



NHSC/PACS Web Tutorials

Running the PACS Spectrometer pipeline for unchopped line mode

PACS-303

Level 0 to 2 processing

Prepared by Dario Fadda
April 2014

This tutorial will guide you through the interactive reduction of spectra obtained in unchopped line mode. This pipeline works for three different modes:

- (i) unchopped line standard;
- (ii) unchopped bright line;
- (iii) wavelength-switching.

We remind that the archive reduction does not include the transient correction available in the interactive pipeline.

Pre-requisites

The following tutorials should be read before and after this one:

- *PACS-101: How to use these tutorials.*
- *PACS-102: Accessing and storing data from the Herschel Science Archive*
- *PACS-103: Loading scripts*
- *PACS-302: Level 1 to level 2 processing*

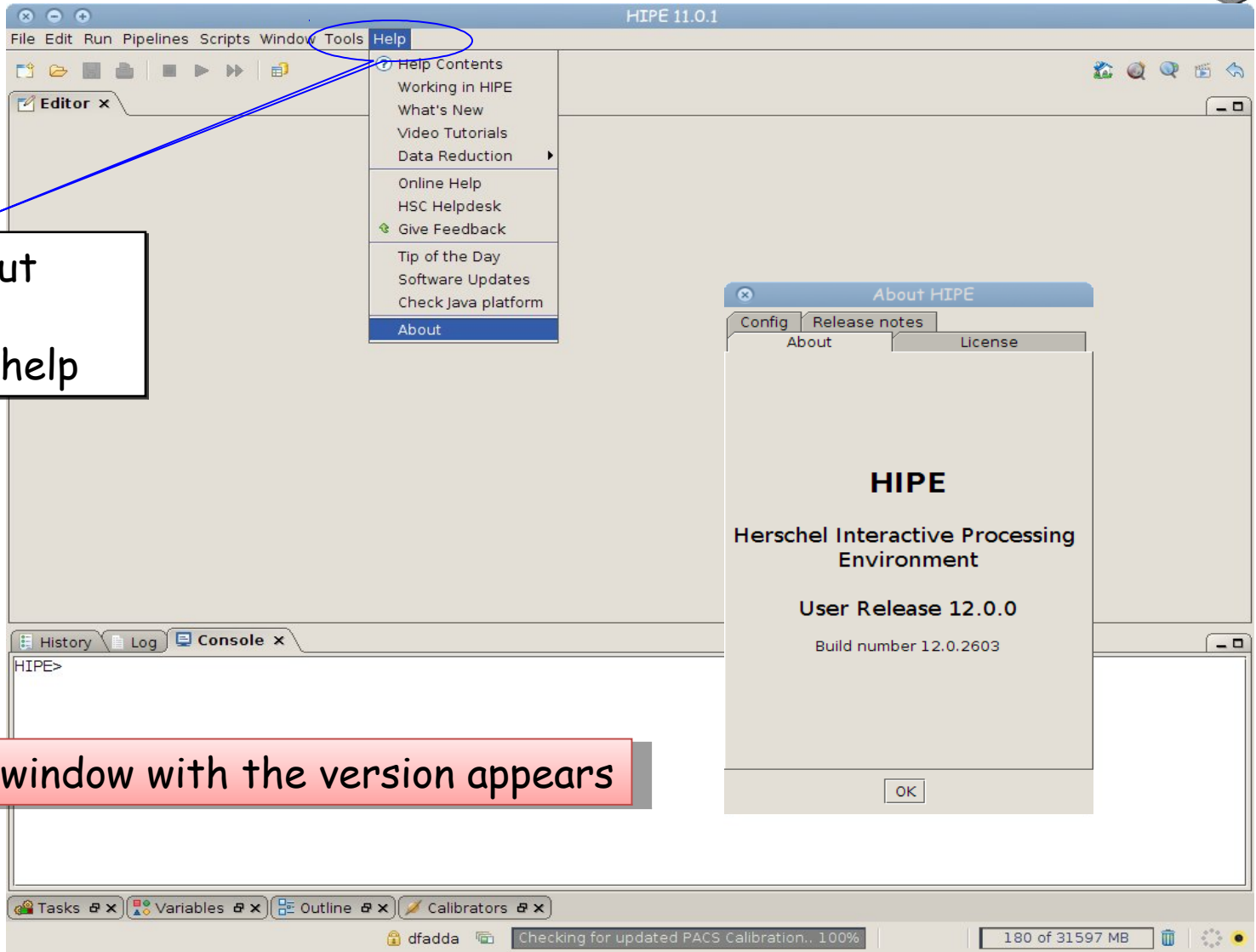
Overview

- 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

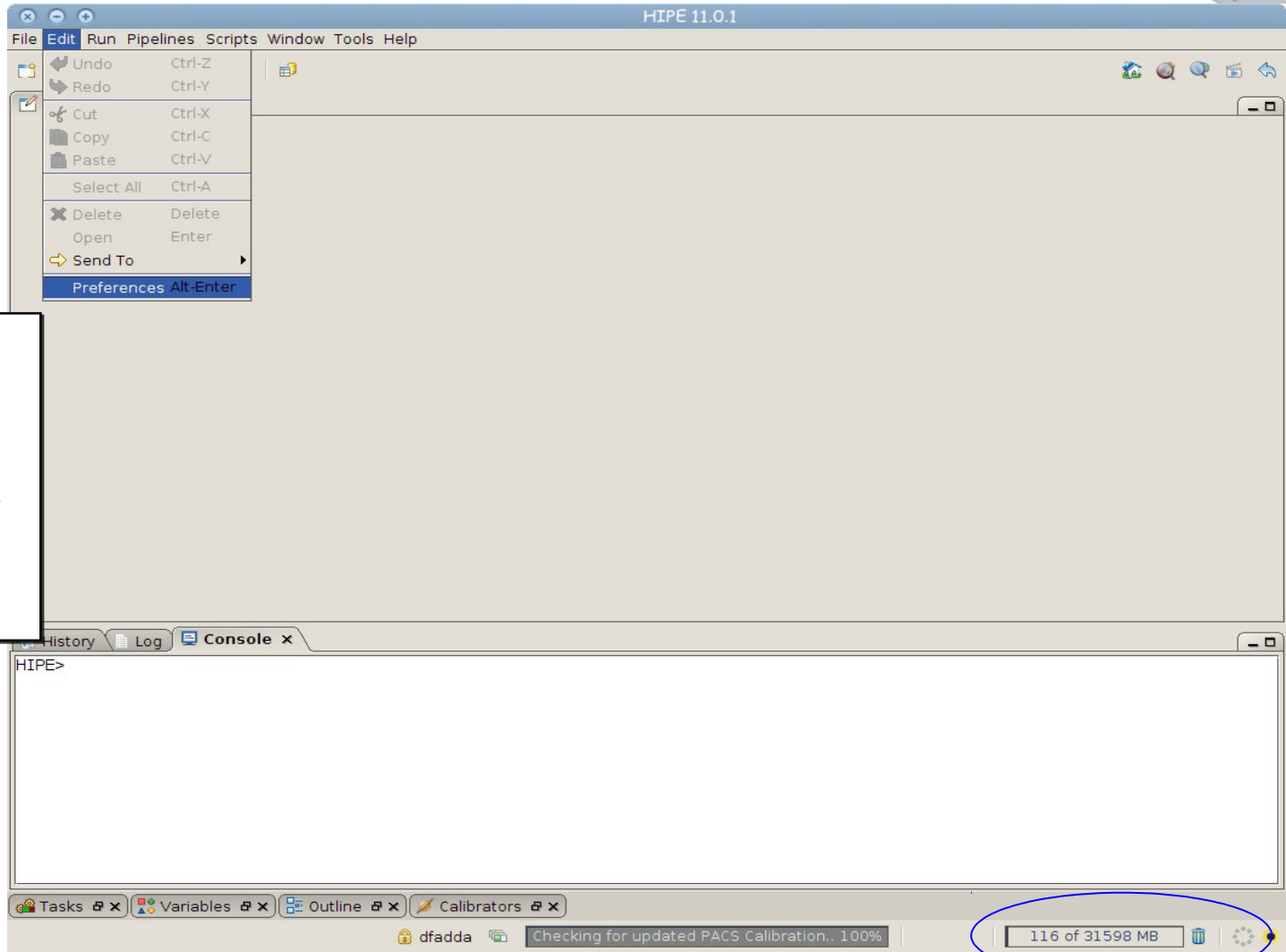
Check HIPE version, memory allocation, and calibration products

The version used for the tutorial is 12.0.2603



Select about from drop down help

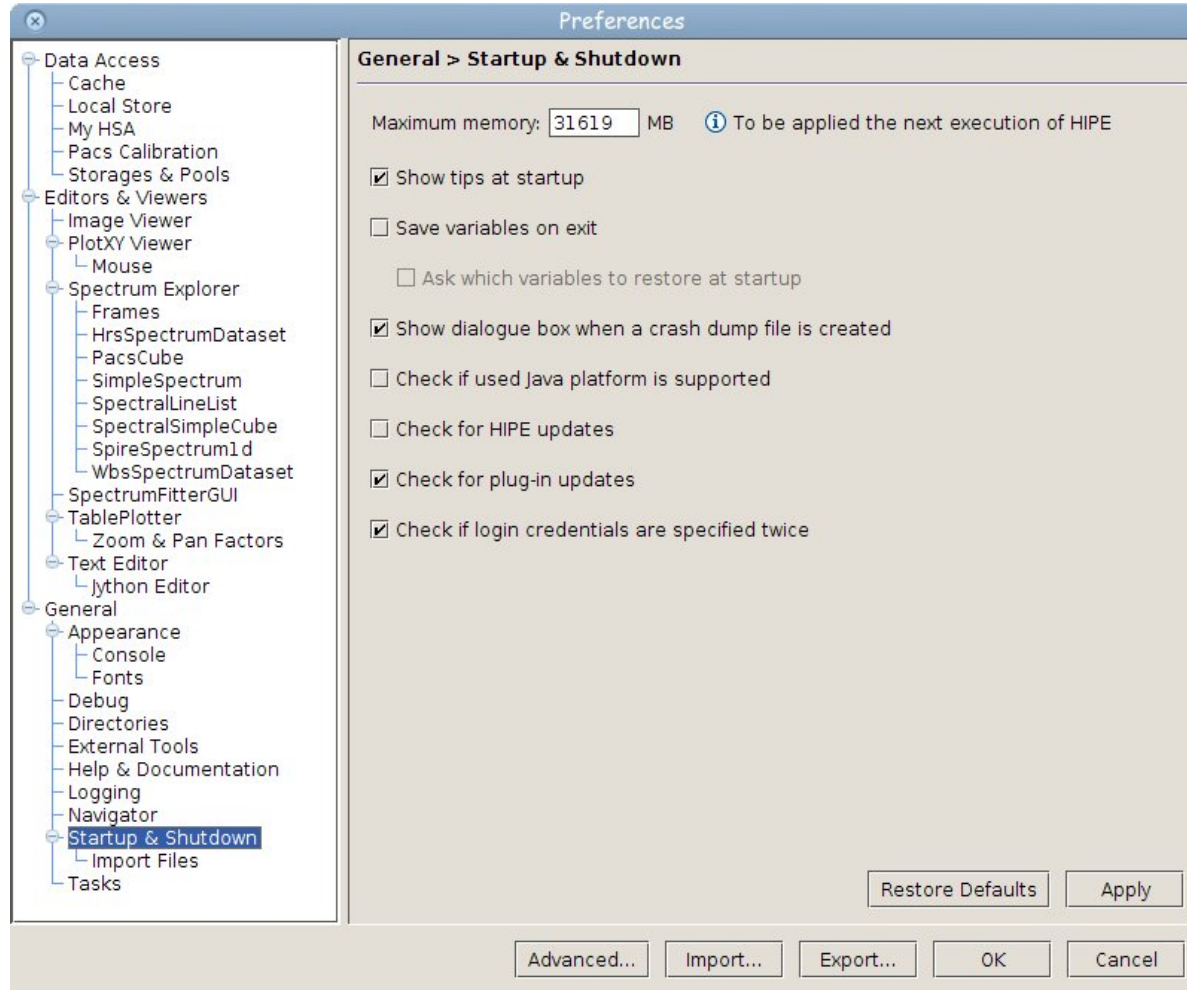
A pop-out window with the version appears



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

Memory used and available

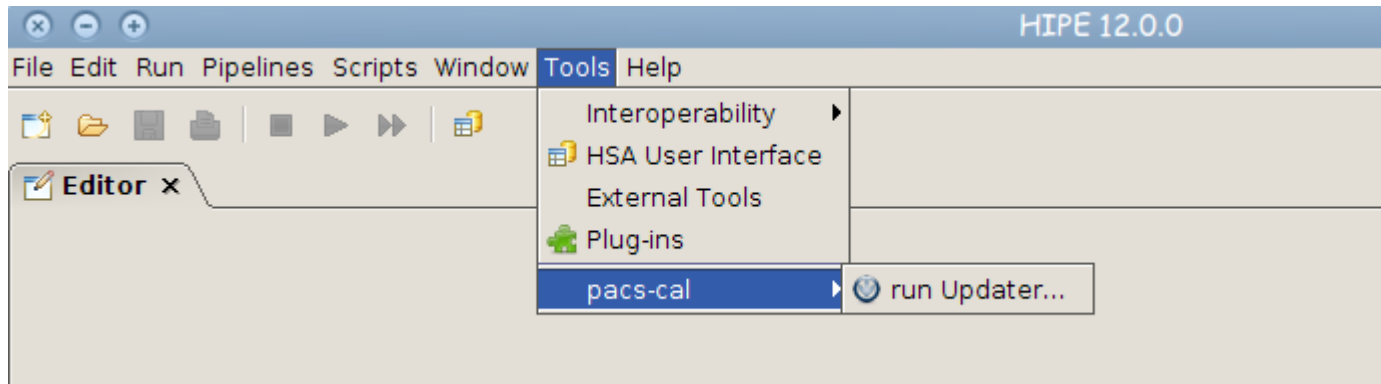
Then click on Startup & Shutdown and change the amount of memory



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.

Calibration

Before running a new reduction, it is a safe habit to check if the latest calibration products are installed. The way to do it is running the Updater.



The screenshot shows the HIPE 12.0.0 software interface. The 'Window' menu is open, and 'Calibration Sets' is selected. The 'Calibration Sets' window displays a grid of calibration sets, with set 65 highlighted. Below the grid, the 'Release Note for Calibration Set v65' is displayed. The release note includes the following information:

Release Note for Calibration Set v65
 Changes for the spectrometer telescope background correction calfiles related to the TelescopeBackground calfile version 8

Updated calibration products
 The following calibration product has been added :

- PCalSpectrometer_TBackCorB2A_FM_v3.fits
- PCalSpectrometer_TBackCorB2B_FM_v3.fits
- PCalSpectrometer_TBackCorB3A_FM_v3.fits
- PCalSpectrometer_TBackCorR1_FM_v4.fits

Reference Information
 Internal ICC repository release 24 February 2014.

ICC Specific Information

- See PACS-5673

The console window at the bottom shows a series of 'HIPE>' prompts. The status bar at the bottom indicates 'dfadda', 'Jython Interpreter 100%', and '5620 of 30569 MB'.

To inspect in detail what is different in the several releases of calibration products, it is possible to access the release notes for any release as well as the list of products with their specific versions.

A new version of the telescope background is now available.

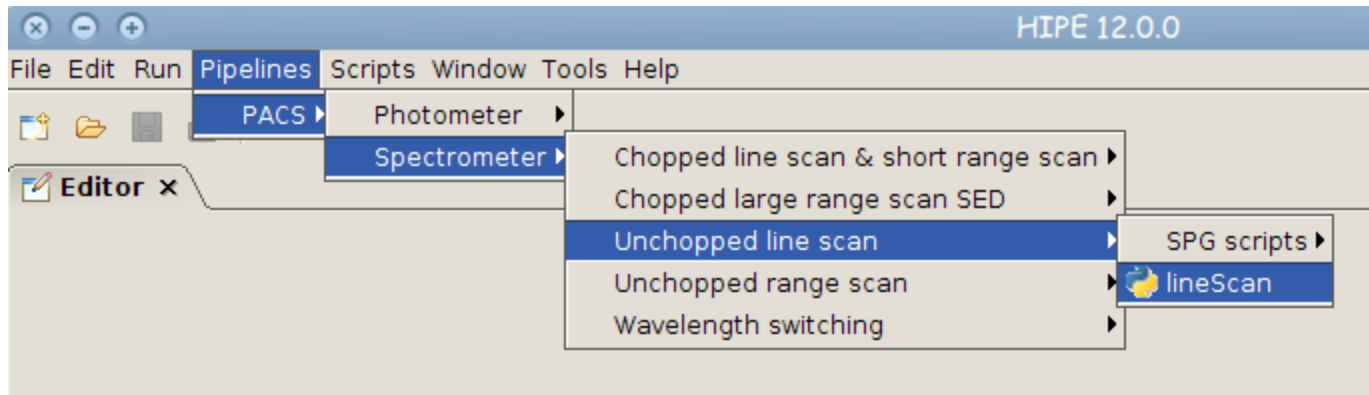
Step 2

Setup

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

Loading the script

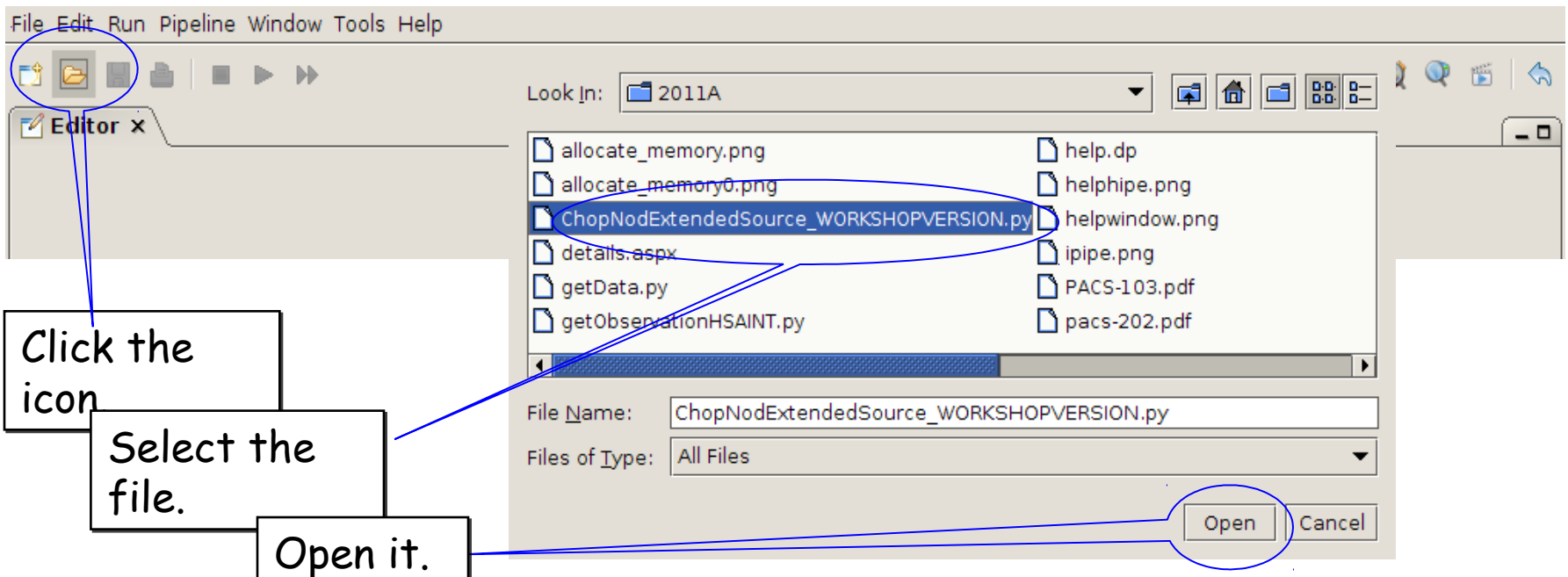
The script used in this tutorial corresponds to the script available directly from the distribution.



In the case you were using a modified script, you should first load it from the directory where it resides.

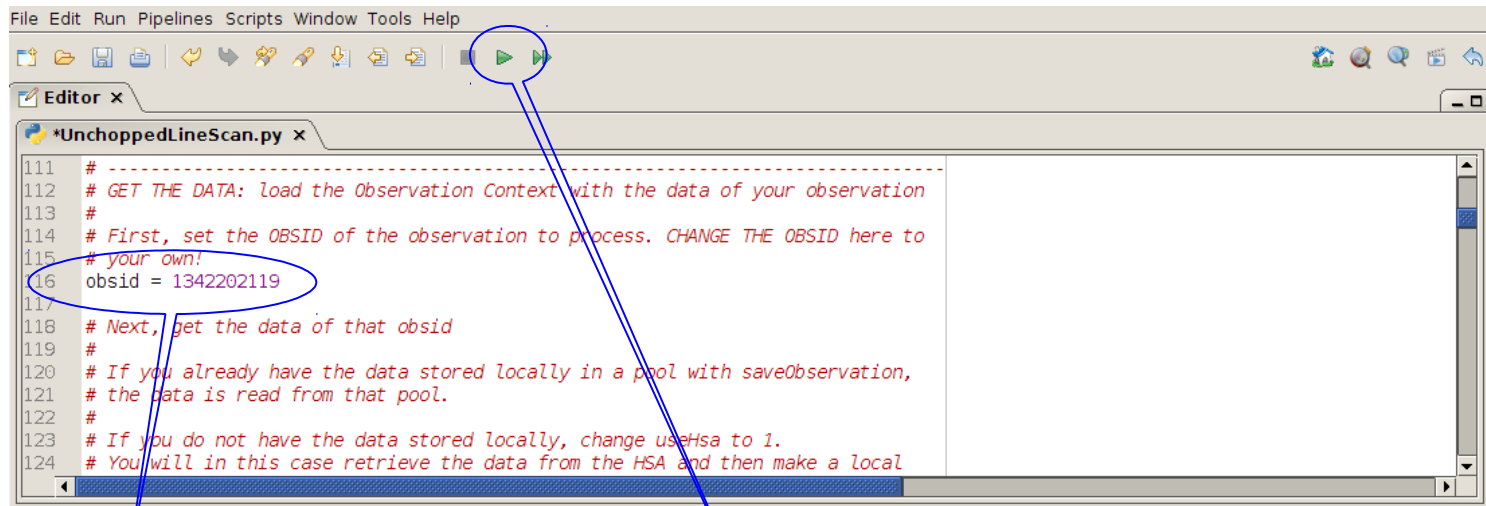
Loading the script

To load a custom script into an HIPE session, just click on the loading icon as shown in the figure. Then 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 Pipelines Scripts Window Tools Help
Editor x
*UnchoppedLineScan.py x
111 # -----
112 # GET THE DATA: load the Observation Context with the data of your observation
113 #
114 # First, set the OBSID of the observation to process. CHANGE THE OBSID here to
115 # your own!
116 obsid = 1342202119
117
118 # Next, get the data of that obsid
119 #
120 # If you already have the data stored locally in a pool with saveObservation,
121 # the data is read from that pool.
122 #
123 # If you do not have the data stored locally, change useHsa to 1.
124 # You will in this case retrieve the data from the HSA and then make a local
```

Modify 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).

```
File Edit Run Pipeline Window Tools Help
ChopNode...RSION.py x
109 # that the poolName is the obsid.
110 #
111 useHsa = 0
112 obs = getObservation(obsid, verbose=True, useHsa=useHsa, poolLocation=None, poolName=None)
113 if useHsa: saveObservation(obs, poolLocation=None, poolName=None)
114
115 # -----
NameError: obs
HIPE> obsid = 1342186799 # M82 - blue
HIPE> useHsa = 0
HIPE> obs = getObservation(obsid, verbose=True, useHsa=useHsa, poolLocation=None, poolName=None)
INFO: using default value for knownLocations ['/home/fadda/hcss/lstore/', '/pools/lstore/*', '/STER/pacsman/PacsPools/HSA_Pacs_DataPools', '/Volumes/pacs-data-mpe', '/Volumes/pacs-data-ivs', '/Users/Shared/data/pools', '/home/fadda/lstore']
INFO: using data pool 1342186799 from directory /home/fadda/.hcss/lstore/
INFO: start querying the storage...
INFO: observation found!
HIPE>
```

Click on this line.

Hit the arrow

Check: observation summary

The next command to use is:

```
obsSummary(obs)
```

Although it comes later in the official pipeline, you can use it already once the observation has been loaded. This can be very instructive, especially if you don't know the lines which have been observed and you want to set the pipeline script to reduce and visualize a particular line.

Check: observation summary

```

History | Log | Console x
HIPE> obsSummary(obs)
Observation Summary:
  OBSID:      1342202119
  Instrument:  PACS
  AOR label:  Calibration_RPSpecFlux_1-RPSpecFlux_433D_stdLine_Unchop_C158_Arp220_0001
  Proposal:   Calibration_rppacs_35
  Target:     Arp 220
  Actual RA:  15h 35m 5.59s
  Actual Dec.: 23° 30' 11.82''
  Redshift:   0.018126 (z)
  Purpose:    ---
  Concat.:    ---
  OD:         440
  Start:      2010-07-28T02:15:16.000000 TAI (1658974516000000)
  Duration:   1964.0 seconds (incl. spacecraft on-target slew time)

AOT and instrument configuration:
  AOT:        PacsLineSpec
  Mode:       Pointed, unchopped grating scan
  Bands:      B2B R1 (prime diffraction orders selected)
  Is bright:  NO (default range mode)
  Nod cycles: 2

Observation block summary:
  | Name(*) | Camera | ID | Band(*) | Wave(*) | WaveMin | WaveMax | Repetitions(*) | ActualRep | Capacitance | OutOfBand | Channel |
  |         |        |   |         | micrometer | micrometer | micrometer |                |           | pF          |            |         |
  | CII C+ | red    | 102 | R1      | 160.600 | 159.040 | 162.242 | 4              | 8         | 0.140      | No        | prime  |
  | -      | blue   | 2   | B2B     | 80.335  | 79.537  | 81.134  | 4              | 8         | 0.140      | No        | parallel |
(*) = requested in HSPOT

System configuration summary:
  SPG pipeline version:      SPG v12.0.0_2491
  Calibration tree version:  64
  SPG pipeline products creation date: 2014-02-12T10:58:40.221000 TAI (1770893920221000)
  Mission configuration:     MC_H52ASTR_P55ASTR_S57ASTR_RP
  Processed to level:        RED: 2.0 BLUE: 2.0
  Quality Control:           PENDING
  Action:                     NONE

Quality comments:
  No comments added

```

We will select: camera = 'red'

Setting the camera

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

```
File Edit Run Pipelines Scripts Window Tools Help
*UnchoppedLineScan.py x
169 # SETUP 1:
170 # - Red or blue camera ?
171 camera = 'red' # camera = 'blue' or 'red'
172
173 # -----
174 # Set up the calibration tree. We take the most recent calibration files,
175 # for the specific time of your observation (obs=obs)
176 #
177 # This tree contains pointers to all the calibration files that the pipeline
178 # tasks use (when calTree=calTree is specified in a task's call).

History Log Console x
HIPE> obsid = 1342202119
HIPE> useHsa = 0
HIPE> obs = getObservation(obsid, verbose=True, useHsa=useHsa, poolLocation=None, poolName=None)
getObservation is retrieving the observation from pool '1342202119' at: '/home/fadda/.hcss/lstore/1342202119'
HIPE> if useHsa: saveObservation(obs, poolLocation=None, poolName=None)
HIPE> verbose = 1
HIPE> updateObservationContext = 0
HIPE> camera = 'red' # camera = 'blue' or 'red'

# -----
# 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
# print calTree below. That version points to a unique set of calibration files.
# If you print the common or spectrometer branches of the tree, you can see
# the version numbers of the individual calibration files this calTree version
# corresponds to.
HIPE>
```

We select camera = 'red'

199 of 28106 MB

Setting the calibration tree

Finally, we set the calibration tree. We can check the calibration used on the archival data with: `print obs.meta["calVersion"]`

The screenshot shows a Python editor window titled 'Editor x' with a file named '*UnchoppedLineScan.py x'. The code in the editor is as follows:

```
184 # the version numbers of the individual calibration files this calTree version
185 # corresponds to.
186
187 calTree = getCalTree(obs=obs)
188 if verbose:
189     print calTree
190     print calTree.common
191     print calTree.spectrometer
192
193 # -----
```

Below the editor is a console window titled 'Console x' showing the output of the code:

```
HIPE> if verbose:
    print calTree
    print calTree.common
    print calTree.spectrometer
PACS Calibration Tree
Model   : FM
Scope   : BASE
Version : 65
Branches: [common, photometer, spectrometer]
PacsCalCommon Calibration Products:
```

Annotations in the image include a blue oval around the `print` statements in the code, a blue arrow pointing from this oval to a text box, and another blue arrow pointing from the '65' in the console output to a second text box.

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

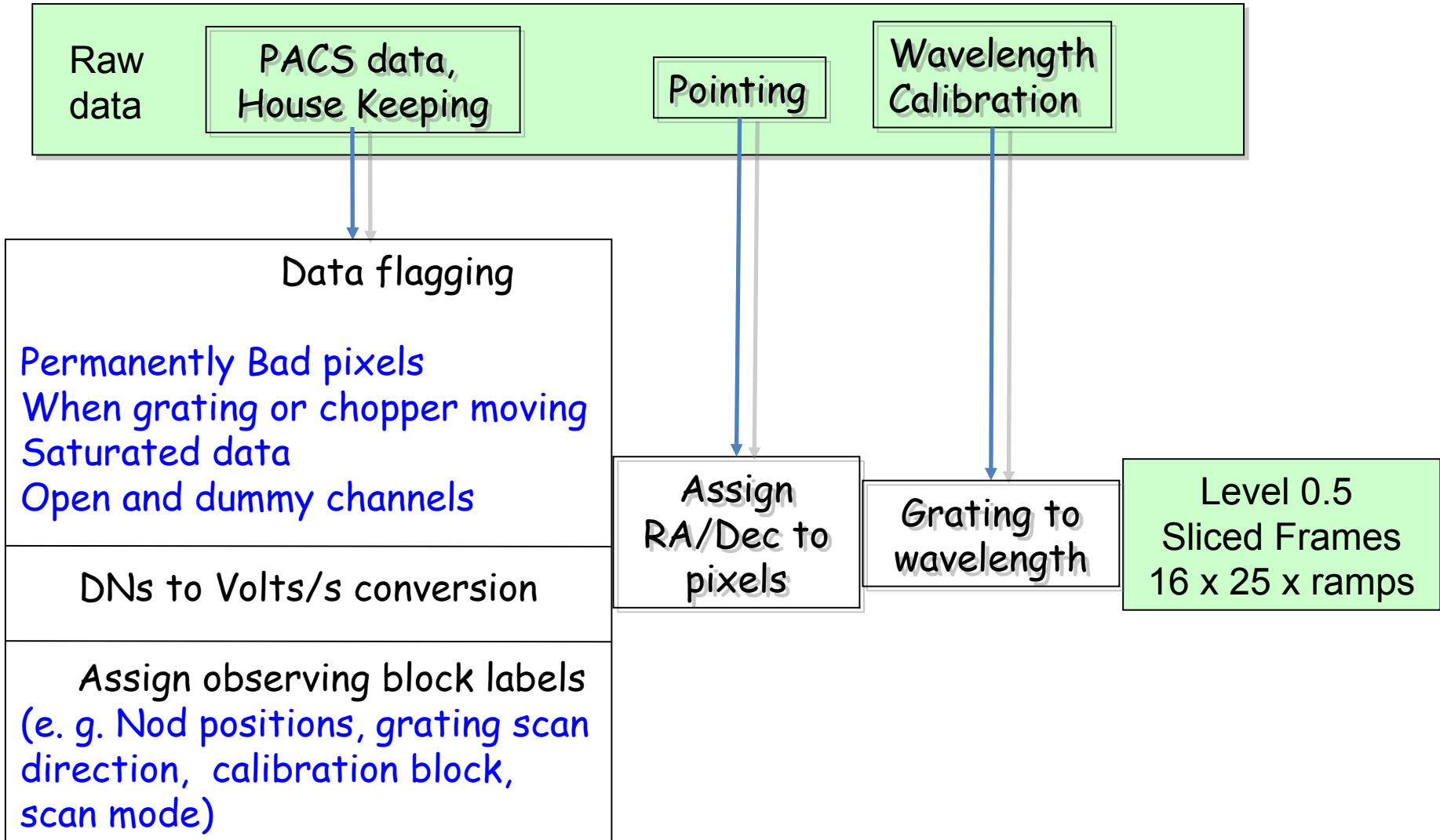
Version 65 includes a revised version of the telescope background model.

Step 3

Run the 0 → 0.5 pipeline

Basic calibration (pointing, wavelength calibration,
slicing)

Level 0 → 0.5

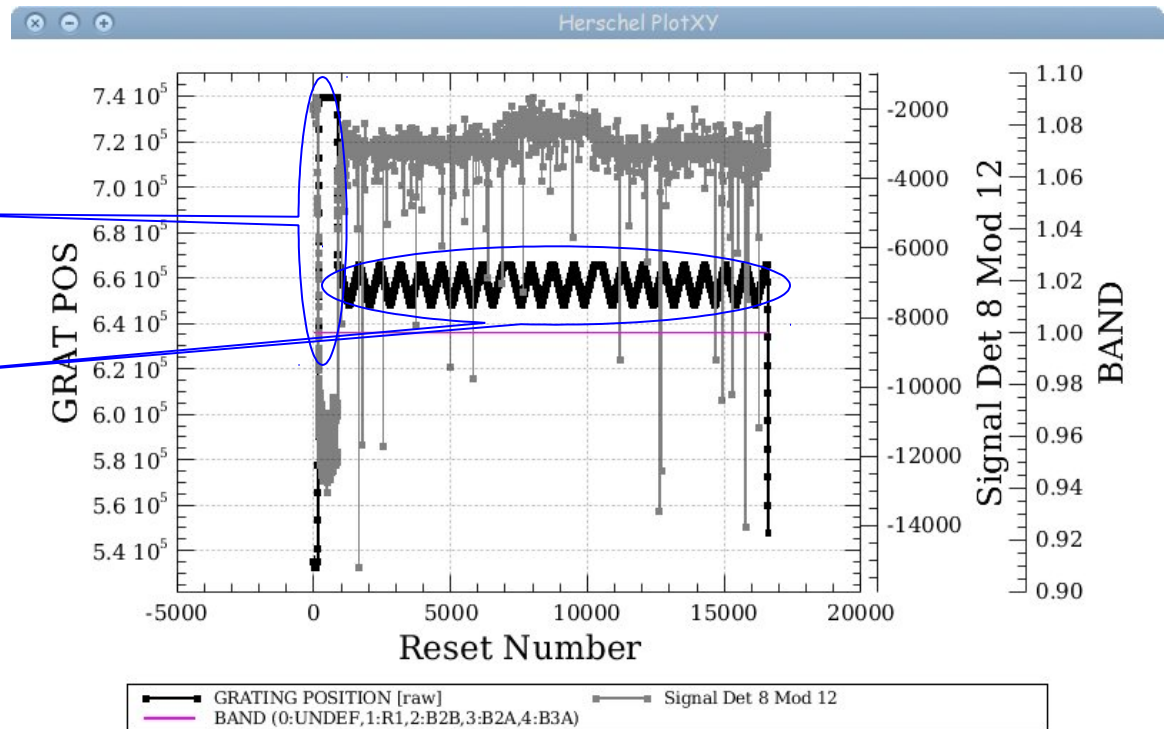


Check: level 0

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.

Calibration block

Grating scans



In our case, after the calibration block, a line is observed in R1.

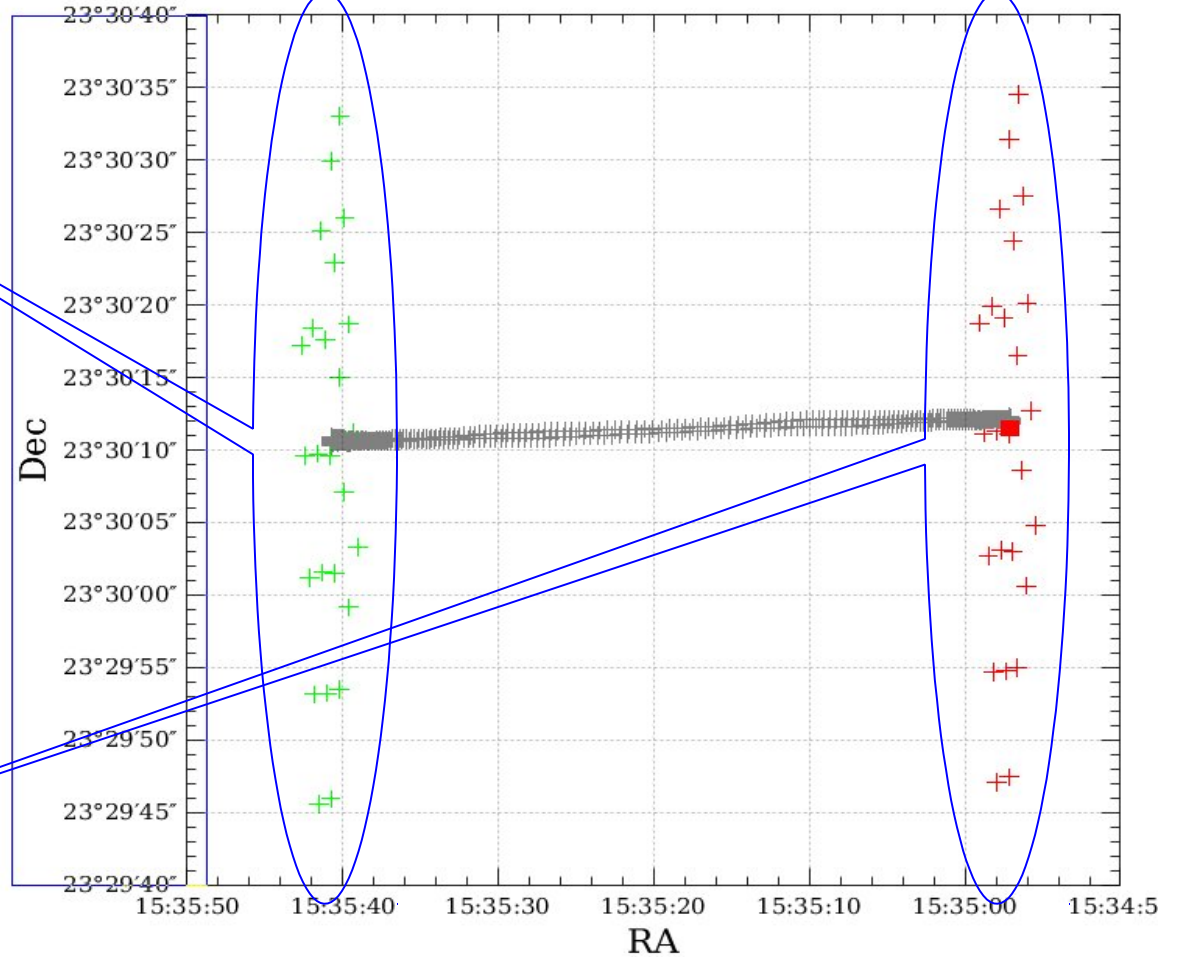
Check: footprint

Herschel PlotXY

PACS footprint and S/C boresight positions

OFF

ON



+ ON Source + OFF Source + Herschel boresight ■ Source

Check: before slicing

```

History Log Console x
HIPE> slicedFrames = flagGratMoveFrames(slicedFrames, dmcHead=slicedDmcHead, calTree=calTree)
HIPE> if verbose:
    # an overview of the slicedFrames contents
    slicedSummary(slicedFrames)
    # Summary of the active (1) and inactive (0) status of every Mask
    maskSummary(slicedFrames)
    # Show the basic data structure, without the signal
    p1 = slicedSummaryPlot(slicedFrames, signal=0)
noSlices: 1
noCalSlices: 1
noScienceSlices: 0
slice#  isScience  onSource  offSource  rasterId  lineId  band  dimensions  wavelengths
0      false      both      both      0 0      [100,101,102] ["R1"]  [18,25,16608]  149.311 - 176.221
Nb of slices: 1
Slice 0
BLINDPIXELS      1
SATURATION        1
RAWSATURATION    0
NOISYPIXELS      0
BADPIXELS         1
UNCLEANCHOP      1
GRATMOVE          1
Slice edges: [0,16608]
HIPE>

```

Only 1 slice

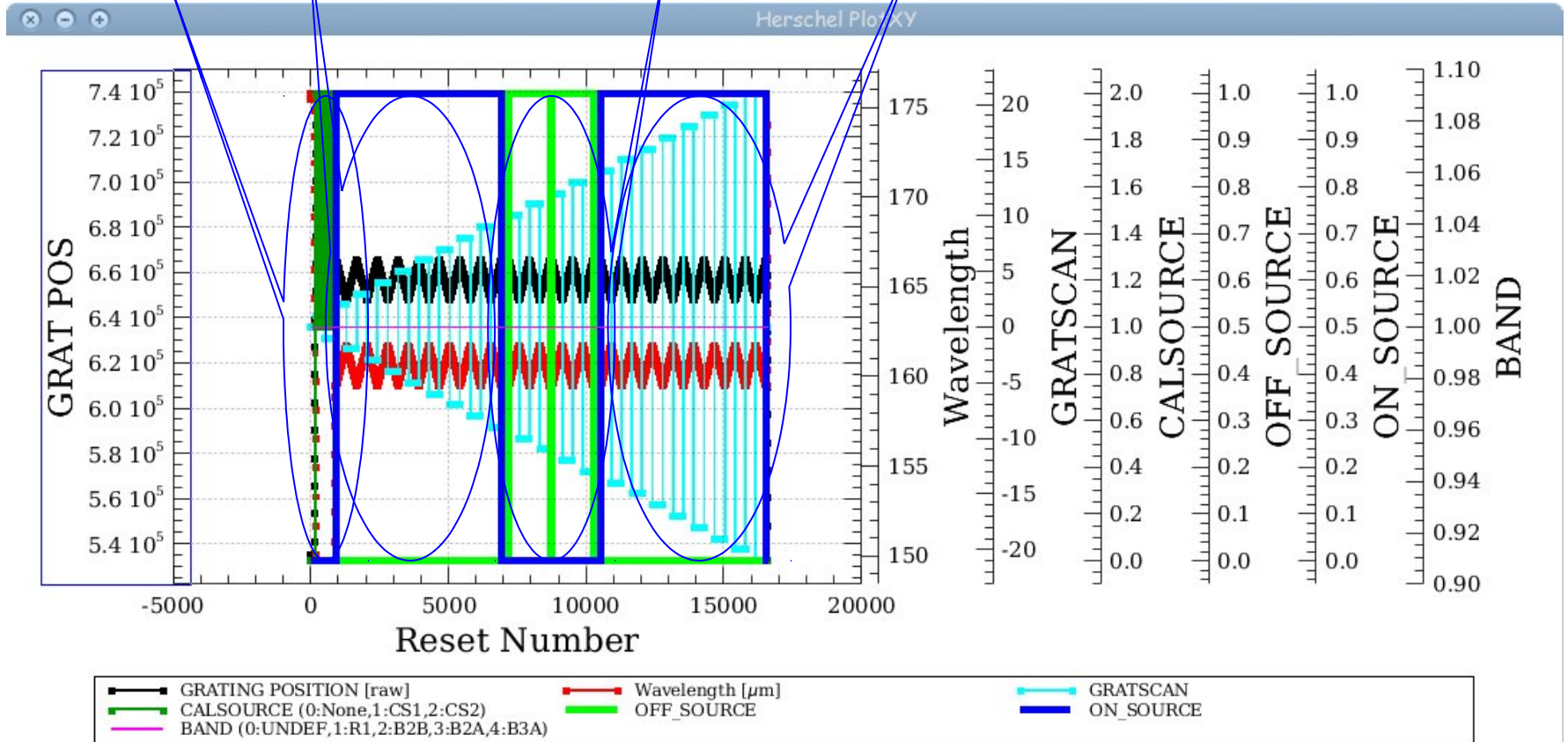
Check: before slicing

Cal Block

ON

OFF

ON



One line with ON, OFF, and ON source positions. Grating scans are numbered positive if upscans and negative if downscans.

Slicing

```
*UnchoppedLineScan.py x
293
294 # Slice the data by Line/Range, Raster Point, nod position, nod cycle, on/off position and per band.
295 # The parameters removeUndefined and removeMasked are for cleaning purposes
296 # Any column in the "BlockTable" can be used as a 'slicingRule', but do
297 # not include/modify the parameter "slicingRules" if you are not 100% aware of what you are doing!
298 # The following rules are the default:
299 # rules = [SlicingRule("LineId",1),SlicingRule("RasterLineNum",1),SlicingRule("RasterColumnNum",1),
300 #         SlicingRule("NoddingPosition",1),SlicingRule("NodCycleNum",1),SlicingRule("IsOutOfField",1),SlicingRule("Band",1)]
301 # A custom rule could e.g. be to slice per scan with: rules = [SlicingRule("Id",1)]
302 slicedFrames = pacsSliceContext(slicedFrames,[slicedDmcHead],removeUndefined=True, removeMasked=True)
303 slicedDmcHead = pacsSliceContext.additionalOutContexts[0]
304
305 # Flag the data affected by the chopper movement in the mask "UNCLEANCHOP"
```

The slicing of the data is performed according to rules made explicit in the pipeline. In our example, one line is observed in four positions (ON, OFF, OFF, and ON). So, we expect 4 slices plus an initial slice containing the calibration block.

Check: after slicing

5 slices !

```

HIPE> if verbose:
    # an overview of the slicedFrames contents
    slicedSummary(slicedFrames)
    p2 = slicedSummaryPlot(slicedFrames,signal=0)
noSlices: 5
noCalSlices: 1
noScienceSlices: 4
slice#  isScience  onSource  offSource  rasterId  lineId      band          dimensions  wavelengths
0       false      no        no         0 0       [101]        ["R1"]      [18,25,679]  149.311 - 150.274
1       true        yes       no         0 0       [102]        ["R1"]      [18,25,6000] 159.040 - 162.242
2       true        no        yes        0 0       [102]        ["R1"]      [18,25,1500] 159.041 - 162.242
3       true        no        yes        0 0       [102]        ["R1"]      [18,25,1500] 159.041 - 162.242
4       true        yes       no         0 0       [102]        ["R1"]      [18,25,6000] 159.040 - 162.242
Slice edges: [0,679,6679,8179,9679,15679]

```

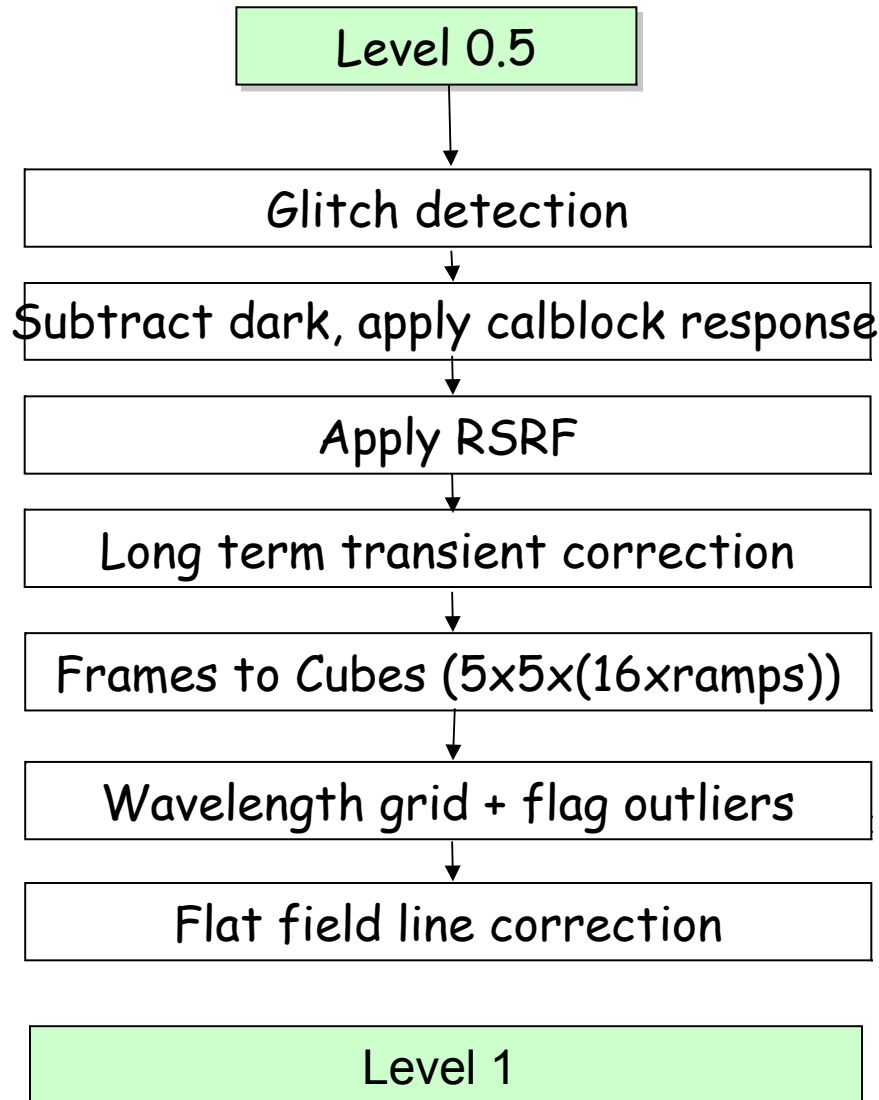
In the description we know which slice is on and off source, the wavelength range covered and the band. From this table, note the lineId number. This will be used later in the pipeline. In this case, we will display line 102.

Step 4

Run the 0.5 → 1 pipeline

Glitch detection, chop differentiation, RSRF, flat

Level 0.5 → 1



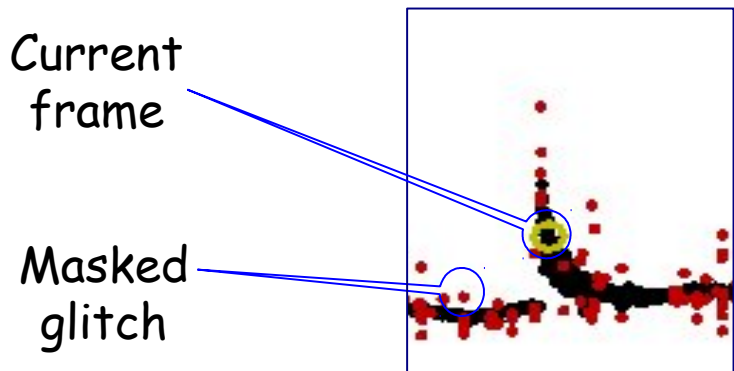
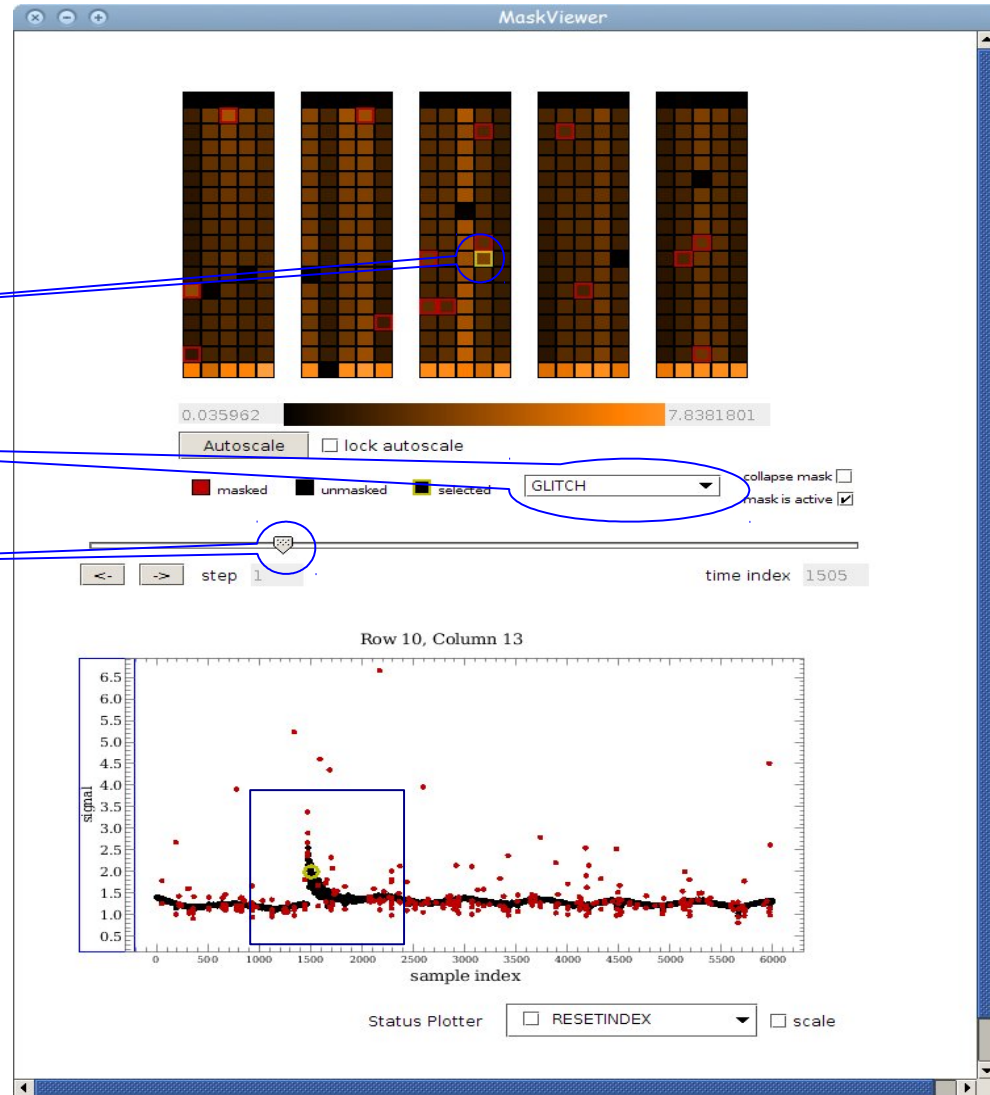
Glitch detection

You can view the signal and the masked readouts with the maskviewer. In this case, we find a transient unmasked after a strong cosmic.

Select a pixel by clicking on it

Select a mask

Select a frame

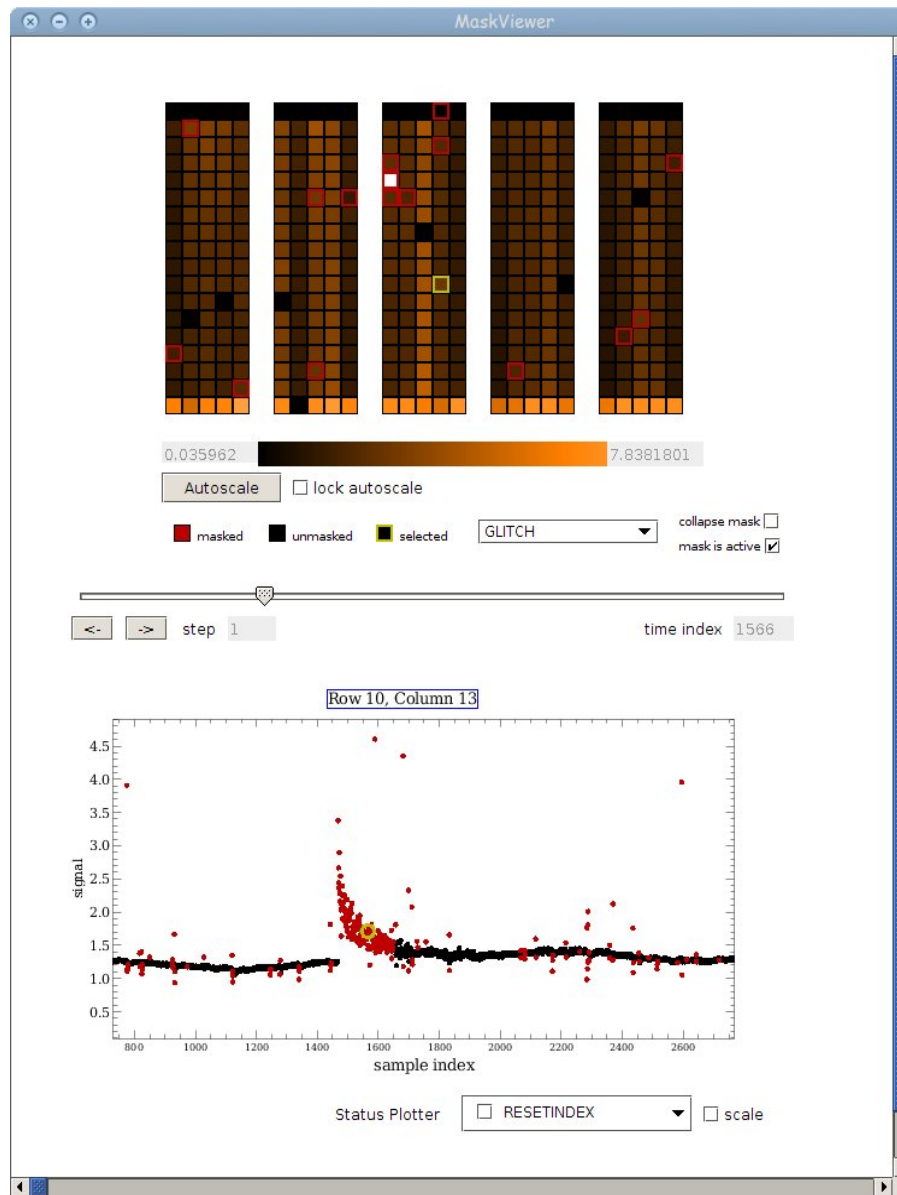


Manual masking

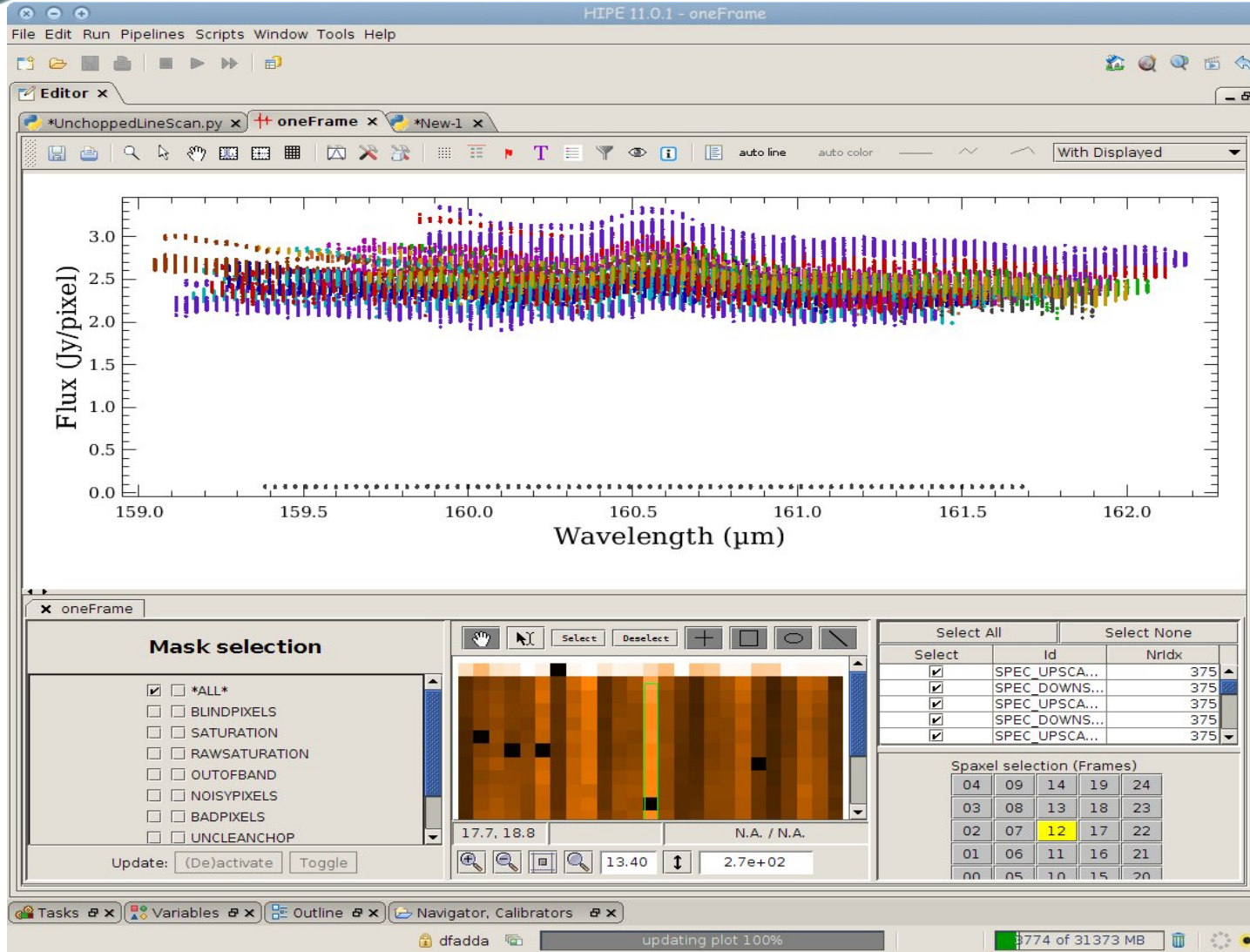
```
*UnchoppedLineScan.py x + oneFrame x *New-1 x
1  # How to use the maskviewer
2  MaskViewer(slicedFrames.get(1))
3
4
5  # Mask part of the signal
6  frame=slicedFrames.get(1)
7  for i in range(1470,1650):
8      frame.setMask("GLITCH",10,13,i,True)
9
10 # Once you have done all editing and checked them,
11 # Replace the original Frames with the edited one
12 slicedFrames.replace(1,frame)
13
14 # Show in MaskViewer the masked set in pixel 10,13 mask GLITCH
15 MaskViewer(slicedFrames.get(1))
16
```

At this point, we can manually mask part of the signal that are compromised, like the one seen with the maskViewer.

We can open a new tab to write these instructions. Part of the first frame is masked and then checked again with the maskViewer.

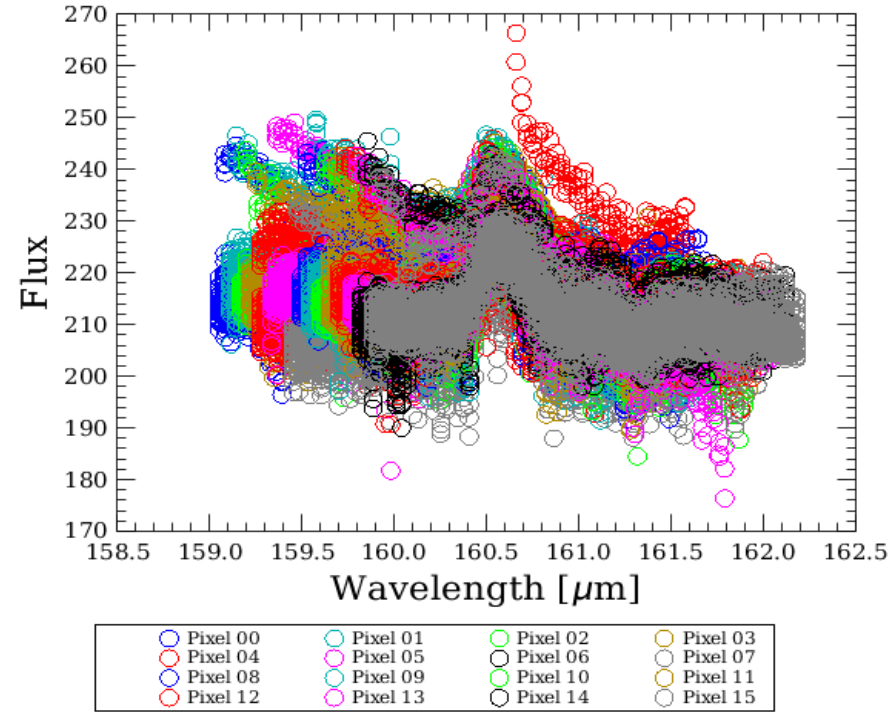
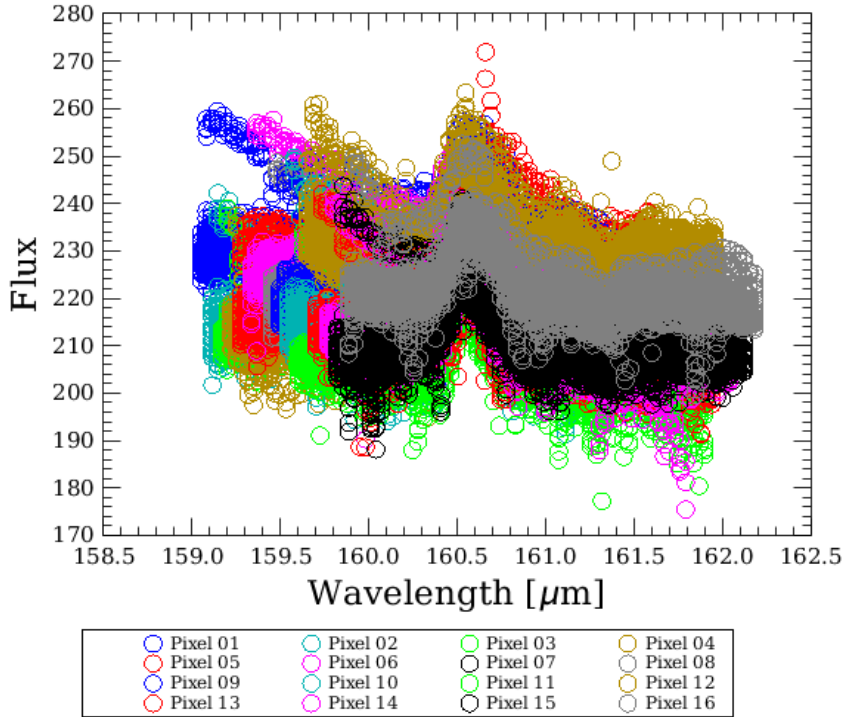


The affected region has been masked and it will not be considered in the following.



Pixels can be now examined with the Spectrum Explorer

RSRF, Dark, Response

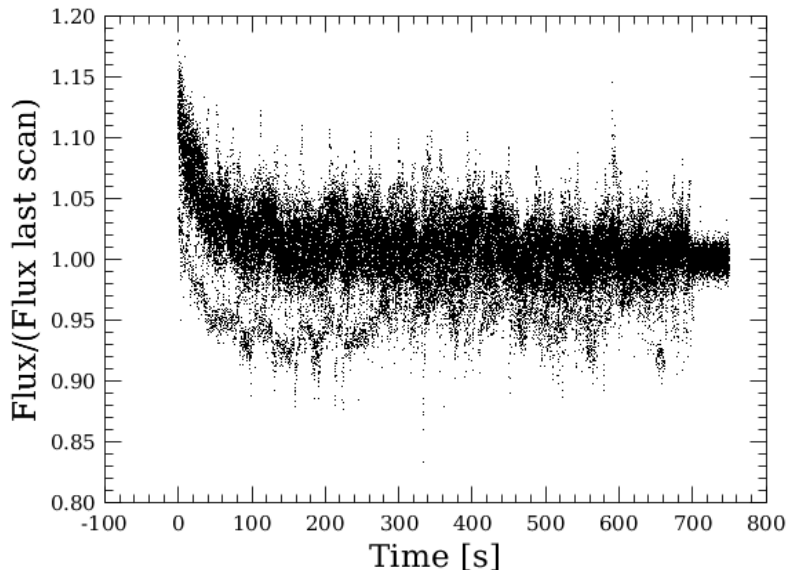


Central module before and after applying several corrections (dark, response, and RSRF)

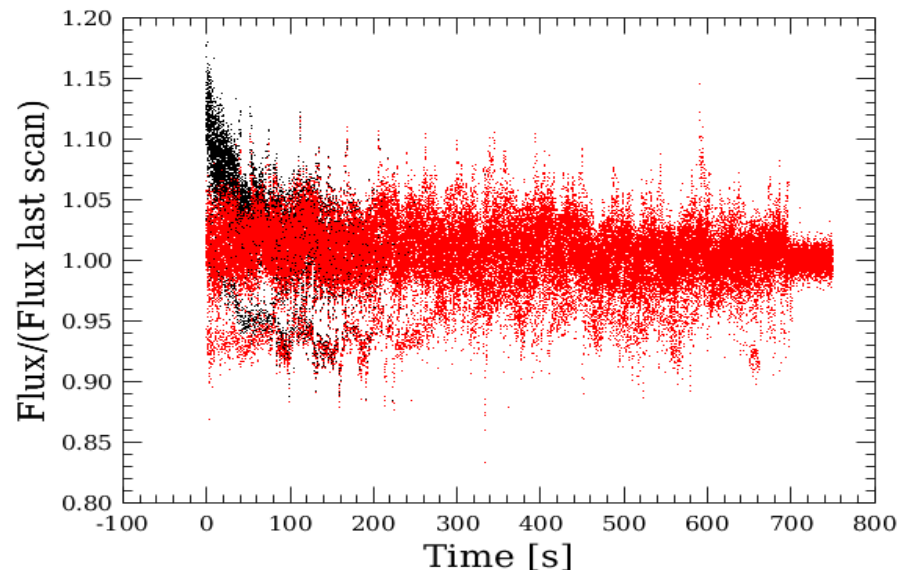
Long term transients

The effect of the transient correction is shown on the central module in the first slice after the calibration block. The plot shows the signal of the spectral pixels in the central module normalized to the signal in the last grating scan. Black and red are before and after the correction, respectively.

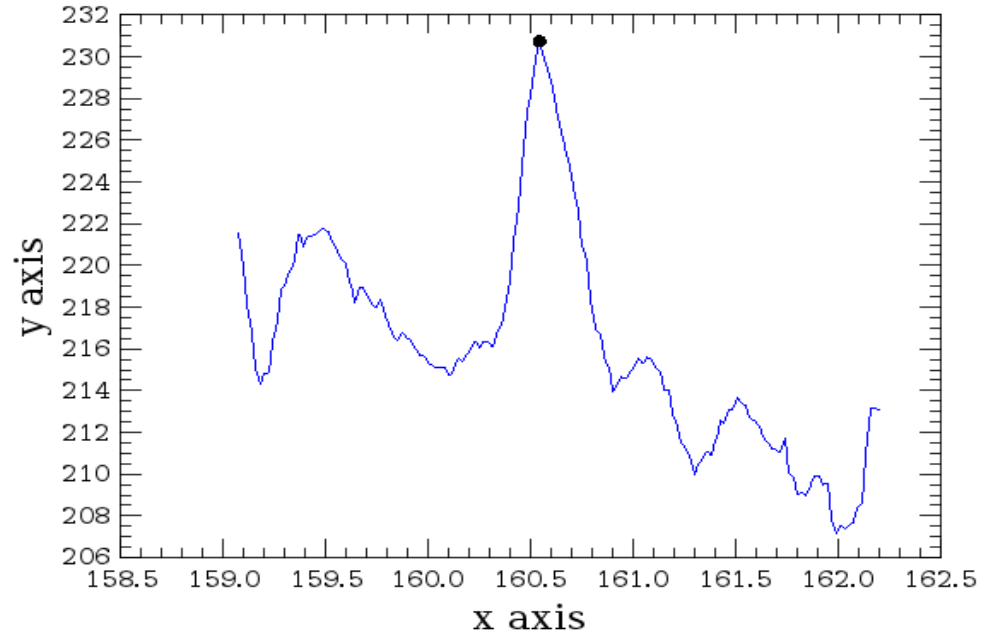
Slice 1 Module = 12



Slice 1 Module = 12



Spectral flat field



The spectral flat-field has been greatly improved in HIPE. Now, each module is explored to detect lines to avoid them when comparing the spectra from different spectral pixels. In verbose mode, the different spectra pop out and the line are identified with a black dot.

Spectral flat field

In the case one knows what to mask (e.g. in case of a known line) or if an absorption line has to be masked, it is possible to enter the list of lines to be masked and do the masking manually.

In this example, the C+ line is masked by changing:

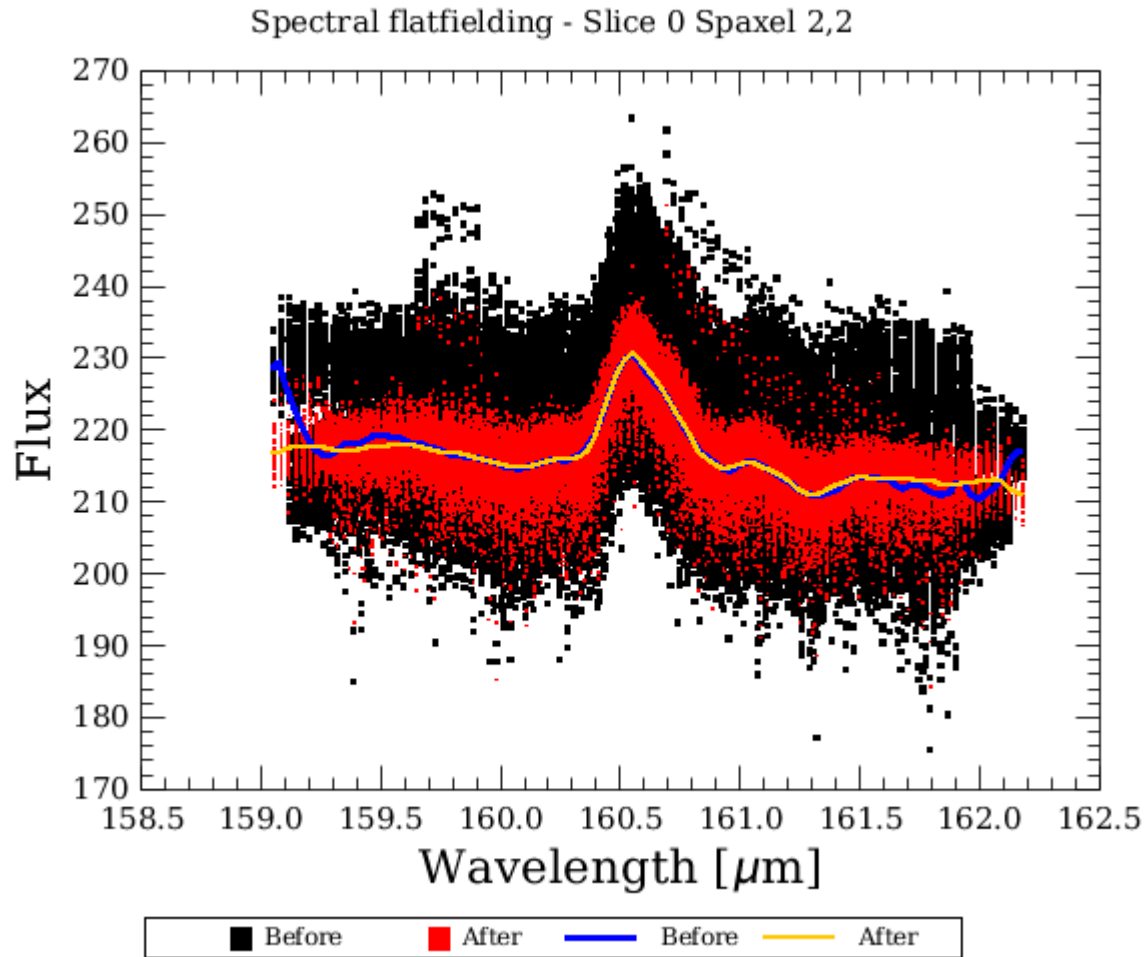
```
LineList = []
```

into

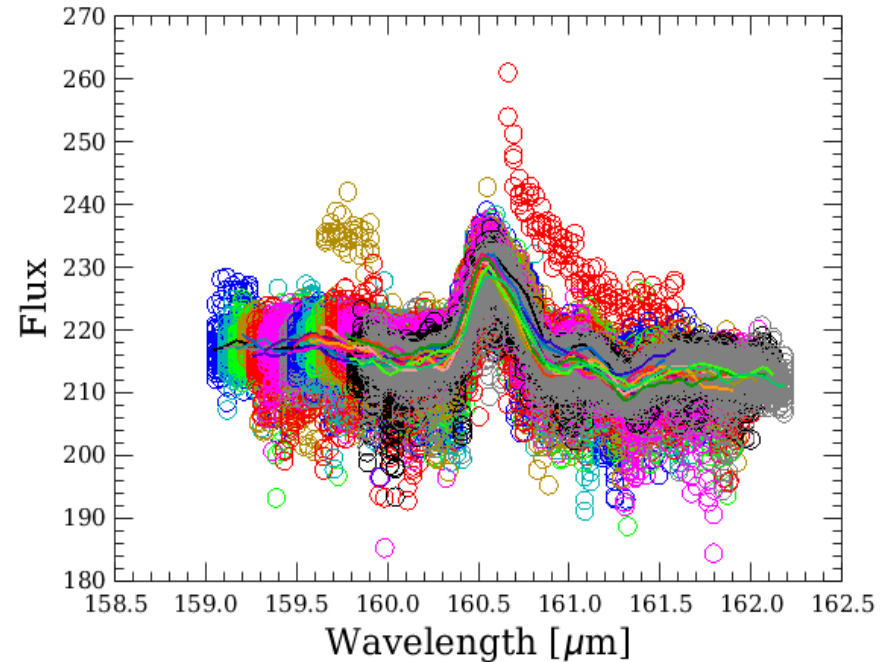
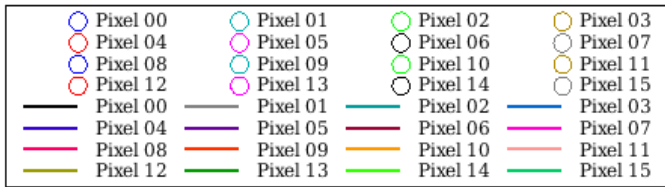
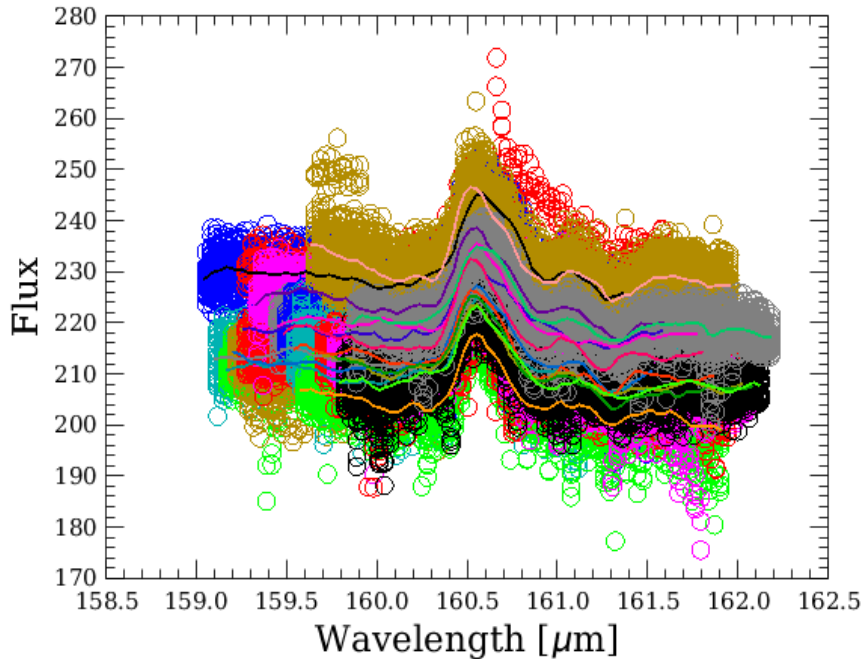
```
LineList = [106.600]
```

```
slicedCubesMask = maskLines(slicedCubes,slicedRebinnedCubes,  
lineList=[106.600], widthDetect=widthDetect, widthMask=2.5, threshold=threshold,  
copy=1, verbose=verbose, maskType="INLINE",calTree=calTree)
```

Spectral flat field

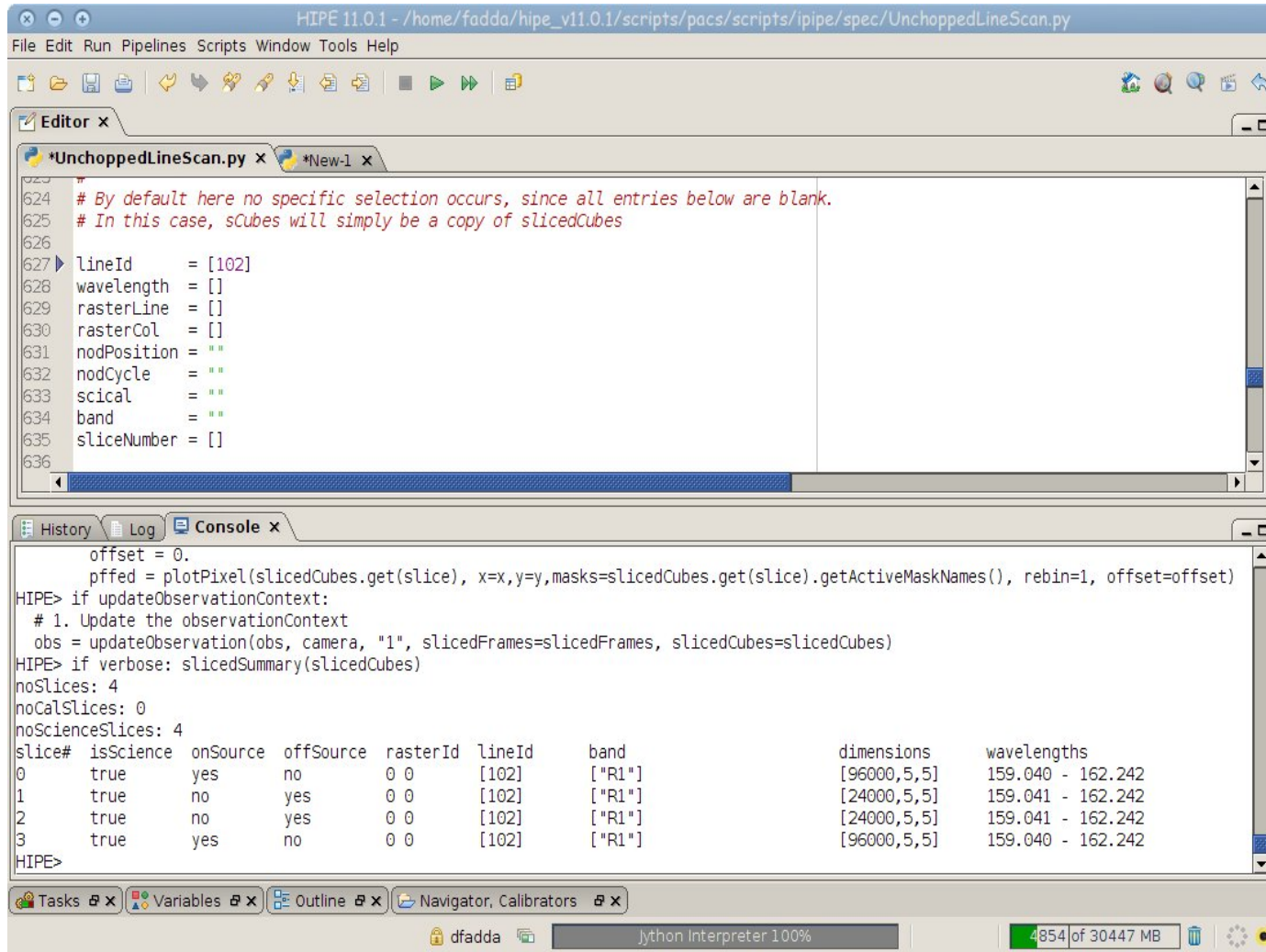


Spectral flat field



Central module before and after the first spectral flat-fielding
 The transient correction has greatly improved the final result.

Level 1 → 2



The screenshot shows a Python IDE window titled "HIPE 11.0.1 - /home/fadda/hipe_v11.0.1/scripts/pacs/scripts/ipipe/spec/UnchoppedLineScan.py". The editor displays the following code:

```

624 # By default here no specific selection occurs, since all entries below are blank.
625 # In this case, sCubes will simply be a copy of slicedCubes
626
627 lineId      = [102]
628 wavelength  = []
629 rasterLine  = []
630 rasterCol   = []
631 nodPosition = ""
632 nodCycle    = ""
633 scical      = ""
634 band        = ""
635 sliceNumber = []
636

```

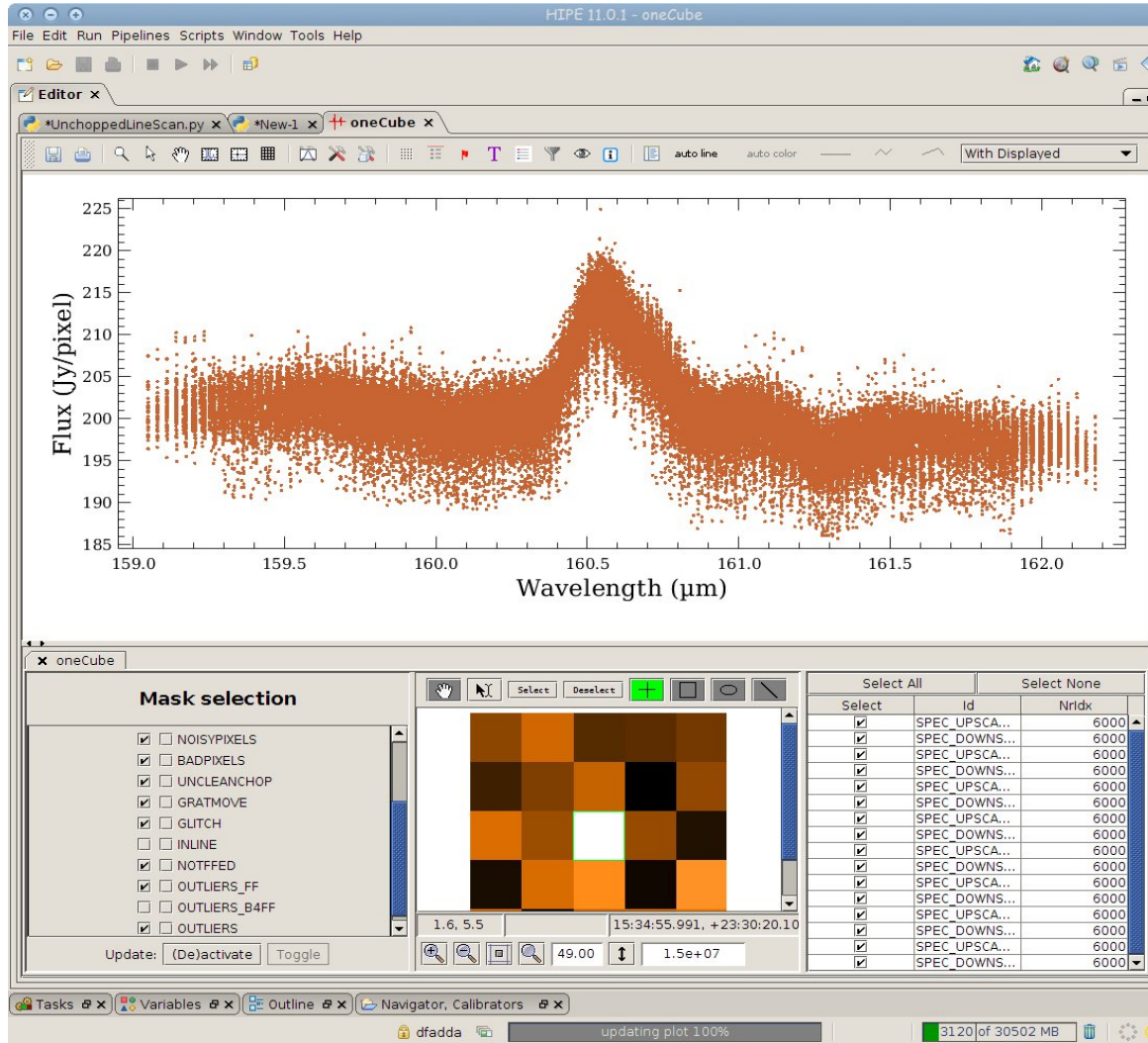
The console output shows the execution of the script, including the output of the `slicedSummary` function:

```

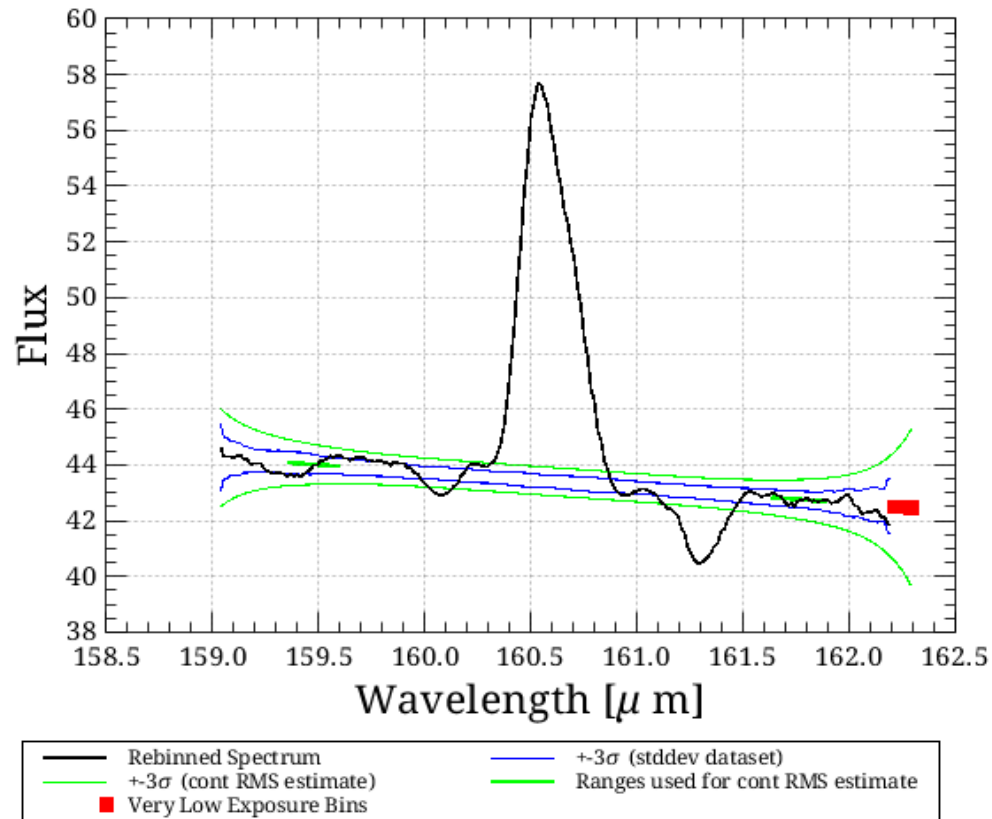
offset = 0.
pffed = plotPixel(slicedCubes.get(slice), x=x,y=y,masks=slicedCubes.get(slice).getActiveMaskNames(), rebin=1, offset=offset)
HIPE> if updateObservationContext:
  # 1. Update the observationContext
  obs = updateObservation(obs, camera, "1", slicedFrames=slicedFrames, slicedCubes=slicedCubes)
HIPE> if verbose: slicedSummary(slicedCubes)
noSlices: 4
noCalSlices: 0
noScienceSlices: 4
slice#  isScience onSource offSource rasterId lineId band dimensions wavelengths
0 true yes no 0 0 [102] ["R1"] [96000,5,5] 159.040 - 162.242
1 true no yes 0 0 [102] ["R1"] [24000,5,5] 159.041 - 162.242
2 true no yes 0 0 [102] ["R1"] [24000,5,5] 159.041 - 162.242
3 true yes no 0 0 [102] ["R1"] [96000,5,5] 159.040 - 162.242
HIPE>

```

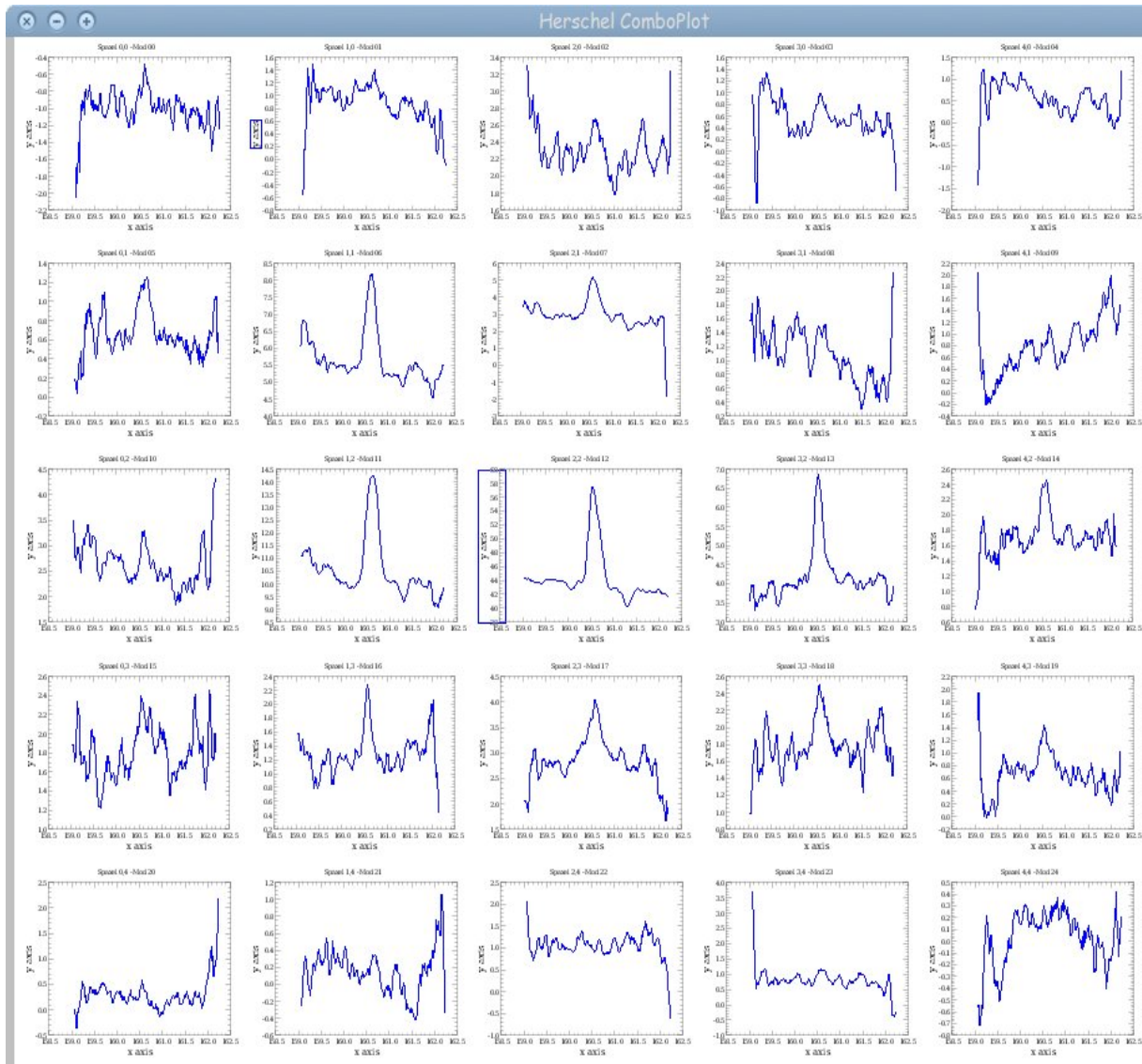
We have reached level 1. To proceed, we choose a line (102) by reading the output of `slicedSummary`. The level two routines are explained in the PACS 302 tutorial.



We can inspect the line with Spectrum Explorer



plotCubeStandardDeviation shows the spectrum for the central pixel with error bands.



After subtracting the OFF cube from the ON cube, a multi-plot feature allows one to see all the spectra in the different spaxels.

Multi-threading

When running interactive pipelines, be sure to use the multi-threading option. **This option is only available for the PACS spectrometer** and it will speed-up your reduction by exploiting all the cores of your machine.



Using this option is extraordinarily simple. Just two lines:

```
Configuration.setProperty("herschel.pacs.spg.common.superThreadCount","4")  
Configuration.setProperty("herschel.pacs.spg.spec.threadCount","8")
```

Some tasks are threaded. The other ones are naturally threaded by exploiting the slicing of the data.

The **superThreadCount** is used for the general threading, while the **threadCount** is used for the threaded tasks.

The optimal choice of the threading parameters depends on the number of cores on your machine and the number of slices. Memory is not an issue, because the first part of the pipeline is unthreaded and puts the entire data in memory. When data are sliced, the total memory used is always the same.

An automatic choice is done by putting:

```
Configuration.setProperty("herschel.pacs.spg.common.superThreadCount","0")  
Configuration.setProperty("herschel.pacs.spg.spec.threadCount","0")
```

Otherwise, a good choice is to put `threadCount` equal to the number of cores and `superThreadCount` equal to the number of slices.

Unchopped range scan

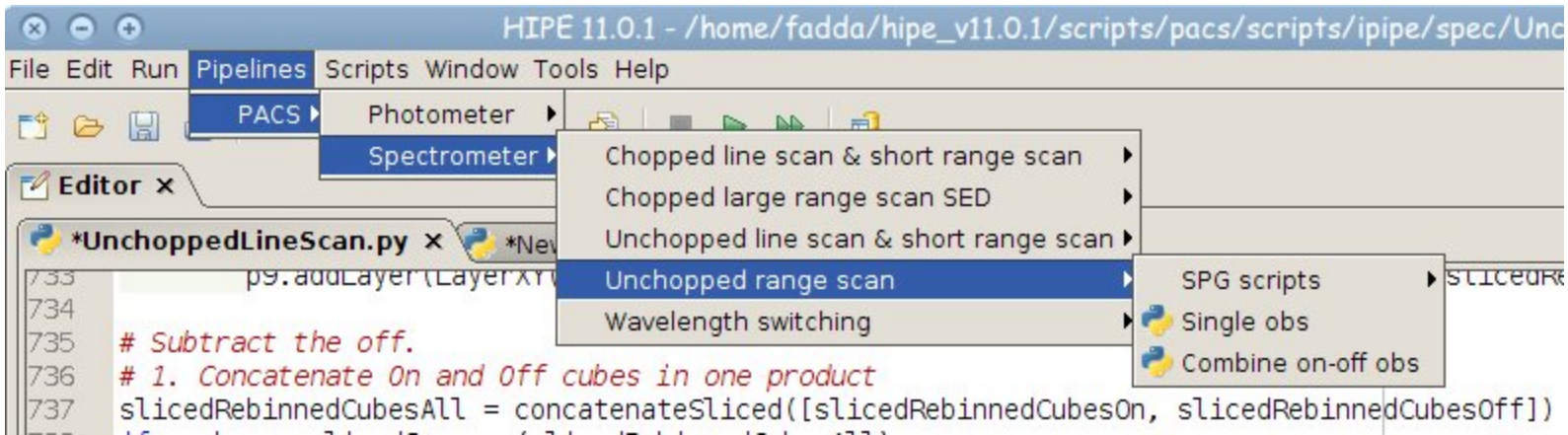
Also for unchopped range scan it is possible to run an interactive script. The difference with the unchopped line scan is that:

A) there is no transient correction module

B) ON and OFF source observations are done in different observations. So two obs-ID numbers are required to reduce the observation properly.

This is done using two scripts: one to reduce each obs-ID and another one to combine them.

Unchopped range scan



The two interactive scripts available to reduce range unchopped observations:

- Single obs for the ON and OFF observations
- Combine the two AORs.