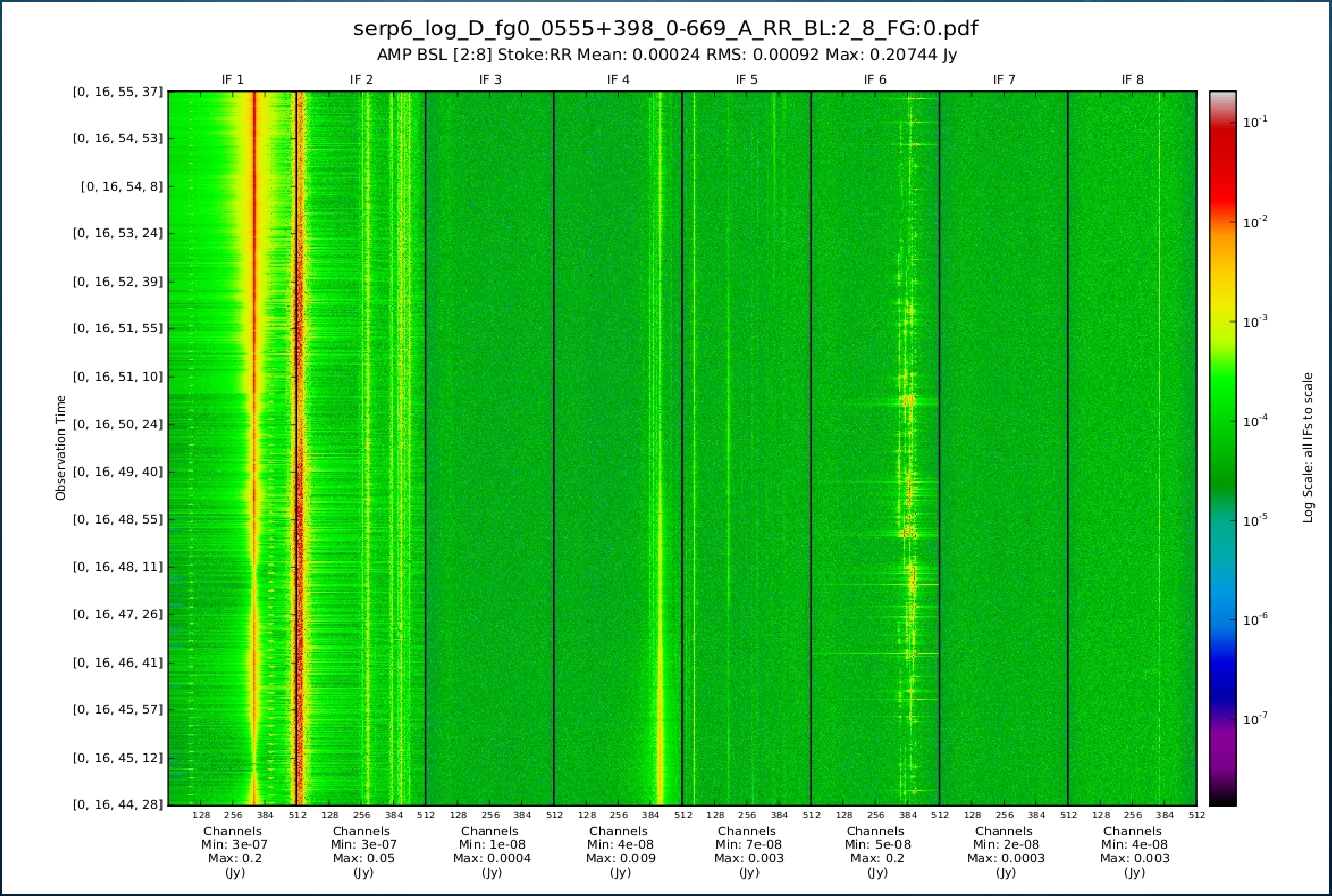




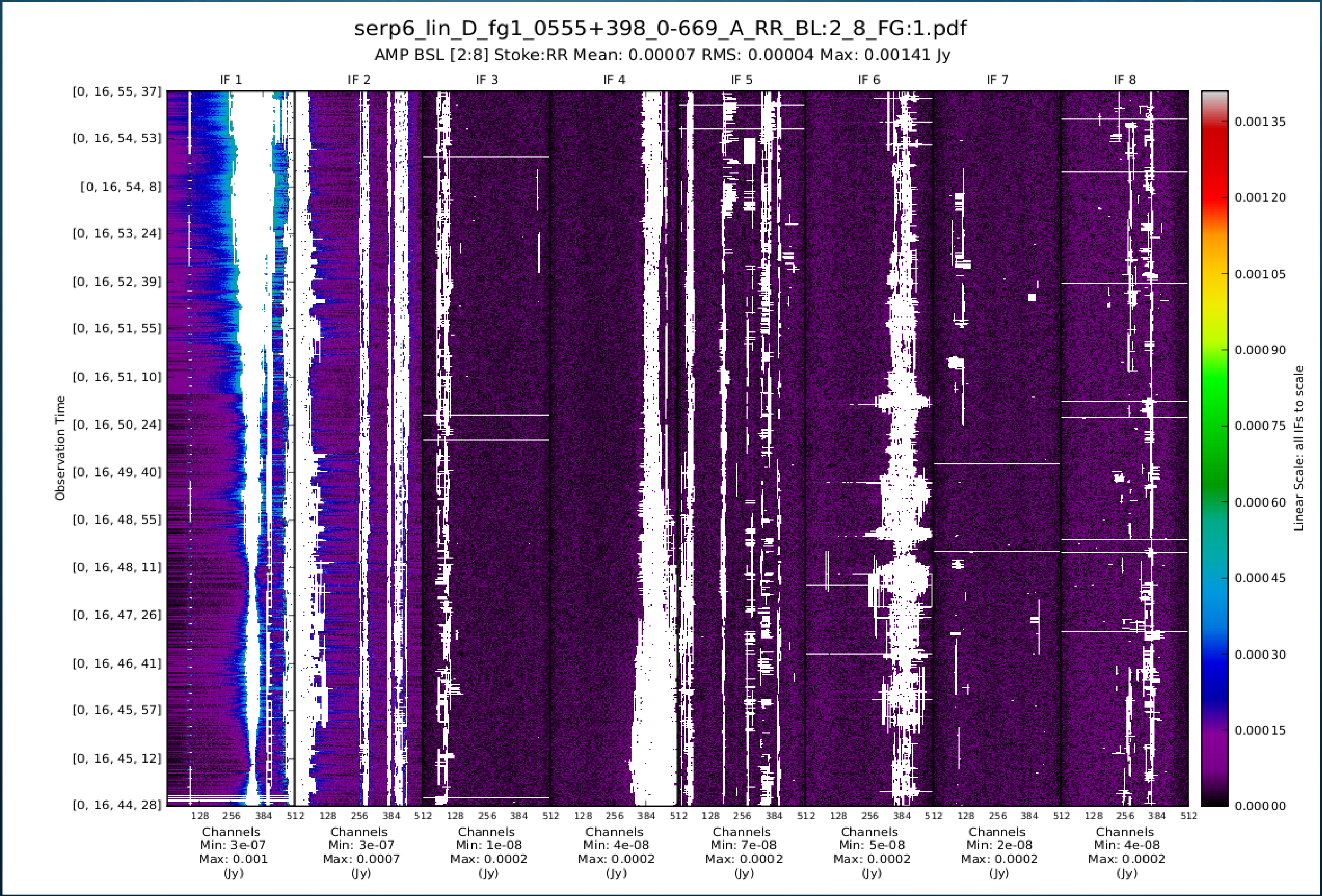


SERPent Luke Peck, UCL

SPPLOT

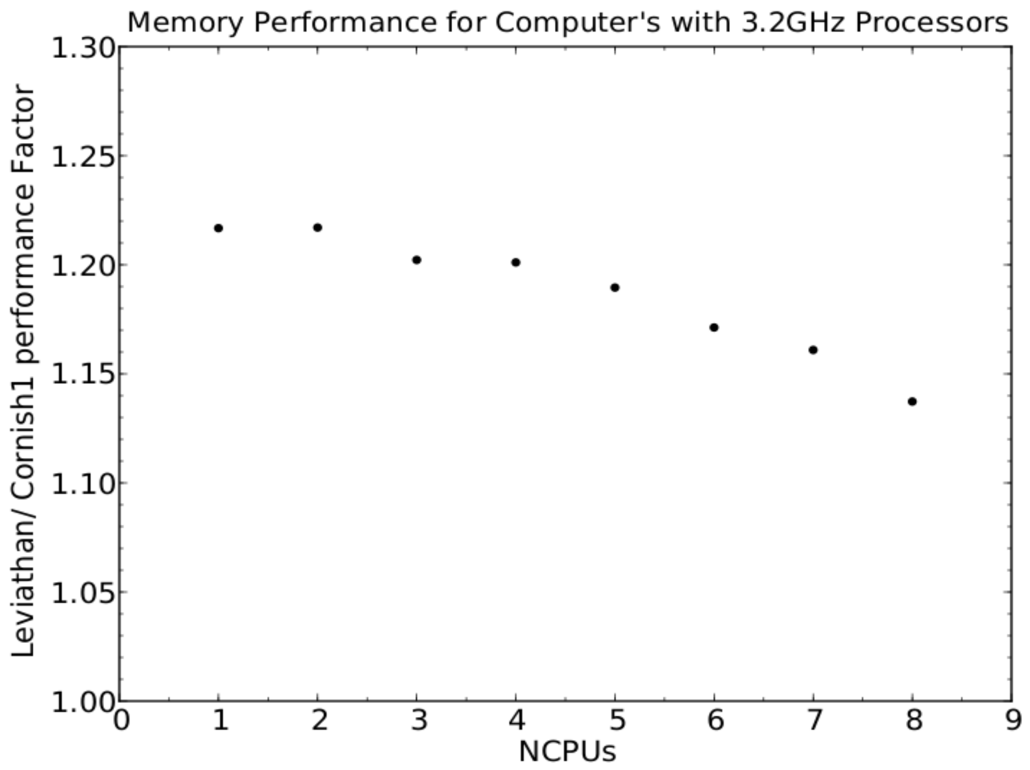


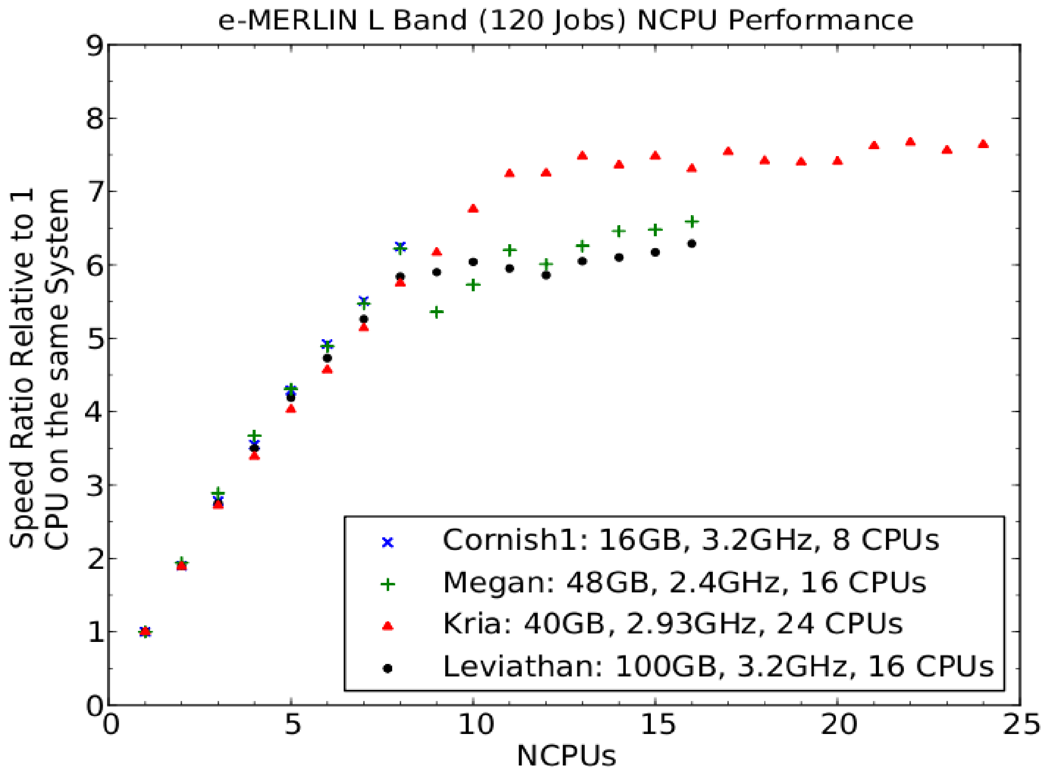
SPPLOTY


serp6_lin_D_fg1_0555+398_0-669_A_RR_BL:2_8_FG:1.pdf
AMP BSL [2:8] Stoke:RR Mean: 0.00007 RMS: 0.00004 Max: 0.00141 Jy
IF 1
IF 2
IF 3
IF 4
IF 5
IF 6
IF 7
IF 8
Observation Time
[0, 16, 55, 37]
[0, 16, 54, 53]
[0, 16, 54, 8]
[0, 16, 53, 24]
[0, 16, 52, 39]
[0, 16, 51, 55]
[0, 16, 51, 10]
[0, 16, 50, 24]
[0, 16, 49, 40]
[0, 16, 48, 55]
[0, 16, 48, 11]
[0, 16, 47, 26]
[0, 16, 46, 41]
[0, 16, 45, 57]
[0, 16, 45, 12]
[0, 16, 44, 28]
128 256 384 512
Channels Min: 3e-07 Max: 0.001 (Jy)
Channels Min: 3e-07 Max: 0.0007 (Jy)
Channels Min: 1e-08 Max: 0.0002 (Jy)
Channels Min: 4e-08 Max: 0.0002 (Jy)
Channels Min: 7e-08 Max: 0.0002 (Jy)
Channels Min: 5e-08 Max: 0.0002 (Jy)
Channels Min: 2e-08 Max: 0.0002 (Jy)
Channels Min: 4e-08 Max: 0.0002 (Jy)
Linear Scale: all IFs to scale
0.00135
0.00120
0.00105
0.00090
0.00075
0.00060
0.00045
0.00030
0.00015
0.00000

Computational Performance

SERPent is parallelised by baselines and IFs.





Memory becomes less important with increase in number of CPUs

The plateau in performance occurs because of the parallelisation method.

Future Development

1. Flagging Parameters Optimisation

2. Speed Optimisation

3. Incorporating SPLOTY into SERPent

4. Incorporating the Flag Mask into SERPent i.e. Read any FG table into SERPent

5. Incorporate spectral line masks

1. Flagging Optimisation

Work being done at UCL and Manchester (Ian Harrison).

Optimise the flagging parameters for:

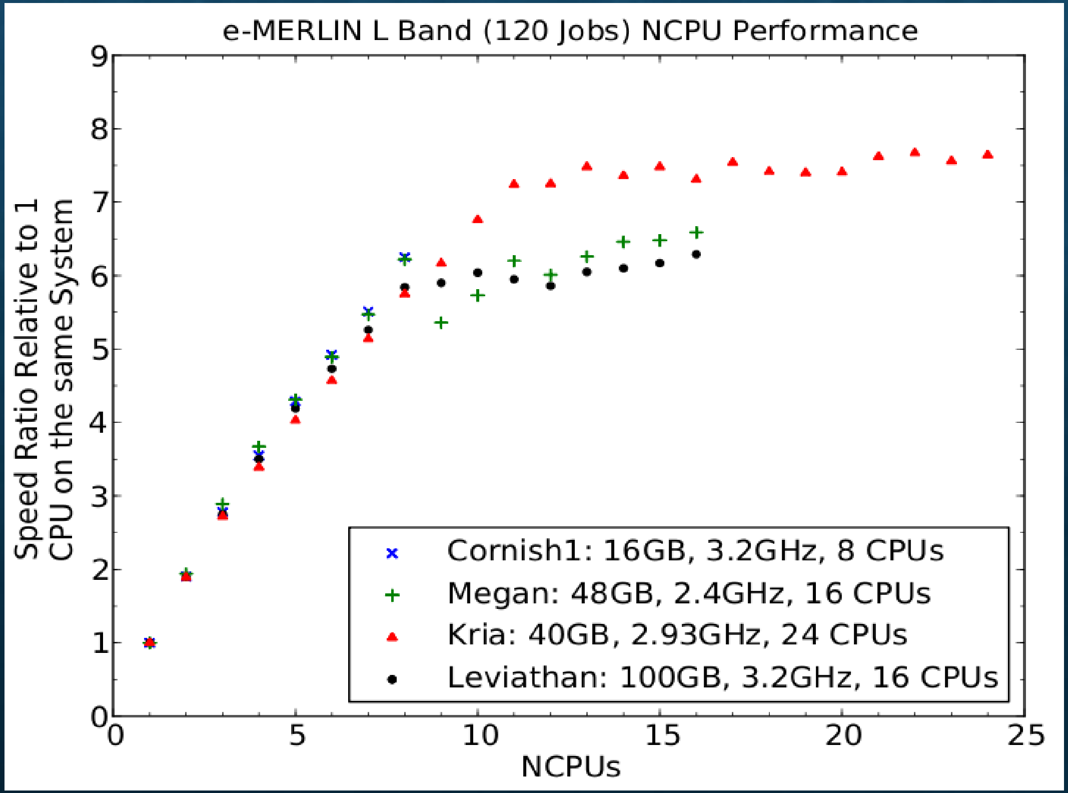
- Each baseline (preferably, or antenna)
- Each regularly observed source (OQ208, 3C286, 0555+398)
- Each observing band (L, C and K)
- Each IF for L-band?

2. Speed Optimisation

For Python 2.7 and higher use the parallelisation module multiprocessing.

Parallelisation Queue method shown to be best (Singh et al. 2013 A&C)

For Python 2.6 and earlier use the existing Fork process in Python.

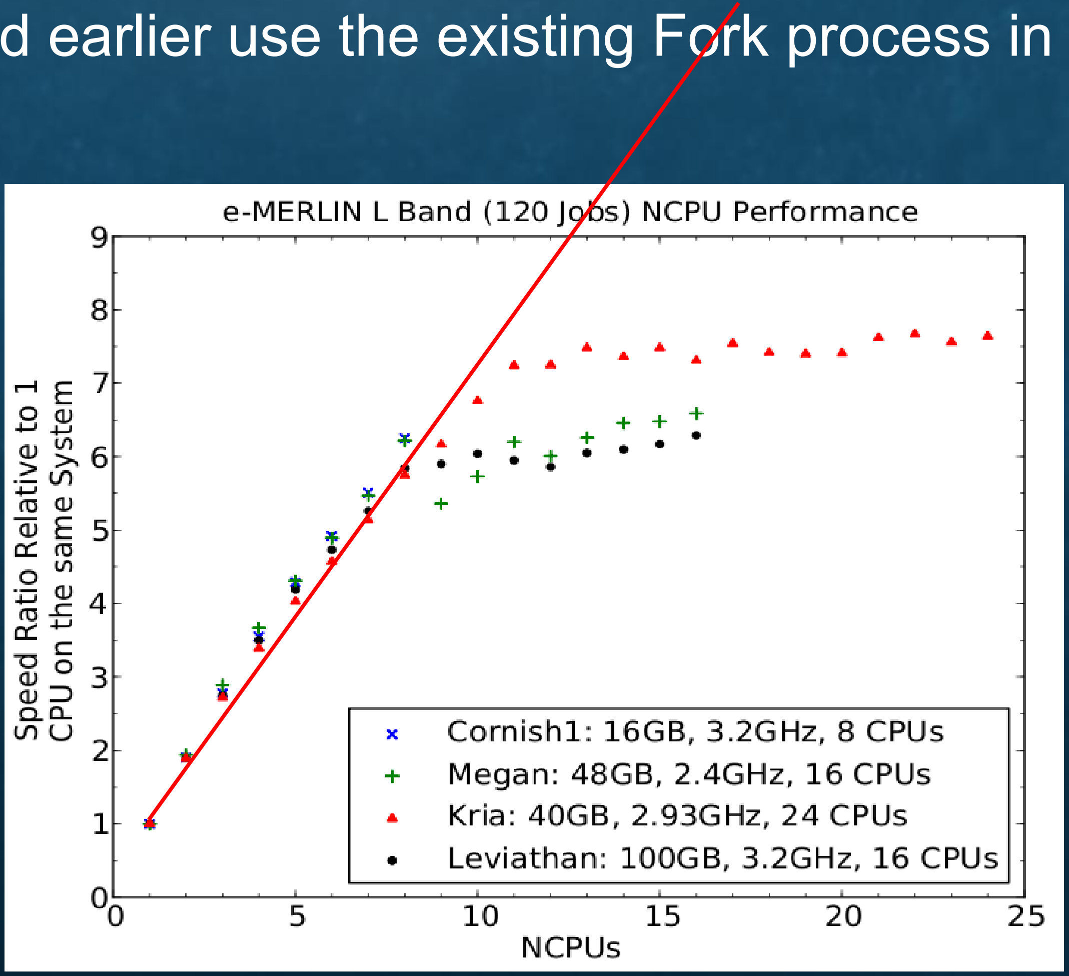


2. Speed Optimisation

For Python 2.7 and higher use the parallelisation module mprocessing.

Parallelisation Queue method shown to be best (Singh et al. 2013 A&C)

For Python 2.6 and earlier use the existing Fork process in Python.



Source Detection and Flux Extraction Algorithm

- Floodfill algorithm finds islands of pixels.
- The sum of these pixels is the uncorrected flux for that source.
- The RMS of the image is calculated from the noise distribution of the image and the noise contribution for each pixel is determined.
- This noise is subtracted from every pixel in the island to give the true flux of the island.
- The PP method assumes islands are a single source, i.e. Does not consider blended sources.
- Because the PP method analyses each connected pixel in an island, no source structure is assumed i.e. Gaussian.

Source Detection and Flux Extraction Algorithm

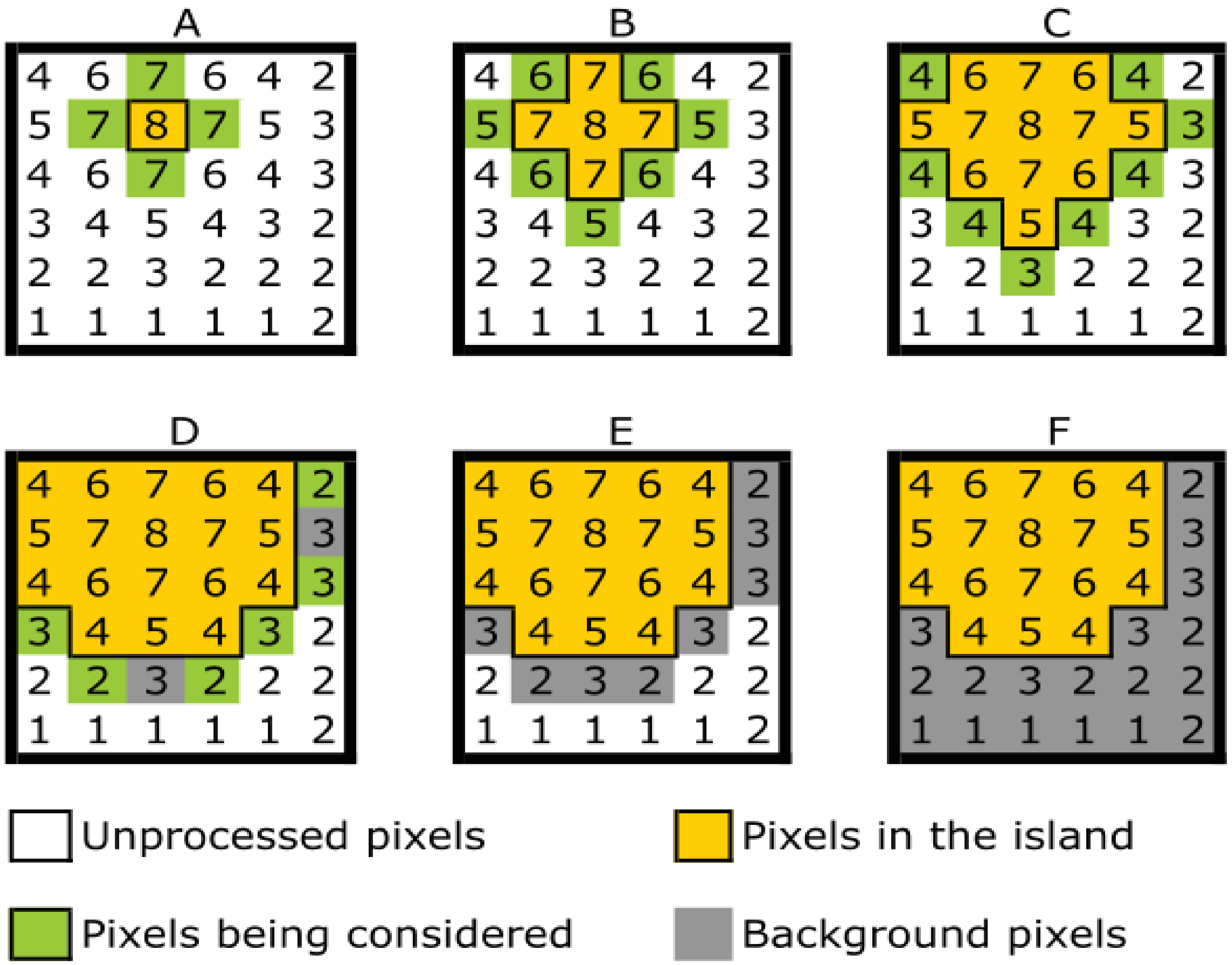
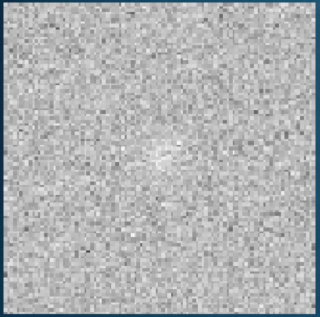
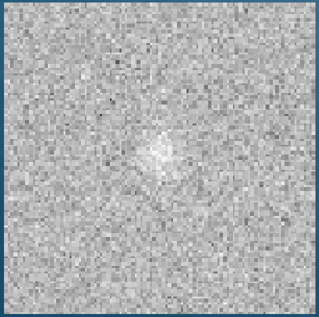
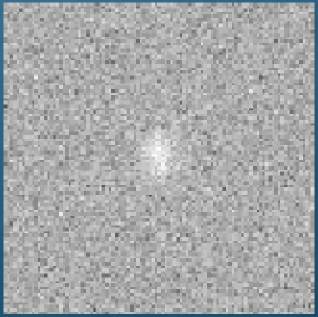


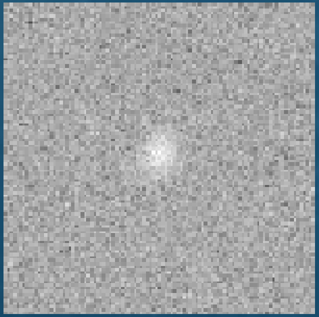
Figure from Hancock et al. 2012

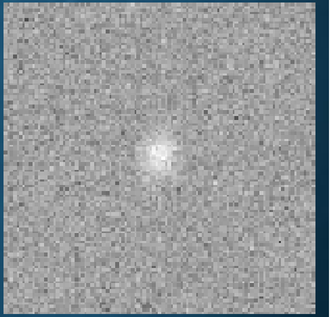
Simulated Point Sources

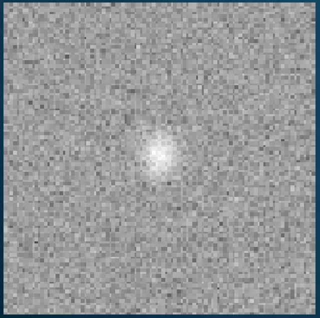

SNR = 2

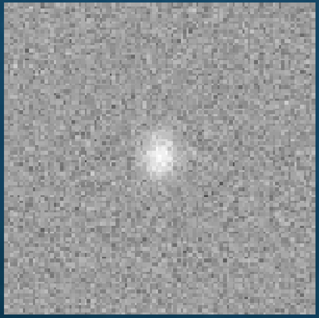

SNR = 3

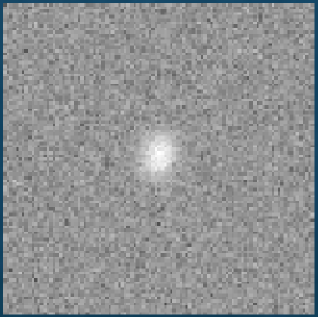

SNR = 4

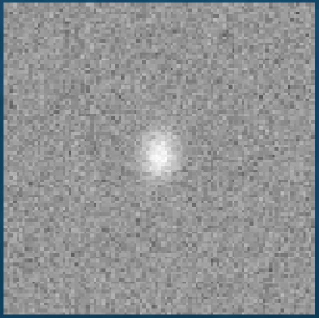

SNR = 5

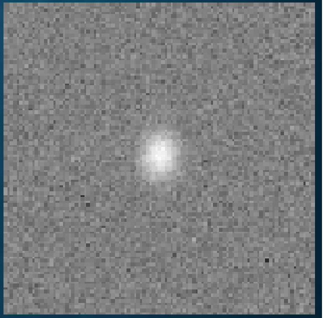

SNR = 6

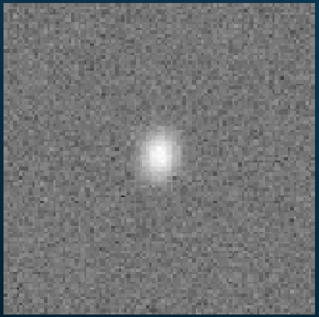

SNR = 7

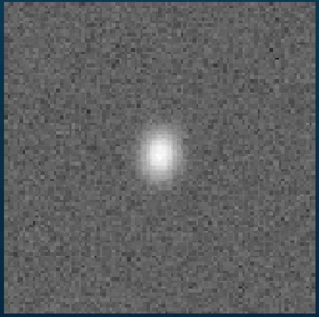

SNR = 8

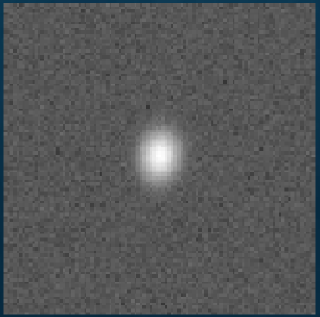

SNR = 9

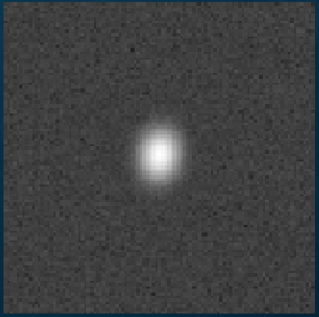

SNR = 10


SNR = 15


SNR = 20


SNR = 30


SNR = 50


SNR = 100

SERPent

Luke Peck, UCL

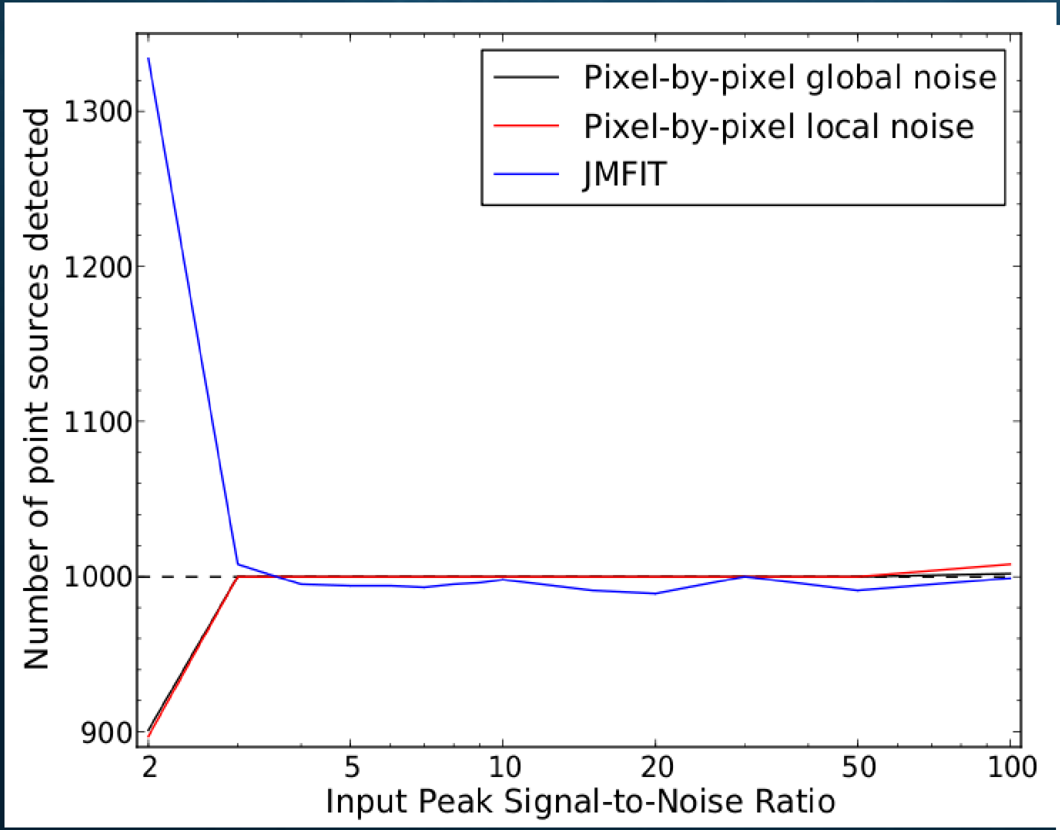
Source Detection Results

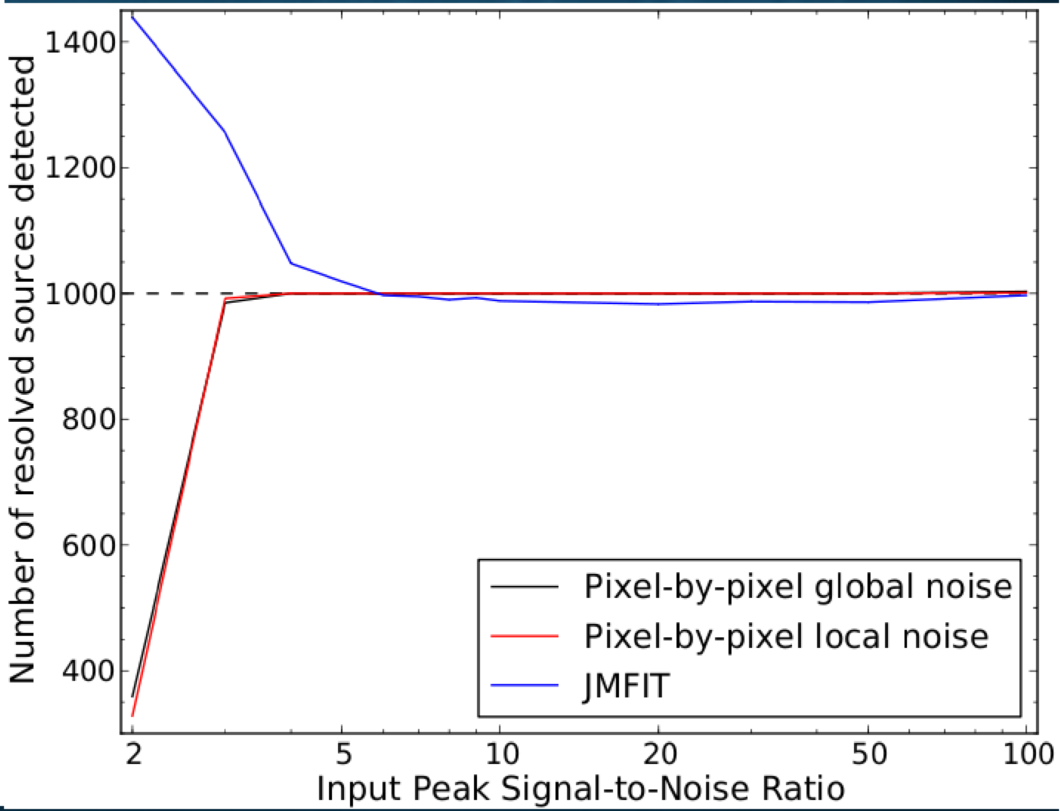
Point Sources

JMFIT has false positive results for SNR < 4. PP Method has false positive results for SNR = 100 (probably due to CLEAN).

Resolved Sources

JMFIT has false positives for SNR < 6. PP method showed no false positives.

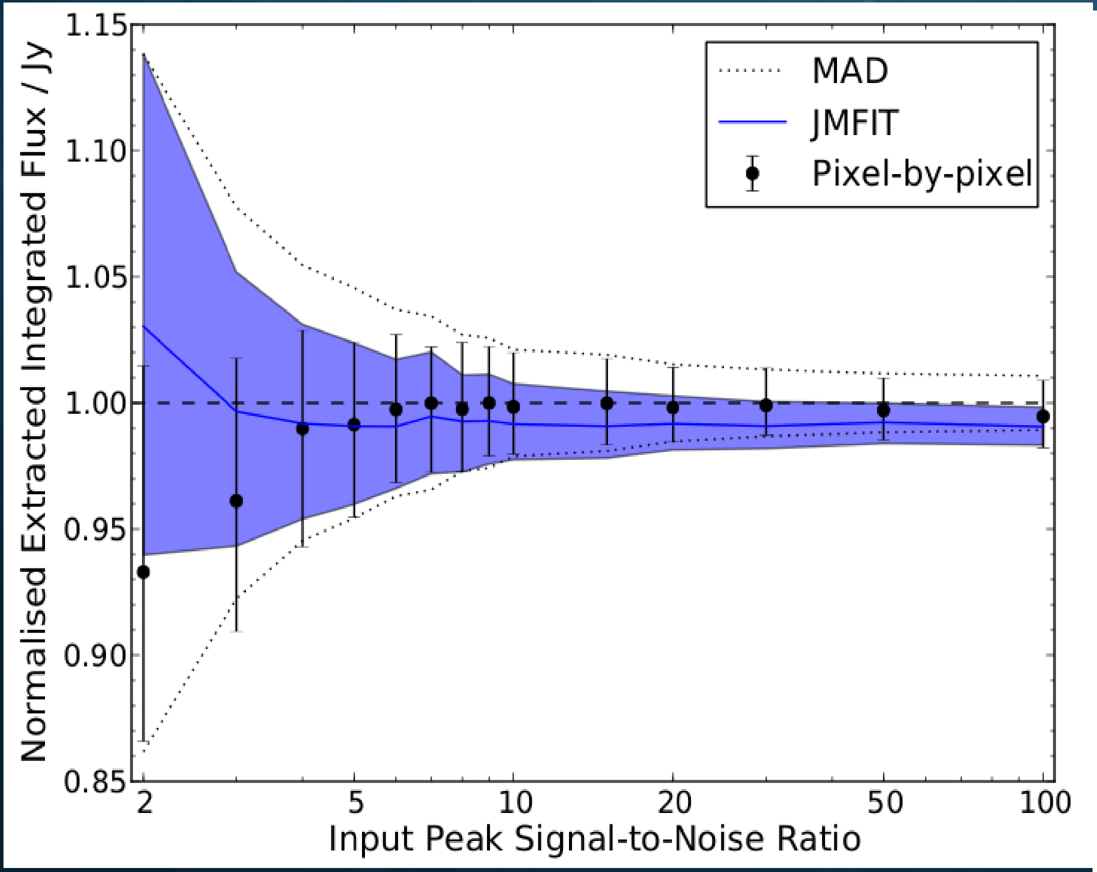




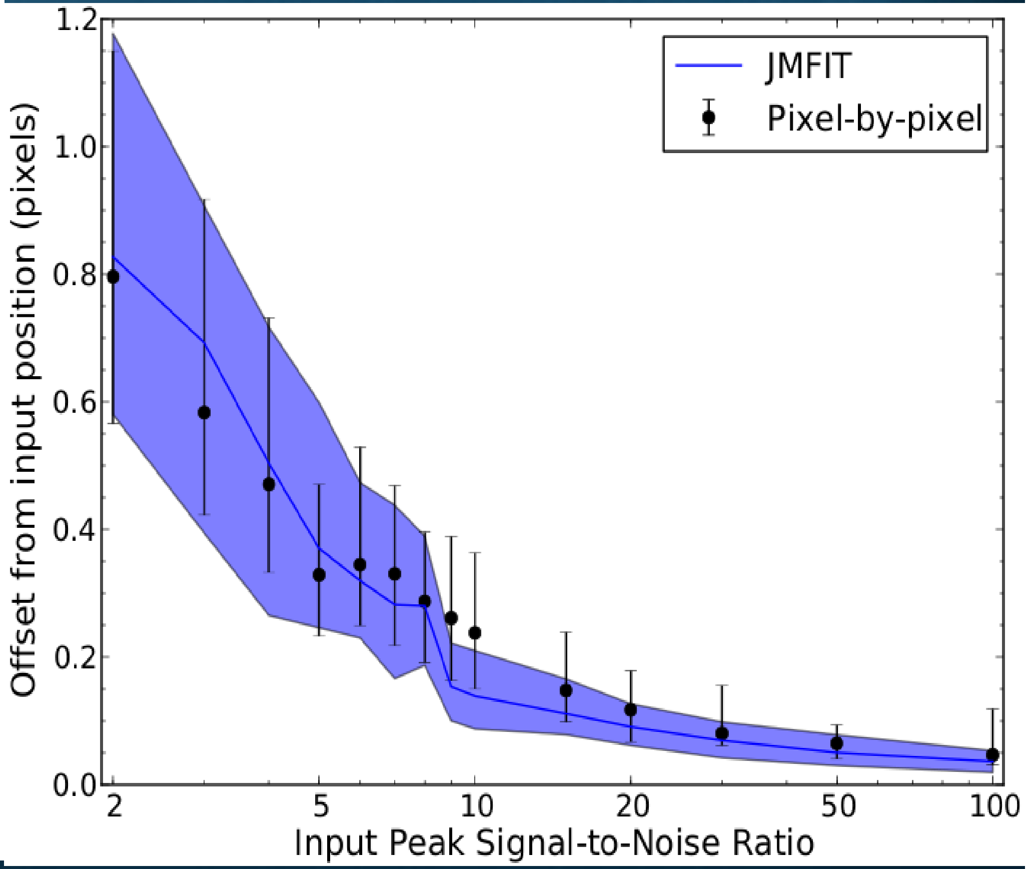
PP Method Performance

Point Sources

Flux Extraction



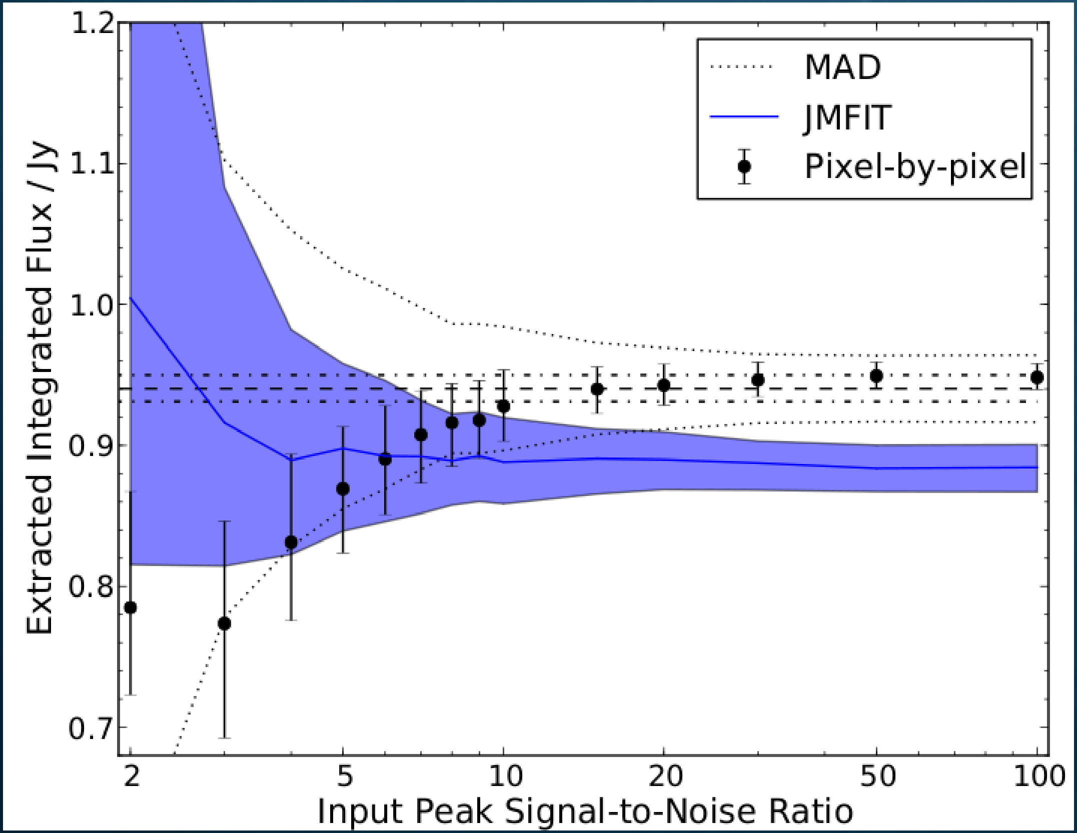
Position Determination



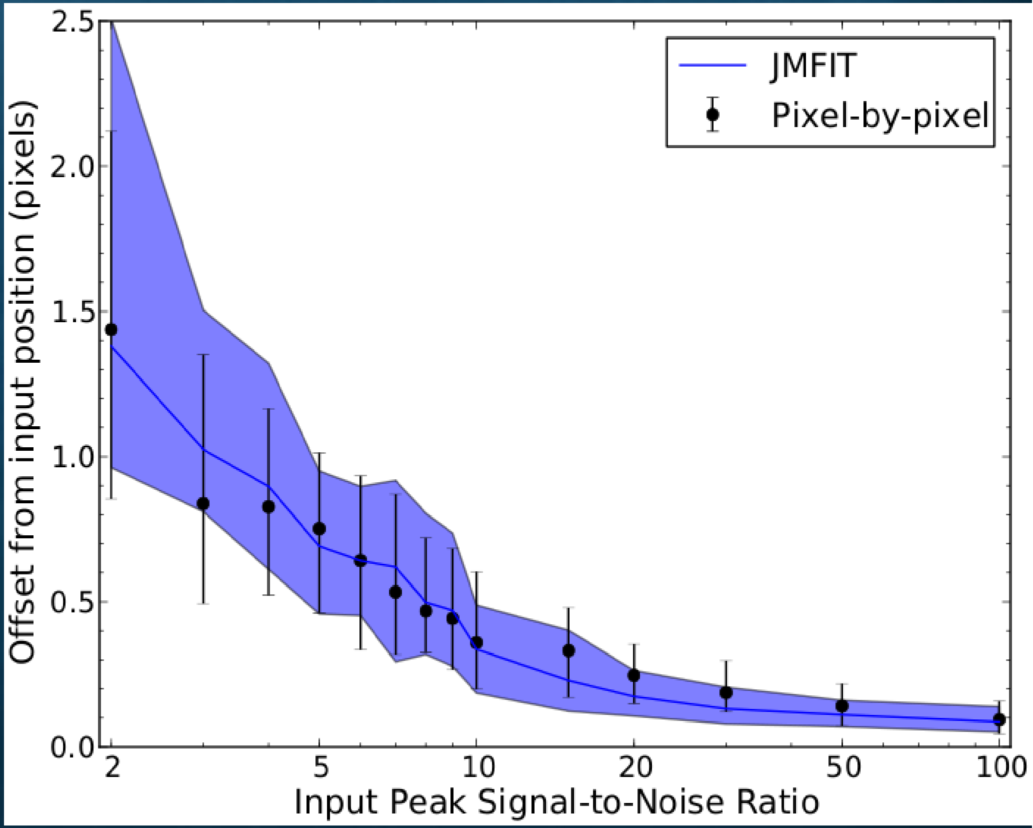
PP Method Performance

Resolved Sources

Flux Extraction



Position Determination



Thank you for listening!

Luke Peck, Danielle Fenech, Jack Morford, Raman Prinja, Jeremy Yates

UCL