NHSC/PACS Web Tutorials

Running the PACS Spectrometer pipeline for CHOP/NOD Mode

PACS-301

Level 0 to 1 processing

Prepared by Dario Fadda
September 2012

Introduction

This tutorial will guide you through the interactive spectrometer pipeline from loading raw data into HIPE to obtain calibrated data with astrometry in the case of chop/nod mode.

Pre-requisites

The following tutorials should be read before this one:

- ***PACS-101***: *How to use these tutorials.*
- ***PACS-102***: *Accessing and storing data from the Herschel Science Archive*
- ***PACS-103***: *Loading scripts*

Overview

Step 1 Check HIPE version and memory

Step 2 Setup

Step 3 Run the 0 → 0.5 pipeline

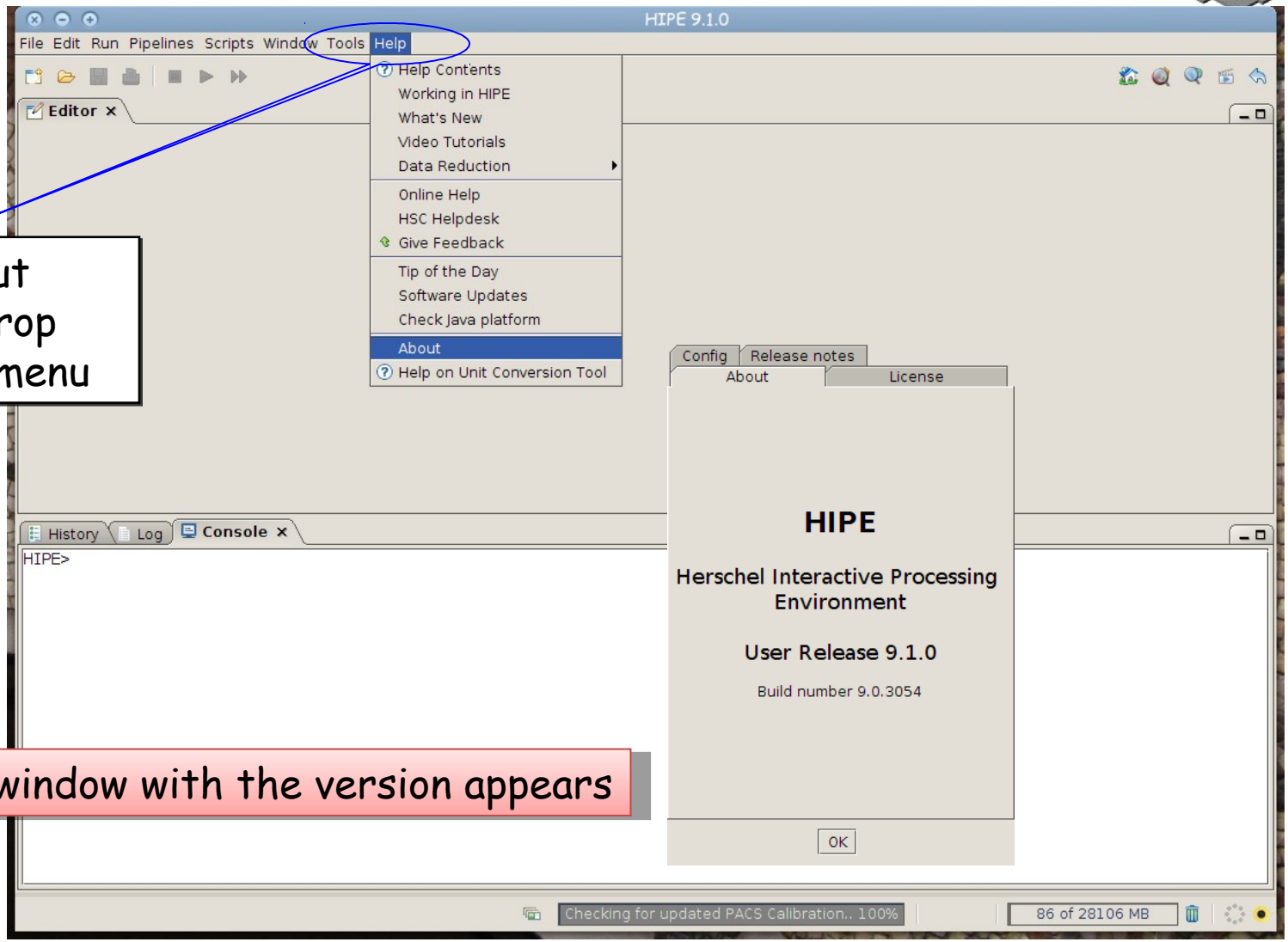
Step 4 Run the 0.5 → 1 pipeline

Step 1

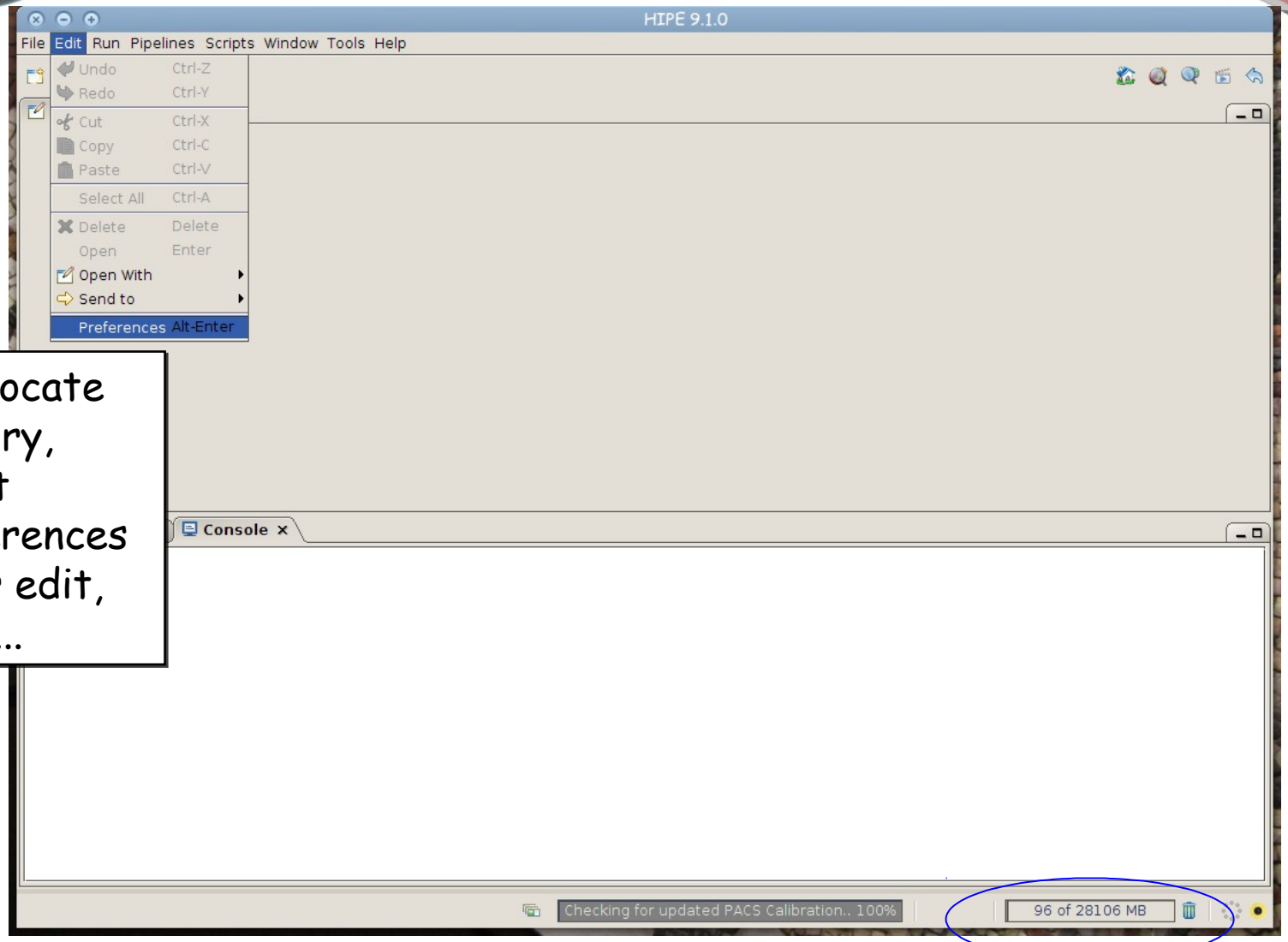
Check HIPE version and memory allocation

The version used for the tutorial is 9.0.3054

Select about from the drop down Help menu



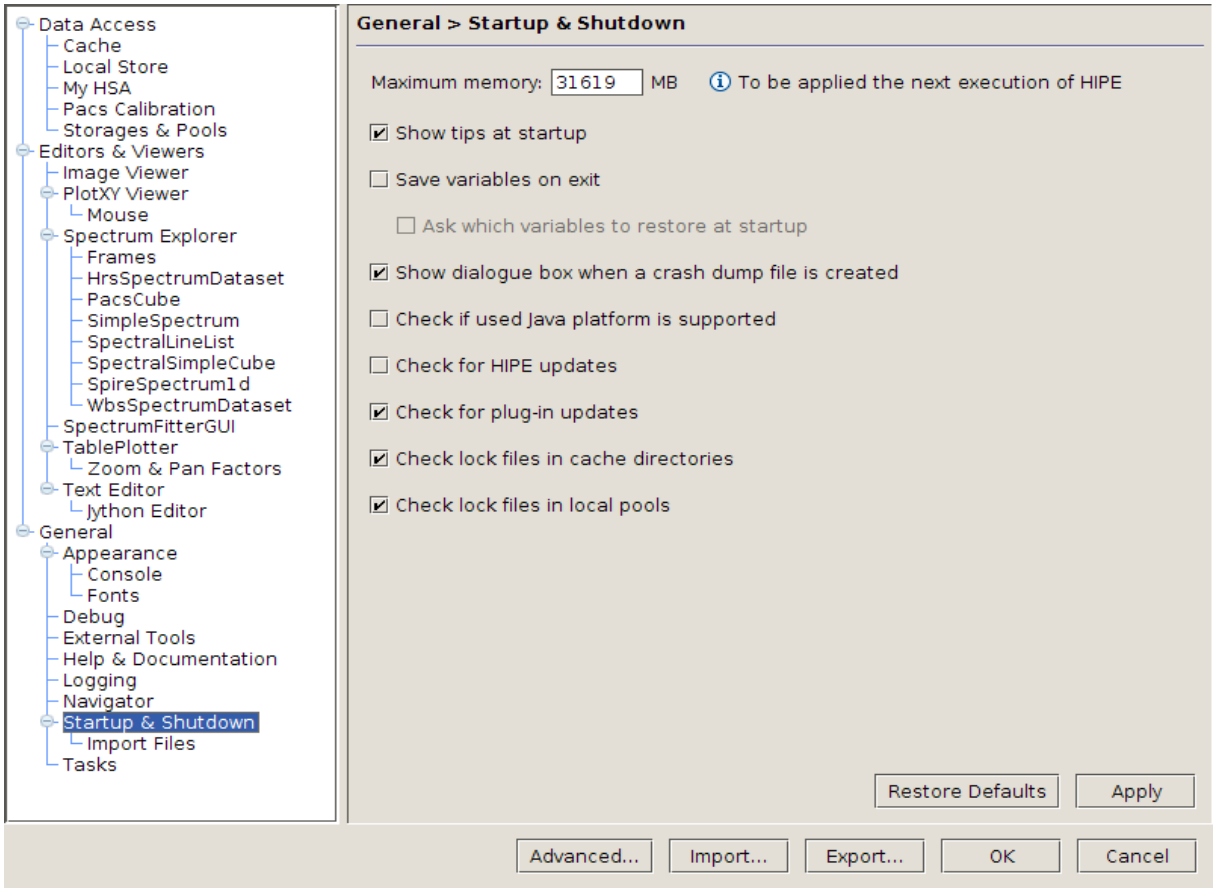
A pop-out window with the version appears



To allocate memory, select preferences under edit, then …

Memory used and available

Then click on Startup & Shutdown and change the amount of memory



The allocated memory should be smaller than the total RAM of your computer. You have to exit and start a new session to use the new amount of memory.

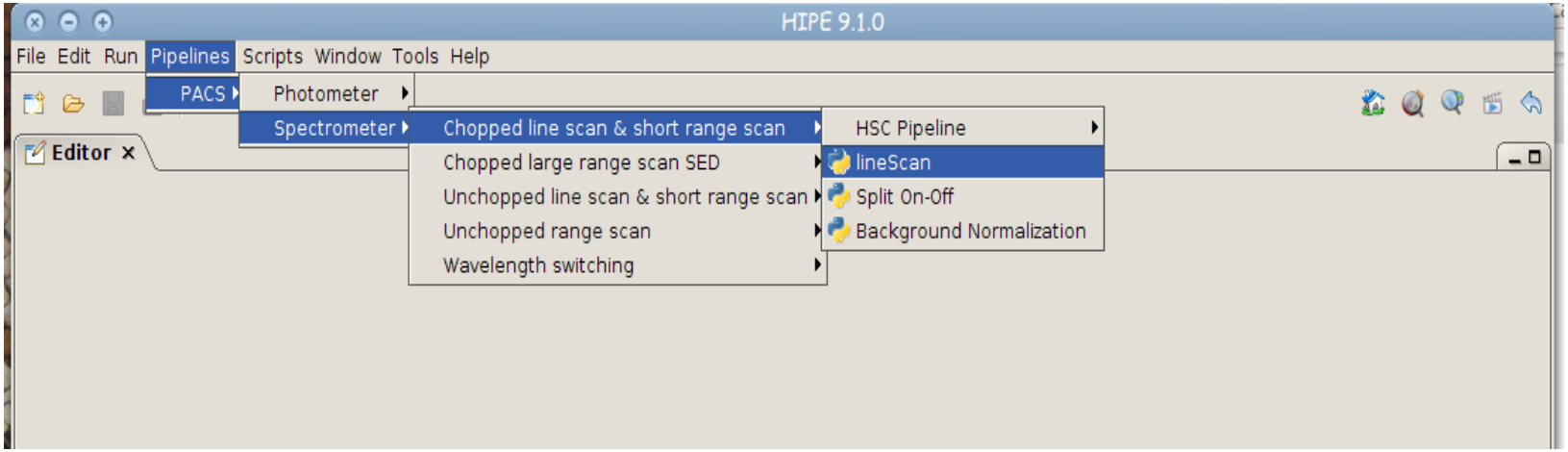
Step 2

Setup

Load pipeline script, load observation, check data, and select the camera

Loading the script

The script used in this tutorial corresponds to the script available directly from the distribution.



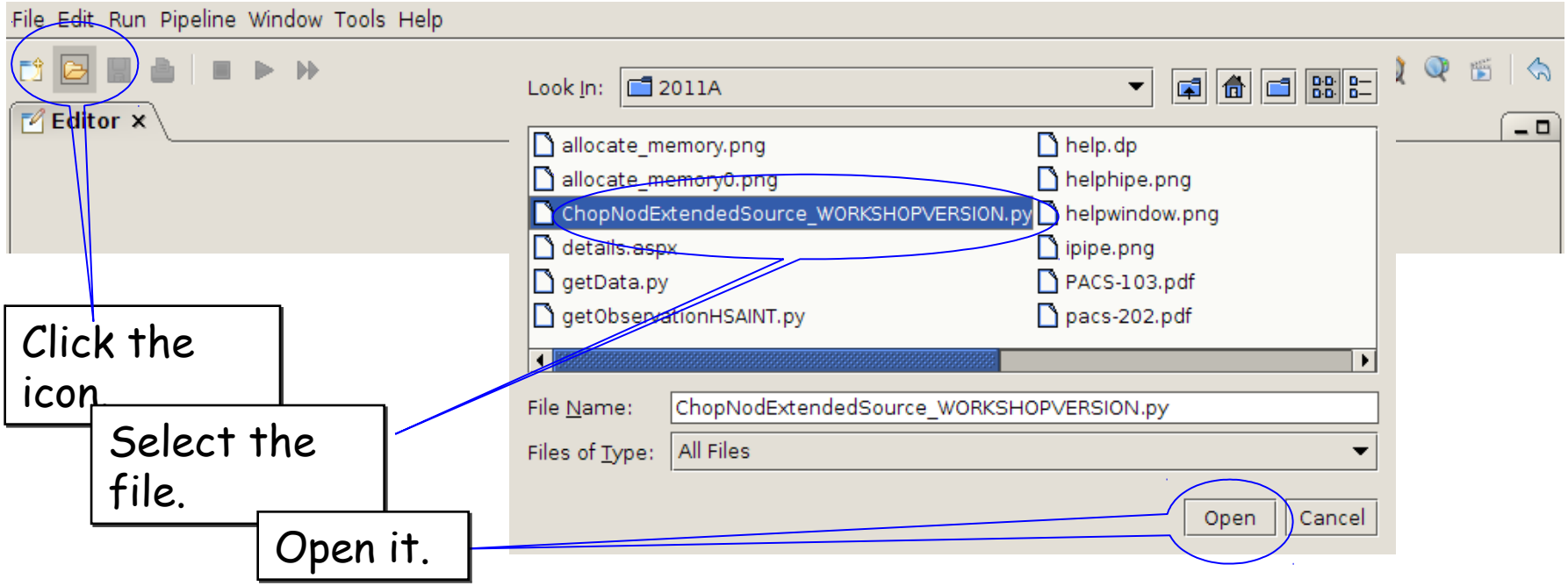
There are two other interactive scripts available:

a) Split on-off: allows one to have separate reductions of the on-source and off-source signals;

b) Background Normalization: uses the emission of the telescope as reference signal to calibrate the signal

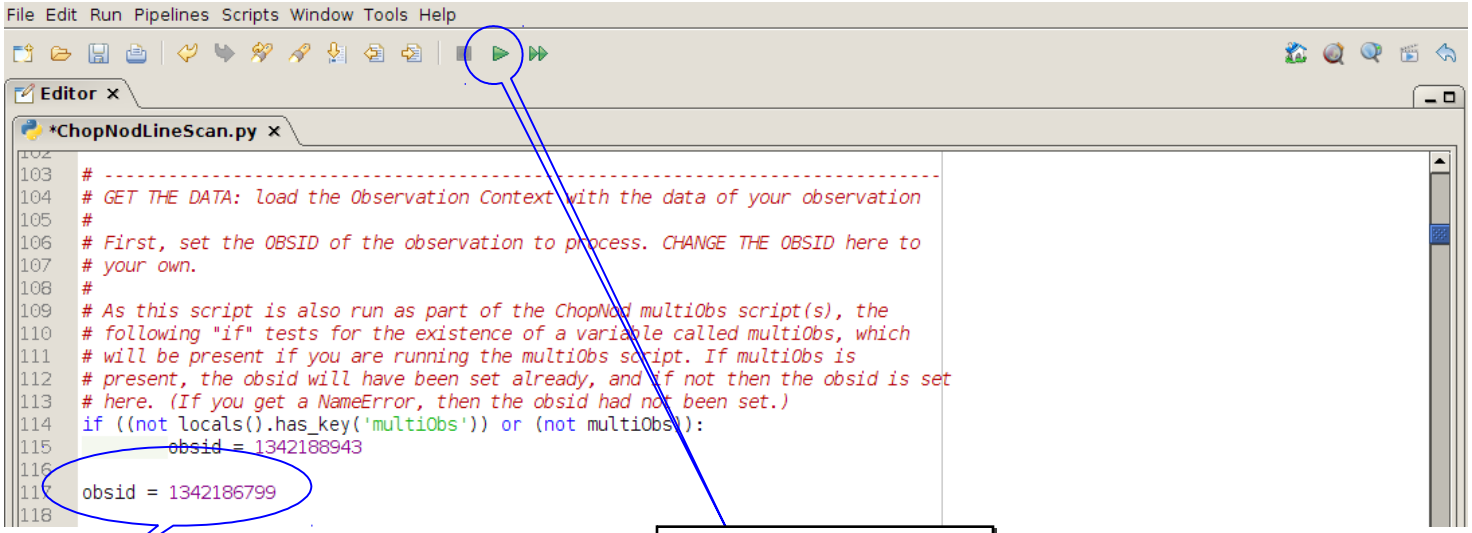
Loading the script

In the case you saved a modified version and you want to load it for analysis of other data, you can access it directly from HIPE clicking on the yellow "folder" icon.



Loading the observation

Once the script is loaded, one can simply step through the lines to execute it one by one. The first thing to do is loading the OBSID relative to the observation chosen.
In the case of this tutorial, the observations has been already saved into a pool which has to be put into your ~/.hcss/lstore directory which is created once installing HIPE. The only thing to do is to write the correct obsid number and then start clicking the green arrow ….

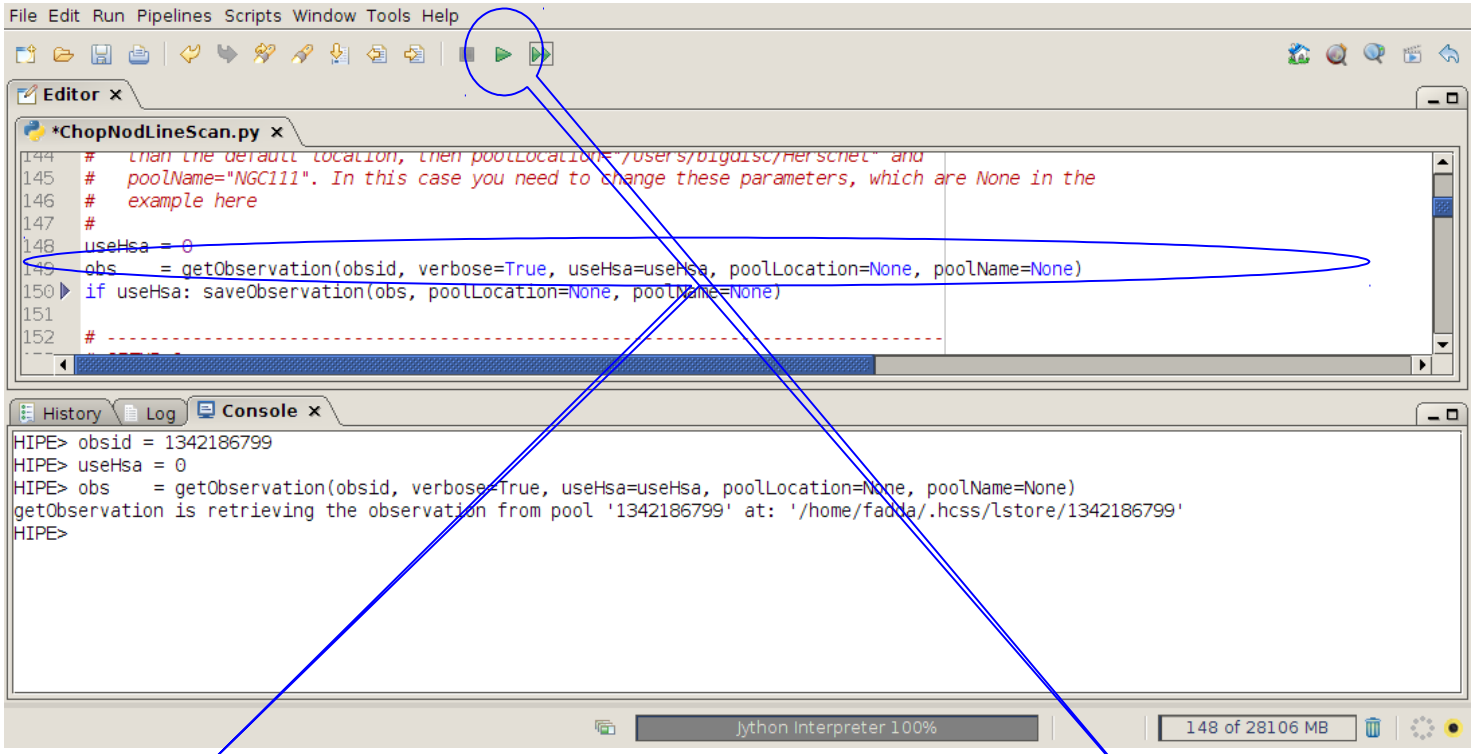


Add this line and click on it.

Hit the arrow

Loading the observation

Next step, we load the observational context (a structure containing all the observational data, information about them and calibration data).



Click on this line.

Hit the arrow

Observation summary

```
HIPE> obsSummary(obs)

Observation summary
    OBSID:       1342186799
    Instrument:  PACS
    AOR label:   NearGalPACS-SB-01-blue
    Proposal:    SDP_esturm_3
    Target:      M82
    Redshift:    0.000677 (z)
    Concat.:     Undef.
    OD:          178
    Start:       Sun Nov 08 07:31:26 PST 2009
    Duration:    582.0 seconds (incl. spacecraft on-target slew time)

AOT and instrument configuration
    AOT:         PacsLineSpec
    Mode:        Pointed, Chop/Nod
    Bands:       B3A + R1 (prime diffraction orders selected)
    Is bright:   YES (shortened spectral range mode)
    Chopper:     large throw
    Nod cycles:  1

Observation request summary (HSpot Line/Range Editor Table)
    Number of requested primary lines/ranges: 2
    List of requested primary line centres [microns]:
        Line/range 1 :   63.223 microns, 3 repetitions, ID OI 63
        Line/range 2 :   57.359 microns, 3 repetitions, ID NIII 57
```

We will select: camera = 'blue'

Unless you know exactly what is in your data, after loading them it is advisable to inspect them. You can do this from the console by writing:

obsSummary(obs)

In this case, we discover that two lines have been observed in the blue range of the PACS spectrometer. So, we will have to select the "blue" camera otherwise we will just reduce the parallel "red" data.

Setting the camera

File Edit Run Pipelines Scripts Window Tools Help

Editor ×

*ChopNodLineScan.py ×

```
# SETUP 1:
#     - Red or blue camera ? As before, we test for whether this script is
#     being run within a multiObs script, in which case the camera will already
#     have been set
if ((not locals().has_key('multiObs')) or (not multiObs)):
        camera     = 'blue'

# ----------------------------------------------------------------------------
# Set up the calibration tree. We take the most recent calibration files,
# for the specific time of your observation (obs=obs)
#
# This tree contains pointers to all the calibration files that the pipeline
# tasks use (when calTree=calTree is specified in a task's call).
# From that calibration tree, certain calibration files are used by each task.
# The "Version" of the calibration tree can be found from the simple
```

We select camera = 'blue'

History | Log | Console ×

```
    Level 2 status: Processed

Quality comments
     ["This observation was performed correctly by the instrument/spacecraft.
Pipeline processed up to L2.
Quality checked by HSC calibration scientists team.
QC comments:  Passed quality control, with the caveats described in the PACS chopped line scan and high sampling range scan AOT
release note.
"]


HIPE> if ((not locals().has_key('multiObs')) or (not multiObs)):
        camera     = 'blue'
HIPE> print camera
blue
HIPE>
```

Jython Interpreter 100% | 103 of 28106 MB

After selecting the camera, we can check what camera we selected by simply printing: "print camera"

Setting the calibration tree

Finally, we set the calibration tree.

File Edit Run Pipelines Scripts Window Tools Help

Editor

UnchoppedRangeScan.py

```
# corresponds to.

calTree = getCalTree(obs=obs)
if verbose:
        print calTree
        print calTree.common
        print calTree.spectrometer
```

History | Log | Console

```
HIPE> print obs.meta["calVersion"]
{description="Version of Calibration Tree", string="PACS_CAL_32_0"}
HIPE> if verbose:
        print calTree
        print calTree.common
        print calTree.spectrometer
PACS Calibration Tree
  Model   : FM
  Scope   : BASE
  Version : 41
  Branches: [common, photometer, spectrometer]

PacsCalCommon Calibration Products:
  chopperAngle                    : FM, 3
  chopperAngleRedundant           : FM, 3
  chopperJitterThreshold          : FM, 2
  chopperSkyAngle                 : FM, 2
  csResistanceTemperature         : FM, 1
  filterWheel2Band                : FM, 2
```

Jython Interpreter 100% | 56 of 28106 MB

Read the time stamp of our obs and apply the calibration from the used distribution.

Version 41 and later ones incorporate several improvements wrt version 32 (archive reduction)

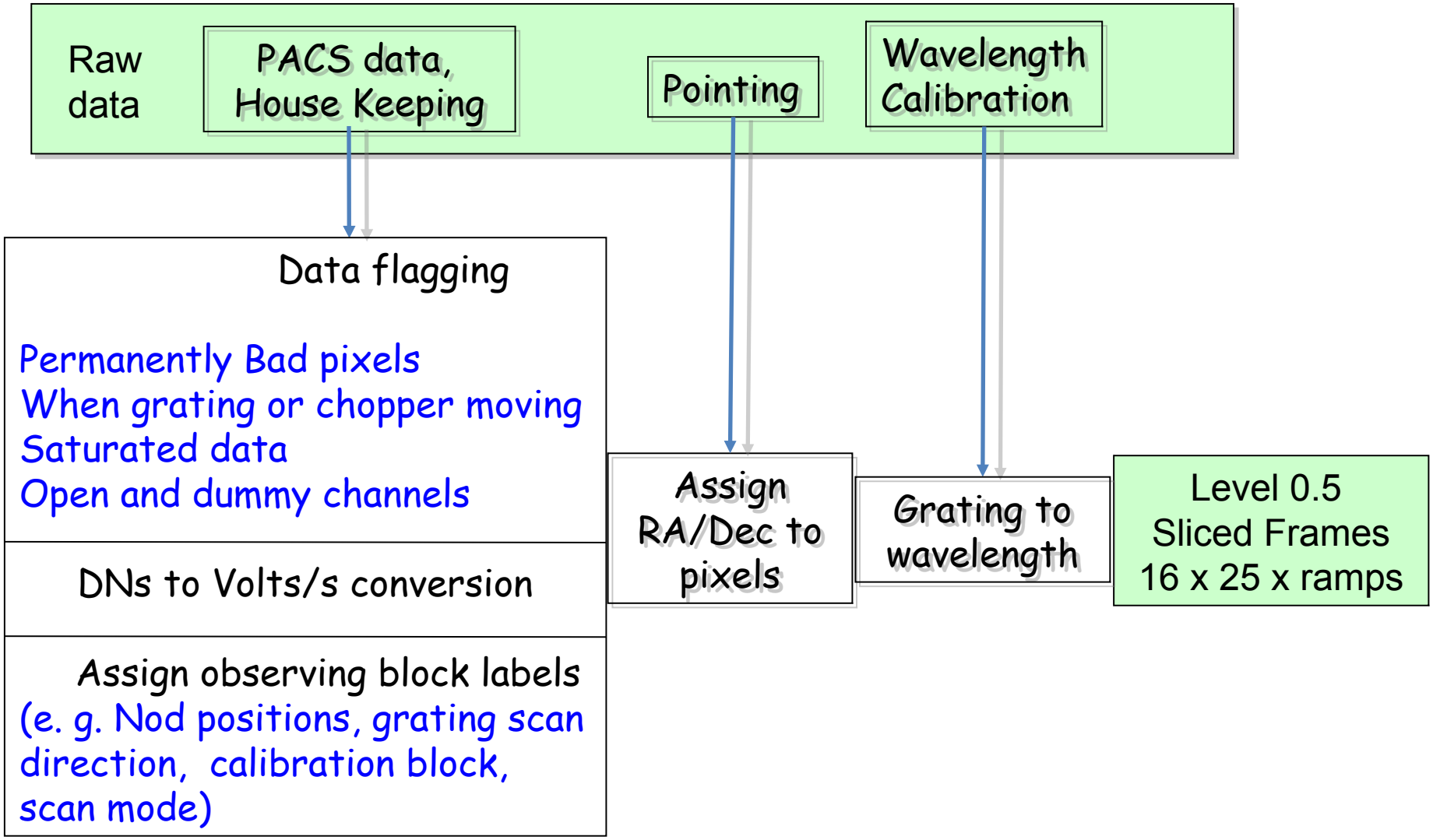
print obs.meta["calVersion"] shows the calibration used in the archive

Step 3

Run the 0 → 0.5 pipeline

Basic calibration (pointing, wavelength calibration, slicing)

Level 0 → 0.5

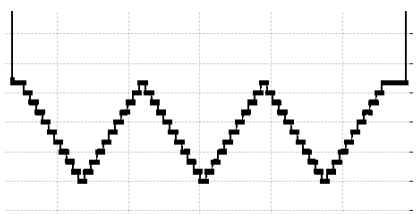

Raw data
PACS data, House Keeping
Pointing
Wavelength Calibration
Data flagging
Permanently Bad pixels
When grating or chopper moving
Saturated data
Open and dummy channels
DNs to Volts/s conversion
Assign observing block labels
(e. g. Nod positions, grating scan direction, calibration block, scan mode)
Assign RA/Dec to pixels
Grating to wavelength
Level 0.5
Sliced Frames
16 x 25 x ramps

Check: level 0

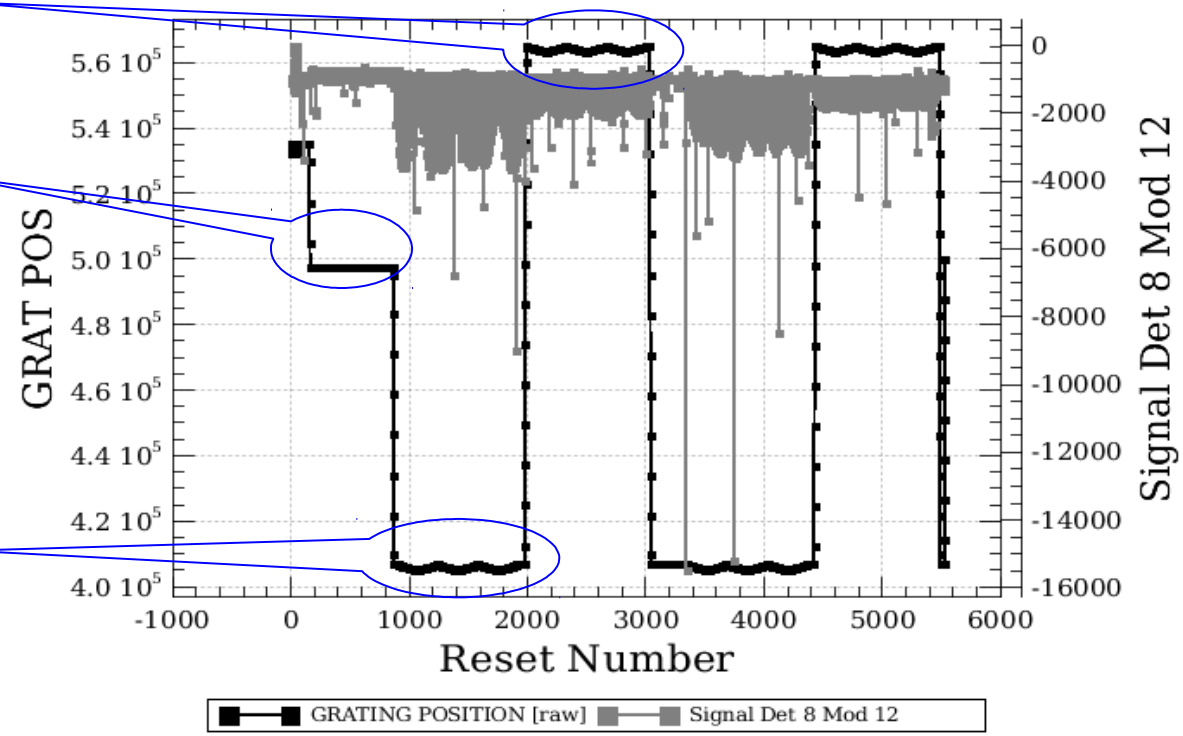
From now on, we will step through the script line by line using the green arrow on the menu bar. The first step consists in extracting the 0-level products from the observation context.

2nd line

Calibration block



1st line, 3 repeats

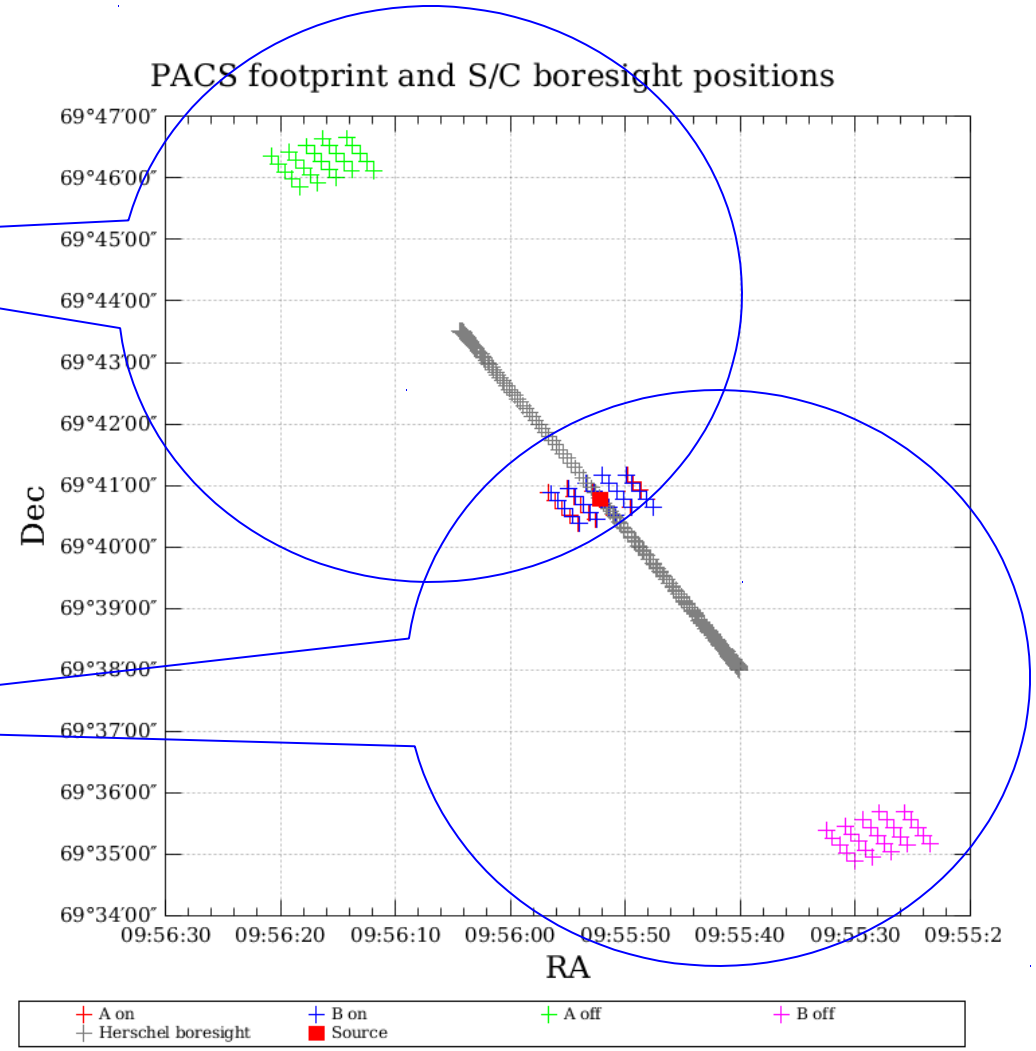


In our case, after the calibration block, we can identify two different lines observed 3 times in the two nod positions.

Check: footprint

Nod A

Nod B



Check: before slicing

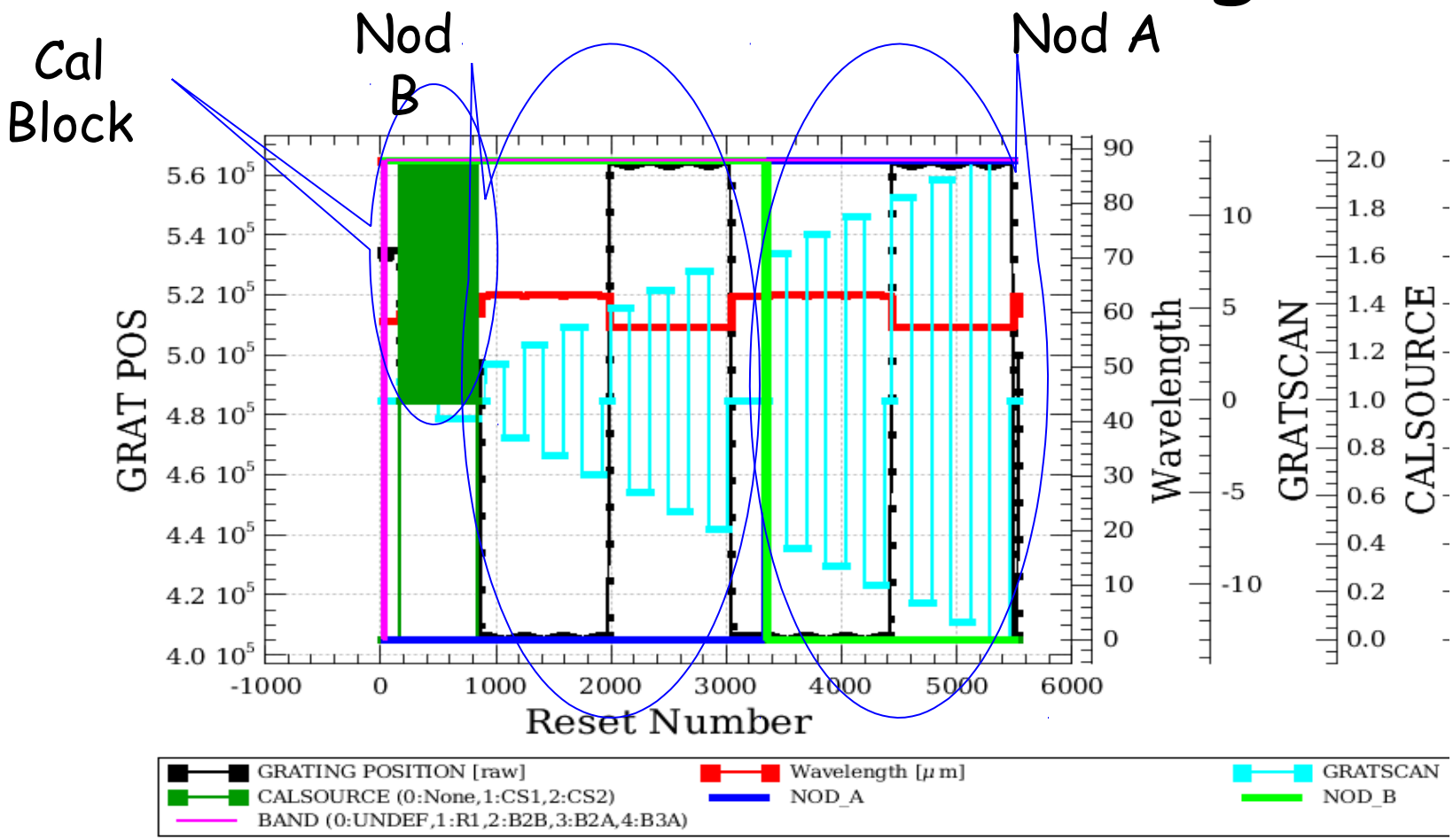
Only 1 slice

History | Log | Console

```
HIPE> slicedFrames = flagChopMoveFrames(slicedFrames, dmcHead=slicedDmcHead, calTree=calTree)
HIPE> slicedFrames = flagGratMoveFrames(slicedFrames, dmcHead=slicedDmcHead, calTree=calTree)
HIPE> if verbose:
        # Summary of the slices
        slicedSummary(slicedFrames)
        # Summary of the active (1) and inactive (0) status of every Mask
        maskSummary(slicedFrames)
        # Plot the instrument movements, without the signal
        p1 = slicedSummaryPlot(slicedFrames,signal=0)
noSlices: 1
noCalSlices: 1
noScienceSlices: 0
slice#  isScience  nodPosition   nodCycle  rasterId  lineId      band                        dimensions      wavelengths
onSource  offSource
0       false      ["","A","B"]  0         0 0       [0,1,2,3]   ["B2B","B3A","UNDEF"]       [18,25,5536]    57.213 - 88.133  both
   both
Nb of slices: 1
Slice  0
BLINDPIXELS          1
SATURATION           1
RAWSATURATION        0
NOISYPIXELS          0
BADPIXELS            1
UNCLEANCHOP          1
GRATMOVE             1
Slice edges:  [0,5536]
HIPE>
```

Jython Interpreter 100% | 1420 of 28106 MB

Check: before slicing



There are two lines (two wavelengths in red). Grating scans are numbered positive if upscans and negative if downscans.

Slicing

*ChopNodLineScan.py

```
# Slice the data by Line/Range, Raster Point, nod position, nod cycle,
# on/off position and per band.
# The parameter removeUndefined is for cleaning purposes
# Virtually any column in the "BlockTable" can be used as a SlicingRule, but do
# not modify the SlicingRules if you are not 100% aware of what you are doing!
rules = [SlicingRule("LineId",1),SlicingRule("RasterLineNum",1),SlicingRule("RasterColumnNum",1),\
         SlicingRule("NoddingPosition",1),SlicingRule("NodCycleNum",1),SlicingRule("IsOutOfField",1),SlicingRule("Band",1)]
slicedFrames = pacsSliceContext(slicedFrames, slicingRules = rules, removeUndefined=1)
```

The slicing of the data is performed according to rules made explicit in the pipeline. In our example, two lines are observed in two nodding positions. So, we expect 4 slices plus an initial slice containing the calibration block.

Check: after slicing

5 slices !

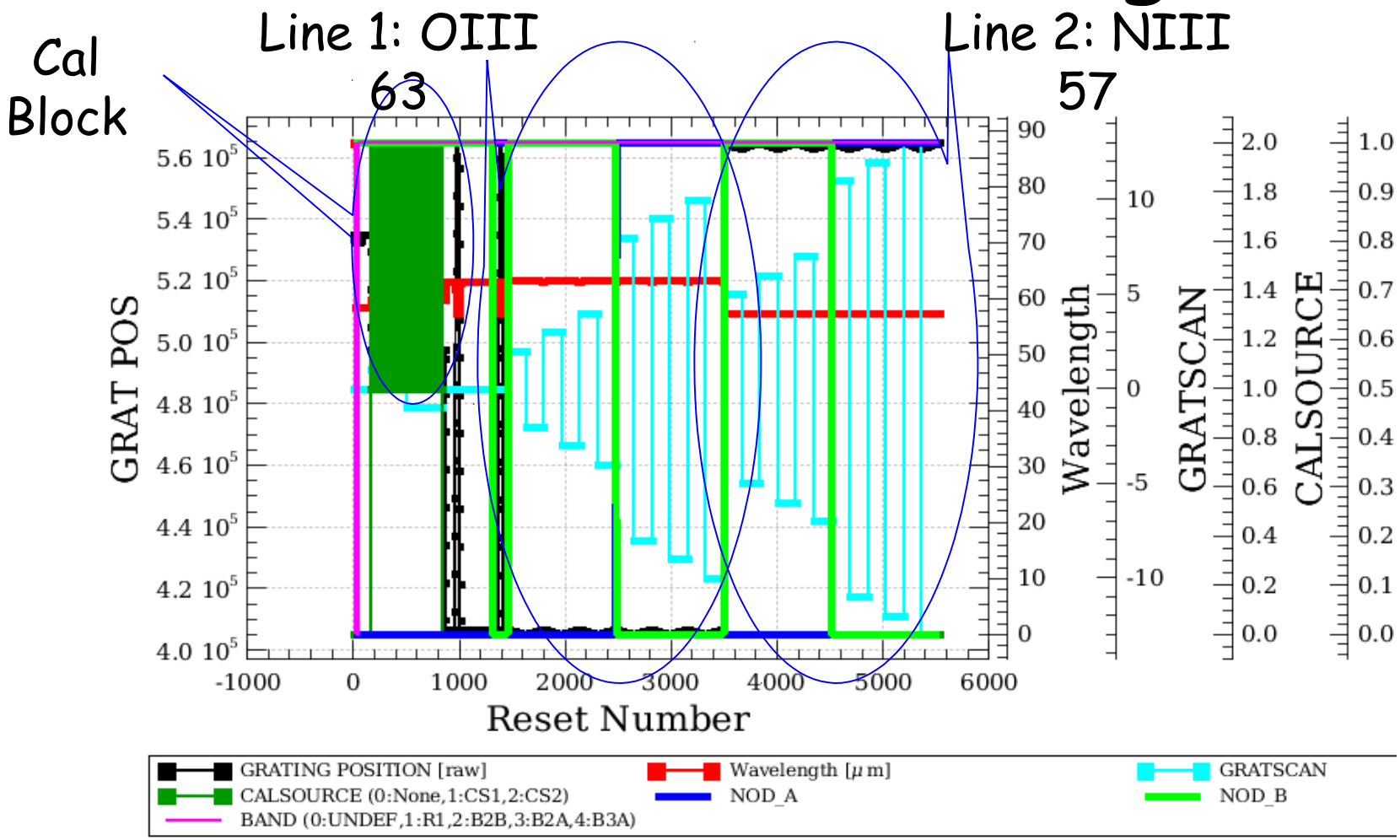
Line 1 – B & A nodes

Line 2 – B & A nodes

History | Log | Console ×

```
# get/saveObservation are also accepted by saveSlicedCopy and readSliced.
# See their description given above (for getObservation).
noSlices: 5
noCalSlices: 1
noScienceSlices: 4
slice#  isScience  nodPosition   nodCycle  rasterId  lineId      band                    dimensions      wavelengths
onSource  offSource
0       false      ["B"]         0         0 0       [1]         ["B3A"]                 [18,25,679]     59.816 - 60.067  no
   no
1       true       ["B"]         1         0 0       [2]         ["B3A"]                 [18,25,1019]    63.093 - 63.379  both
   both
2       true       ["A"]         1         0 0       [2]         ["B3A"]                 [18,25,1019]    63.093 - 63.379  both
   both
3       true       ["B"]         1         0 0       [3]         ["B3A"]                 [18,25,1019]    57.213 - 57.548  both
   both
4       true       ["A"]         1         0 0       [3]         ["B3A"]                 [18,25,1019]    57.213 - 57.548  both
   both
Slice edges:  [0,679,1698,2717,3736,4755]
HIPE>
```

Check: after slicing



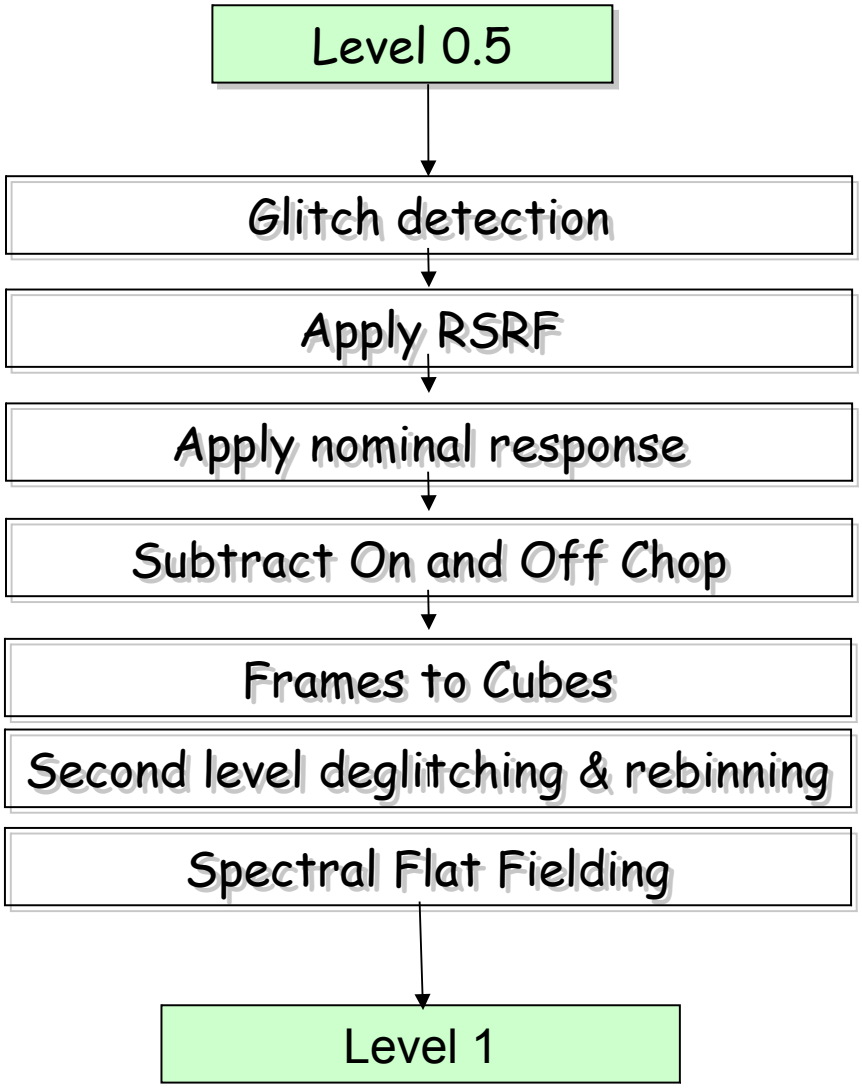
There are four slices (calibration, nod A and B for the 1st line, nod A and B for the 2nd line).

Step 4

Run the 0.5 → 1 pipeline

Glitch detection, chop differentiation, RSRF, flat

Level 0.5 → 1


Level 0.5
Glitch detection
Apply RSRF
Apply nominal response
Subtract On and Off Chop
Frames to Cubes
Second level deglitching & rebinning
Spectral Flat Fielding
Level 1

Glitch detection

You can check manually the points flagged as glitches or masked for other reasons using the maskviewer

Select a pixel by clicking on it

Select a mask

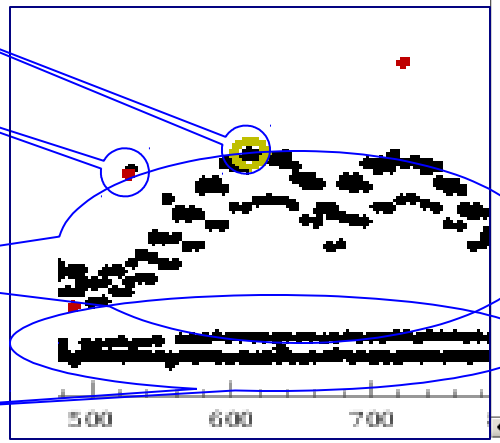
Select a frame

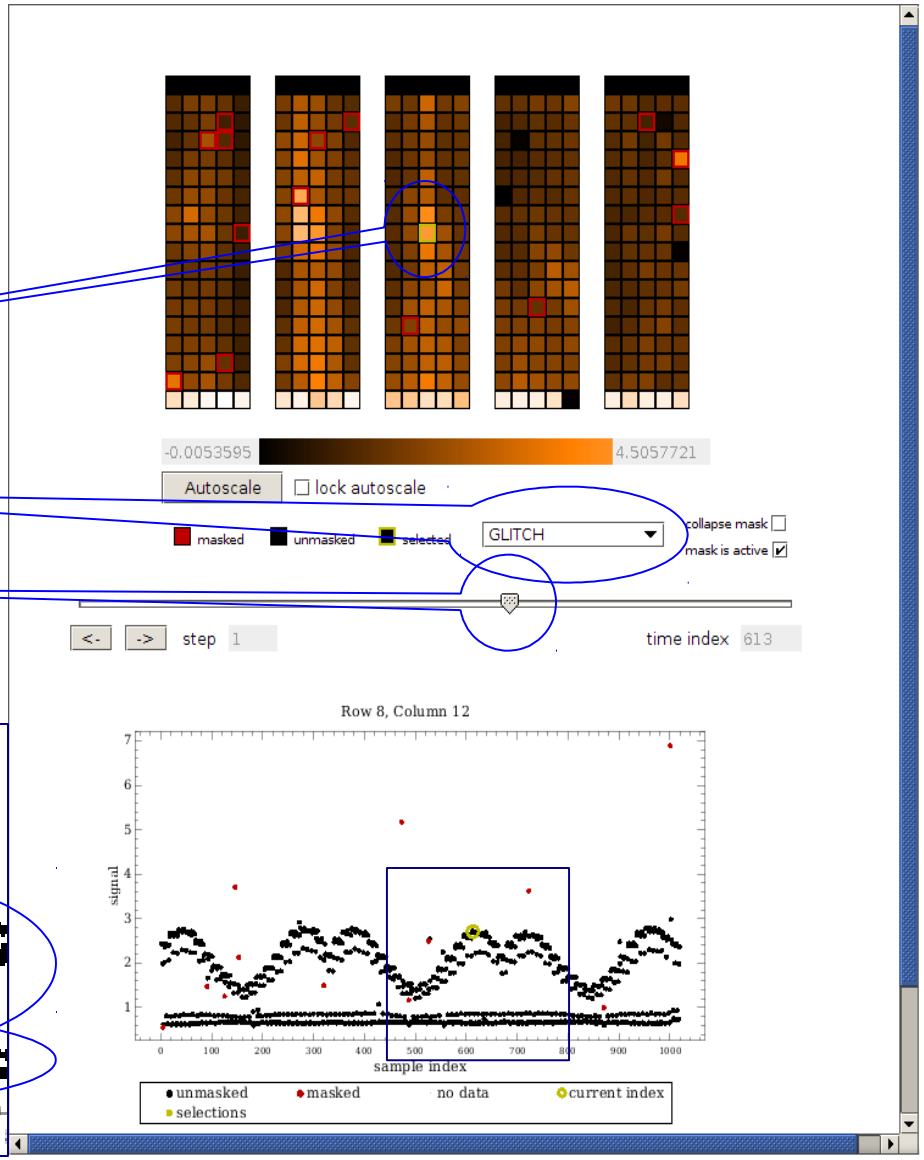
Current frame

Masked glitch

ON signal

OFF signal

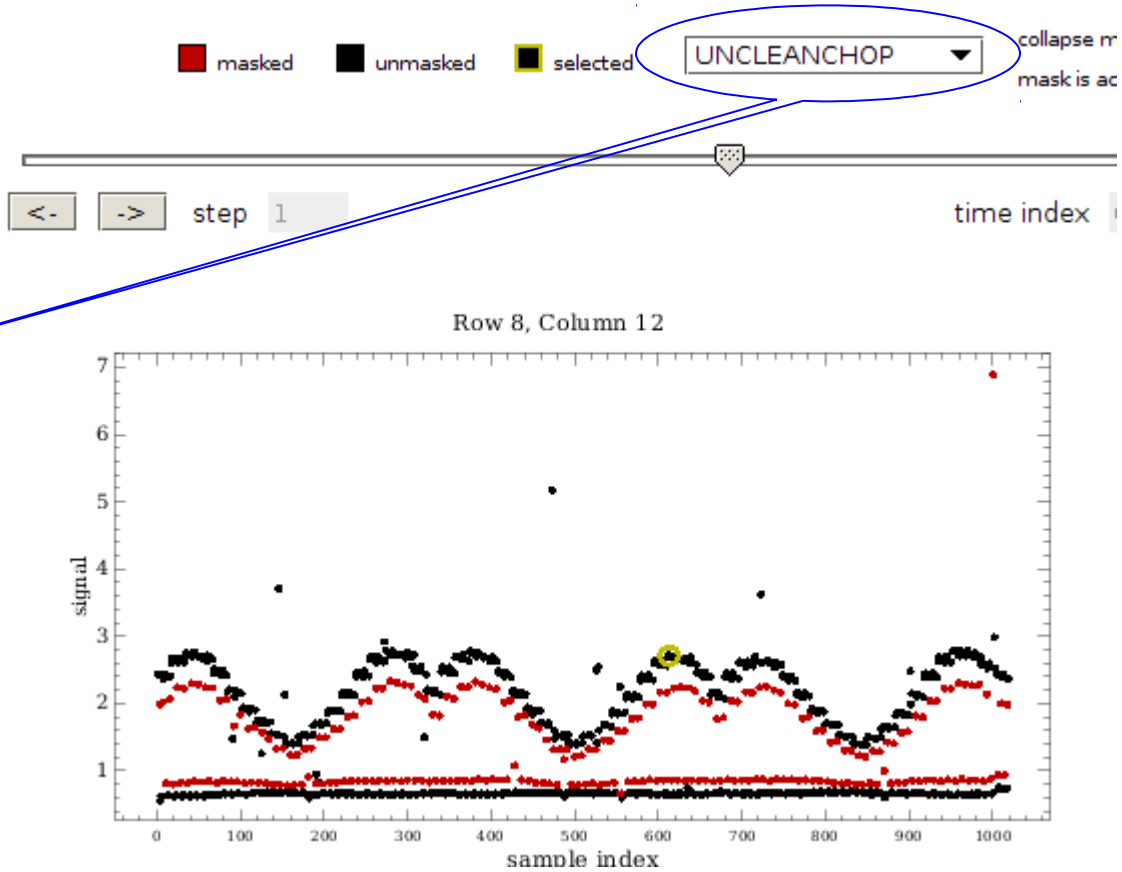




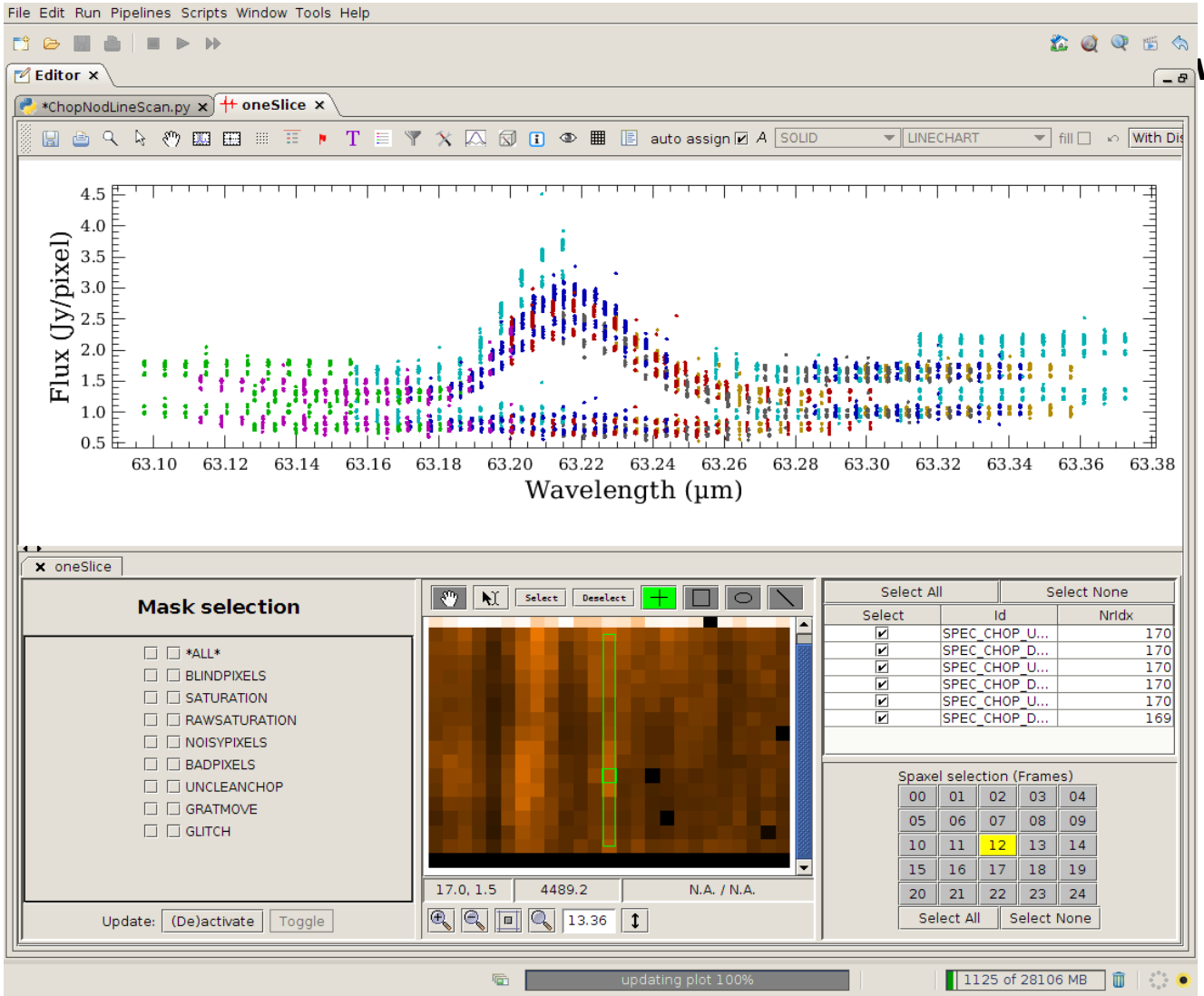
More masks

It is possible to explore other masks

Select unclean chop



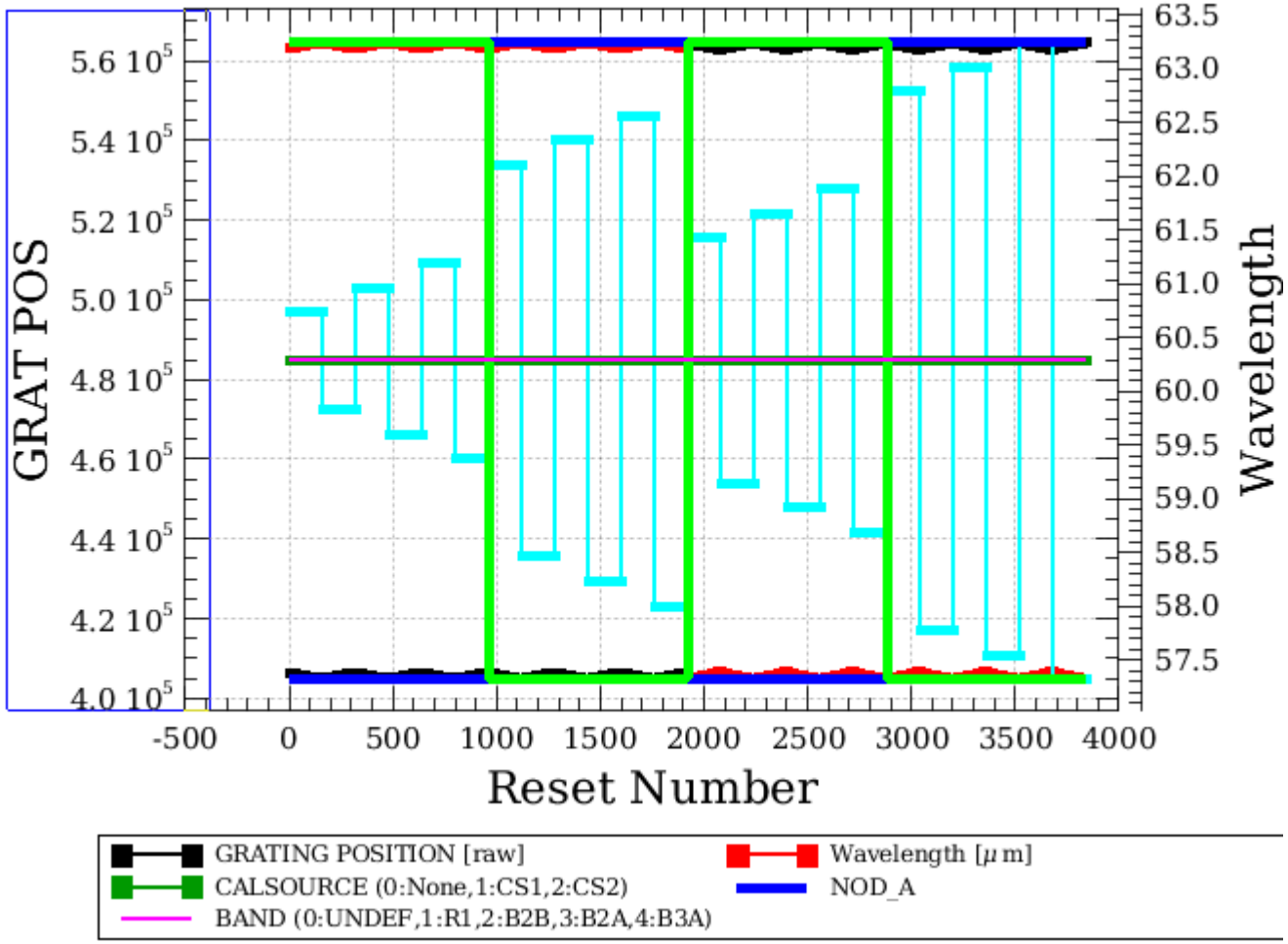
In this case, it is clear why there is a second group of points for the ON and OFF positions. These corresponds to signals obtained when the chopper was not yet in the correct position.



A further inspection of your data is now possible using the Spectrum Explorer. Several options are available such as selection of pixels and different masks for the first slice.

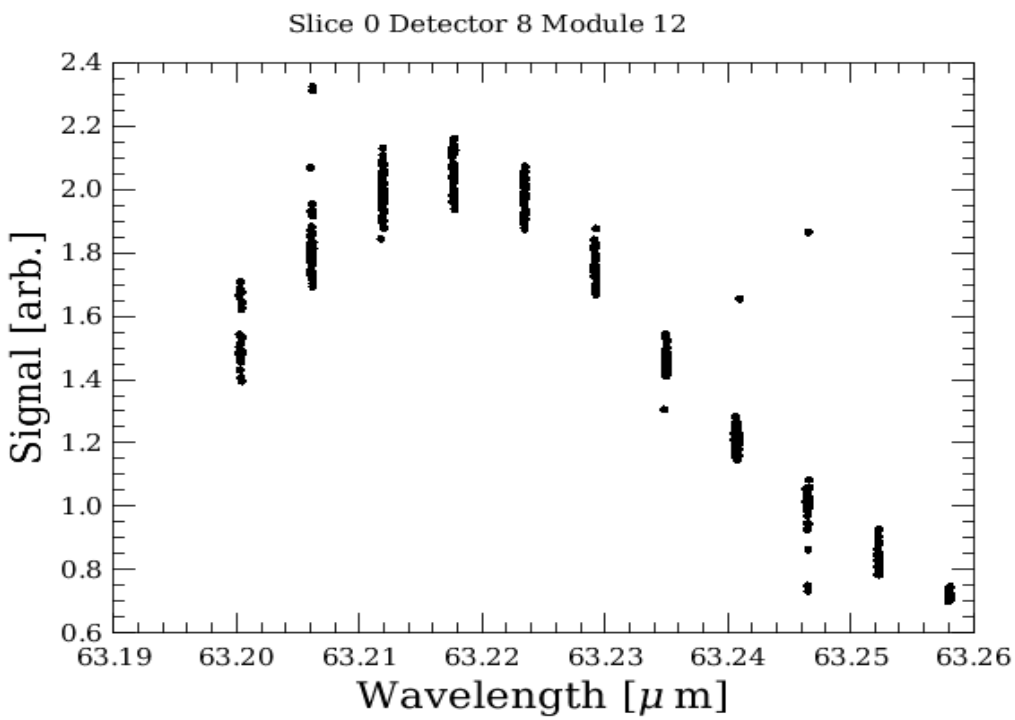
Chop differentiation

After chop differentiation, the calibration block is excluded from the data



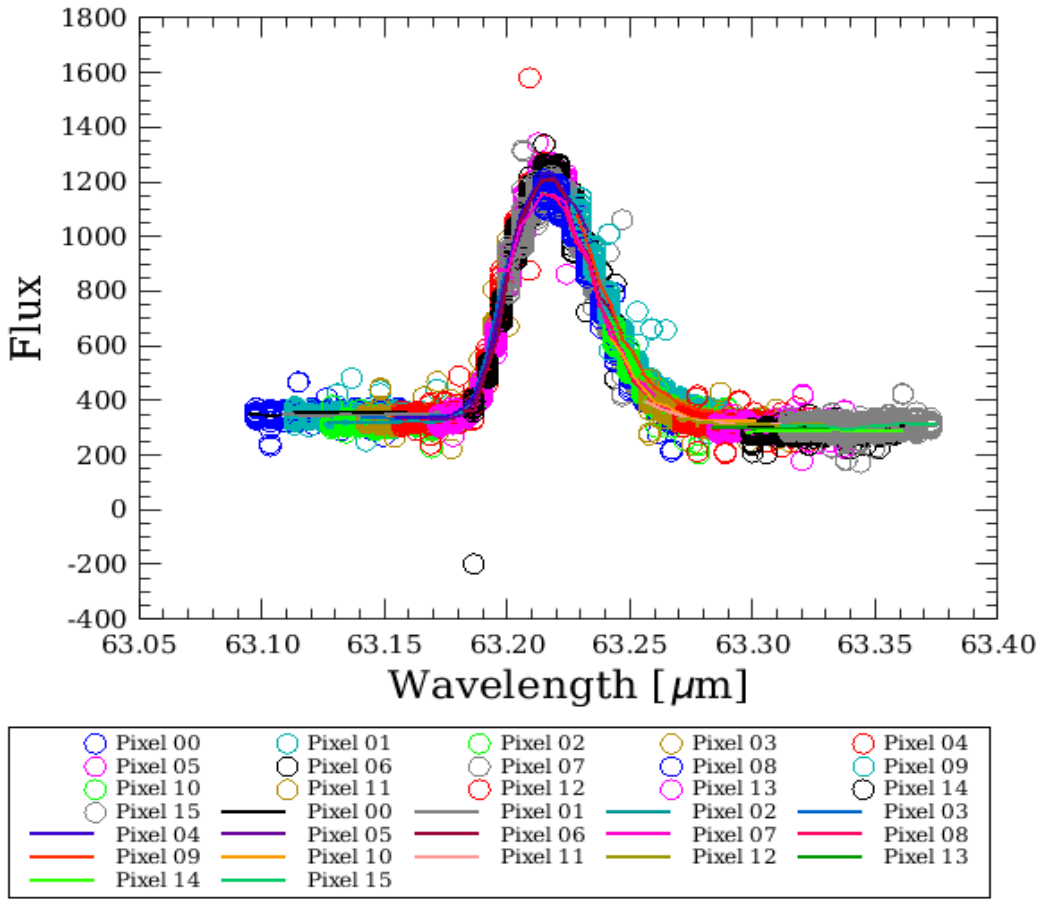
Chop differentiation

The data are only on the ON position (OFF being subtracted)



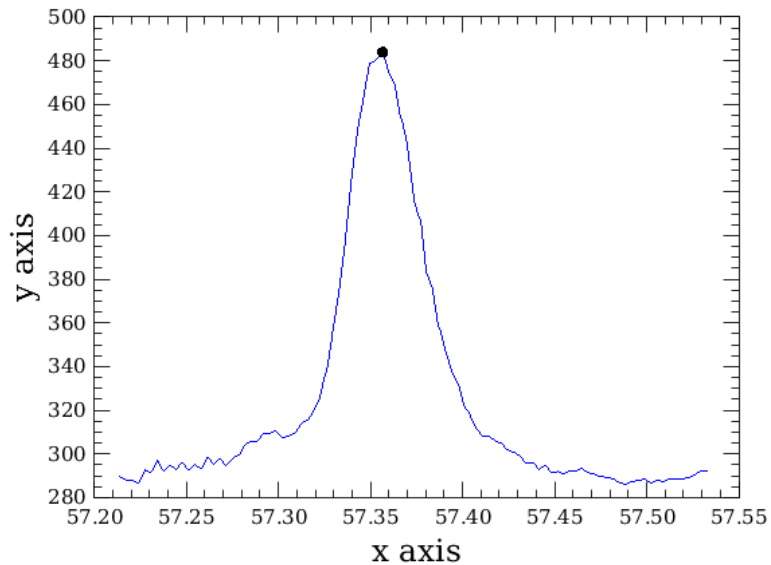
RSRF and response

After applying RSRF and response corrections
we have a first look at the spectrum



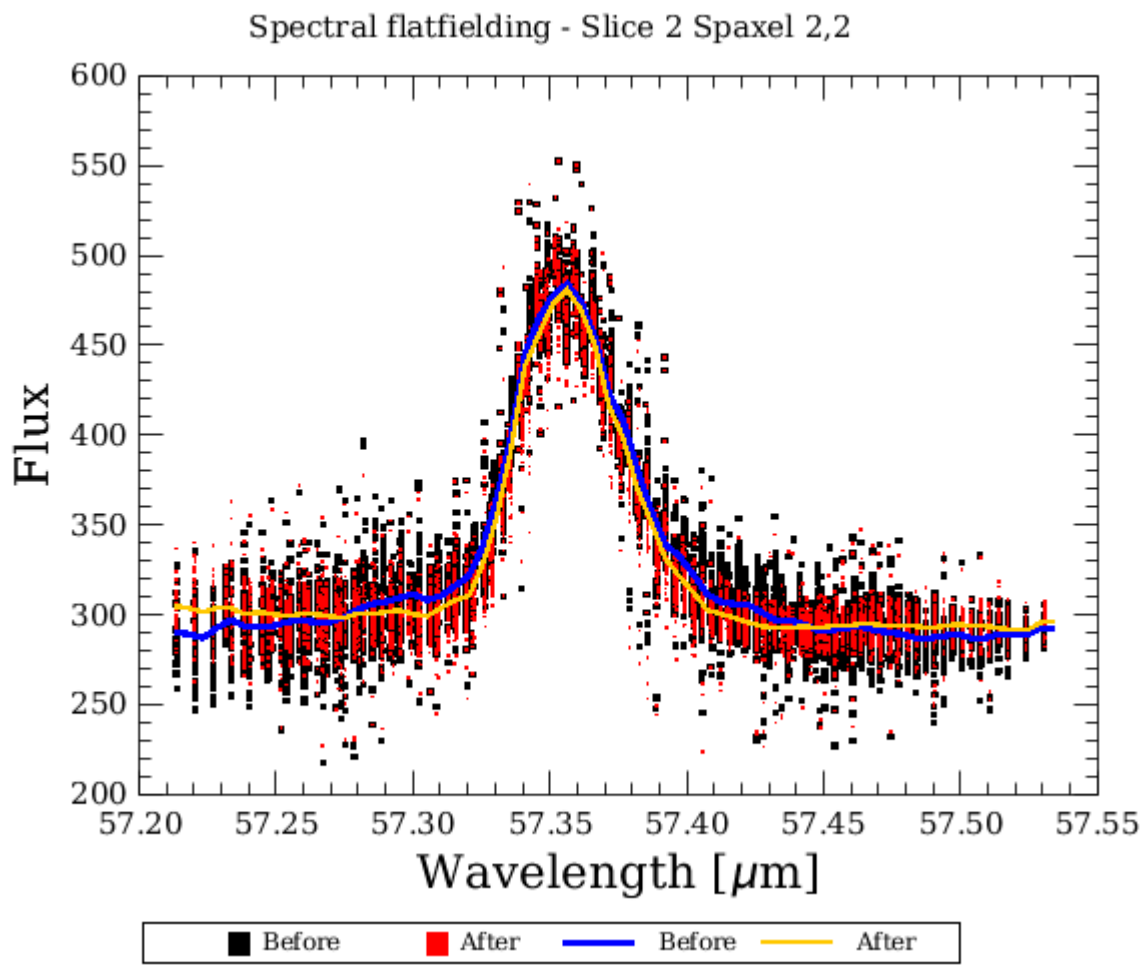
nhsc.ipac.caltech.edu/helpdesk

Spectral FlatField

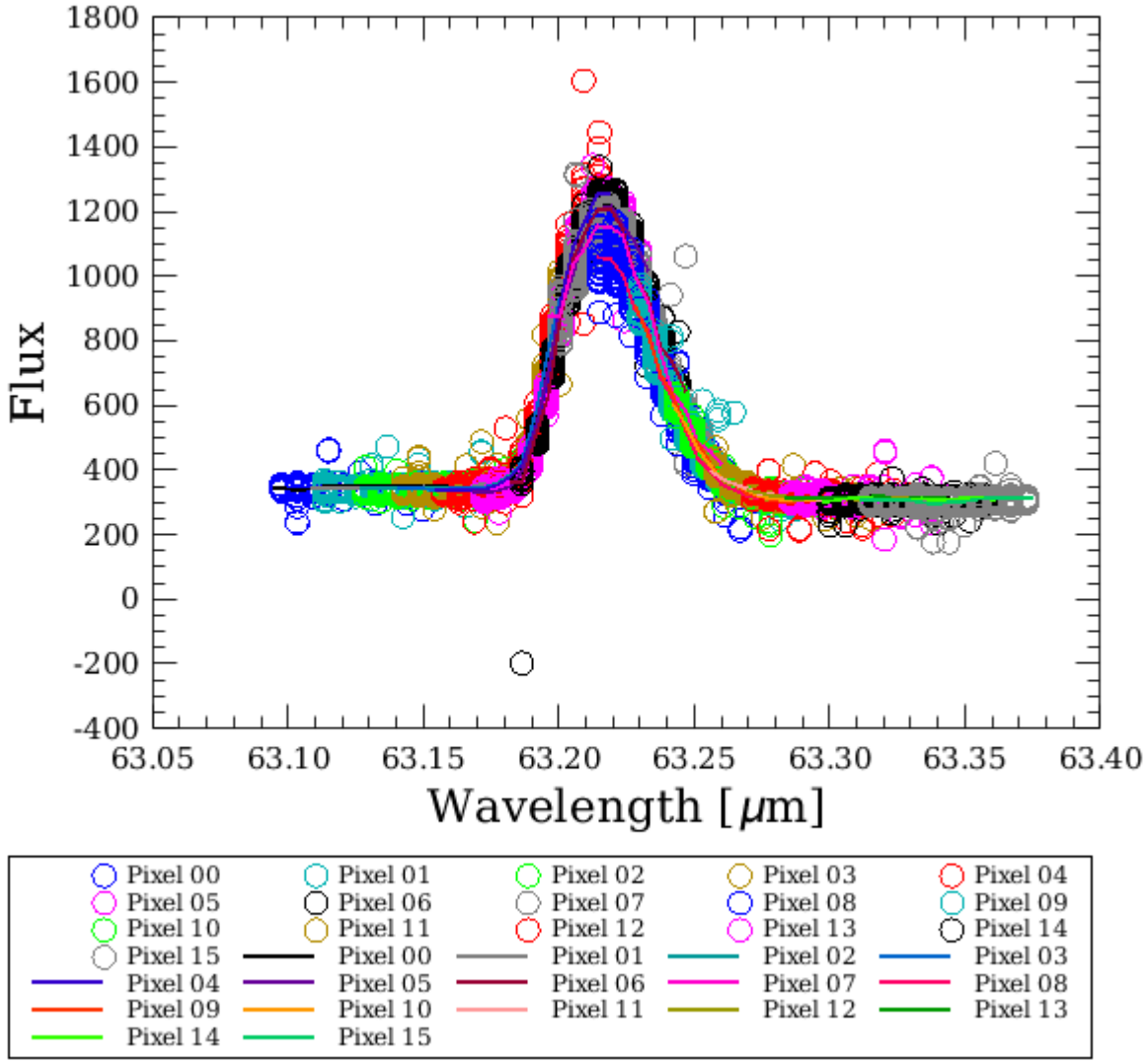


As a default, the code will search for lines in all the pixels and then mask them before computing the spectral flat field.
It is possible to give directly the list of lines to be masked via the parameter lineList = [57.36], for instance.

Spectral FlatField



nhsc.ipac.caltech.edu/helpdesk



At this point, the frames are converted in calibrated cubes and we have reached level 1 !