



NHSC/PACS Web Tutorials

Running the PACS Spectrometer pipeline for CHOP/NOD Mode

PACS-301

Level 0 to 1 processing

Prepared by Dario Fadda
February 2011

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

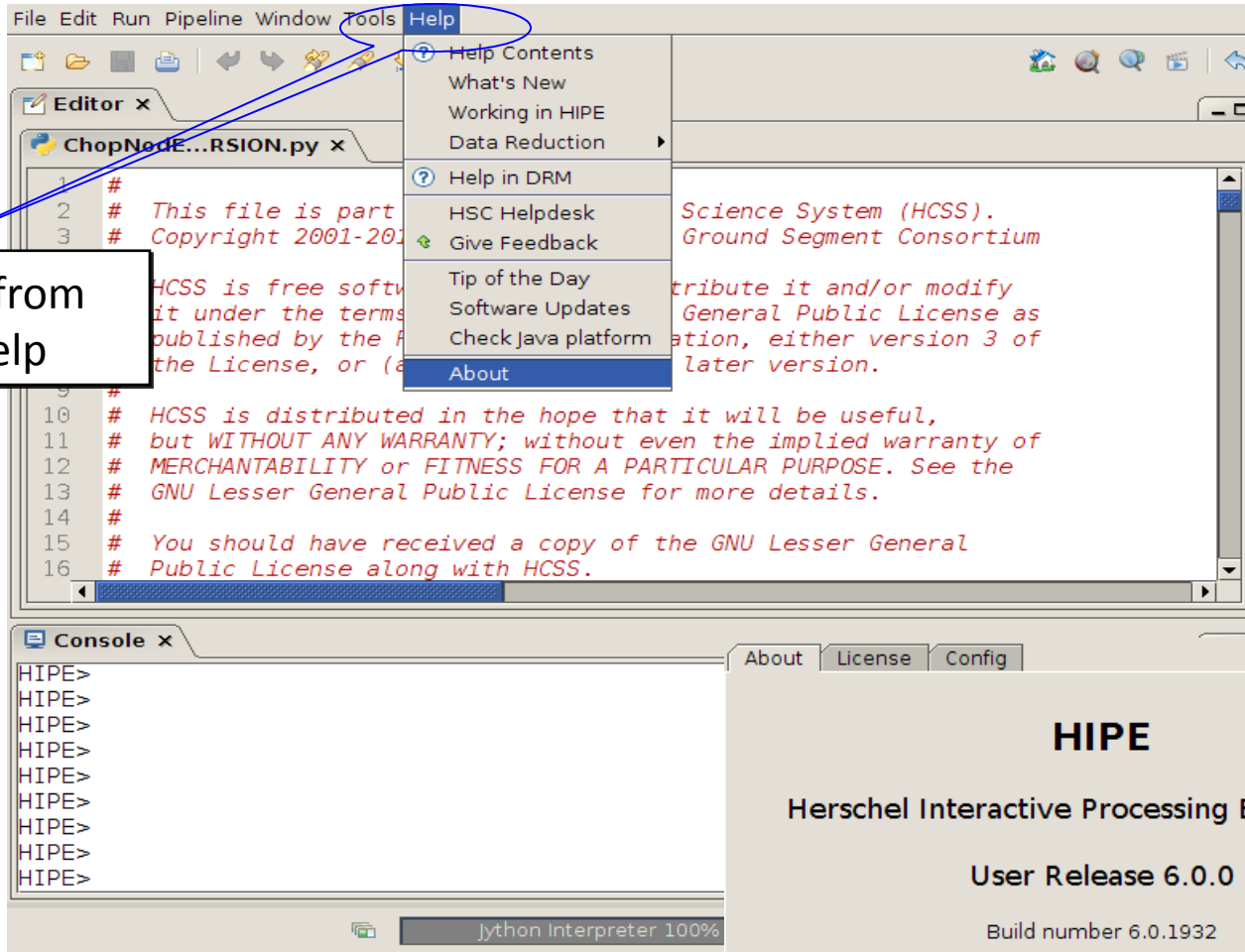
- Step 1** Check HIPE version and memory
- Step 2** Setup
- Step 3** Run the 0 → 0.5 pipeline
- Step 4** Run the 0.5 → 1 pipeline



Step 1

Check HIPE version and memory allocation

The version used for the tutorial is 6.0.1932



Select about from drop down help

A pop-out window with the version appears

To allocate memory, select preferences under edit, then ...

The screenshot shows a software application window with a menu bar (File, Edit, Run, Pipeline, Window, Tools, Help) and a toolbar. The 'Edit' menu is open, showing options like Undo, Redo, Cut, Copy, Paste, Delete, Open, Open With, Send to, Select all, Incremental search, Find/replace..., Go to line, Toggle comment, Shift right, Shift left, and Preferences (highlighted). The main text area contains red text, including a copyright notice for the Herschel Common Science System (HCSS) and the GNU Lesser General Public License. Below the text area is a 'Console' window with multiple lines of 'HIPE>' prompts. At the bottom of the application window, a status bar shows 'jython interpreter 100%' and a memory usage indicator '706 of 6372 MB' which is circled in blue.

Memory used and available

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

General > Startup & Shutdown

Maximum memory: MB ⓘ To be applied the next execution of HIPE

- Show tips at startup
- Check cache for locks
- Save variables on exit
 - Ask which variables to restore
- Inform about dump files
- Check Java platform
- Check for updates

Restore Defaults Apply

Advanced... Import... Export... OK Cancel

The allocated memory should be smaller than the total RAM of your computer

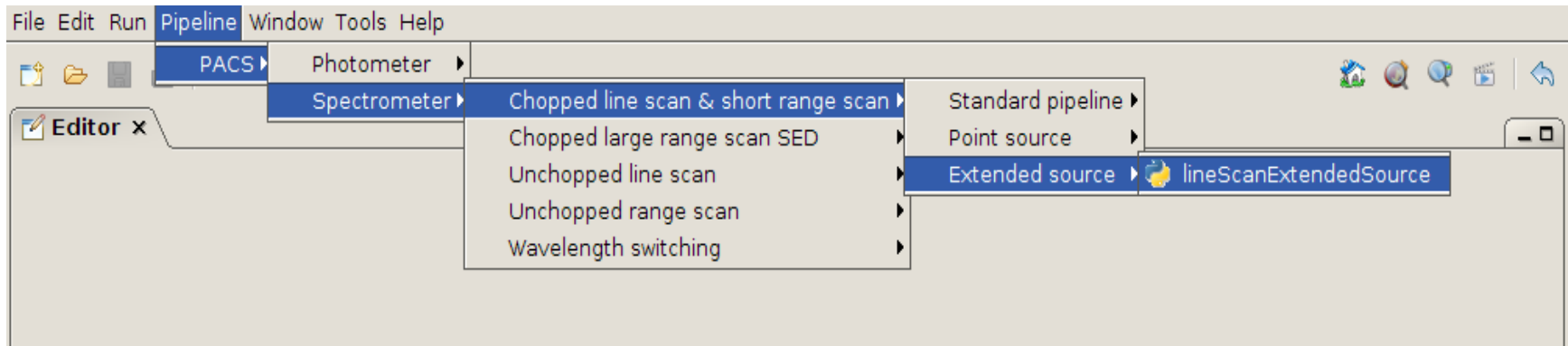
Step 2

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.

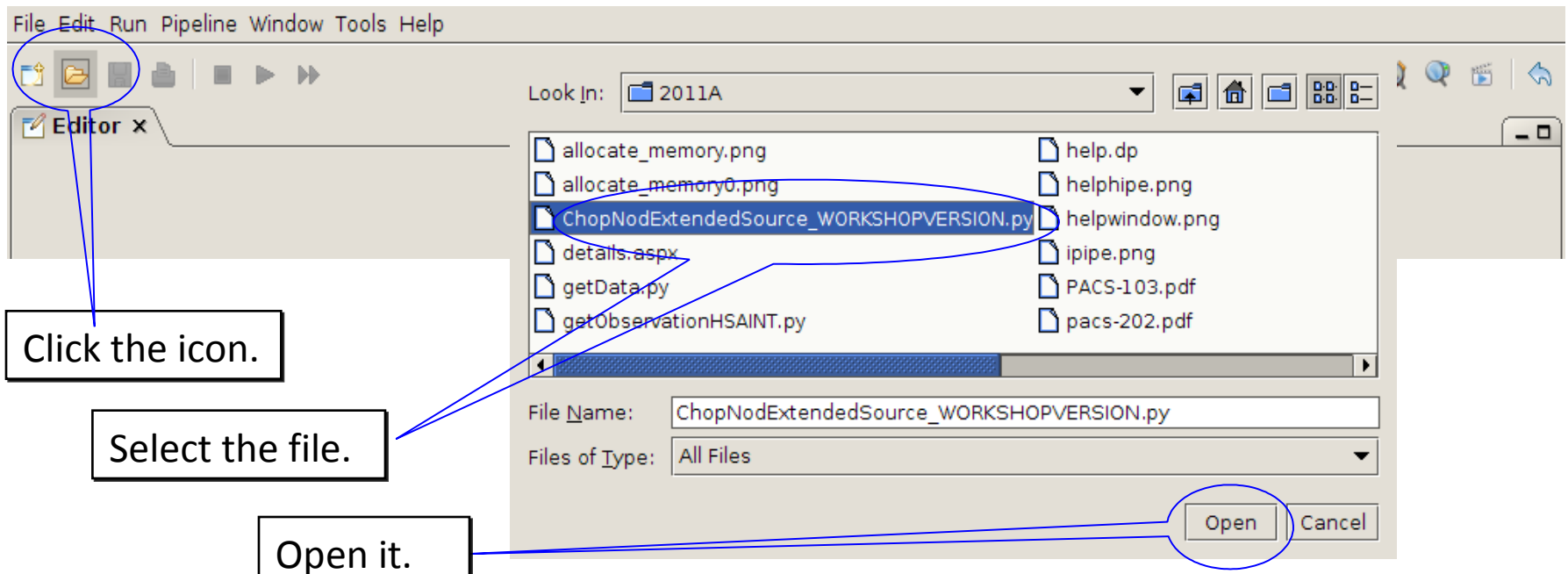


We added a few lines to load the correct set of data and make a few further checks before starting the reduction.

Loading the script

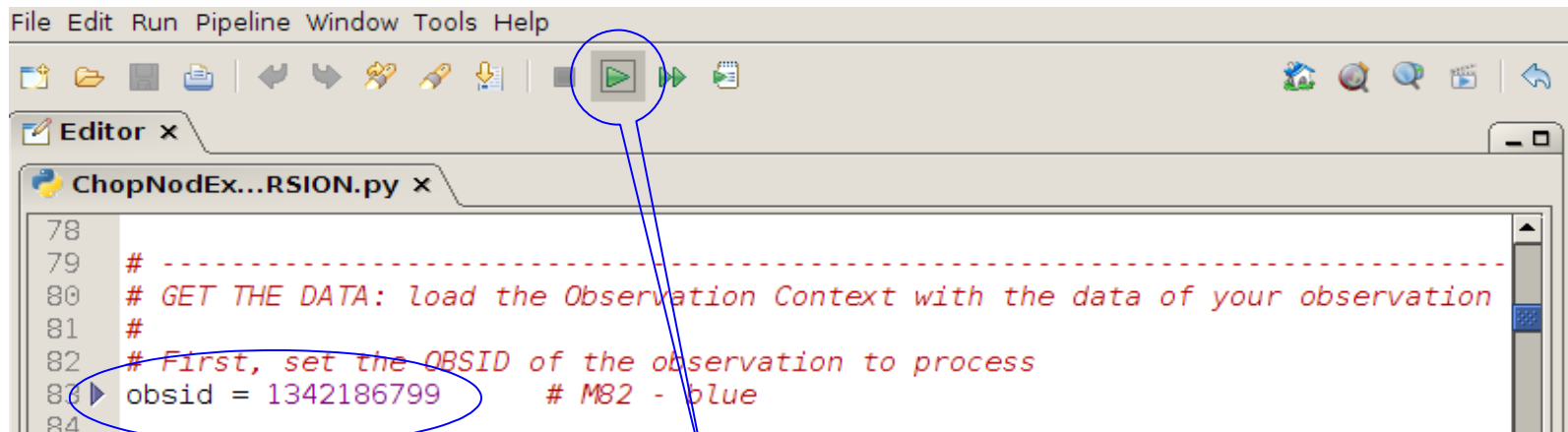
The modified script can be found at the IPAC website:
ADD WEBSITE HERE

To load the script into your hiipe session, just click on the loading icon as shown in the figure. The search the location where you put the file using the pop-up window and finally load it into the session.



Loading the observation

Once the file is loaded, one can simply step through the lines to execute it one by one. In this tutorial, we will explain how to modify some lines to explore different observations and lines and to check the results of the main operations on the data. 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.



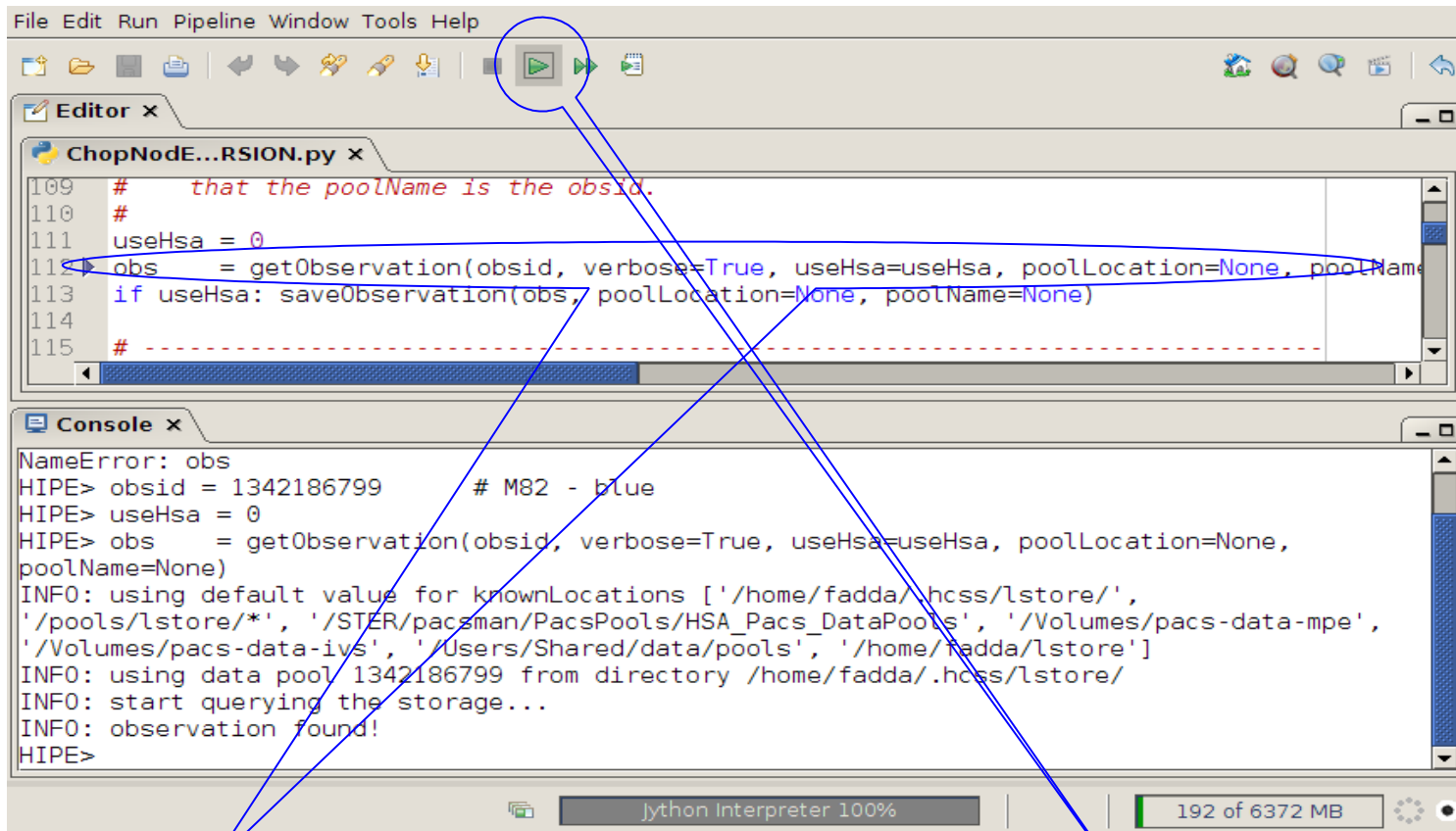
```
File Edit Run Pipeline Window Tools Help
ChopNodEx...RSION.py x
78
79 # -----
80 # GET THE DATA: load the Observation Context with the data of your observation
81 #
82 # First, set the OBSID of the observation to process
83 ▶ obsid = 1342186799 # M82 - blue
84
```

Click on this line.

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

Check: observation summary

```
Console x
HIPE> if verbose: obsSummary(obs)

Observation summary
OBSID: 1342186799
AOR Label: NearGalPACS-SB-01-blue
Proposal: SDP_esturm_3
Target: M82
Redshift: 6.77E-4 (z)
Observing Day (OD): 178
Observing started: Sun Nov 08 07:31:26 PST 2009
Total duration (incl. slew): 582.0 seconds
AOT: PacsLineSpec
Observing mode: Pointed, Chop/Nod
Sampling density for: bright lines
Chopper throw: large
Nod cycles: 1

Observation block summary
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

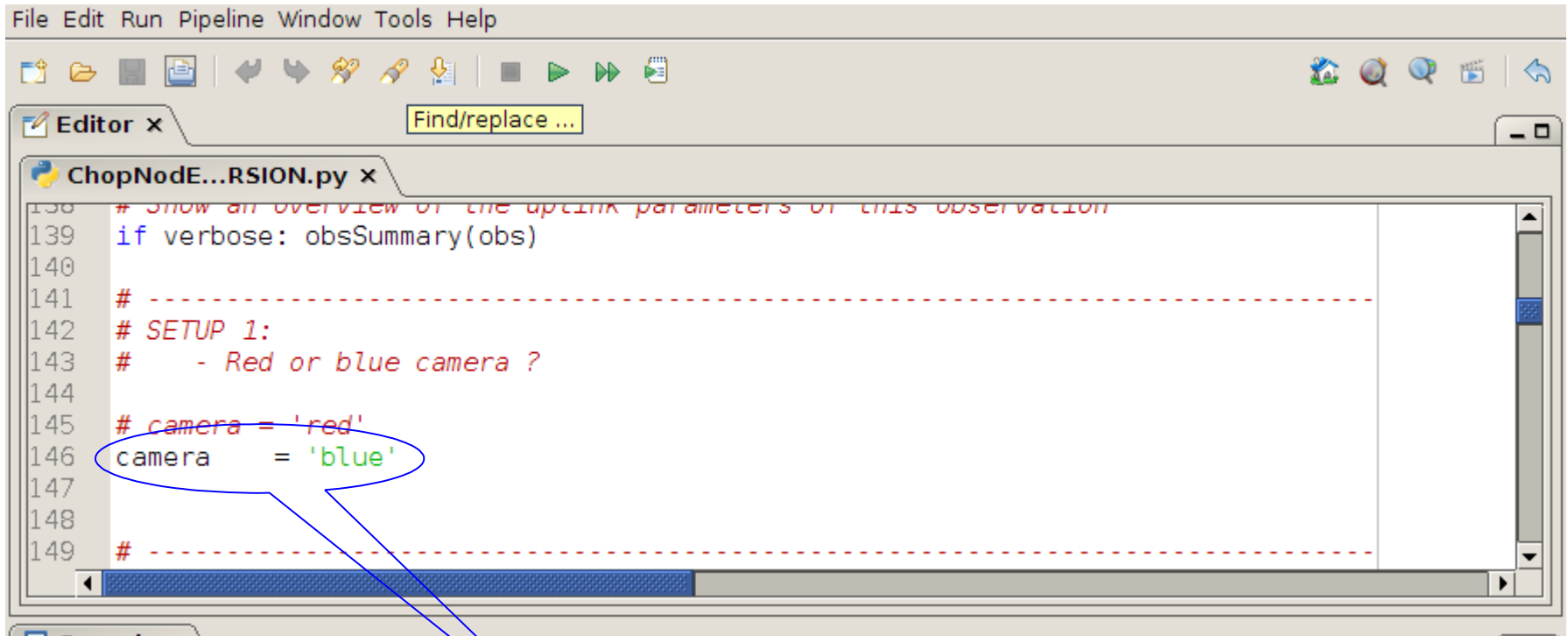
System configuration
SPG version of archived products: v4.1.0
SPG products creation date: Mon Oct 11 12:45:10 PDT 2010
Mission configuration: MC_HPhase3_P33ASTR_S34ASTR_SDP

Archived products status
Level 0 status: LEVEL0_PROCESSED
Level 1 status: LEVEL1_PROCESSED
Level 2 status: LEVEL2_PROCESSED
```

We will select: camera = 'blue'

Setting the camera

Once we decide the line to explore, we can set the camera to blue or red.



```
File Edit Run Pipeline Window Tools Help
Find/replace ...
ChopNode...RSION.py x
138 # show an overview of the uplink parameters of this observation
139 if verbose: obsSummary(obs)
140
141 # -----
142 # SETUP 1:
143 #   - Red or blue camera ?
144
145 # camera = 'red'
146 camera = 'blue'
147
148
149 # -----
```

We select camera = 'blue'

Setting the calibration tree

Finally, we set the calibration tree.

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

Version 8 and later ones incorporate different scale factors and include improved flat field corrections.

```

File Edit Run Pipeline Window Tools Help
ChopNodeE...RSION.py x
155 # From that calibration tree, certain calibration files are used by each task.
156 # If you print the common and spectrometer branches of the tree, you can see
157 # which versions of the calibration files will be used.
158 calTree = getCalTree(obs=obs)
159 if verbose:
160     print calTree
161     print calTree.common
162     print calTree.spectrometer
163
Console x
PACS Calibration Tree
Model : FM
Scope : BASE
Version : 13
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
obcpDescription : FM, 4
siam : FM, 5
timedep : FM, 13

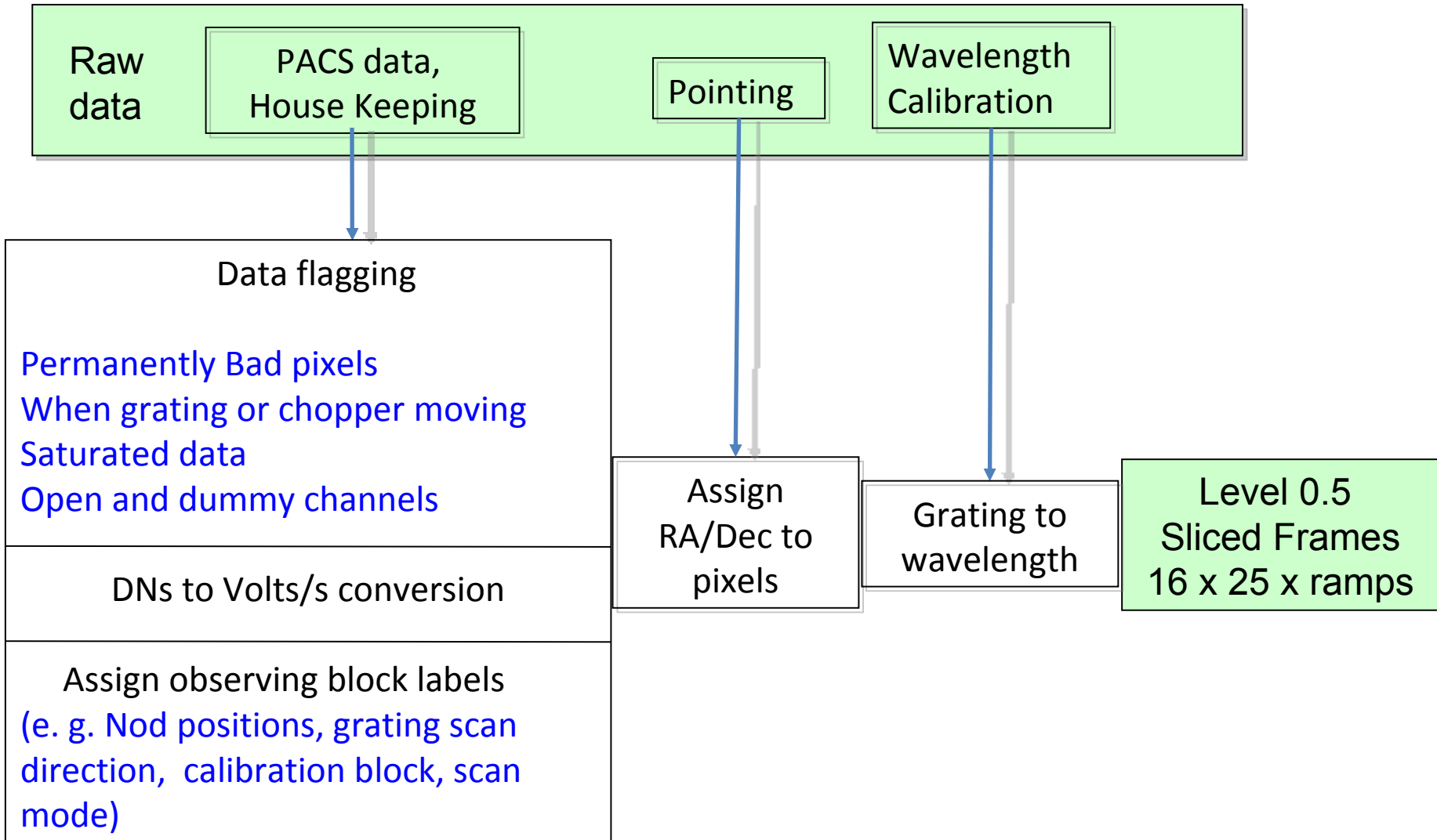
PacsCalSpec Calibration Products:
absoluteCapacitance : FM, 3
arrayInstrument : FM, 8
badPixelMask : FM, 1
calSourceFlux : FM, 4
capacitanceRatios : FM, 5
chopperThrowDescription : FM, 2
crosstalkMatrix : FM, 1
darkCurrent : FM, 2
    
```

Step 3

Run the 0 → 0.5 pipeline

Basic calibration (pointing, wavelength calibration, slicing)

Level 0 → 0.5

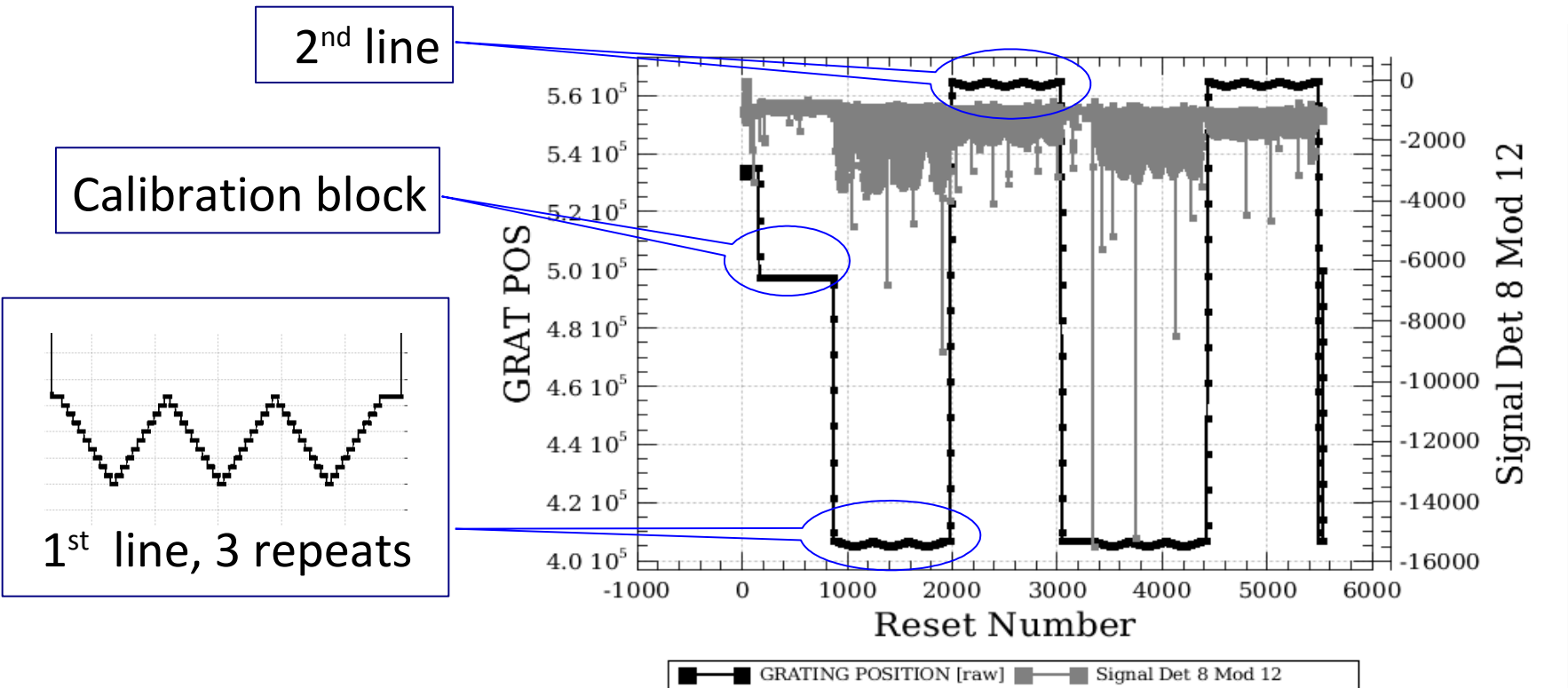


```
ChopNodEx...RSION.py x
164
165 # -----
166 # Extract the level-0 products from the ObservationContext
167 #     Get and prepare the level0 product that you will pipeline on.
168 #     First we copy the metadata from the ObservationContext to the level0 product
169 #     Then we extract the level0 product from the ObservationContext
170 pacsPropagateMetaKeywords(obs,'0', obs.level0)
171 level0 = PacsContext(obs.level0)
172
173 # For your camera, extract the Frames (scientific data), the rawramps (raw data
174 # for one pixel), and the DMC header (the mechanisms' status information,
175 # sampled at a high frequency)
176 slicedFrames = SlicedFrames(level0.fitted.getCamera(camera).product)
177 slicedRawRamp = level0.raw.getCamera(camera).product
178 slicedDmcHead = level0.dmc.getCamera(camera).product
179
180 if verbose:
181     # Get an overview of the basic structure of the data, prior to any processing
182     slicedSummary(slicedFrames)
183     # Plot the grating position & raw signal of central pixel
184     p0 = slicedSummaryPlot(slicedFrames,signal=1)
185
```

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.

Check: level 0

Let's check first the basic structure of the data. It is important to verify if the observation has been performed correctly.



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

Some flagging

```
ChopNodEx...RSION.py x
186 # -----
187 #           Processing           Level 0 -> Level 0.5
188 # -----
189
190
191 # flag the saturated data in a mask "SATURATION" (and "RAWSATURATION": this
192 # uses the raw data we get for some pixels)
193 # used cal files: RampSatLimits and SignalSatLimits
194 slicedFrames = specFlagSaturationFrames(slicedFrames, rawRamp = slicedRawRamp, calTree=calTree)
195
196 # Convert digital units to Volts, used cal file: Readouts2Volts
197 slicedFrames = specConvDigit2VoltsPerSecFrames(slicedFrames, calTree=calTree)
198
199 # Identify the calibration blocks and fill the CALSOURCE Status entry
200 slicedFrames = detectCalibrationBlock(slicedFrames)
201
202 # Add the time information in UTC to the Status
203 slicedFrames = addUtc(slicedFrames, obs.auxiliary.timeCorrelation)
204
```

Adding pointing products

```
ChopNodEx...RSION.py x
204
205 # Add the pointing information to the Status
206 #   Uses the pointing, horizons product (solar system object ephemeris),
207 #   orbitEphemeris products, and the SIAM cal file.
208 # NOTE for SSOs: you should contact the HSC helpdesk to check whether the
209 #   pointing is currently correctly handled for moving objects.
210 slicedFrames = specAddInstantPointing(slicedFrames, obs.auxiliary.pointing, calTree = calTree, orbit
211
212 # Extend the Status of Frames with the parameters GRATSCAN, CHOPPER, CHOPPOS
213 # used cal file: ChopperThrowDescription
214 slicedFrames = specExtendStatus(slicedFrames, calTree=calTree)
215
216 # Convert the chopper readouts to an angle wrt. focal plane unit and the sky
217 # and add this to the Status, used cal files: ChopperAngle and ChopperSkyAngle
218 slicedFrames = convertChopper2Angle(slicedFrames, calTree=calTree)
219
220 # Add the positions for each pixel (Ra and Dec datasets)
221 # used cal files: ArrayInstrument and ModuleArray
222 slicedFrames = specAssignRaDec(slicedFrames, calTree=calTree)
223
224 if verbose:
225     # show footprint of your observation
226     slicedPlotPointing(slicedFrames)
227
```

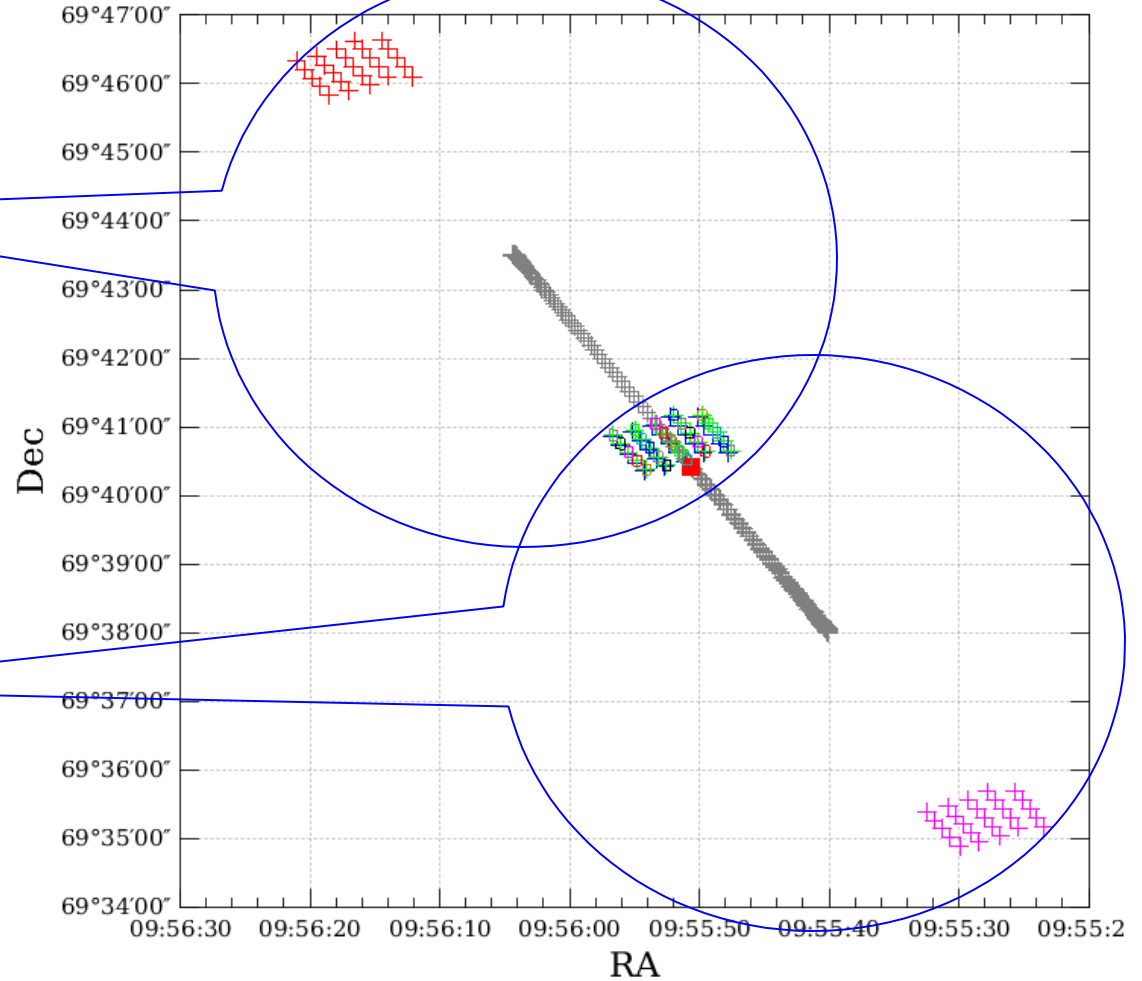
At this point, astrometry is added to the data. At each pixel a coordinate is assigned and we can check if the pointing was performed correctly.

Check: footprint

PACS footprint and S/C boresight positions

Nod A

Nod B



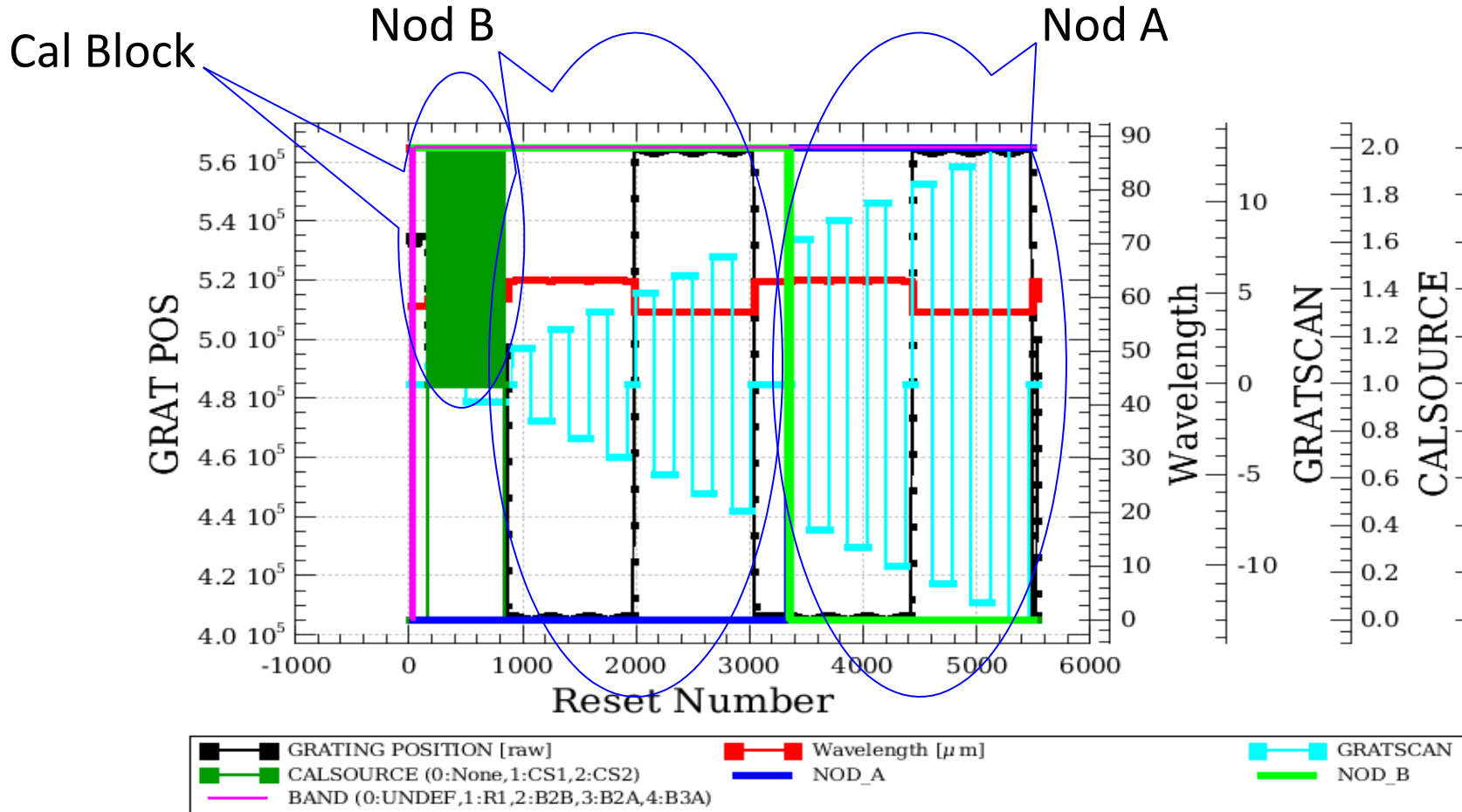
```
ChopNodEx...RSION.py x
228 # Add the wavelength for each pixel (Wave dataset), used cal file: WavePolynomes
229 slicedFrames = waveCalc(slicedFrames, calTree=calTree)
230
231 # Correct the wavelength for the spacecraft velocity.
232 # Uses the pointing, orbitEphemeris and timeCorrelation product.
233 slicedFrames = specCorrectHerschelVelocity(slicedFrames, obs.auxiliary.orbitEphemeris, obs.auxiliary
234
235 # Find the major logical blocks of this observation and organise them in the
236 # BlockTable attached to the Frames; used cal file: ObcpDescription
237 slicedFrames = findBlocks(slicedFrames, calTree = calTree)
238
239 # Flag the known bad or noisy pixels in the masks "BADPIXELS" and "NOISYPIXELS"
240 # used cal files: BadPixelMask and NoisyPixelMask
241 # -> by default the bad pixels will be excluded later when final cubes are built, the noisy pixels
242 slicedFrames = specFlagBadPixelsFrames(slicedFrames, calTree=calTree)
243
244 # Flag the data affected by the chopper movement in the mask "UNCLEANCHOP"
245 # Uses the high resolution Dec/Mec header and the cal files ChopperAngle and ChopperJitterThreshold
246 slicedFrames = flagChopMoveFrames(slicedFrames, dmcHead=slicedDmcHead, calTree=calTree)
247
248 # Flag the data effected by the grating movement in the mask "GRATMOVE"
249 # Uses the high resolution Dec/Mec header and the cal file GratingJitterThreshold
250 slicedFrames = flagGratMoveFrames(slicedFrames, dmcHead=slicedDmcHead, calTree=calTree)
251
252 if verbose:
253     # Summary of the slices
254     hgyr6745454te slicedSummary(slicedFrames)
255     # Summary of the active (1) and inactive (0) status of every Mask
256     maskSummary(slicedFrames)
257     # Plot the instrument movements without the signal included
258     p1 = slicedSummaryPlot(slicedFrames,signal=0)
259
```

Check: before slicing

Only 1
slice

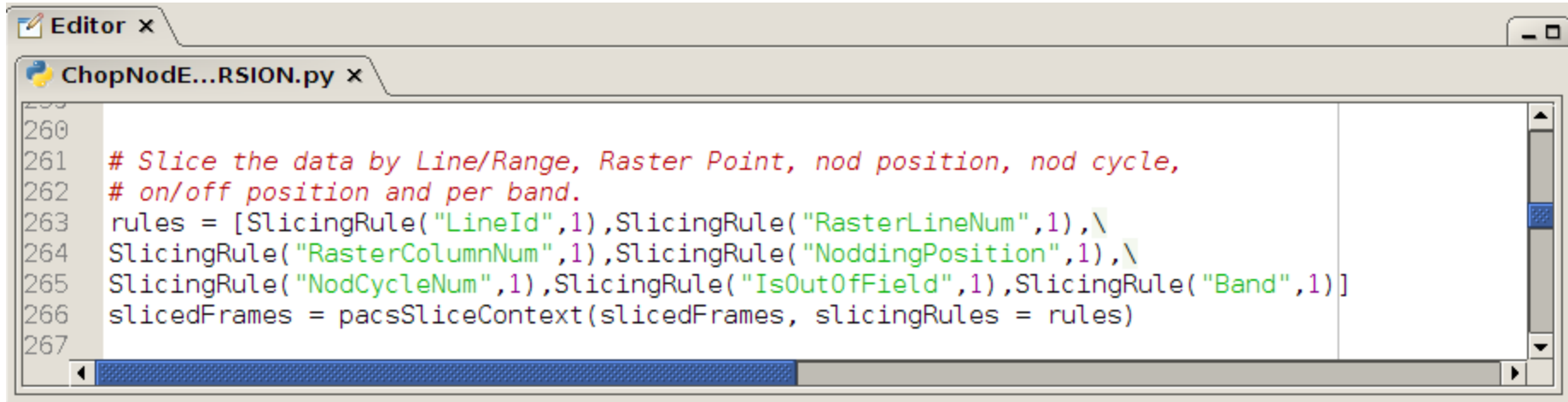
```
Console x
HIPE> if verbose:
..... # Summary of the slices
..... slicedSummary(slicedFrames)
..... # Summary of the active (1) and inactive (0) status of every Mask
..... maskSummary(slicedFrames)
..... # Plot the instrument movements without the signal included
..... pl = slicedSummaryPlot(slicedFrames,signal=0)
.....
noSlices: 1
noCalSlices: 1
noScienceSlices: 0
slice# isScience nodPosition nodCycle rasterId lineId band
dimensions wavelengths
0 false ["", "A", "B"] 0 0 0 [0,1,2,3] ["B2B", "B3A", "UNDEF"]
[18,25,5536] 57.213 - 88.119
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>
```


Check: before slicing



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

Slicing



```
Editor x
ChopNodE...RSION.py x
260
261 # Slice the data by Line/Range, Raster Point, nod position, nod cycle,
262 # on/off position and per band.
263 rules = [SlicingRule("LineId",1),SlicingRule("RasterLineNum",1),\
264 SlicingRule("RasterColumnNum",1),SlicingRule("NoddingPosition",1),\
265 SlicingRule("NodCycleNum",1),SlicingRule("IsOutOfField",1),SlicingRule("Band",1)]
266 slicedFrames = pacsSliceContext(slicedFrames, slicingRules = rules)
267
```

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

```

HIPE> rules =
[SlicingRule("LineId",1),SlicingRule("RasterLineNum",1),SlicingRule("RasterColumnNum",1),SlicingRule("NoddingPosition",1),SlicingRule("NodCycleNum",1),SlicingRule("IsOutOfField",1),SlicingRule("Band",1)]
HIPE> slicedFrames = pacsSliceContext(slicedFrames, slicingRules = rules)
HIPE> if verbose:
..... slicedSummary(slicedFrames)
..... p2 = slicedSummaryPlot(slicedFrames,signal=0)
.....
noSlices: 5
noCalSlices: 1
noScienceSlices: 4
slice#  isScience  nodPosition  nodCycle  rasterId  lineId  band
dimensions  wavelengths
0      false    ["","B"]    0         0 0      [0,1]    ["B2B","B3A","UNDEF"]
[18,25,1460]  57.213 - 88.119
1       true     ["B"]       1         0 0      [2]      ["B3A"]
[18,25,1019]  63.093 - 63.379
2       true     ["A"]       1         0 0      [2]      ["B3A"]
[18,25,1019]  63.093 - 63.379
3       true     ["B"]       1         0 0      [3]      ["B3A"]
[18,25,1019]  57.213 - 57.548
4       true     ["A"]       1         0 0      [3]      ["B3A"]
[18,25,1019]  57.213 - 57.548
Slice edges: [0,1460,2479,3498,4517,5536]
HIPE>

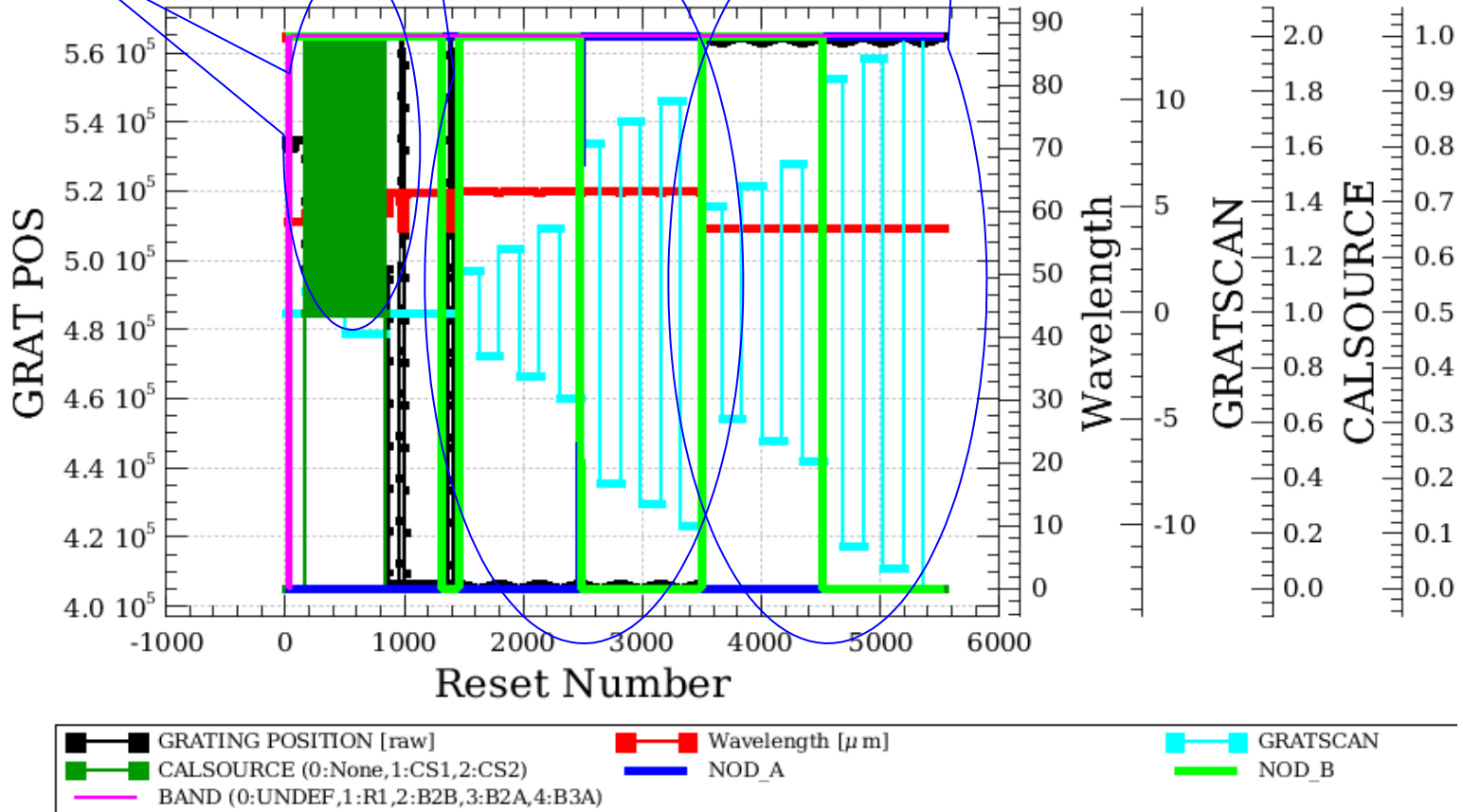
```

Check: after slicing

Line 1: OIII 63

Line 2: NIII 57

Cal Block



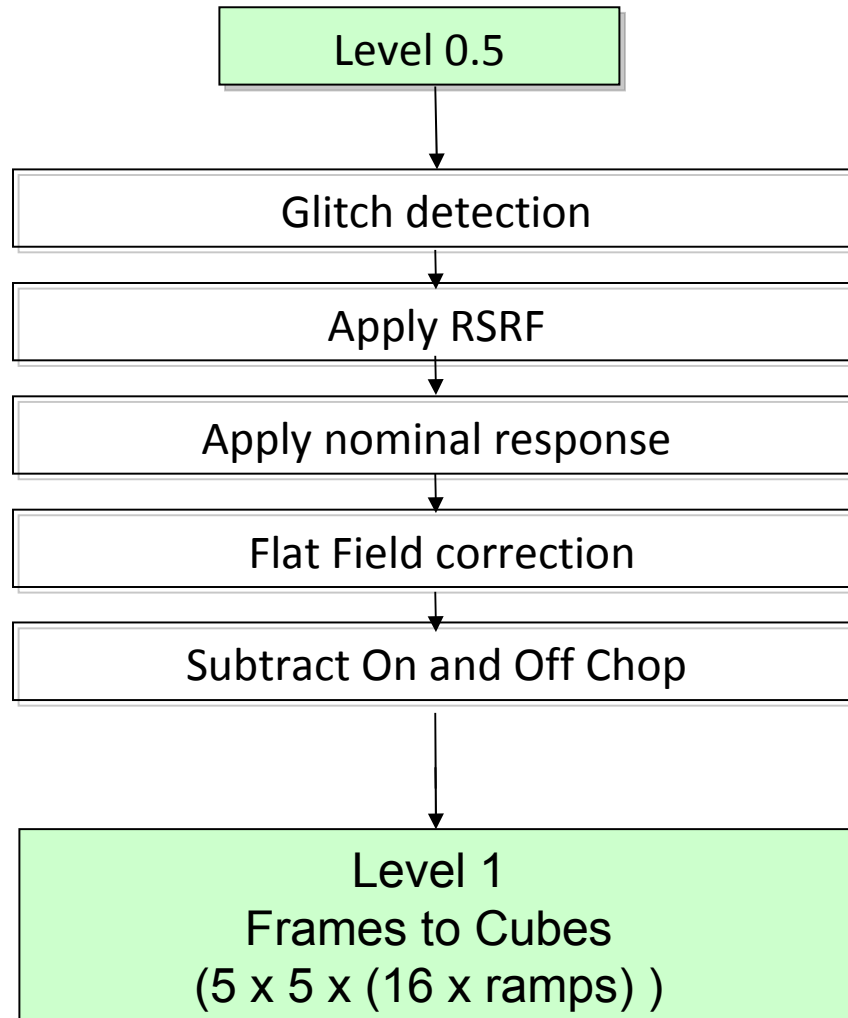
There are four slices (calibration, nod A and B for the 1st line, nod A and B for the 2nd line).

Step 4

Run the 0.5 → 1 pipeline

Glitch detection, chop differentiation, RSRF, flat

Level 0.5 → 1



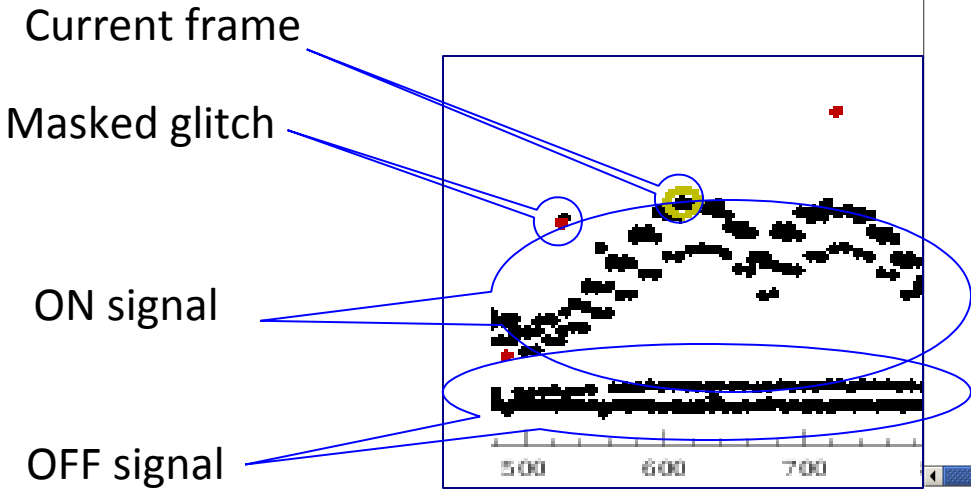
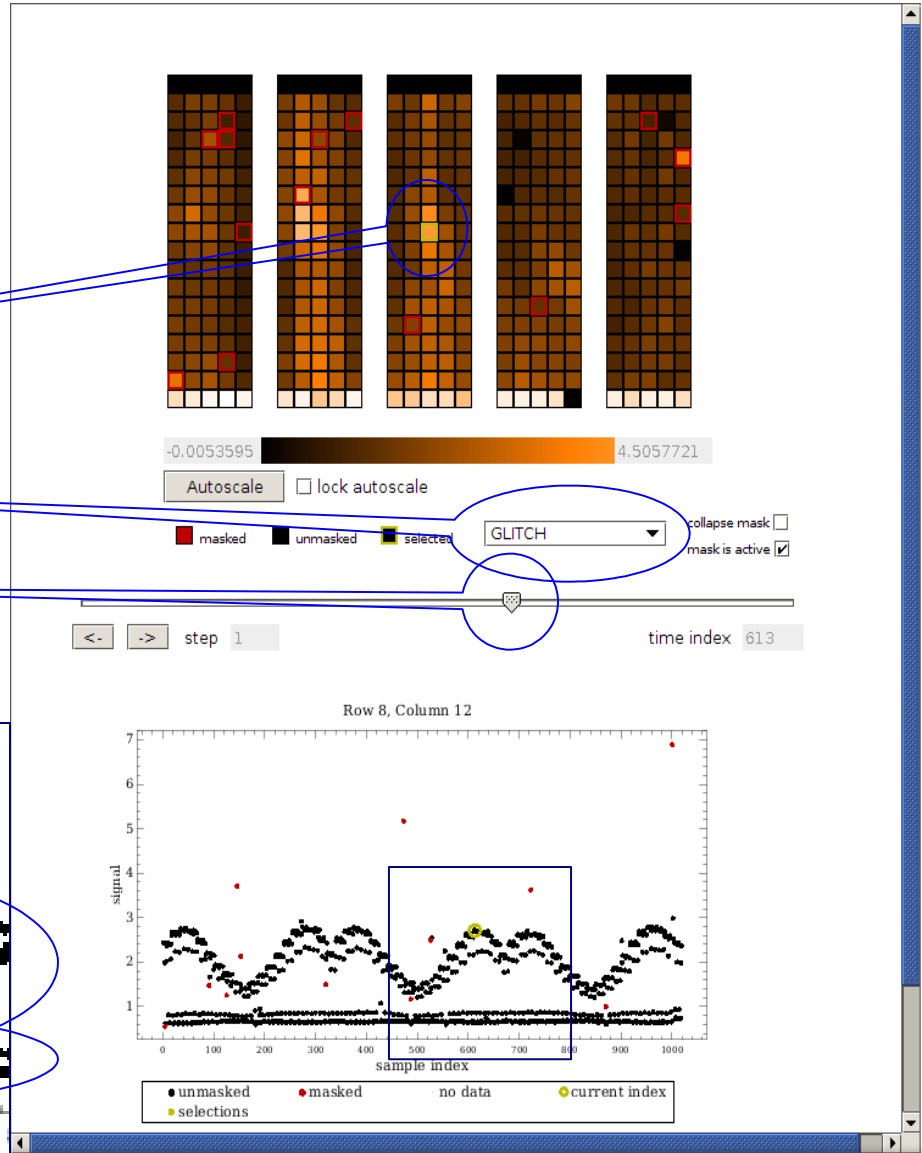
```
ChopNodEx...RSION.py x
297
298 # De-activate all masks before running the glitch flagging
299 slicedFrames = activateMasks(slicedFrames, StringId([" "]), exclusive = True)
300
301 if verbose: maskSummary(slicedFrames,slice=0)
302
303 # Detect and flag glitches ("GLITCH" mask)
304 slicedFrames = specFlagGlitchFramesQTest(slicedFrames)
305
306
307 if verbose:
308     slicedSummary(slicedFrames)
309     # Summary plot, including the signal
310     p3 = slicedSummaryPlot(slicedFrames,signal=1)
311     # Plot of signal vs wavelength for the central pixel for a single slice (you can chose any s
312     # Detector signal: you will see the on and off chop spectral data together
313     # on this plot, as they have not been subtracted from each other yet
314     # (compare this plot to p6 below)
315     slice = 1
316     p4 = plotSignalBasic(slicedFrames, slice=slice)
317     # Inspect timeline of signals and masked signals via a viewer
318     MaskViewer(slicedFrames.get(slice))
319
```

At this point, a deglitching code creates a glitch mask. Sometimes, this deglitching can be too aggressive. It is therefore possible to ignore this mask and detect outliers using data redundancy (see tutorial level 1 → level 2)

Glitch detection

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

- Select a pixel by clicking on it
- Select a mask
- Select a frame



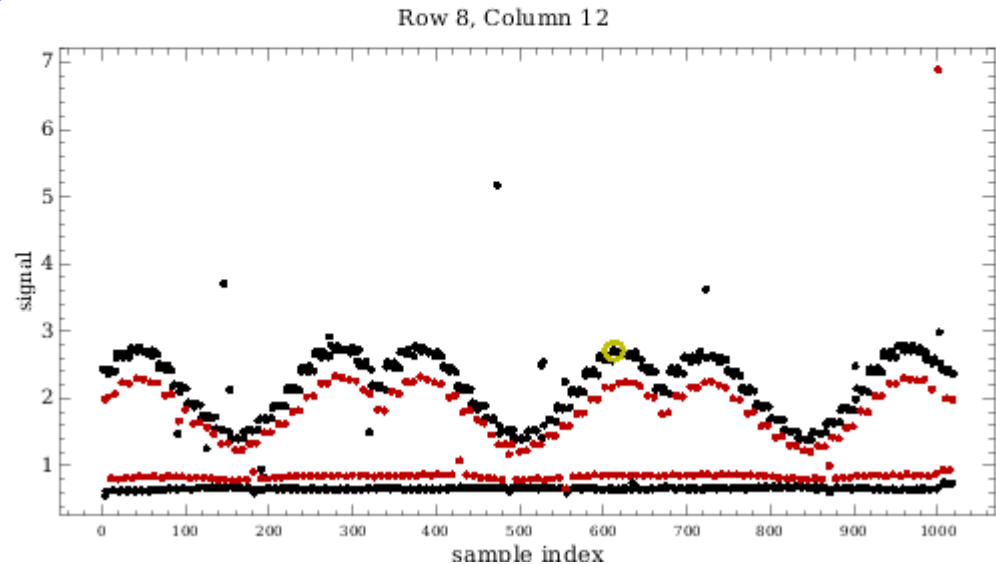
More masks

It is possible to explore other masks

■ masked ■ unmasked ■ selected collapse m mask is ac

<- -> step 1 time index

Select unclean chop



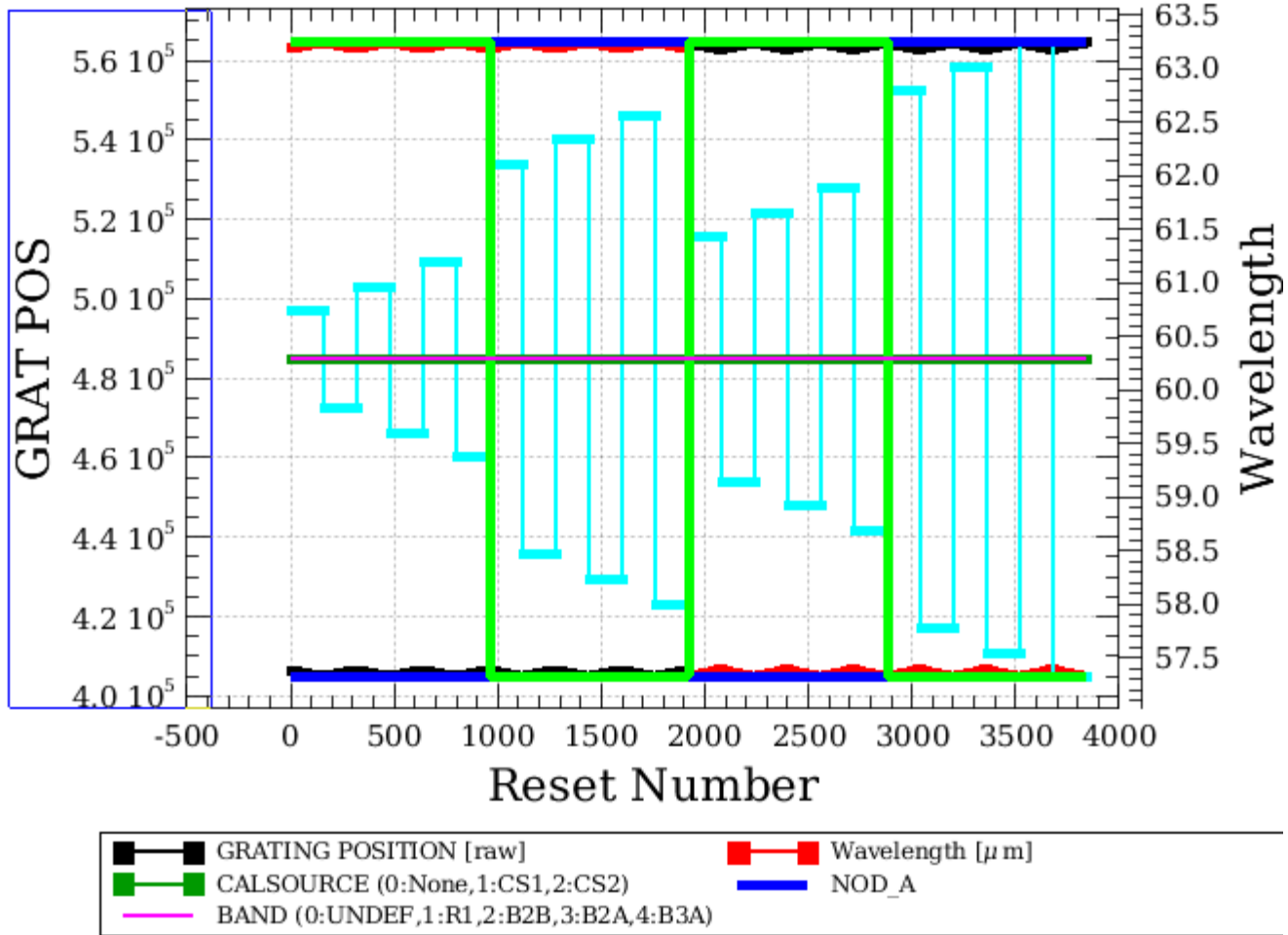
In this case, it is clear why there is a second group of points for the ON and OFF positions. These corresponds to signals obtained when the chopper was not yet in the correct position.

Chop differentiation

```
ChopNodEx...RSION.py x
348
349 # Compute the differential signal of each on-off pair of datapoints, for each chopper cycle
350 # The calibration block is cut out of the slicedFrames, so only the scientific slices remain.
351 slicedFrames = specDiffChop(slicedFrames, scical = "sci", keepall = False, normalize=False)
352
353 if verbose:
354     # Data Structure. Only science blocks are left
355     slicedSummary(slicedFrames)
356     p5 = slicedSummaryPlot(slicedFrames, signal=0)
357     # Detector signal: signal is now differential, with the offs having been
358     # subtracted from the ons (compare to p4 above)
359     p6 = plotSignalBasic(slicedFrames, slice=0)
360
361
```

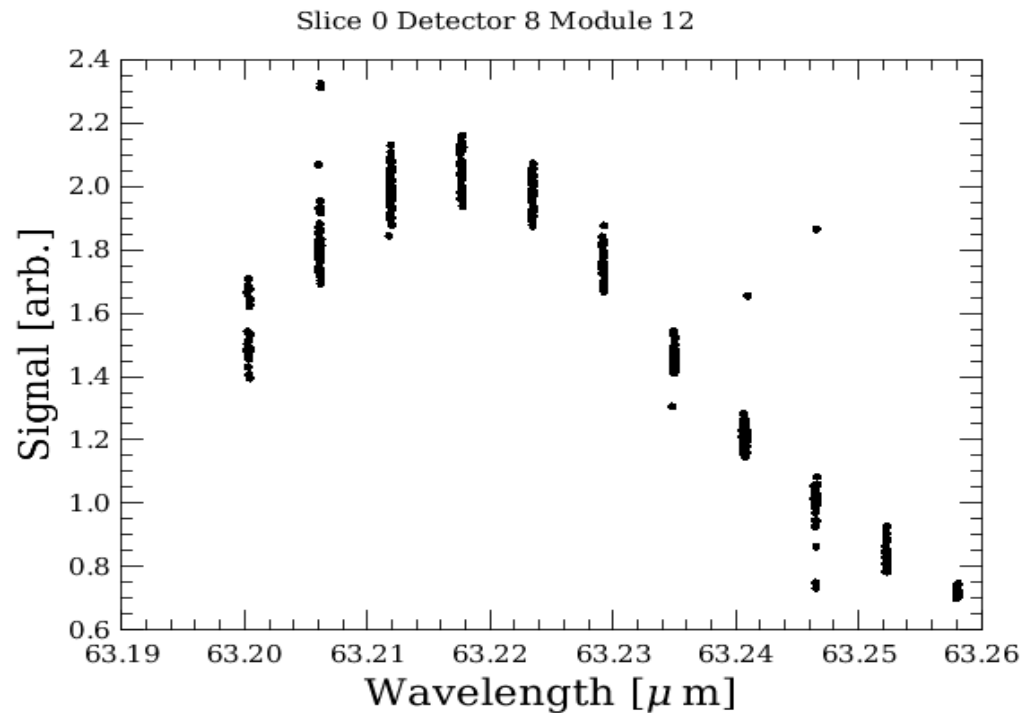
After chop differentiation, the calibration block is excluded from the data

Chop differentiation



Chop differentiation

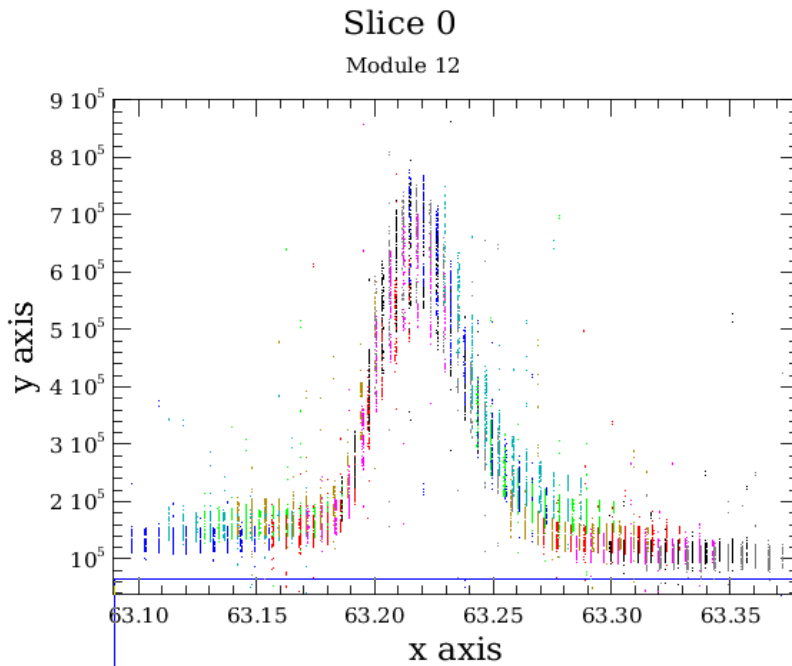
The data are only on the ON position (OFF being subtracted)



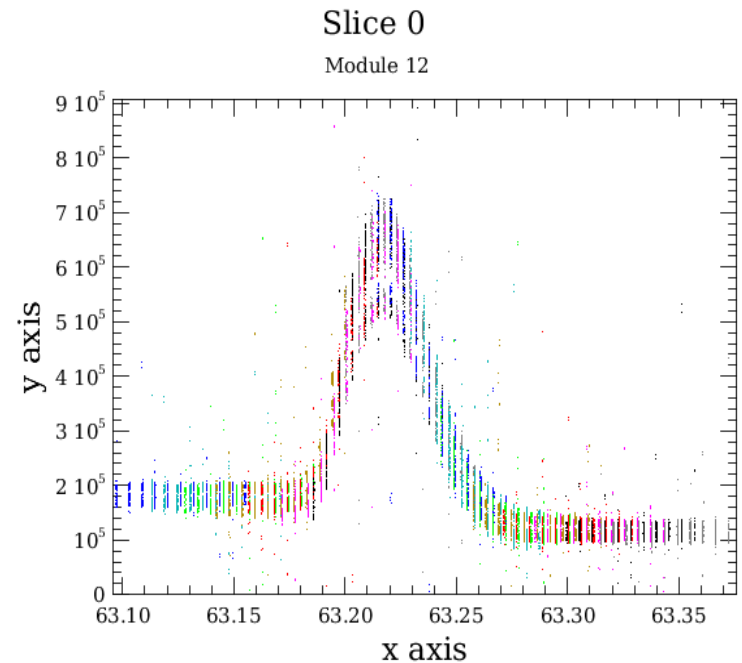
```
ChopNodEx...RSION.py x
373
374 # Divide by the relative spectral response function
375 # Used cal files: rsrfr1, rsrfb2A, rsrfb2B or rsrfb3A
376 slicedFrames = rsrfCal(slicedFrames, calTree=calTree)
377
378 # Divide by the response
379 # Used cal file: nominalResponse
380 slicedFrames = specRespCal(slicedFrames, calTree=calTree)
381
382 if verbose:
383     slice = 0
384     frame = slicedFrames.get(slice)
385     module = 12
386     flux = frame.signal[1:17,module,:]
387     wavelengths = frame.wave[1:17, module,:]
388     pwave = PlotXY( line=Style.NONE,titleText="Slice "+str(slice),subtitleText=" Module "+str(mod
389     for i in range(16):
390         pwave[i] = LayerXY(wavelengths[i,:], flux[i,:],line=Style.NONE,symbol=Style.DOT)
391     del frame, wavelengths, flux
392
393
394 # Refine the spectral flatfield.
395 # Parameters:
396 #     minWaveRangeForPoly (microns). If your spectral ranges are > this value, a poly is fit to the
397 #     spectra, of order "polyOrder", on a pixel by pixel basis independently for each module/spaxel
398 #     This (continuum) fit is then used normalise the individual 16 pixels to the module/spaxel me
399 #     minWaveRangeForPoly (microns). If your spectral ranges are < this values, the median of the spe
400 #     is calculated, on a pixel by pixel basis independently for each module/spaxel.
401 #     This value is then used normalise the individual pixels to the module/spaxel median.
402 #     --> the values of polyOrder and minWaveRangeForPoly can be adjusted by the user
403 #     In both cases the correction is a division.
404 #
405 # See also the specFlatFieldRange URM entry and the PDRG Chap. 3 for more information.
406 slicedFrames = specFlatFieldRange(slicedFrames,polyOrder=5, minWaveRangeForPoly=4., verbose=1)
```

RSRF, response, flat field

Check of the central module (all spectral pixels)

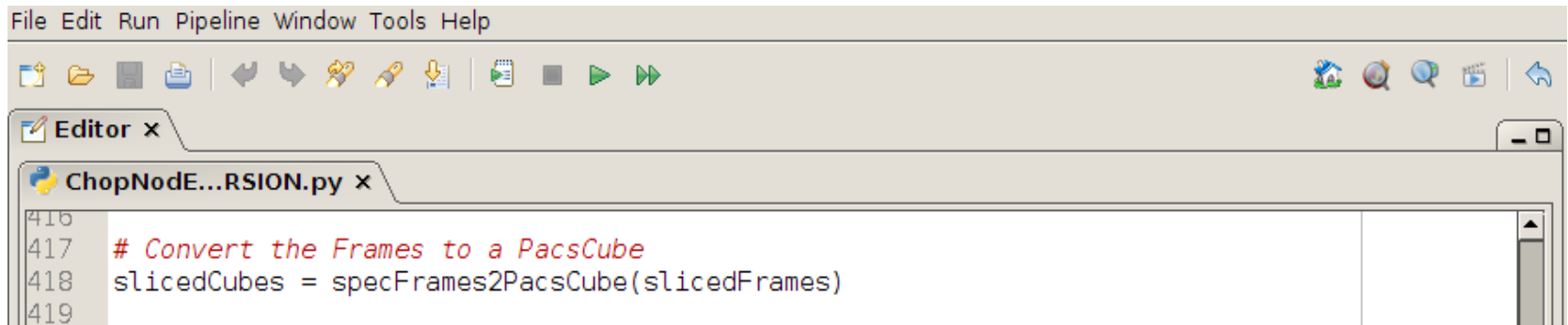


RSRF & response



Flat field

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



```
File Edit Run Pipeline Window Tools Help
[Icons]
Editor x
ChopNodE...RSION.py x
416
417 # Convert the Frames to a PacsCube
418 slicedCubes = specFrames2PacsCube(slicedFrames)
419
```