HERMES Pipeline V 6.0
User Manual

Introduction

This sixth version of the pipeline, summer 2014, was tested on the HRF, LRF, WRF and LRF-WRF data on unbinned frames. We assume the installation is done by your local Hermes DRS guru, but this installation now uses pip in order to force needed packages into the wanted version. The pipeline was tested on several platforms and flavors of linux (linux SUSE, linux Ubuntu, mac).

A normal installation contains the pipeline, which is installed in the “pipeline” folder, as well as additional tools which were developed to test the pipeline. The latter were added in the “tools” folder. This cookbook document is found in the “pipeline/run/doc” folder.

The main purpose of this manual is to serve as a cookbook for both novel as experienced HERMES-DRS users. The data is reduced preliminary during observations to allow the observer to assess easily the quality of the data in real-time. Moreover, the full pipeline is ran every morning, allowing the observer to have the fully reduced data ready by the start of the next night. A web-based logbook system is in development at the telescope which will allow the observer to communicate events to the PIs of the programs.

This manual refers to the software living in:

http://hermescvs.ster.kuleuven.be/repos/mercator/software/releases/DRS/hermes

that holds the revision number 8510.

Document history:

19.07.2009 – “First draft” by Y. Frémat, H. Hensberge, L. Dumortier (ROB)
23.07.2009 – “Comments concerning the scripts” by H. Van Winckel.
24.07.2009 – “Changes” by H. Van Winckel.
24.07.2009 – Release 1 by Y. Frémat and H. Van Winckel
18.08.2009 – “Explaining the setup” quick jump-in guide by L. Dumortier.
10.09.2009 – “Input, Output and syntax details” by L. Dumortier.
22.09.2009 – “Graphical User Interfaces” by L. Dumortier
30.11.2009 – “With Radial Velocity Calculations” by A. Jorissen
23.02.2010 – Release 2 by L. Dumortier and H. Van Winckel
06.05.2010 – Y.F. addition of rebinning (option in applyDRS) and mergeDRS.
14.09.2010 – order offset determination is now automatic, and merging happens in the applyDRS. 3 modes rebinning.
26.09.2010 – Preparing branch for release 3 – Yves Fremat, Hans Van Winckel, Alain Jorissen, Sophie Van Eck, Nadya Gorlova and Louis Dumortier
08.10.2010 – Release 3
15,01,2010 – nightDRS is now multiprocessing, This is much faster, Hans Van Winkel,
15.02.2011 – Updates relative to the COP determination. - Y.F.
03.03.2011 – New log system now always writes in a file. Louis Dumortier
12.05.2011 – New release 4 – LD + HVW
30.09.2011 – starlist.py added in the tools
20.12.2011 – transforming the software for multi-fiber purposes.
20.03.2012 – twoFibDrift is added.
30.05.2012 – New Release 5
29.11.2012 – Modified hermesVr.py to include the possibility of fitting rotationally-broadened profiles – A. Jorissen
30.06.2014 – Code adapted to pyfits 3.2.2, added pip requirement files.
Bugs removed and code enhanced.

Table of Contents

General rules

1) Naming Conventions for image files:

1.Every raw file is saved in the raw folder of the night and is composed of: a unique eight (former six) digit unique number, underscore, fibername, underscore, exposure type and "`.fits`" as lowercase extension. This is the file index is also stored in the fits header under: "UNSEQ"

2.Fiber is one of `{"HRF", "LRF", "WRF", "LRF_WRF"}`, uppercase only.

3.Exposure is one of `{"BIAS", "DARK", "FF", "TH", "OBJ", "OBJ_TH"}`, uppercase only.

4.Fiber type is determined in the fits header under the keyword "FIBMODE".

5.Exposure type is determined in the fits header under the keyword "EXPTYPE".

6.Irrespective of the other name attributes, there are NEVER two raw input files with the same sequence number. However, reduction programs are producing several output files as intermediate step or final product, with the same sequence number of the raw file, but with different attributes.

7.The fiber and the mode shown in the name are only there to help finding the good file (globbing) without reading each and every header. The programs read the fiber and mode from the header of the files, not from the file name. Renaming a file will NOT be sufficient to change the fiber nor the mode.

8. From this release on, the extractorders program in two fiber mode is producing two files : "_LRF_OBJ_2F" and "_WRF_TH_2F". This addiction of the "2F" suffix was needed to distinguish them from standard LRF or WRF files, mostly to let "twoFibDrift" find them without confusion.

Examples of raw inputfiles:

```
240501_HRF_FF.fits
240511_HRF_TH.fits
240530_HRF_OBJ.fits
00304774_LRF_WRF_OBJ_TH.fits
```

2) File Indices

In every command, the keyword will be followed by at least one file index. This tells the program on which file to work.
You can also give the file name, but only the (first) index found in that name will be extracted to look at a list of files beginning with that index.
The index is extracted by a routine using regular expression that searches for the first complete integer in the string. It will return heading zero's as part of the index. No signs or decimal positions are treated as part of the index. The index number is strictly positive.

Examples :

input file name	index
abc123456_HRF_TH.fits	**123456**
000454545_HRF_TH.fits	**000454545**
-78912-TH.fits	**78912**
gh789456.78_TH.fits	**789456**
FF123TH789.fits	**123**

Renaming a file is not a good idea : The DRS programs relies on the file names to select file lists of a certain type, and on the fits header content to detect the real content of an individual frame. Header and file names have to match ! Do not rename a file unless it is the final product of the DRS reduction !

3) Blacklists

Blacklists are list of indexes of files, which the user does not want to treat.

There are two blacklists incorporated into the system. The first one is a global one and it contains known corrupted (FF, TH, OBJ) files as well as binned images. This global blacklist is automatically read from the software and is embedded into the software tree. A second blacklist can be defined by the user and can contain sequence numbers of files which the user does not want to treat in his/her session. Both lists are read by the “absoluteDRS” script and merged to create the internal file reject system.

Local blacklist

One may create a file called “blacklist.xml” in the local hermesRun folder, to contain a list of indexes (or file names from whom the indexes will be extracted) that will NOT be treated by the absoluteDRS script. This facility is made to except certain files from global treatments. The format of the blacklist file is in .xml and is as follows:

```
<indexes>                          <- your preferred global name for this list (mandatory)
    <shiftedBy16pixels>              <-  the name you give for the first index (be explicit, it will help)
        123456                           <-    the first file index
    </shiftedBy16pixels>
    <N007>                           <-  index names may not begin with a figure
        007007                           <-  but indexes obviously may
    </N007>                          <-  always close all your names like this
</indexes>
```

Using this tool, one can ban a file forever without having to retype its index over and over again. But blacklist.xml is personal, hence, it is located in your folder (usually in hermesRun).

Global blacklist

There is a similar file in the “nights” folder. That file is maintained by the Leuven team: it holds indexes of known corrupted files.
Changes of this file can only be done by people of the software development team and/or the Mercator team of the IvS. If you install the software yourself, please do also update regularly, otherwise ask your DRS responsible to do so.
The file is located in the directory where also the products of the nights are located so that the update should be easily performed on the same time as the nights, on a regular basis.

3) Command Syntax

a) General principles

The syntax and description of each command is detailed here.

Optional parameters are enclosed in brackets **[ ]**

Parameters enclosed in braces **{ }** indicate that you must enter one and only one of the possible choices shown enclosed. Those parameters are separated by commas that have not to be typed.

Points **....** indicate that the preceding parameter can be repeated.

Single quotation marks and parentheses must be entered where shown.

A space *must* be entered to separate all parameters and keywords.

All keywords are referring to python programs, and on some Linux flavors they may run without invoking the python command itself.

For every script of the pipeline, the syntax is described and at least one example is shown.

Do not use a keyword more than once in the same command. Only the first one will be considered. No error message will be issued when this is done.

b) Grammar rules

One can always invoke a program by its name in the following way :

```
python progName.py
```

On some Linux flavors, a shorter way can work :

```
progName.py
```

This will usually lead the program to try to display a Graphical User Interface (GUI) when possible (not every program has the ability, nor every computer has). When no GUI is displayed, a syntax help screen will guide you, it generally begins with the word “USAGE” and shows up something like this :

```
USAGE  :

python absoluteDRS.py -f nindex -l nindex [-o nindex] [-b] [-s 1] [-r 17] [-d] [-e
OBJ]"

-f nindex : Number of the first file in the series (found in the DataInput path)
-l nindex : Number of the last file in the series (found in the DataInput path)

OR

python absoluteDRS.py -i nindex [-o nindex] [-b] [-s 1] [-r 17] [-d] [-e OBJ]

-i nindex : Number of a single file to proceed (relative to the DataInput path)


-o: Name of the model order positions file to use (found in the DataOutput path)
      Optional. A new one will be calculated when not given.
-b: Optional. When found, skip the background measurement. Faster.
-s: Optional. Step in background calculations. Step is 1 by default, higher
      values will speed up background calculation. Debug purpose.
-r: Optional. Specify an offset to the order model when instrument has a little
      shift.
-d: Optional. When present, write debug files in the debugPath
-e: Optional. Exposure type. Accepted values are FF, BIAS, TH, OBJ
                  When found, file selection will be limited to those
                  having that exposure mentioned in the name.
-c: Optional. Remove Cosmics.
```

It is a useful short reminder of the parameters you can use.

c) Details of the Grammar Rules

• All parameters and keywords must be separated by a space.

• Every keyword that requires a parameter must be followed by a space and that parameter.

• Some keywords have no parameter at all. They work as booleans : when they are present, the program will do that action, otherwise not. Example : -d stands for “debug information required” in almost every program. When not found, no debug information will be written in the Debug folder.

• The order in which the keywords are given is of no importance, but they have to be followed by their parameter value when required.

It is the same to write

```
-s 1 -d -r 17
```

than

```
-d -r 17 -s 1
```

or

```
-r 17 -s 1 -d
```

is equivalent.

d) Most used mandatory keywords

-i stands for input. The number that follows is the index of the input file.

-f stands for first. The number that follows is the index of the first input file.

-l stands for last. The number that follows is the index of the last input file.

The couple of keywords -f -l and the -i keyword are mutually exclusive :

{ -i, [-f -l] }. Use -i ***or*** -f and -l.

When -f and -l indices are given, the files pointed to by those indexes are included in the selection.

-o stands for order model. Generally this keyword is followed by the file index of the order model we want to use.

-e stands for exposure. Has to be in {TH, OBJ, BIAS, FF}.

e) Most used optional keywords

-d stands for debug. When present, the program will write files with intermediate reduction steps into the Debug folder.

-r specifies that the instrument has a shift against the instrument model.

-s stands for step. In some parts, long computations can be done only every s step, to fasten up the program. Used in debug.

-p stands for plot. When present, programs that include graphics will show them. In any case, the graphics will be written to disk as .png files. They are by default not shown on screen to facilitate the running in background or on virtual machines.

Installing the software

Several steps must be done before running :

1) Set the PYTHONPATH

1. If you install the software yourself: Create a folder called “hermes” and copy the complete content of the Hermes folder of the latest svn release into it. If the software is installed on you local system, ask your local instructor.

2.There must be an environment variable named PYTHONPATH which points also to the Hermes folder (This can be different with IDEs (integrated development environment) like eclipse, komodo, wing, and others). For example:

```
export PYTHONPATH=/home/software/DRS/hermes/:$PYTHONPATH
```

Under Linux, either the installation is done by your local instructor and you just source the rc file, or one should edit e.g. /etc/profile and add the export instruction as shown above, with the path adapted to your needs. The /etc/profile is valid for every user and is read at login. Therefore, the first time, you will have to retype the export in your shell, for every shell, or restart the machine to be sure.

Type “**`cd $PYTHONPATH`**” to check that it works.

2) Set the input and output paths

All the raw frames are assumed to be in a directory known by the pipeline as the *DataInput* path.

All the results of the pipeline will be stored in a directory known by the pipeline as the *DataOutput* path.

When debug information is asked, files are written in the d*ebugPath* path.

Analyses results such as those produced by computeRadialVelocity are written in a global path defined by the *AnalysesResults* path

DebugPath *and* *AnalysesResults* are defined in the hermesConfig.xml file located in the hermesRun dir in your home dir. If it does not exist, it will be created and the variables will be given the default values. One can always create that configuration file with default values by running “setup.py”.

One can use the system variable “HERMESRUN” to specify a folder where the folder “hermesRun” will be found, or written if not found.

To run the setup.py script. In the hermes folder, execute :

```
python setup.py
```

It will create a file if it is not found, in the “hermesRun” folder in the home dir. That folder will also be created when needed.

```
hermesConfig.xml
```

One should first check this .xml file and adapt the xml keywords for your local needs. The file hermesConfig.xml is located in the “hermesRun” folder in the home folder. Under windows, the home folder is the folder that contains the “my documents” folder.
The pipeline assumes that the data is stored on your system in the standard way in which a folder like YYYYMMDD/raw contains the raw data.

Example of a hermesConfig file and paths :

```
<hermes>
      <Nights>sftp://user@hserver:22/data/nights/</Nights>
      <CurrentNight>sftp://user@hserver:22/data/nights/20110725</CurrentNight>
      <Raw>raw</Raw>
      <Reduced>/home/user/nights</Reduced>
      <AnalysesResults>/home/user/hermesRun</AnalysesResults>
      <DebugPath>/home/user/hermesDebug</DebugPath>
      <ConsoleLogSeverity>debug</ConsoleLogSeverity>
</hermes>
```

In the currentNight path, folders are known to carry original files and produced files.

•The *DataInput* path is made of the **raw** folder in the **CurrentNight** path. The CurrentNight is defined to by the "**CurrentNight**" xml tag of the config file. Those files should never been modified, or better : the raw folder is normally write protected.
Example : "Nights/20100912/raw"

•The **reduced** folder pointed to by the "Reduced" tag of the config file will receive the results written by the programs.

Change these paths to reflect your local situation before running the software.

From previous release on, it is possible to have the results written in your local home dir, or use the DRS in background and point to the input files situated on some server while the hermes program runs from a virtual machine and writes its results on your desktop pc. It also makes it possible to keep a copy of the original input and output files, and to run some programs without modifying the original files. The disk carrying the original input and output files may now be read-only.

3) Testing the installation

It is really easy to test the local installation of the hermes pipeline.
Three things must be correctly set-up in order to run the pipeline commands:

1) The PYTHONPATH must point to the "hermes" folder in which the sofware is deployed:

type "cd **$PYTHONPATH/pipeline/run**"
this should lead you to your local hermes "run" folder, or, for users using a central software repository, to that folder. Aliases can easily be defined so that you run the pipeline/run scripts from where-ever in your system.

2) The Python Libraries must be installed :

Users running the Hermes DRS from a central software repository will normally find everything installed correctly.
For other users, to test the presence of the needed python libraries, use the "hermes.py" program found in the "pipeline" folder.

type "**python hermes.py**"

This tests that all needed libraries are present, and up to date to a sufficient level of trust.

You should see this kind of message (version values are supposed to evolve)

```
>>>

     This tests your Python installation for Hermes.
     ===============================================

You are running Python version 2.7.3 (default, Feb 27 2014, 20:00:17)
[GCC 4.6.3]
Your numpy version 1.8.1 is up to date or sufficient.
Your scipy version 0.14.0 is up to date or sufficient.
Your pyfits version 3.2.2 is up to date or sufficient.
Your pyephem version 3.7.5.1 is up to date or sufficient.
Your matplotlib version 1.3.0 is up to date or sufficient.
/usr/lib/python2.7/dist-packages/gtk-2.0/gtk/__init__.py:127:
RuntimeWarning: PyOS_InputHook is not available for interactive use of
PyGTK
  set_interactive(1)
pygtk is found  {'2.0': '/usr/lib/python2.7/dist-packages/gtk-2.0'}

>>>
```

The "pyfits" version is critical with this release and has to be at least 3.2.2

The development was done and tested on UBUNTU 11.04, with mostly all standard packages installed from the package manager, except for pyfits and pyephem.

3) Installing needed libraries

This new release introduces the ability to install all needed packages automatically, with the version hermes needs. To do this, you must have the utility called “pip” installed :

Install pip

To install or upgrade pip, securely download `get-pip.py`.

Then run the following (which may require administrator access):
`python get-pip.py`

If setuptools (or distribute) is not already installed, get-pip.py will install setuptools for you.

To upgrade an existing setuptools (or distribute), run `pip install -U setuptools`

To enable the use of pip from the command line, ensure the Scripts subdirectory of your Python installation is available on the system PATH. (This is not done automatically.)

Install Hermes with pip

go to the installation folder (the “hermes” called one) and issue this command :

```
svn up
sudo pip install -r HermesDRS-requirements.txt
```

Manually install hermes

Commands to install those packages may vary following your distribution.

```
Pyfits      3.2.2
pyephem     3.7.3.4
```

Were installed from sources. Just download the sources compressed file, unzip it, and in the unzipped folder, go type
`sudo python setup.py`

In order to unleash the power of the GUI's, install GTK and pygtk, at least version 2.2, before adding matplotlib. GTK is a general package to download and unzip in a folder, say “GTK”, that folder has to be added to your path. pygtk is installed by **`python setup.py install`**

4) The input, output, debug and results paths must be set :

Those paths are read from the “hermesConfig.xml” file in a hermesRun folder in the home folder. The “setup.py” program will help you doing that in an easy graphical output,

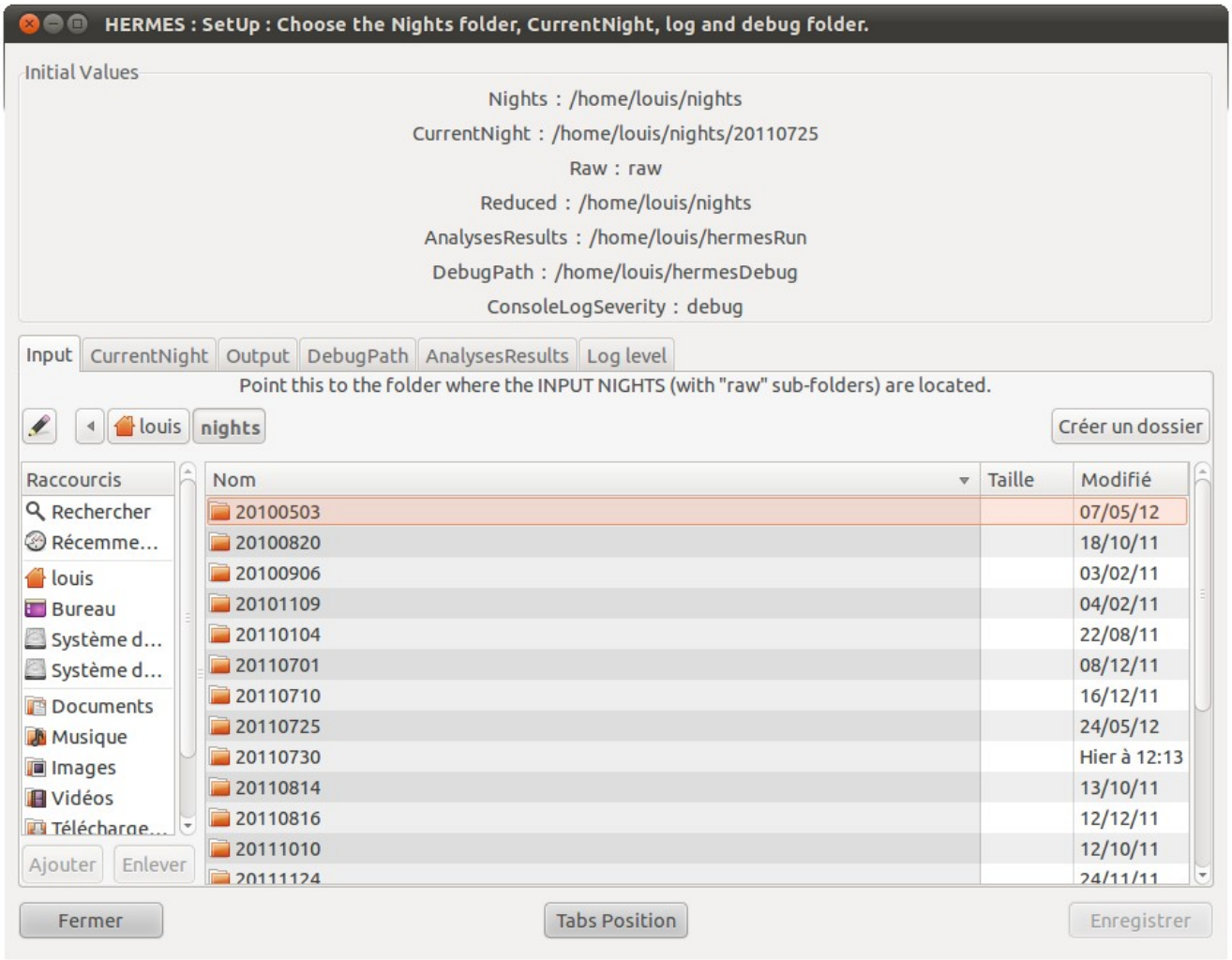
Running the software: cookbook for a full reduction process

STEP 1: Folders setup

This step must be done only once at the first run of the software implementation. It creates the configuration files only when they do not exist yet. Those configuration files are used by the software to locate the input files, and where the results and the debug information will be written. It also upgrades the config files from previous releases.

The file “hermesConfig.xml” is written in a folder called “hermesRun” in your home dir. That folder also contains all local information of and from your runs. Under windows, the home folder is the one that contains the “my documents” folder.

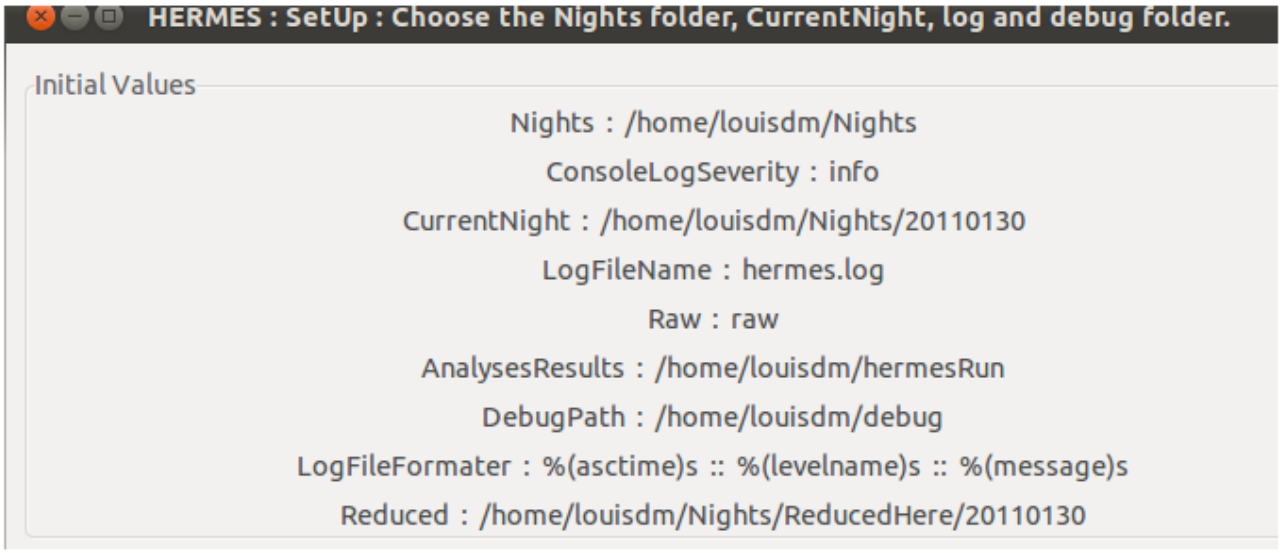
Edit the file “hermesConfig.xml” to make the software point to the folders you want, or alternatively use the “setup.py” utility to get an interactive way to define the paths. The setup.py shows up this interface :



Select the paths in the five tabs and record the changes to make the package run on those paths.

The graphic interface of the setup.py is divided in three parts.

First top part shows up what there is in the file when initially loaded. It is an easy way to remember how it was before the user changed it, without having to reload the file.

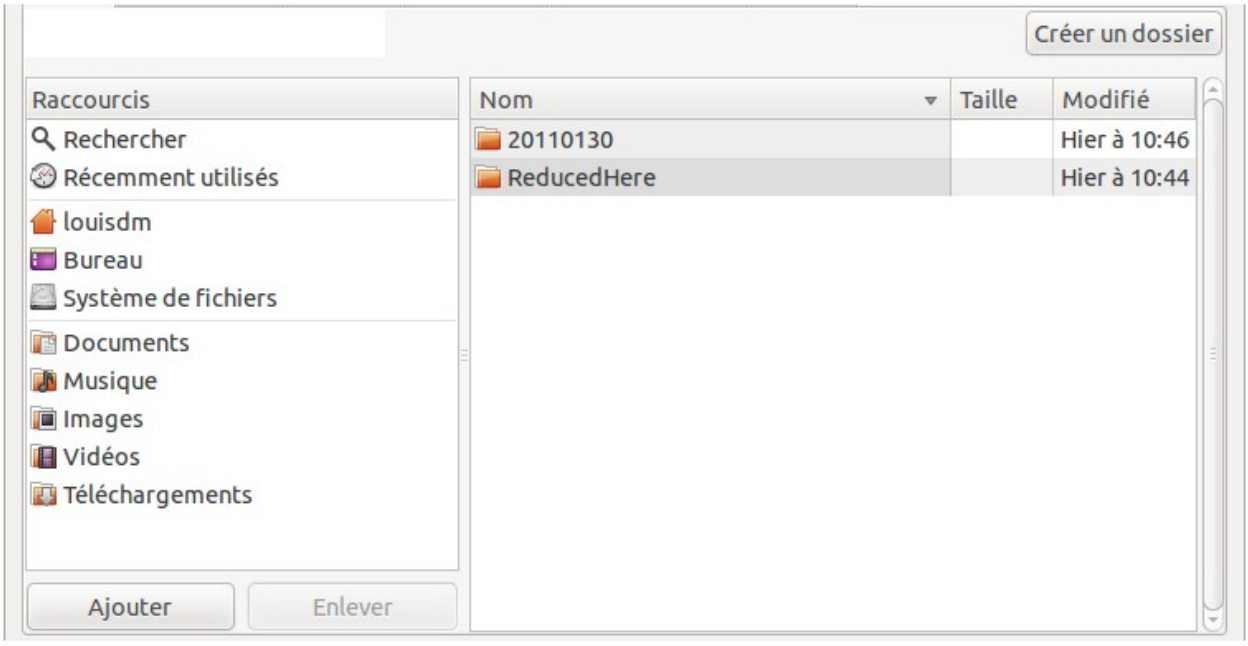


In the middle there is the selection system for the paths and other settings.

Here the tab “Nights” is selected, and the path associated is $HOME/Nights

One should select the correct path for every tab.

There are easy ways to select existing folders without typing them:



On the **left**, the panel is divided in three parts :

Search a place, or look in the recently used subdirectories.

Below this, the “add” button will add any of your common places to this list so that you can then access it immediately.

On the right side, there is a “create folder” button.
And below, the file system let you navigate in your folders to choose the correct one.

Where to run ?

The python scripts as front-ends to run the underlying software components, are placed in folder **`hermes/pipeline/run.`**

One can easily go there by doing :

```
cd $PYTHONPATH/pipeline/run
```

But the software is fully adapted to be run from anywhere, as long as the pythonpath is correctly set and the config files are pointing to the right input and output folders. Therefore, users running software from a central software repository should better go to their home/hermesRun folder, where all their local config files are to be found. One can easily make aliases to call python.

STEP 2 : Check the order alignment (automatic within a allowed range)

For various reasons, the reference position of the orders on the CCD frame may vary (slightly or significantly) with time. In the previous versions of the pipeline it was asked to the user to manually measure this offset using the **`checkOrders.py`** script and to provide its value to the **`absoluteDRS.py`** script (option **`-r`**) while extracting the Flat Fields.

From the 3rd release on, this order offset is automatically estimated by the pipeline before extracting the Flat Fields.

The value is stored in the header of the extracted FlatField (e.g. 00297216_HRF_FF_ext.fits) or corresponding orderpositions (e.g. 00297216_HRF_FF_modOrdpos.fits) under the keyword:

`HIERARCH shiftApproximateOrderPositions`

However, one can still check the correctness of the automatically derived order offset using the **`checkOrders.py`** script which is still maintained (see Page 65).

Note that if the order offset is larger than 14 pixels, the construction of a new instrument model is required.

STEP 3 : Extraction of the Flat Fields

This step works on FF exposure type files found in the input folder.

This step involves input files with an FF as last part of their name. All indexes shown on this page refer to file names continuing with "**`_FF.fits".`**
Example : 240509 is for **`240509_HRF_FF.fits.`**

Knowing the order offset from step 3, you can start the reduction of your flatfield files.

Syntax :

```
python absoluteDRS.py {-i nindex, -f nindex -l nindex} [-r int] [-d]
[-e {FF TH OBJ}]
```

where nindex stands for the file index and int for the shift, as integers. Optional "-d" parameter will generate debug data when found and optional "-e" parameter will select only files with the given exposure type in the series. During the FF extraction procedure, the cross-order profile for the night is computed (see Illustration 1) and used to extract the orders.

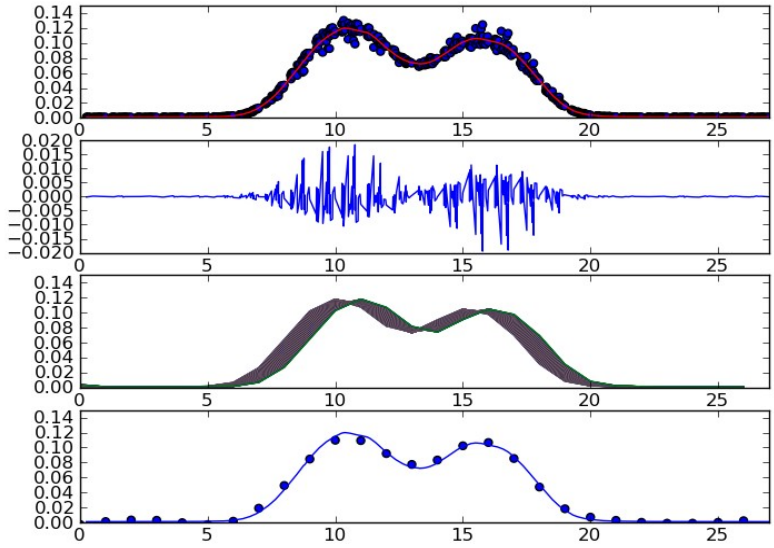


Illustration 1: From the upper to the lower panel: Reconstructed COP (dots) is compared to its smoothed version; Difference between the smoothed and reconstruced COP; Discretized COP; Comparison between the model COP (dots) and the night COP.

Examples:

To reduce a FF (flatfield) file, invoke the absoluteDRS script in that way:

```
python absoluteDRS.py -i 240509 -r 17 [-d]
```

Where: “240509” is the index of the flatfield file **240509_HRF_FF.fits**, and “17” the order offset found in STEP 2. If no offset was found, simply type:

```
python absoluteDRS.py -i 240509
```

The pipeline operations which are preformed are: cutting the prescan region, bias subtraction, ADU to photon units conversion, measurement and modeling of the order positions, measurement and modeling of the background, subtract the background, extract the spectrum and create a file with the order positions (modOrdPos).

You can also ask to work on an average of a series of flatfield files:

```
python absoluteDRS.py -f 240496 -l 240499 [-d] [-e FF]
```

in that case, “240496” is the index of the first FF file and “240499” the index of the last one. All files are supposed to be found in the *DataInput* directory.
Optional parameter “-d” allows the print out of debug data.
Optional parameter “-e” will allow to select only the “FF” files in the series.

STEP 4 : Extraction of the wavelength calibration frame

This step works on TH exposure type files found in the input folder.

The positions of the orders are stored in the *DataOutput* directory in a file whose name finishes with "**`*_FF_modOrdpos.fits`**" (passed in the "-o" parameter). With the index of this file you can begin to reduce your other frames, starting with your wavelength calibration frames.

Wavelength calibration frames are in the dataInput folder with names terminating with (for the HRF fibre): "**`_HRF_TH.fits`**" and alternatives for the different fibres.

Syntax :

```
python absoluteDRS.py {-i nindex, -f nindex -l nindex} -o nindex
                      [-b] [-d] [-e {FF TH OBJ}]
```

where nindex stands for the file index as integers.
Note that one can use the -i OR the couple -f and -l keywords, to indicate the calibration frame *_TH.fits.
-o refers to the index of the *_FF_modOrdPos file found in the output dir.
-b, when present, do skip compute the background (faster).
-d, when present, produces debug files.
-e assures to select only one kind of files when different ones are comprised between your first and last index.

Examples:

There is no real need to subtract the background for the wavelength calibration frame, so one can run the script in a faster way.

```
python absoluteDRS.py -i 240513 -o 240509 -b [-d]
```

where "-i" gives the index of a wavelength calibration file (e.g. 240513 for file 240513_HRF_TH.fits) which is supposed to be stored in the *DataInput* path, "-o" gives the index of the model order positions file which is assumed to be in the *DataOutput* path (e.g. 240509 for file 240509_HRF_FF_modOrdpos.fits), and the parameter "-b" is used to tell the script not to care for the background and extract without background subtraction.
Optional "-d" allows the print out of debug data.

To average files before extraction, one can use the group input syntax:

```
python absoluteDRS.py -f 240511 -l 240515 -o 240509 [-d] -e TH
```

Where "240511" is the index of the first Th-Ar-Ne spectrum and "240515" is the last of the series.
Option "-d" allows the print out of debug data.
Option "-e" assures you will only select "TH" files in the series.

STEP 5 : Extraction of an Object frame

This step works on OBJ exposure type files found in the input folder.

To extract the object spectra, the syntax is similar to the extraction of the wavelength calibration spectrum, except that background subtraction is here scientifically mandatory:

Syntax :

```
python absoluteDRS.py -i nindex -o nindex [-b] [-d] [-c [-fs -fc]]
```

Where

"-b" is optional. When found, the background should not be subtracted. Default is that the background IS extracted.

"-i nindex" is the index of the object file stored in the *DataInput* path with a name terminating with "**_OBJ.fits**".

"-o nindex" is the index of the file containing the order positions stored in the *DataOutput* path with a name terminating with "**_FF_modOrdpos.fits**".

"-d" is the usual debug switch.

"-c" removes cosmics when present (longer).
This option allows the use of :
"-fc" to force a critical value
"-fs" to force a safe value.
Those two parameters are optional.
When not given, default values comes from the instrumentModel.

Since, by default, the background is subtracted and we have to subtract it, the following syntax is most used:

```
python absoluteDRS.py -i nindex -o nindex [-d]
```

Example:

```
python absoluteDRS.py -i 240529 -o 240509 -c
```

Options for absoluteDRS.py

The absoluteDRS.py script is very versatile:
Here is the complete list of options.

[-b]
when present, skip the background computation (faster)

[-c]
remove cosmics. When present, those options are read :

[-fs ff.f]
fsafe : argument. Only used for cosmics removal.
ff.f is float. Default value comes from instrumentModel when not given.

[-fc ff.f]
fcrit : argument. Only used for cosmics removal.
ff.f is float. Default value comes from instrumentModel when not given.

[-d]
debug mode, creates lots of intermediary result files in the debugPath.

[-e]
exposure type, from ["OBJ", "TH", "FF"].
This will limit the file type to those you select

[-i index]
index of the input file

[-f index]
index of the first file, used with “-l”

[-l index]
index of the last file, used with “-f”. The two parameters give first and last index of a series. Files found in the “blacklist.xml” file in the “hermesRun” folder AND in the “pipeline/config” folder are removed from the list. This may lead to an empty list.

[-m int]
when present, number of maximum input files kept after removing the blacklisted ones. Default value is 10, which fits in 4GB ram. When not present, the software tries to use the largest available amount of memory and logs the limit.

[-o index]
model order positions file in DATAOUTPUT folder, *modOrdpos*

[-s int]
when present, specifies a step for the background computation (used when -b is not found).

[-r int]
when present, introduces an order offset. int stands for values between -50 and +50

[-ffall]
treat all “_FF.fits” files. Files found in the “blacklist.xml” file in the “hermesRun” folder are removed from the list. This may lead to an empty list.

[-thall]
treat all “_TH.fits” files. Files found in the “blacklist.xml” file in the “hermesRun” folder are removed from the list. This may lead to an empty list.

[-RF FIBER]
Accepted values are ['LRF', 'HRF', 'WRF'] . Used to select only images of one type. optional. When not specified, default HRF will be proceeded.

Help given by absoluteDRS when invoked without parameters :

USAGE :

```
python absoluteDRS.py {[-f 240511 -l 240515] [-i 240505] [-ffall] [-thall]} [-o
240509] [-b] [-s 1] [-r 17] [-d] [-e {FF TH OBJ}] [-m 10] [-RF {HRF LRF WRF}]

 -f: Number of the first file in the serie
         (relative to the DataInput path)
 -l: Number of the last file in the serie
         (relative to the DataInput path)
         this will treat all files between both indexes included,
         except blacklisted ones.
   OR
 -i: Number of a single file to proceed
         (relative to the DataInput path)
   OR
 -ffall  : treat all ff files (except blacklisted ones)
   OR
 -thall  : treat all th files (except blacklisted ones)

 -m: Number of maximum input files kept after removing the blacklisted ones.
              Default value is 10, which fits in 4GB ram.

 -o: Number of the model order positions file to use.
              (relative to the DataOutput path)
              Optional. A new one will be calculated when not given.

 -b: Optional. When found, skip the background measurement (faster).

 -s: Optional. Step in background calculations.
              Step is 1 by default, higher values will
              speed up background calculation. Debug purpose.

 -r: Optional. Specify an offset to the order model
              when instrument has a little shift.

 -d: Optional. When present, writes debug information in the DebugPath

 -e: exposure type. Accepted values are  ['OBJ', 'TH', 'FF']
              Used to select only images of one type.

 -c: Optional. When present, remove cosmics.
     -fs -fc : Optional.
              Safe and critical limits for cosmics removal.

 -RF: FIBER type. Accepted values are  ['LRF', 'HRF', 'WRF']
              Used to select only images of one type.
             optional. When not specified, default HRF will be proceeded.

------------------------------------------------------------
actually:
                  Nights : /home/louis/nights
       ConsoleLogSeverity : debug
             CurrentNight : /home/louis/nights/20110725
                DataInput : /home/louis/nights/20110725/raw
               DataOutput : /home/louis/nights/20110725/reduced
           ModelDetection : True
          AnalysesResults : /home/louis/hermesRun
                    Night : 20110725
                DebugPath : /home/louis/hermesDebug
                  Reduced : /home/louis/nights
```

STEP 6 : Determination of the wavelength dispersion

This step works on TH ext type files found in the output folder.

To determine the wavelength dispersion from one of your wavelength calibration spectra, you can use the dispersionDRS.py script in the following way:

Syntax :

```
python dispersionDRS.py [-i] nindex [-d]
```

Examples:

```
python dispersionDRS.py -i 240511

python dispersionDRS.py -i 240511 -d
```

Where “240511” is the index of the extracted wavelength calibration spectra. This file is expected to be found in the *DataOutput* path with the name “**240511_TH_ext.fits**”.
Optional “-d” parameter writes debug data in intermediary fits files.
This operation will lead to the following graph:

This script runs the “computeRadialVelocity” component as well, but without graphs.

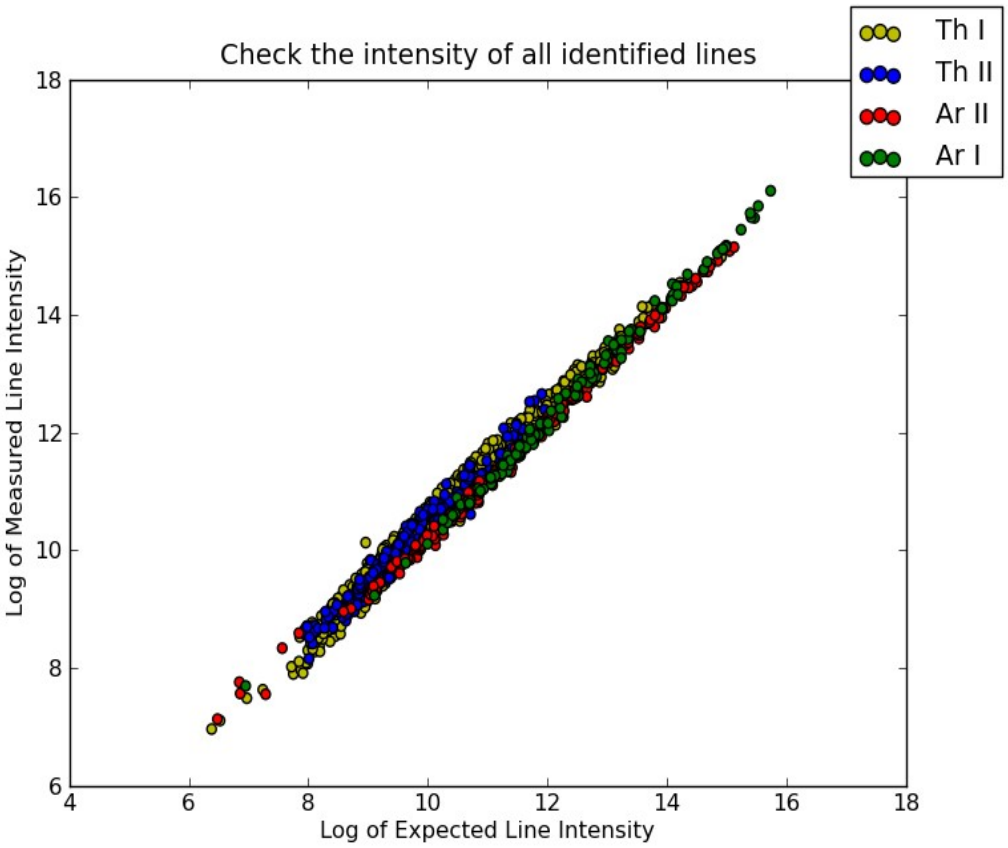


Illustration 2: Plot of the (roughly) estimated line surface (dex) as a function of the expected value.

Expected values (intensity and position) are coming from the instrumentmodel and were obtained by reducing a reference frame with the pipeline of the valid period.

The file produced by this script is:

`240511_HRF_TH_ext_wavelengthScale.fits`

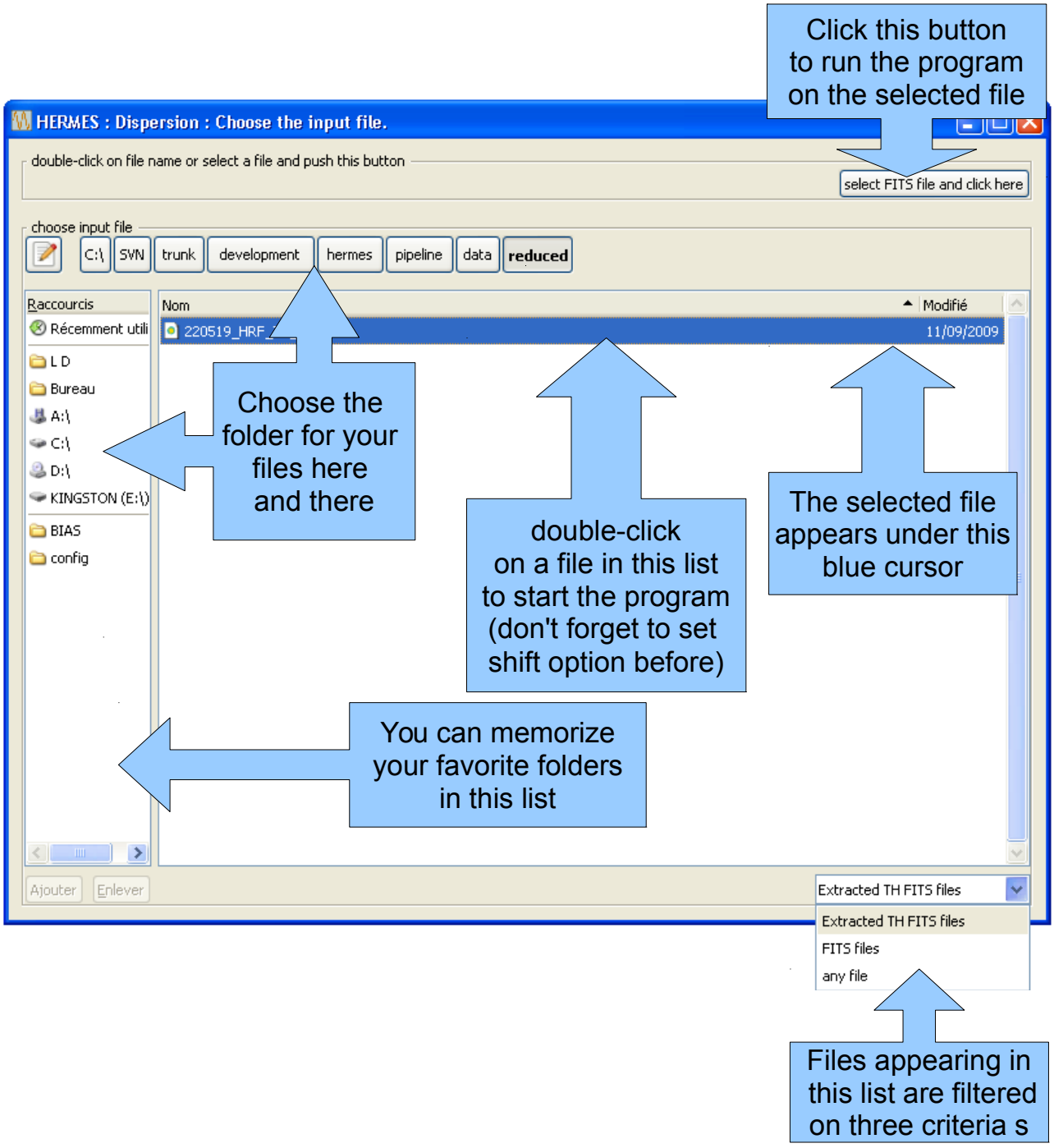
It contains the wavelength value in angstroms for every pixel and this for for every column and spectral order.

Determination of the wavelength dispersion in a GUI

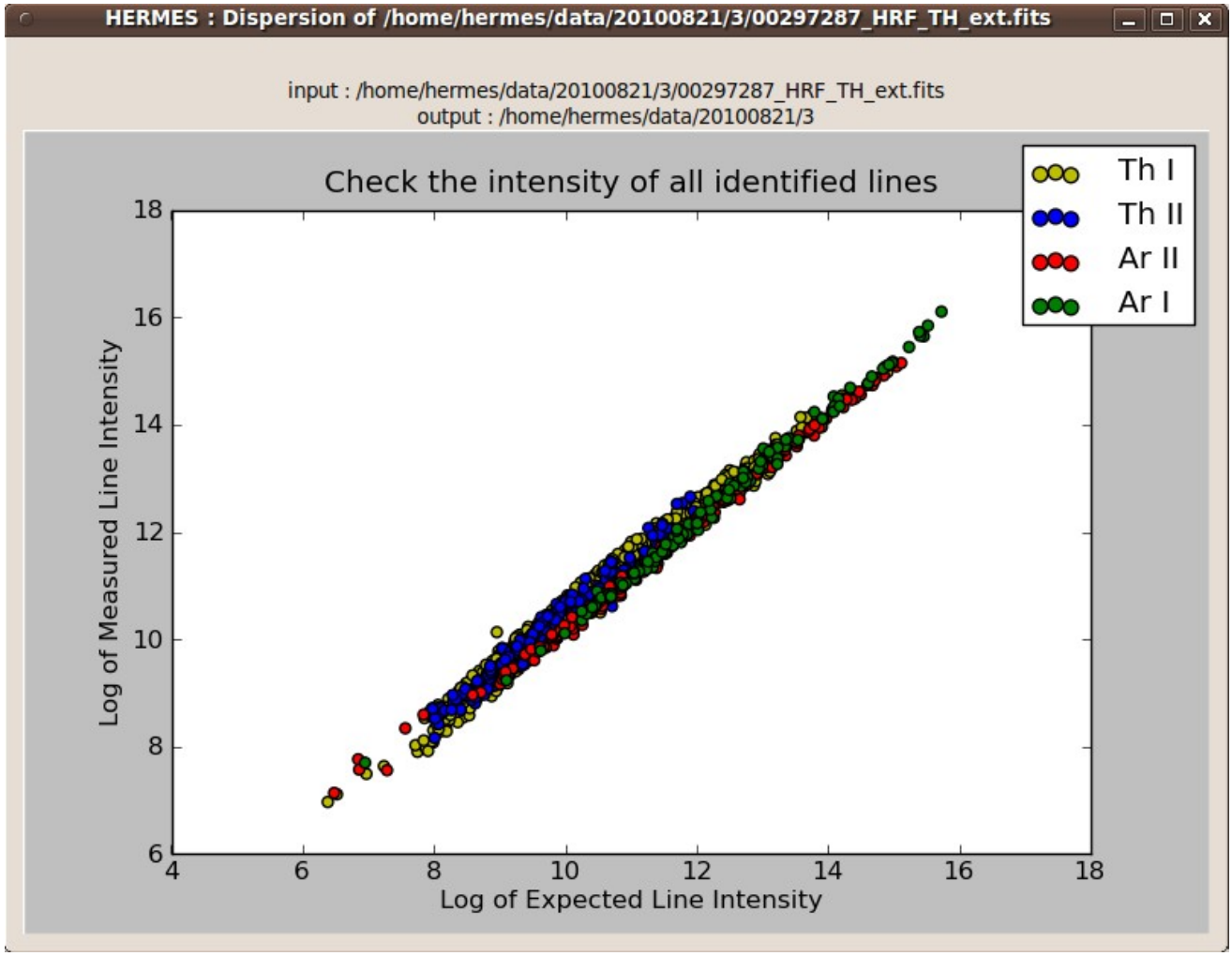
There is also an interactive way to run the software in a graphical user interface to allow the user to choose the input file interactively.

```
python  dispersionDRS.py
```

The program will try to start the GTK engine used by our GUI. If it does not succeed, you will be prompted to give the full command-line arguments.



When launched as a GUI (started without arguments), the graph is shown in the frame. You have to close the window to end the program.



When run from the command-line, without GUI, the graph is shown without a frame and is automatically closed when the program finishes.

In both cases, the graph is written in a file.

This script runs the “computeRadialVelocity” component as well with the default ThArNe mask of the release, but this without graphs.

STEP 7 : Apply the wavelength dispersion and merge the orders

This step works on ext and wavelengthScale type files found in the output folder.

Finally, you can use the applyDRS.py script to apply the wavelength dispersion to the spectra and produce the output spectrum. The script does 3 things:

1. It applies the wavelength dispersion to the extracted spectrum (with or without flatfielding with the normalise extracted flatfield);
2. It rebins the spectrum in 3 different ways:
   1. with a constant wavelength step
   2. with a constant logarithmic wavelength step
   3. with a wavelength step fixed order per order (the step is taken so that the number of points after and before rebinning are equal).
3. It merges the orders.
4. It prints the S/N ratio per pixel computed (before rebinning) at the middle of each order

Syntax :

```
python applyDRS.py -i nindex -w nindex [-ff nindex] [-c] [-t]
```

where:

-i is the index of the _OBJ.fits file found in the InputDir, on which you want to apply the dispersion.

-w is the index of the **_HRF_TH_ext_wavelengthScale.fits** file containing the wavelength dispersion and located in the output dir,

-ff is the index of the extracted flatfield file (if dividing the spectrum by the flatfield is required). Use this to correct the instrument response.

-c tells the program to search for extracted files where cosmics were removed.

-t This option generates the output of the merged full spectra in ascii as well. The extend is .txt by default.

The files for wavelength dispersion and flat-fielding are stored in the *DataOutput* path because they are produced by previous steps and the program will search them there.

Make sure the hermesConfig.xml points to the correct folder.

Examples:

```
python applyDRS.py -i 240529 -w 240511 -ff 240509
```

When no division by the flatfield is required:

```
python applyDRS.py -i 240529 -w 240511
```

The output is stored in the *DataOutput* path.

The names are build with the part of the input file that holds the index, the exposure and the fiber, followed with the order number and ".txt" extension when no clipped input was found or required, or "_c.txt" extension when use of the clipped file was asked and that file was found.

The main ouput files provided by the procedure are (e.g. for index file 00297246):

00297246_HRF_OBJ_ext_**log**_merged.fits
00297246_HRF_OBJ_ext_**wavelength**_merged.fits

When option -t is used, txt files are also produced :

00297246_HRF_OBJ_ext_**log**_merged.txt
00297246_HRF_OBJ_ext_**wavelength**_merged.txt

The codes **log** and **wavelength** are referring to the merged spectrum rebinned in log(λ) and linear λ scale. The ascii version is saved in the "*.txt" files, which contain the wavelength scale (first column) and the corresponding fluxes (second column). The fits version contains the fluxes while the wavelength scale or log(λ) scale is defined in the header by the following standard keywords:
CRVAL1: log(λ) or λ value of the first pixel.
CDELT1: corresponding step (in log(λ) or λ)
CTYPE1: rebinning mode (wavelength or logarithmic)

Take care: the merged 1D spectrum is corrected for the barycentric correction using the BVCOR fits keyword but only in the log-merged version. In the natural logarithmic wavelength scale, the barycentric correction is just an offset and hence it can be applied without the need of an extra rebinning. In the linear wavelengthscale merged versions the barycentric correction is NOT applied.

Help given by applyDRS when invoked without parameters :

```
>>>


USAGE  :

python applyDRS.py -i nnnnnn -w nnnnnn [-ff nnnnnn] [-p] [-c] [-t]
 -i  nnnnnn : index of the input file (found in the DataOutput path)
              This file is named "*_ext_CosmicsRemoved.fits" or "*_ext.fits"
 -w  nnnnnn : index of the wavelength scale file (found in the DataOutput path)
              This file is named "*wavelengthScale.fits"
 -ff nnnnnn : (optional) index of the flatfield (found in the DataOutput path)
              provides a flatfield to divide the spectrum.
 -p         : (optional) draws a series of plots.
 -c         : try to work on file where cosmics were removed.
              If not found, try to locate and use the regular file.
 -t         : To ask the printing out of the rebinnined orders in ascii format.

--------------------------------------------------------------------------------
actually:
                   Nights : C:\nights
              CurrentNight : C:\nights\20090926
                 DataInput : C:\nights\20090926\raw
                DataOutput : C:\nights\20090926\reduced
            ModelDetection : True
          AnalysesResults : C:\Documents and Settings\L D\hermesRun
                 DebugPath : C:\debug
                   Reduced : C:\nights\20090926\reduced

(use  hermes/setup.py  to change the paths.)

>>>
```

STEP 8 : Determine the radial velocity

This step works in order-pixel space so on the _OBJ_ext type files found in the output folder. The associated wavelengthcalibration (_wavelengthScale.fits) is found in the same outputfolder and by default the wavelengthScale file is used with the nearest sequence number.

This is a command-line utility that computes the radial velocity by cross correlation the extracted spectrum with a mask of individual lines. Both emission and absorption spectra can be used. The default mask is the one determined by a high resolution Arcturus spectrum, but the user can provide own spectral masks at will :

```
python hermesVR.py -i nnnnnn [-w nnnnnn] [-v ff.f] [-b ff.f] [-e] [-t] [-L or -LL] [-p] [-c][-m {n pathToMask}] [-f] [-ROT] [-o nn]
```

where :

-i is mandatory and is followed by the index of an object input file in the form "_ext.fits". It is globbing in all 'reduced' folders of the nights unless option “-n” is used. Note that in the 2 fibers mode, index should be typed NNNN_LRF (for processing the OBJ spectrum) or NNNN_WRF (for processing the TH spectrum) where NNNN is the file index.

-w optional : index of the wavelength-scale file found in the outputpath as “*_wavelengthsScale.fits”

-v optional : initial value for velocity, as a float.
When not provided, the program runs in two steps, scanning the velocity scale [-300, +300 km/s] with larger steps first.

-m optional: mask number as integer, or complete path to mask.
When value is 1, the Arcturus mask is used (constructed from the NOAO spectrum), else for any other integer, ThArNe.
Both files are taken from the path in the instrumentConfig.xml file; there is a “Vrmasks” folder in the config section made to retain the most useful ones.
The user can also give a complete pathToMask to point to any other fits file containing a mask. A mask file has i=1, N lines of two columns:
central wavelength of line_i , relative contrast of line_i (ex: 0.2 = line core has intensity (1 – 0.2) of the continuum value; this value is used to weight the contribution of line_i in the CCF)

Available masks:
HermesArcturusMask.fits (default)
HermesCMask.fits
HermesRMask.fits
HermesF0Mask.fits (main sequence)

HermesG2Mask.fits (main sequence)
HermesM4Mask.fits (main sequence)
HermesMIIIMask.fits
HermespAGBMask.fits
HermesThArNeMask.fits
HermesTelluricsMask.fits

-b optional. Barycentric velocity correction [Barycorr] as float [Vr = Vr_obs + Barycorr].

-e optional. Stands for CCF in emission, e.g. when correlating with a ThArNe object file (as opposed to absorption by default)

-t optional. CCF on Th-Ar : sets emission to True and mask to 2.

-L optional. Forces a larger velocity window (120 km/s, as opposed to default 60 km/s), useful for, e.g., Mira stars with wide CCFs (10-15 km/s)

-LL optional. Forces an even larger velocity window (300 km/s, as opposed to default 60 km/s), useful for CCFs of width 50 km/s

-c optional. Use _OBJ_ext_CosmicsRemoved.fits file as input instead of _OBJ_ext.fits The cosmic clipped version of the extracted science spectrum should always be used if possible.

-f optional. Do not use flat-fielded frame. Otherwise, look for _FF_ext.fits file in the DataInput directory to be used to flat-field the spectrum before computing the CCF.

-p optional. Plot the graphs on screen. The graphs are always written to disk in the *debugPath*.

-a n optional. Add next n files (the images are summed prior to the computation of the cross correlation). Take care: the barycentric correction is not applied prior to the sum so you will smear out the small difference in BVCOR between the frames.

-n Search for input file only in current night (set by the setup.py in the hermesconfig.xml file). When not present, all nights are searched.

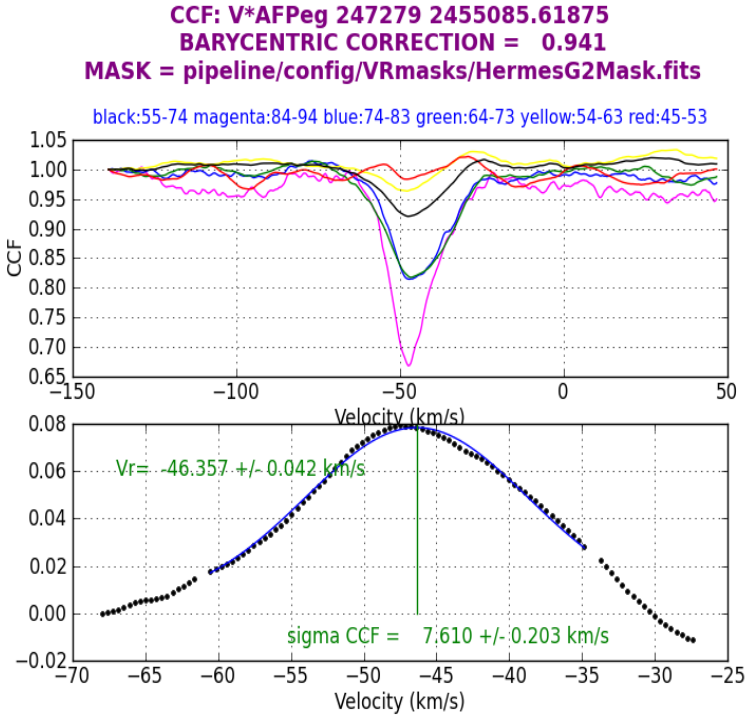
-ROT optional. Use a rotationally-broadened profile to fit the CCF. The option -f is recommended in conjunction with -ROT to avoid a slope in the CCF due to the spectrograph blaze function. A better precision will be obtained when -ROT and -f are used in conjunction.

-o nn optional. Use order nn as check of mask match (48 <= nn <= 94).
Default nn = 54 (Halpha).

Graphs are as follows:

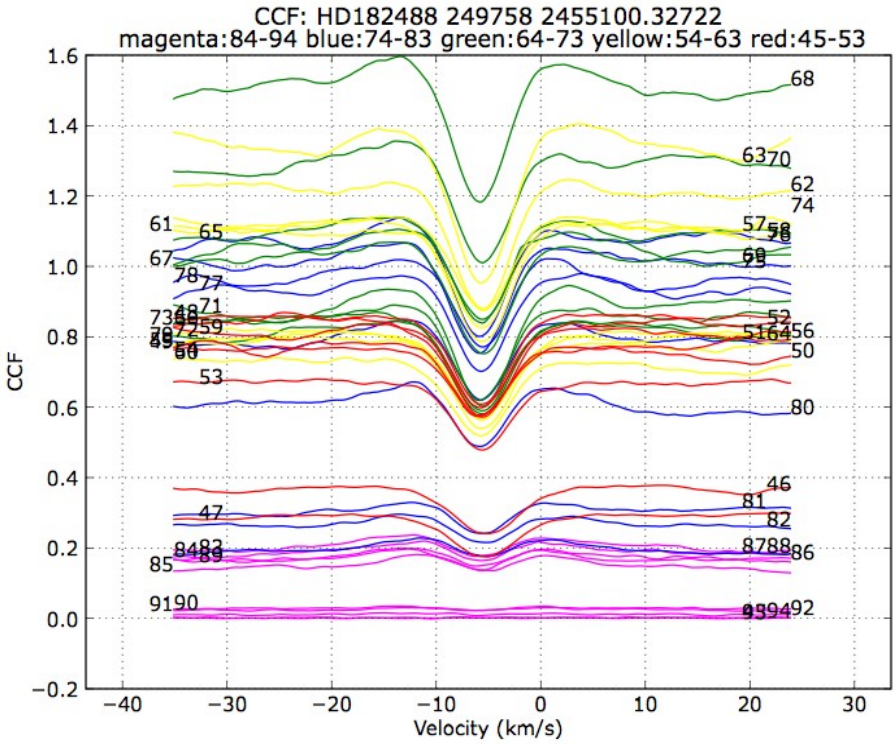
nnnnnn_VR.png:

The various curves display the CCFs computed from various order sets, as indicated in the header. The black line (orders 55-74) provides the best compromise between absence of tellurics and maximum signal in the case of G-K stars. The lower panel displays the Gaussian fitted on the **core** of the CCF, as represented by the blue line.



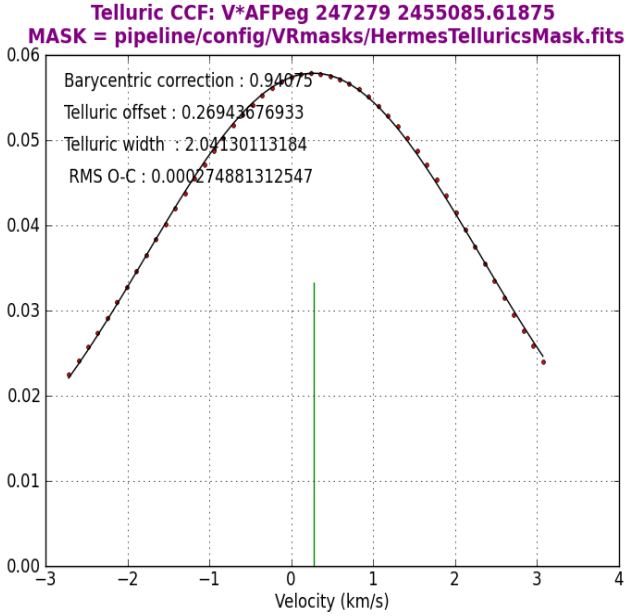
nnnnnn_AllCCF.png:

The CCFs for each order, as indicated by the label. Same color code as the previous figure. The ordinate axis is arbitrarily normalized to unity on the first point of order 69.



nnnnnn_VR_telluric.png:

The fit on the telluric lines with a telluric mask. This is useful in case there is an important drift due e.g. to atmospheric pressure variations between the target spectrum and the Th-Ar calibration spectrum, if taken too fart apart. The stellar velocity on the telluric reference frame is obtained by **subtracting** the telluric offset (with its sign!) from the stellar velocity (see also below).



nnnnn_AllCCF.data:

The CCFs for all orders in tabular form.

First comes a header:

Spectral file index : 247279
Wavelength file : /media/EE3029193028EA73/HERMES_DATA/20090910/reduced/247296_ HRF_TH_ext_wavelengthScale.fits
Flat field file : None
Mask file : pipeline/config/VRmasks/HermesG2Mask.fits
Star (header) : V*AFPeg
HJD : 2455085.6187454
Vr from orders 55-74 : -46.3572246877 +- 0.0421920595877 km/s
Vr ord.55-74 telloffset : -46.6266614571 +- 0.0423019547346 km/s
Vr Drift (2F frame) : 0.0 km/s
Telluric offset : 0.26943676933 +- 0.00304720890582 km/s
Telluric width : 1.62633629327 km/s
Barycentric correction : 0.94075 km/s

The **Vr Drift (2F frame)** field provides the shift in RV between the external Th-Ar frame (taken at the end/beginning of night) and the internal one, in case of a 2F frame (it is set to 0.0 km/s otherwise). The field **Vr from orders 55-74** already includes that correction, i.e., it is calibrated with the internal Th-Ar frame. The field **Telluric width** contains the width of the CCF fitted to the telluric lines.

Then comes the data blocks:

order	depth	Vr +/-	nLines	width +/- (gaus)	<S/N> (spec)	RMS O-C sqrt(chi2/dof)
		km/s		km/s		
54-74	0.07 +/- 0.00	-46.357 +/- 0.042	1134	7.61+/- 0.20	48.1	0.002
47-53 (red)	0.03 +/- 0.00	-96.412 +/- 0.143	116	4.18+/- 0.23	122.8	0.004
54-63	0.04 +/- 0.00	-48.453 +/- 0.070	490	6.87+/- 0.30	56.0	0.002
64-73	0.21 +/- 0.01	-44.642 +/- 0.062	644	10.58+/- 0.54	32.7	0.004
74-83	0.23 +/- 0.01	-44.781 +/- 0.081	1109	9.28+/- 0.47	15.9	0.007
84-94 (viol)	0.29 +/- 0.01	-46.561 +/- 0.094	1122	6.21+/- 0.29	9.4	0.016
order 93	0.98 +/- 0.23	17.669 +/- 0.246	34	0.91+/- 0.25	2.7	0.389
order 92	0.17 +/- 0.02	-45.118 +/- 0.666	128	5.12+/- 0.68	4.2	0.078
order 91	0.34 +/- 0.01	-45.488 +/- 0.305	140	7.88+/- 0.32	5.0	0.057
order 90	0.25 +/- 0.01	-49.143 +/- 0.402	122	9.99+/- 0.43	6.1	0.049
order 89	0.29 +/- 0.01	-46.517 +/- 0.237	144	6.66+/- 0.25	10.6	0.041
order 88	0.28 +/- 0.01	-46.593 +/- 0.187	147	5.34+/- 0.19	12.5	0.035
...						
order 48	0.06 +/- 0.01	-13.599 +/- 0.489	18	4.47+/- 0.50	216.1	0.021

The *Vr from orders 55-74:* provides the best estimate of the velocity for most spectral types.

Nevertheless, the table provides for comparison the velocity derived from all the individual orders, with the following fields:

depth :	the relative depth of the CCF core
Vr +/- :	the radial velocity and its error (based on the error on the central position of Gaussian fitted on the CCF, as given by the python routine *scipy.optimize.leastsq*)
nLines :	Number of lines used to compute the CCF
width +/- :	the sigma of the Gaussian fitted to the CCF
<S/N> :	the average S/N in the extracted spectrum, for the corresponding wavelength range
RMS O-C :	the root-mean-square of the O-C (i.e., CCF – fitted Gaussian). Equivalent to sqrt(chi2/dof), where chi2 is the chi square of the fit and dof the number of degrees of freedom (number of CCF points – number of parameters, here 4)

Then comes a block providing the 500 velocity abscissae of the CCF:

Vr(i): [-129.08296373 -128.96484219 -128.84672066 -128.72859912 -128.61047759 -128.49235605 -128.37423452 -128.25611298 -128.13799144 -128.01986991
…
-70.61280348 -70.49468195 -70.37656041 -70.25843887 -70.14031734]

And then the 55 non-normalized CCFs, each with 500 data points:

CCF order 49 lambda range: 7189.26698052-7341.58465166 Nlines=40
[21680.38830213 21677.20188657 21676.3916209 21673.74008459 ...

Contents of the file velocities.data:

Every run of hermesVR.py adds one line to this file, in the following format:

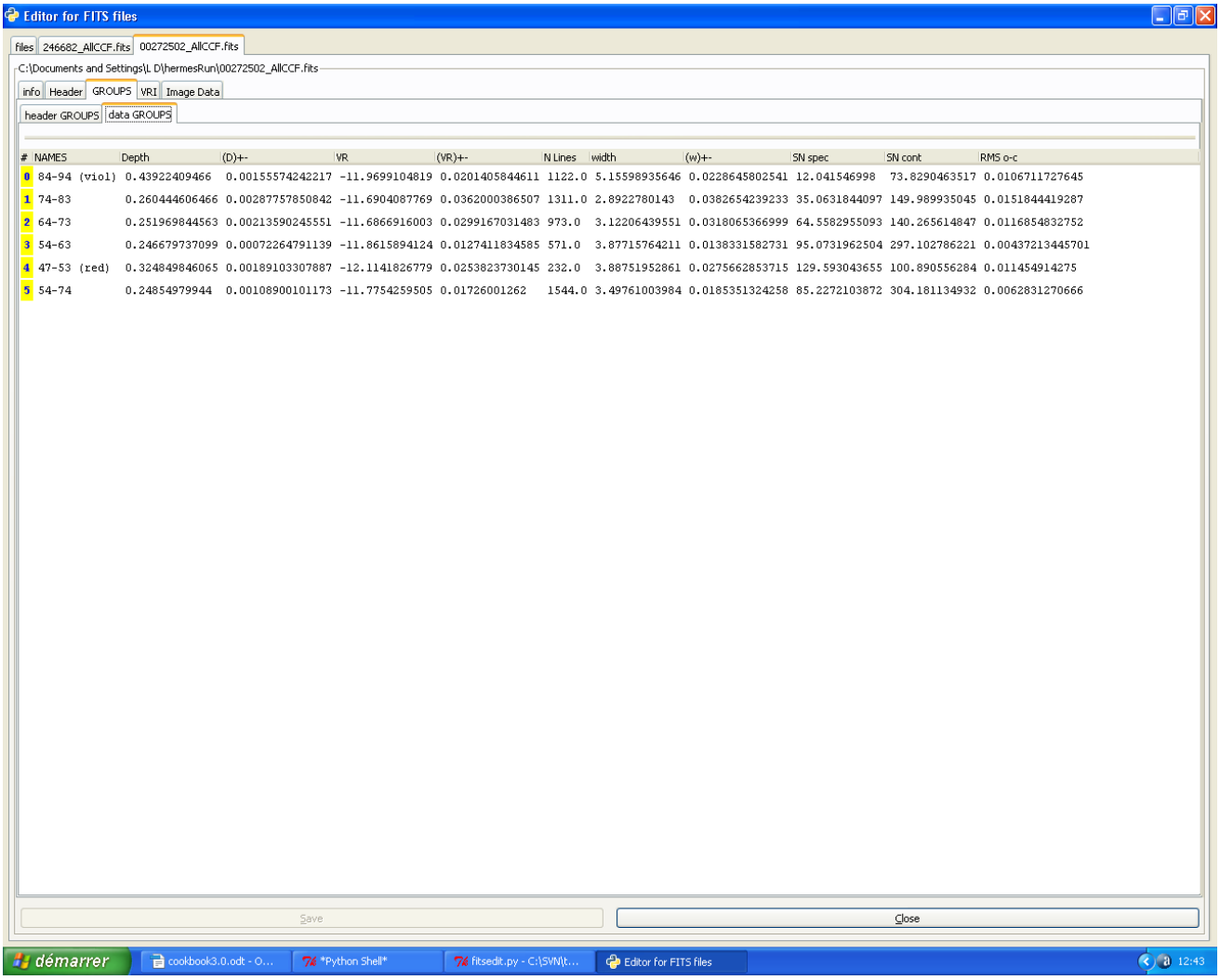
Index_file	Star	HJD	Texp (s)	BaryCorr (km/s)	RV_Drift (km/s)	Tell_off	+- Err	Tell_width (km/s)	Vr	Err(Vr)	Nlines spect.	Depth CCF	Width	S/N	O-C
247279	V*AFPeg	2455085.6187	520	0.941	0.00	0.269	0.005	1.66	-46.357	0.042	1134	0.07 0.00	7.61 0.2	48.1	0.002

The info is provided for the favoured set of orders only, i.e., 54-74.

Contents of **velocities.data** file:
Field 1: unique_number
Field 2: star_name
Field 3: HJD
Field 4: exp_time
Field 5: barycentric correction
Field 6: RV drift (2F frames only)
Field 7: telluric offset
Field 8: tell_offset_error
Field 9: telluric CCF width
Field 10: Vrad(55-74) (reduced to solar-system barycentre and corrected for Th-AR RV drift if 2F frame - not corrected for telluric line offset)
Field 11: err on Vrad
Field 12: n_lines_used
Field 13: depth_CCF
Field 14: error on depth_CCF
Field 15: sigma_CCF
Field 16: error on sigma_CCF
Field 17: signal_to_noise in spectrum (orders 55-74)
Field 18: wavelength_calibration_file
Field 19: mask_file
Field 20: FF_file

Contents of file nnnnnn_AllCCF.fits

All results are also found in a fits file. It contains the same data as the ascii file.



Help given by hermesVR when invoked without parameters :

When invoked without parameters, it lists the most important options and tells the folders and files in a syntaxic help :

```
USAGE  :

USAGE  :

python hermesVR.py -i 240529 [-p] [-w 240511] [-v 25.] [-b 30.] [-n] [-L or -LL]
[-e] [-c] [-f] [-g] [-WRF] [-m {1 2 pathToMask}] [-ROT] [-o 54]

 -i   : to provide the index of the input file
      : in the 2 fibers mode, index should be NNNN (for processing the OBJ spectrum
        in LRF mode) and switch -WRF may be added (for processing the TH spectrum)
        where NNNN is the file index .
[-w] : to provide the index of the wavelength scale file (found in the DataOutput
        path)
[-v] : estimate of the radial velocity
[-b] : barycentric correction
[-n] : when set, search input file only in the DataOutput path, else search in all
        nights.
[-L] : wider velocity window
[-e] : correlation on emission lines
[-t] : correlation on Th-Ar spectrum with Th-Ar mask
[-m] : mask type {1: Arcturus (default) 2: Th-Ar, or complete-path-To-Your-Mask}
[-p] : show plots
[-f] : do NOT use flat-fielded frame
[-a] : add some consecutive files ( -a 3 reads four files and adds them )
[-c] : when set, work on the clipped extracted spectra. Recommended
[-g] : give a file name, a path, or both, for the log file.
[-o nn] : plot order nn for checking purposes (48 <= nn <= 94). Defaulted to 54
             (Halpha)
[-WRF] : When processing two-fiber mode files, do the WRF (TH frame).
         When not present, the LRF will be done instead.
[-ROT] : use a rotationally-broadened profile to fit the CCF.
[-findmask] : try all masks in the VRmasks folder till a correlation is found
              (begin with Arcturus).


--------------------------------------------------------------------------
actually, when "-n" option is provided, files are only searched in :
/Users/ajorisse/Desktop/Alain/HERMES_DATA/reduced/20090926
(use  hermes/setup.py  to change the paths.)

where you have those "_OBJ_ext.fits" possible indexes for the -i :
249863 249864 249865 249866 249867 249868 249869 249870 249871 249872 249873
249874 249875
```

A list of radial-velocity standard stars to derive the mask zero-point:

From Udry S., Mayor M., Queloz D., 1999

Towards a New Set of High-Precision Radial-Velocity Standard Stars

In: Precise Stellar Radial Velocities, ASP Conference Series #185, IAU Colloquium 170. Eds. J. B. Hearnshaw and C. D. Scarfe. ISBN: 1-58381-011-0 (1999), p. 367

http://cdsads.u-strasbg.fr/abs/1999ASPC..185..367U

HD	Vr (km/s)	HD	Vr (km/s)	HD	Vr (km/s)	HD	Vr (km/s)
3765	-63.30	82106	29.75	140538	19.00	182572	-100.35
10780	2.70	82885	14.40	144579	-59.45	190007	-30.40
32923	20.50	90343	9.55	145742	-21.85	190404	-2.60
38230	-29.25	101177	-16.95	151541	9.40	193664	-4.50
42807	6.00	109358	6.25	154345	-46.95	196850	-21.05
50692	-15.05	115404	7.60	158633	-38.60	197076	-35.40
62613	-7.85	125184	-12.40	159222	-51.60	210667	-19.50
65583	14.80	128165	11.25	164922	20.15	221354	-25.20
73667	-12.10	131977	26.85	168009	-64.65		
79210	10.65	139323	-67.20	182488	-21.55		

An updated list should be posted on the IAU Commission 30 (Radial Velocities) web site, but is not yet available:

http://www.ctio.noao.edu/science/iauc30/iauc30.html

http://obswww.unige.ch/~udry/std/std.html

These objects are given in program number 1 and all observers should have measured radial velocity standards per night.

STEP 9 : Determine the drift in two fib mode

The two-fiber mode encountered for input files with suffix "_LRF_WRF_OBJ_TH" must be reduced in two steps, in this order :

1. absoluteDRS on the WRF files (use the -RF WRF switch) to produce the "_WRF_TH_2F_ext" files
2. abdoluteDRS on the LRF files to produce the "*_LRF_OBJ_2F_ext" and the "*_LRF_OBJ_2F_ext" files.

This produces files with a forthcoming suffix "2F" necessary for the next step to distinguish between true two-fiber mode files and regular one-fiber LRF ones, if any.

Goal : determine the drift between the current frame and the wavelength scale files shot at the beginning of the night.

Command Line :

```
python twoFibDrift.py -i nnnnnnnn [-w nnnnnnnn]
```

where -i is mandatory and is the number of the input file to be processed

and -w is optionnal and is the number of the wavelength scale file to be used

Help displayed :

```
USAGE  :

python twoFibDrift.py -i nnnnnnnn [-w nnnnnnnn]

 -i: Number of a single file to proceed
         (relative to the DataInput path)

 -w: Optional. When present, that particular index, if found, is used
               as wavelength scale. If not, the closest wavelengthscale
               for that fiber will be used.

------------------------------------------------------------
actually:
                   Nights : /home/louis/nights
       ConsoleLogSeverity : debug
             CurrentNight : /home/louis/nights/20110725
                DataInput : /home/louis/nights/20110725/raw
               DataOutput : /home/louis/nights/20110725/reduced
           ModelDetection : True
          AnalysesResults : /home/louis/hermesRun
                    Night : 20110725
                DebugPath : /home/louis/hermesDebug
                  Reduced : /home/louis/nights

(use  hermes/setup.py  to change the paths.)
```

Reducing a full night at once: nightDRS.py

* To easily reduce all OBJ files found in the *DataInput* path, use script named nightDRS.py which launches all the above scripts.

One can use it in the following way:

```
python nightDRS.py {-n YYYYMMDD [{-oa folder -oh folder}] -l } [-d] [-t] [-r i] [-g logfile] [-b blackList] [-m max n files] [ -c [-fc] [-fs]]
```

Where -n or -l parameter is mandatory. Parameters “-o?”, “-r”, “-t” and “-d” are optional.

-n must be followed by the name of the folder that contains the “raw” folder one wishes to reduce.
-l parameter will choose the last folder in alphabetical order in the “Nights” folder. When the convention YYYYMMDD is followed, this will make it possible to automatically reduce the last raw folder in a crontab job, for example (a job that runs daily at a fixed hour).

Parameters -oa and -oh are mutually exclusive. They stand for “output absolute” and “output hermes”. They allow the user to have his results in another folder than the default one.
Note that this script do not use the “reduced” path set in the hermesConfig.xml file. The “-oa” option uses the given folder for output and the “-oh” builds a path relative to the hermes path.

Option “-d” allows the print out of debug data.

Option “-t” will also produce ascii versions of the merged spectra.

Option “-r” allows to take into account an order offset related to a change of the instrument configuration (see option “-r” in the absoluteDRS script).

Option “-b” allows the use of an alternate blacklist (in the xml format).

Option “-g” allows the use of an alternate path to write the logFile to. Useful when no write rights are allowed to some folders.

Option “-m” let you manage the memory by limiting the number of stacked files in the Ffall or Thall options used for absoluteDRS. When not provided, the software tries to stack as much as possible frames and logs the amount it reaches.

Option “-c” permits to give specific critical and safe boundaries for the extractorder process.

Paths for nightDRS

When no **-o?** options are given, paths are constructed as follows :

- folder found in the “nights” tag in the config file.
- path correctly build for your operating system from there on to the night given in the -n parameter OR path correctly build for your operating system from there on to the last folder in alphabetical order in the “Nights” tag when “-l” switch is found on the command line.
- “raw” is added for the input folder
- “reduced” is added to name the output folder

example:

the “nights” tag in the config file says “/home/nights”
-n parameter is 20100912
input will be : /home/nights/20100912/raw
output will be : /home/nights/20100912/reduced
If the “-o?” switch is present, the output will be redirected to that folder.

Note: The nightDRS.py script checks if some files coming from the reduction process already exist. If they exist in the *DataOutput* path, they will not be computed again. The best way to start from scratch is therefore to empty your *DataOutput* directory, or to make the hermesConfig.xml file point to a new or empty output folder, or to give a new name in the “-oa” parameter. It will be created when missing.

The script will trace the orders on a merged file of (by default) 10 flatfield files. The wavelengthcalibration is performed on all concatenated calibration frames. If several of those series were obtained, the wavelenghtcalibration is performed several times. The object files are extracted by default using the cosmic-clipping module and the extracted and wavelengthcalibrated orders are extracted and written as ASCII files in the ORDERS directory.

Help given by nightDRS when invoked without parameters :

```
USAGE  :

python nightDRS.py {-n YYYYMMDD [{-oa folder -oh folder}] -l } [-d] [-t] [-r
i] [-g logfile] [-b blackList] [-m max n files] [ -c [-fc] [-fs]]

This script allows to proceed all files in one directory.

      input files are read from the "YYYYMMDD/raw" folder and reduced results
      are written in the "reduced" folder of the night : "YYYYMMDD/reduced".
      Path to the input folder containing the nights is read from the "input"
      tab (also called "nights").
      Path  to  the  output  folder  containing  the  nights  is  read  from  the
      "output"   tab. If the output night or reduced folder does not exists,
      they will be     created.
     Paths to debug, results and nights folders, are read from the respective
      tabs. the names "raw" and "reduced" cannot be changed.
     The "YYYYMMDD" parameter comes from one of the :

mandatory parameters :
 -n:   which night is to be automatically reduced, (usually year months day
YYYYMMDD)
      optional parameters :
     -oa: uses an absolute path for output.
     -oh: uses a relative path to the "hermesrun" folder for output.
          "-oa" and "-oh" creates all missing folders when needed.
          The "-o?" options override the usage of the "reduced" folder.
   OR
 -l:  treats the last folder in the "input" folder, easy for automatizing.

optional parameters :
 -r:  shift as integer
 -d:  produces debug intermediary files in debugPath.
 -g:  use path, filename or both as alternate log file.
 -b:  use blackList file name. When not present, habitual "blacklist.xml" is
      used (always located in the "nights" folder).
 -t:  also produces *.txt files of extracted results.
 -c:  Optional. When present, remove cosmics.
      -fs -fc : Optional.
             Safe and critical limits for cosmics removal.
 -m:  Number of maximum input files kept after removing the blacklisted ones.
             Default value is 10, which fits in 4GB ram.

------------------------------------------------------------
actually:
                  Nights : /home/louis/nights
       ConsoleLogSeverity : debug
             CurrentNight : /home/louis/nights/20110725
                DataInput : /home/louis/nights/20110725/raw
               DataOutput : /home/louis/nights/20110725/reduced
           ModelDetection : True
          AnalysesResults : /home/louis/hermesRun
                    Night : 20110725
                DebugPath : /home/louis/hermesDebug
                  Reduced : /home/louis/nights

use python ../../setup.py to change the paths.
```

Overview of Files treated by the Hermes software

Input and Output files by program

Five paths are defined in the “hermesConfig.xml” configuration file
The “reduced” path is the place where the programs are writing their results..
The “raw” stands for the path to raw images.
Some of our programs are working on results of other programs, and have to find their input in the output folder.
The “debug path” is the place where programs do write debug information when asked for.
The “AnalysesResults” point to the path where results of the radial velocity module will be placed.

program	files read for input	from folder	output files written in output or in results folders
CheckOrders.py	_HRF_FF.fits	raw	-
absoluteDRS.py	_HRF_FF.fits	raw	_HRF_FF_modOrdpos.fits
	_HRF_FF_modOrdpos.fits	reduced	
	_HRF_FF_COP.fits	reduced	Night COP.
	_HRF_FF_COP.png	reduced	See Illustration 1
	_HRF_FF_mslit.fits	reduced	Proposed mslit widths.
	_HRF_FF_templateordercenters.fits	reduced	Template for the order positions.
	_HRF_OBJ.fits	raw	
dispersionDRS.py	_TH_ext.fits	reduced	_HRF_TH_ext_wavelengthScale.fits
applyDRS.py	_HRF_TH_ext_wavelengthScale.fits	reduced	an ORDERS folder containing ascii “.txt” files for each order. When the “-text” option is used, writes the “OBJ_REBIN” fits file.
	_HRF_OBJ_ext_log_merged.fits	reduced	Log-scaled merged spectrum.
	_HRF_OBJ_ext_log_Rebin.fits	reduced	Log-scaled spectrum (not merged)
	_HRF_OBJ_ext_wavelength_merged.fits	reduced	Wavelength scaled order merged spectrum.
	_HRF_OBJ_ext_wavelength_Rebin.fits	reduced	Rebinned wavelength scaled spectrum (not merged).
	_HRF_OBJ_ext_log_merged.txt	reduced	Text version of the log scaled merged spectrum.
	_HRF_OBJ_ext_wavelength_merged.txt	reduced	Test version of the wavelength scaled merged spectrum.
hermesVR.py	_OBJ_ext.fits	reduced	- a line added to “velocities.data” in the “Analyses_Result” folder - The figures nnnnnn_AllCCF.png nnnnnn_VR.png: - The file nnnnnn_AllCCF.dta - The file nnnnnn_AllCCF.fits

Components used by scripts from the “run” folder

Most important scripts are using components. Here is a list of components used by the scripts of the “run” folder.

CheckOrders.py	This script is stand-alone and does not use any component. There is no debug option nor debug files.
absoluteDRS.py	averageImages.py computeBiasPrescan cutPrescanRegion subtractBiasCCD searchOrderPositions modelSearchedOrderPositions extractOrdersSimpleSumNoCosmics convertADUtoPhotonUnits measurebackground modelbackground subtractbackground
dispersionDRS.py	linePositions wlModel computeRadialVelocity
applyDRS.py	readFitsImage rebin merge
firstLook.py	checkimagesanity computebiasprescan cutprescanregion subtractbiasCCD convertADUtophotonunits estimateorderposition measurebackground modelbackground subtractbackground extractordersSimpleSumNoCosmics
hermesVR.py	readFitsImage computeRadialVelocity
twoFibDrift.py	readFitsImage computeRadialVelocity

Header keys written by Hermes programs

The reduction process adds some header keys in the fits files. As they become more numerous, here is a list of them, grouped by the program that wrote them :

ComputeRadialVelocity

```
    header [ "VRMASK" ]  = ( maskfile, program )
    header [ "RVDRIFT" ] = self.rvDrift
    header [ "VR_TELL" ] = Vrtellcorr
```

rebin

```
    header [ 'HIERARCH rebinMethod' ] = ( self.rebinMethod, "rebin" )
    header [ 'HIERARCH binSize%(0)03d' % {"0": order } ]      = ( binSize  [ order, 0 ],"rebin" )
    header [ 'HIERARCH coordinate%(0)03d' % {"0": order } ]   = ( startBins [ order, 0 ] + 0.5 *
binSize [ order, 0 ], "rebin" )
    header [ 'HIERARCH firstUsefulBin%(0)03d' % {"0": order}] = ( int ( firstUsefulBin [ order ] ),
"rebin" )
    header [ 'HIERARCH lastUsefulBin%(0)03d' % {"0": order }] = ( int ( lastUsefulBin [ order ] ),
"rebin" )
```

remove2dflatfield

```
    header [ twoDflatFielding ]         = instrumentModel [ header [ "observingMode" ] + "2Dff" ] )
```

convertADUtophotonunits

```
    header [ "IMAGUNIT" ] = ( "electrons", "convertADU2photonsUnits" )
    header [ "HIERARCH electronsPerADU" ] = ( m.gain, "convertADUtoPhotonUnits" )
```

cutprescanregion

```
    header [ "HIERARCH cutFirstRow" ] = ( FR,    name )
    header [ "HIERARCH cutFirstCol" ] = ( FC,    name )
```

extractorders

```
    header [ "HIERARCH BlemishedPixelsMasked" ] = ( self.BlemishedPixelsMasked,         name )
    header [ "HIERARCH compareExtractionMethods" ] = ( self.compareExtractionMethods,      name )
    header ["HIERARCH checkValidityCrossorderProfile"] = self.checkValidityCrossorderProfile,name)
    header [ "HIERARCH areVirtualEdgeOrdersAdded" ]  = ( self.areVirtualEdgeOrdersAdded,  name )
    header [ "MORDPOS" ] = ( self.results [ "MORDPOS" ] , msg )
    header [ "NUMPY" ]    = ( numpyVersion, "version used for this extraction" )
    header [ "PYFITS" ]   = ( pyfits.__version__, "version used for this extraction" )
    header["DATEEXTR"]=(datetime.now().strftime('%Y-%m-%d %H:%M:%S'), "date & time of extraction")
    header [ "COSMICS" ] = ( numberOfRemovedCosmics, name )
    header [ "FSAFE" ]   = ( self.fsafe, name )
    header [ "FCRIT" ]   = ( self.fcrit, name )

    if isTwoFibTH:
        ## YF 20120413 # So that computeradialvelocity knows which of the 2 fib we use
        if "EXPTYPE" in header:
            header [ "EXPTYPE" ] = ( "TH", "Exposure type", name )
        if "BVCOR" in header:
            header [ "BVCOR" ]   = ( 0., "BVCOR reset to zero by extractOrders", name )
        elif "VHELIO" in header:
            header [ "VHELIO" ]  = ( 0., "VHELIO reset to zero by extractOrders", name )

        if "EXPTYPE" in header:
            header [ "EXPTYPE" ] = ( "OBJ", "Exposure type", name )
```

linepositions

```
    tbhdu.header [ "TTYPE" ]  = ( "MODEL", "Table type", "linepositions" )
    tbhdu.header [ "ASHIFT" ] = ( averageShift, "Average Shift (pixels)", "linepositions" )
    tbhdu.header [ "DISP" ]   = ( dispersion, "Shift dispersion (pixels)", "linepositions" )
```

MeasureBackground

```
    header [ "HIERARCH allowedDistance" ]    = ( self.allowedDistance,                   name )
    header [ "HIERARCH halfLengthMedian" ]   = ( self.halfLengthMedian,                  name )
    header [ "HIERARCH useOrderPositionsFromTemplate" ] = ( self.useOrderPositionsFromTemplate,
name )
```

merge

```
    header[HR+'startOverlap%(0)03d%(1)03d'%{"0":pOrder, "1":kOrder}] = (startOverlap, "merge")
    header[HR+'endOverlap%(0)03d%(1)03d'%{"0":pOrder,"1":kOrder}] = (endBinOverlap, "merge")
    header [ 'CRPIX1' ] = ( 1. , "merge" )
    header [ 'CRVAL1' ] = ( mergedWavelengths [ 0 ] , "merge" )
    header [ 'CDELT1' ] = ( step [ 0 ] , "merge" )
    if ( mergedWavelengths [ 0 ] < 10. ):
        header [ 'CTYPE1' ] = ( "log(wavelength)"  , "merge")
    else:
        header [ 'CTYPE1' ] = ( "WAVELENGTH",  , "merge")
```

modelbackground

```
    header [ "HIERARCH halfLengthFilterCrossOrder",=(self.halfLengthFilterCrossOrder, name )
    header [ "HIERARCH halfLengthFilterAlongOrder",    = ( self.longOrder, name )
    header["HIERARCH halfLengthLocalBackgroundFeature"]=(self.halfLengthLocalBackgroundFeature,
name)
    header["HIERARCH degreeBackPolAlongOrder",         = ( self.gOrder, name )
```

modelorderpositions

```
    header [ "HIERARCH areVirtualEdgeOrdersAdded" ]      = ( self.areVirtualEdgeOrdersAdded, name )
    header [ "HIERARCH PolynomeDegreeInOrderDirection" ] = ( self.polyorder,                 name )
    header [ "HIERARCH PolynomeDegreeInRowDirection" ]   = ( self.polyrow,                   name )
    header [ "HIERARCH tolerance" ]                      = ( self.tolerance,                 name )
    header [ "HIERARCH isDifferential" ]                 = ( self.isDifferential,            name )
    header [ "HIERARCH areRelativeCoordinates" ]         = ( self.areRelativeCoordinates,    name )
    header [ "HIERARCH maxAllowedIterations" ]           = ( self.maxAllowedIterations,      name )
```

modelsearchedorderpositions

```
    name = "msop"
    self.header [ "HIERARCH areVirtualEdgeOrdersAdded" ] = ( self.areVirtualEdgeOrdersAdded, name )
    if ( self.isLevelToSet ):
        self.header = self.__tresholdsInHeader ( self.header )
    else:
        self.header["HIERARCH ccfTresholdForMaskConstruction"]=(self.level,name )
    self.header [ "HIERARCH areRelativeCoordinates" ]   = ( self.areRelativeCoordinates,    name )
    self.header [ "HIERARCH PolynomeDegreeCrossOrder" ] = ( self.polyorder,                 name )
```

```
    self.header [ "HIERARCH PolynomeDegreeCrossOrder" ] = ( self.polyrow,                  name )
    self.header [ "HIERARCH makeTemplateOrdpos" ]       = ( self.makeTemplateOrdpos,       name )
    self.header [ "MODSHIFT" ]                          = ( self.MODSHIFT, name)
    self.header [ "COPUSED" ] = ( str ( template ), str ( inputFile ), name )
    self.header [ "HIERARCH percentageOfBadRows" ] = ( percentage, name )
```

qualityoverlap

```
    name = "qualityOverlap"
    header[HQ+'offsetIndicatorAverageShift' ] = ( bestAverage - minimumShift, name )
    header[HQ+'offsetIndicatorAverage' ]      = ( offsetIndicatorAverage [ bestAverage ], name )
    header[HQ+'offsetIndicatorAverageNoShift']= ( offsetIndicatorAverage [-minimumshift ], name )
    header[HQ+'offsetIndicatorQualityShift' ] = ( bestQuality - minimumShift, name )
    header[HQ+'offsetIndicatorQuality' ]      = ( offsetIndicatorQuality [ bestQuality ], name )
    header[HQ+'offsetIndicatorQualityNoShift']=(offsetIndicatorQuality [ - minimumshift ], name )
```

readfitsimage

```
    header [ "model" ] = ( instrumentDateToUse, "readfits" )
    header [ fname ]  = ( Fparts + name )
```

searchorderpositions

```
    header [ "HIERARCH positionDev" ]  = ( self.positionDev,                    name )
    header [ "HIERARCH rowStep" ]      = ( self.rowStep,                        name )
    ##    next one generates a pyfits message "card is too long, comment is truncated."
    header["HIERARCH shiftApproximateOrderPositions"]=(self.shiftApproximateOrderPositions, name)
    header [ "HIERARCH numberOfBadRows" ] = ( nbOfBadRows, name )
```

subtractbiasCCD

```
    name = "subtractBiasCCD"
    header [ "HIERARCH subtractBias" ]  = ( self.subtractBias,                    name )
    header [ "HIERARCH readoutNoise" ]  = ( self.readoutNoise,                    name )
    header [ "HIERARCH biasToleranceFromDefault" ] = ( self.biasToleranceFromDefault,  name )
    header [ "HIERARCH readoutNoiseToleranceFromDefault"]=(self.readoutNoiseToleranceFromDefault,
name )
    header [ "HIERARCH subtractedBiasCCD" ]   = ( self.results["subtractedBiasCCD"],      name )
    header [ "HIERARCH usedReadoutNoiseCCD" ] = ( self.results["usedReadoutNoiseCCD"],    name )
```

wlmodel

```
    tbhdu.header [ "TTYPE" ] = ( "INTERMEDIATE", "Table type", "wlmodel" )
    header [ "HIERARCH WLMODEL ODEGREE" ]   = ( self.polyorderabs, "wlmodel_WL_calibration" )
    header [ "HIERARCH WLMODEL RDEGREE" ]   = ( self.polyrowabs,   "wlmodel_WL_calibration" )
    header [ "HIERARCH WLMODEL REF ROW" ]   = ( self.middleRow,    "Reference row" )
    header [ "HIERARCH WLMODEL REF ORDER" ] = ( self.middleOrder,  "Reference order" )
    header [ "HIERARCH WLMODEL COEFFS " + str ( k ) + " " + str ( l ) ] = ( coefs [ k , l ] ,
"wlmodel_WL_calibration" )
```

applyDRS

```
    self.header [ keySNR ] = ( signalToNoise [ SNR - firstOrder,  instrumentModel.middleRow ],
"APPLYDRS", after = "FCRIT" )
    hdu.header [ "FILENAME" ] = ( inputFile, "applyDRS" )
```

computebiasprescan

```
    header [ "HIERARCH prescanBiasADU" ]         = ( prescanBiasADU,           name )
    header [ "HIERARCH prescanReadoutNoiseADU" ] = ( prescanReadoutNoiseADU,   name )
    header [ "HIERARCH prescanFilDim" ]          = ( self.prescanFilDim,       name )
    header [ "HIERARCH prescanRejectFactor" ]    = ( self.prescanRejectFactor, name )
    header [ "HIERARCH constantBias" ]           = ( self.constantBias,        name )
    header [ "HIERARCH numberMaskedPixels" ]     = ( numberMaskedPixels,       name )
```

Debug Files produced by component :

Most of the programs can write files with intermediate data to help tracking unexpected or unexplained behaviours. Those files are written by the underlying components. Some of the components are used by several programs.
When invoked with the “-d” option, programs are asking the components to write those files.
The debug files are all found in the “DebugPath” pointed to by the hermesConfig file. See section “2) Set the input and output paths” to change this.

Format is : “**nnnnnnnn_DXXX_explain.fits**”

File names are all beginning with the input file index, formatted as “nnnnnnnn” followed by “_DXXX_” where D is for Debug and XXX stands for a unique number representing the position in the sequence, and then some distinctive part described here. This system permits to keep the files in the pipeline consecutive following order.

Debug files are not necessarily written when debug option “-d” is set, there may also be other conditions, also described here.

component averageImages.py

(only when more than one image is read)

for clipped average computation :

- _D001_medianOverStack.fits : contains the median of the stack.
- _D002_varImageStack.fits : contains the error image
- _D003_meanOverStack.fits : is the average of the stack
- _D004_toleranceMean.fits : is a new average computed only with pixels in limit of tolerances.

always :

- _D005_meanImage.fits : average of image stack
- _D006_meanImageVariance.fits :variance of Image stack / number Unmasked Pixels

component computeRadialVelocity.fits

- _D010_vri.fits : computed vri
- _D011_CCF.fits : computed CCF

component computeBiasPrescan.fits

- _D020_prescan.fits : part of the image on the prescan region
- _D021_prescanUse.fits : part of the image on the prescan region without filter size on the borders
- _D022_prescanMedianUse.fits : median of previous

component convertADUtoPhotonUnits.py

when “imageUnit” in header is NOT "electrons":

- _D030_image.fits : modified image
- _D031_imageVariance.fits : error of modified image

component estimateOrderPositions.py

- _D040_estimateAbsoluteOrderPositions.fits : frame of absolute positions

component extractOrders.py

Names are self-explanatory.

- _D050_extractedImageSimpleSumWithCosmics.fits
- _D051_extractedImageSimpleSumWithoutCosmics.fits
- _D052_extractedImageWeightedExtrWithCosmics.fits
- _D053_extractedImageWeightedExtrWithoutCosmics.fits
- _D054_cosmicsMask.fits
- _D055_overflowMask.fits
- _D056_numberOfBlemishedPixelsPerRow.fits
- _D060_extOverflowMaskNoCosmics.fits
- _D061_extNumberOfBlemishedPixelsPerRowNoCosmics.fits

component linepositions :

when isDifferential is True :

- _D070_tblReferenceLines

Always :

- _D071_WCFDifTable

component measureBackground

•_D080_bkg_' + halfLengthMedian + "_" + allowedDistance

component measureOrderPositions

•_D090_correlationInMeasureOrderPositions.fits

component modelBackground

•_D100_splinedModelAndResiduals.fits : splined model and residuals

component modelOrderPositions

•_D110_mAbsOPos.fits : model Absolute Order Positions
•_D111_trueMAbsOPosWoMask.fits : unmasked true Measured Absolute Positions
•_D112_trueMAbsOPos.fits : true Measured Absolute Positions

component modelSearchedOrderPositions.py

•_D120_masktomodelsearchedorderpositions.fits : consideredMask
•_D121_ominuscsofsearchedpositions.fits : oMinusCs of SearchOrderPositions
•_D122_coefficientsmodelsearchedorderpositions.fits : modelOrderPositionCoefficients

component remove2Dflatfield.py

•_D130_image.fits : image
•_D131_imageVariance.fits : and its variance

component searchorderpositions.fits

•_D140_foundorderpositions.fits :order Positions
•_D141_maximaoftheccf.fits : crossCorr Maxima

component subtractBiasCCD.py

•_D150_subtractbiasCCD” : subtracted image

component wlmodel

•_D160_+polyorderabs+X+polyrowabs+WCFrameFromWLMODEL.fits
extImageWCF
•_D161_+polyorderabs+X+polyrowabs+OMCS4CHECK.fits
selectedLampLinesDescription

component subtractbackground

This component always produces two usefull files in the debug folder

•_t30backgroundsubtracted
•_t31backgroundsubtractedvariance

component computeRadialVelocities

This component writes several graphes in the debugPath.

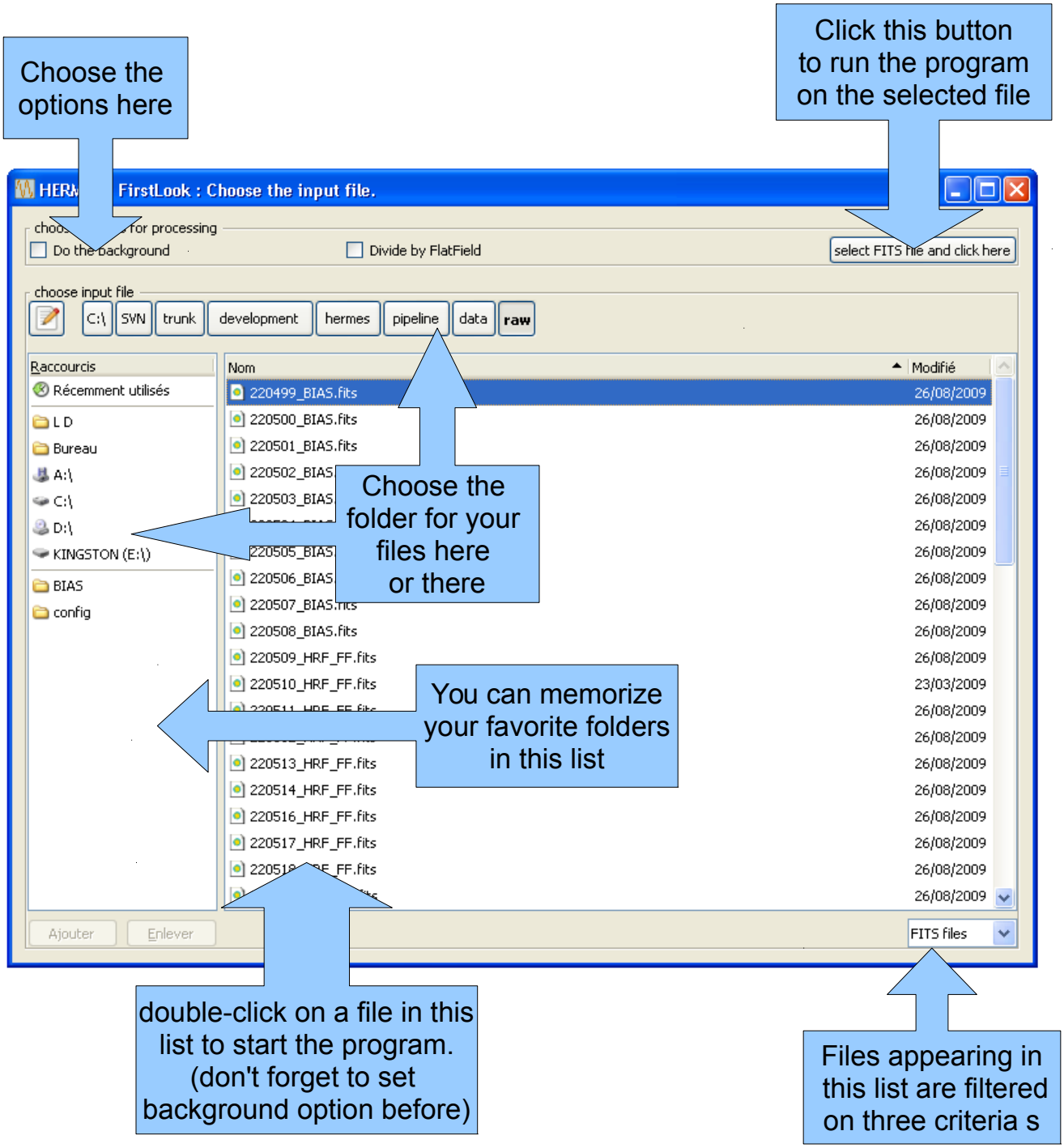
Wavelength for Hermes Orders table

This table shows a sample for wavelengths found in mid orders :

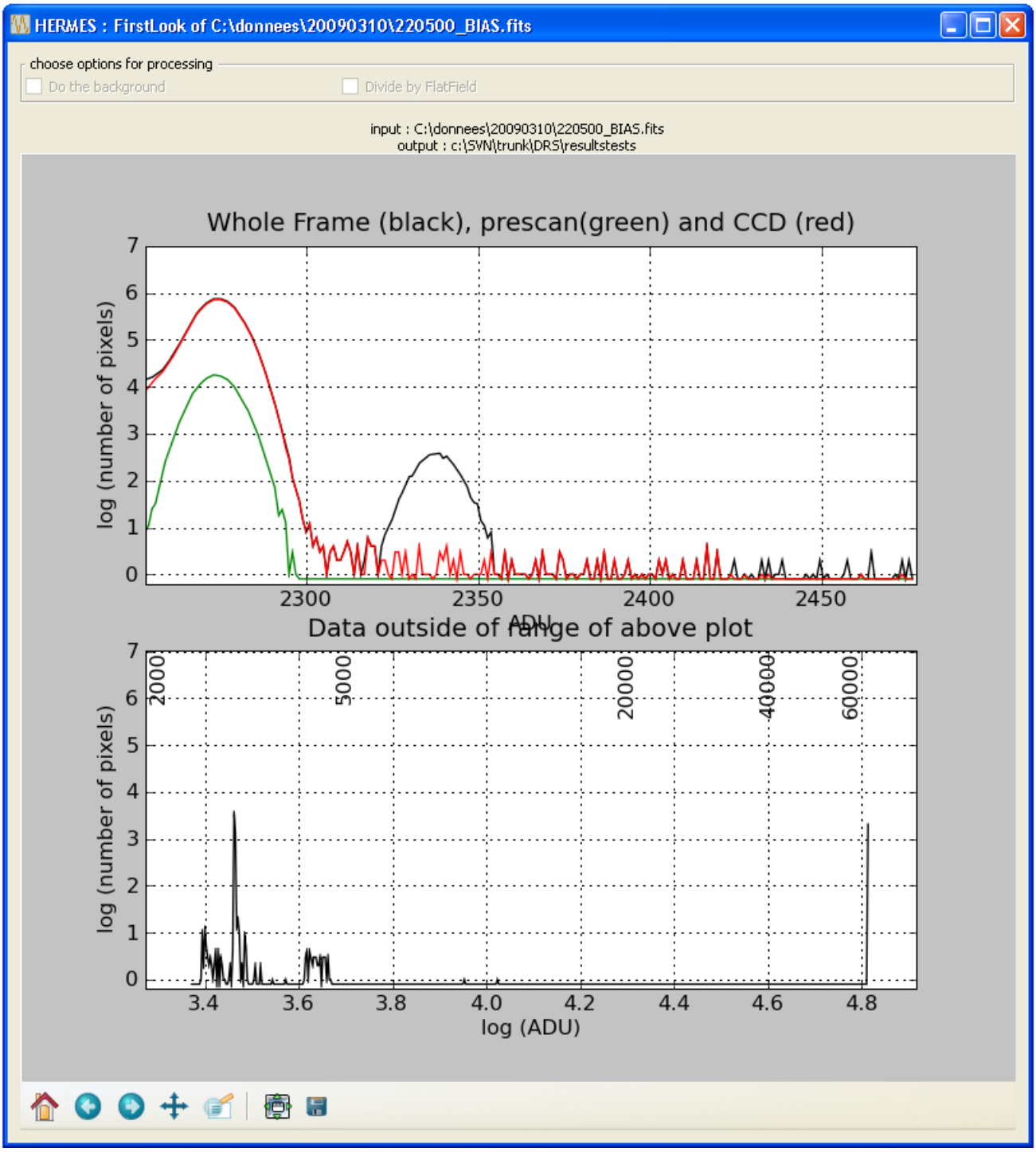
ORDER	WAVELENGTH	ORDER	WAVELENGTH
40	8910.387	69	5165.310
41	8693.061	70	5091.513
42	8486.078	71	5019.794
43	8288.720	72	4950.068
44	8100.333	73	4882.252
45	7920.320	74	4816.269
46	7748.134	75	4752.046
47	7583.276	76	4689.513
48	7425.286	77	4628.604
49	7273.745	78	4569.257
50	7128.265	79	4511.412
51	6988.489	80	4455.013
52	6854.089	81	4400.006
53	6724.760	82	4346.341
54	6600.221	83	4293.968
55	6480.210	84	4242.842
56	6364.485	85	4192.918
57	6252.821	86	4144.156
58	6145.008	87	4096.514
59	6040.849	88	4049.956
60	5940.163	89	4004.445
61	5842.778	90	3959.945
62	5748.534	91	3916.424
63	5657.282	92	3873.848
64	5568.881	93	3832.185
65	5483.201	94	3791.403
66	5400.116		
67	5319.511		
68	5241.277		

A First Look utility.

There is a “firstLook.py” program in the same directory, that you can use to have a first idea of the quality of the data.



After calculating, the program shows up a graph that looks like this :



The firstLook utility can also be called from the command line :

```
python firstLook.py -i nindex [-b]
```

The nindex is mandatory and refers to an index of an input file found in the input folder.
The -b switch is optional. When found, the program removes the background (longer).

Manually check the order alignment or offset

This step works on FF files found in the input folder.

Check whether the reference order positions referenced in the instrumentConfig.xml file are correctly aligned with those found in the flat fields of the night you want to reduce. To do this, run the checkOrders.py script:

syntax :
```
python checkOrders.py -i nindex [-r n]
```

where nindex is the index of the input FF file

examples :
```
python checkOrders.py -i 240552
```
240552 refers to file 240552_HRF_FF.fits

A plot should appear soon (see Erreur : source de la référence non trouvée). By zooming in the figure, you can check whether the reference order positions are well aligned with the exposures obtained during the night. If this is not the case, you can rerun the script providing an offset or shift value, to search for the best one:

```
python checkOrders.py -i 240552 -r 17
```

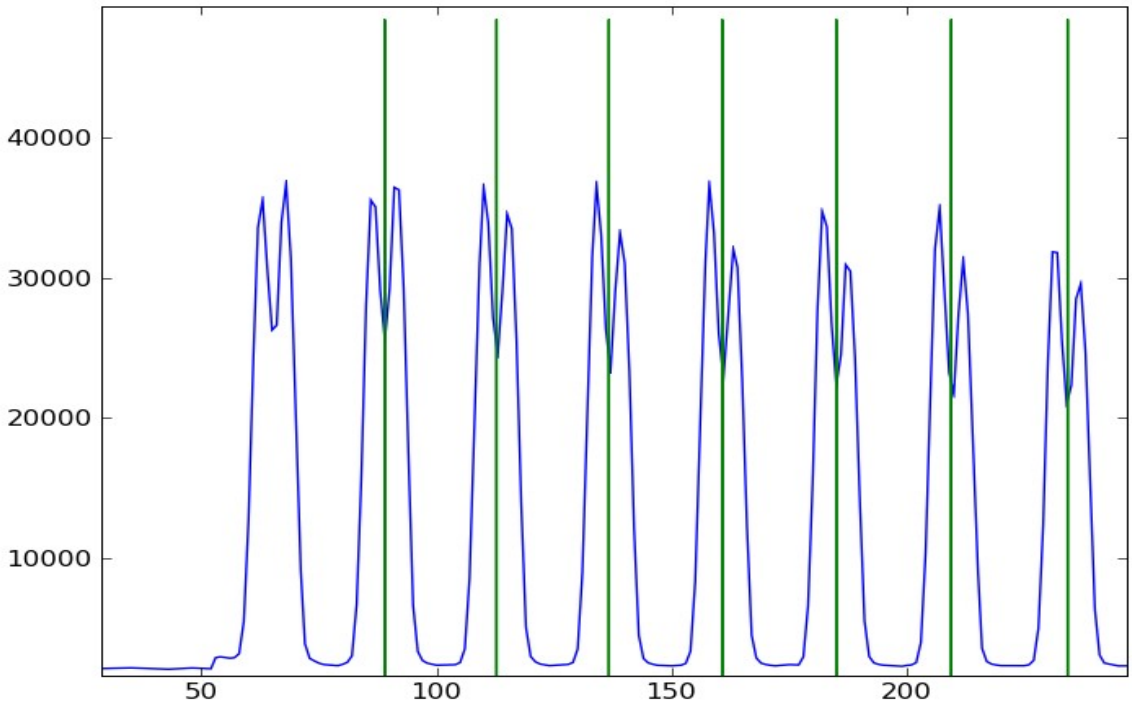


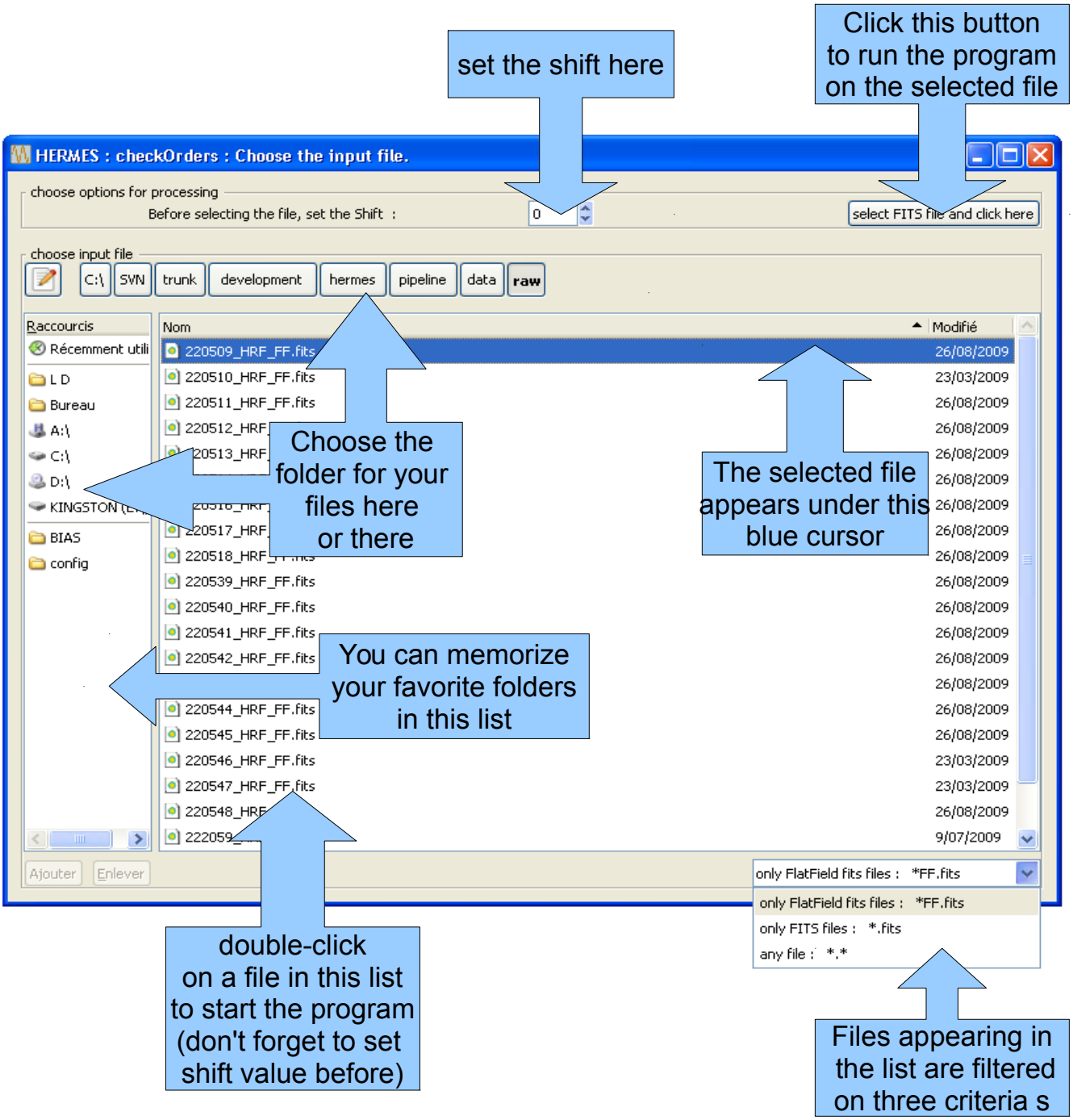
Illustration 3: "python checkOrders.py 240552 -r 17" . Orders are now correctly aligned, meaning that the order offset is about 17 pixels.

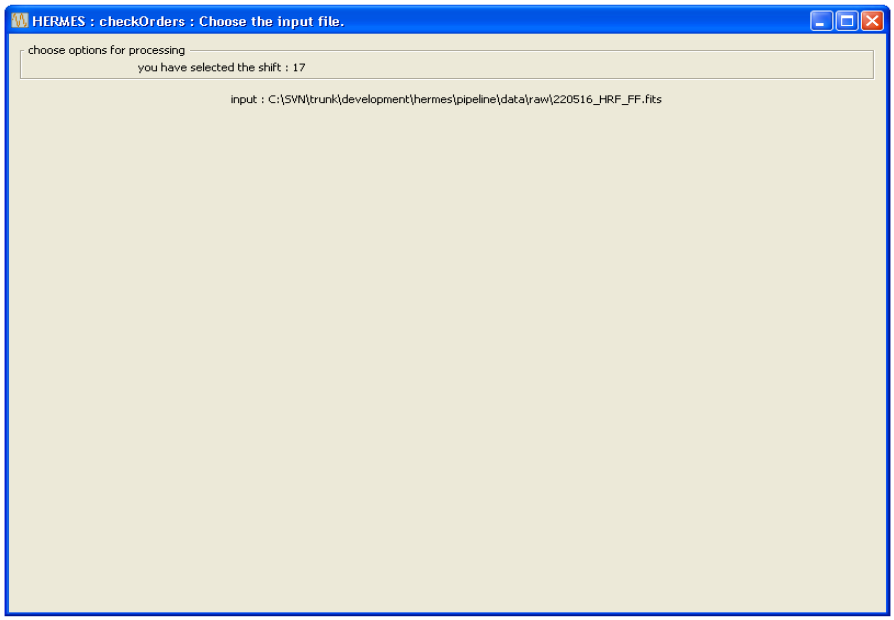
CheckOrders in a GUI

There is also an interactive way to run the software in a graphical user interface where the user can easily choose the input file and shift.

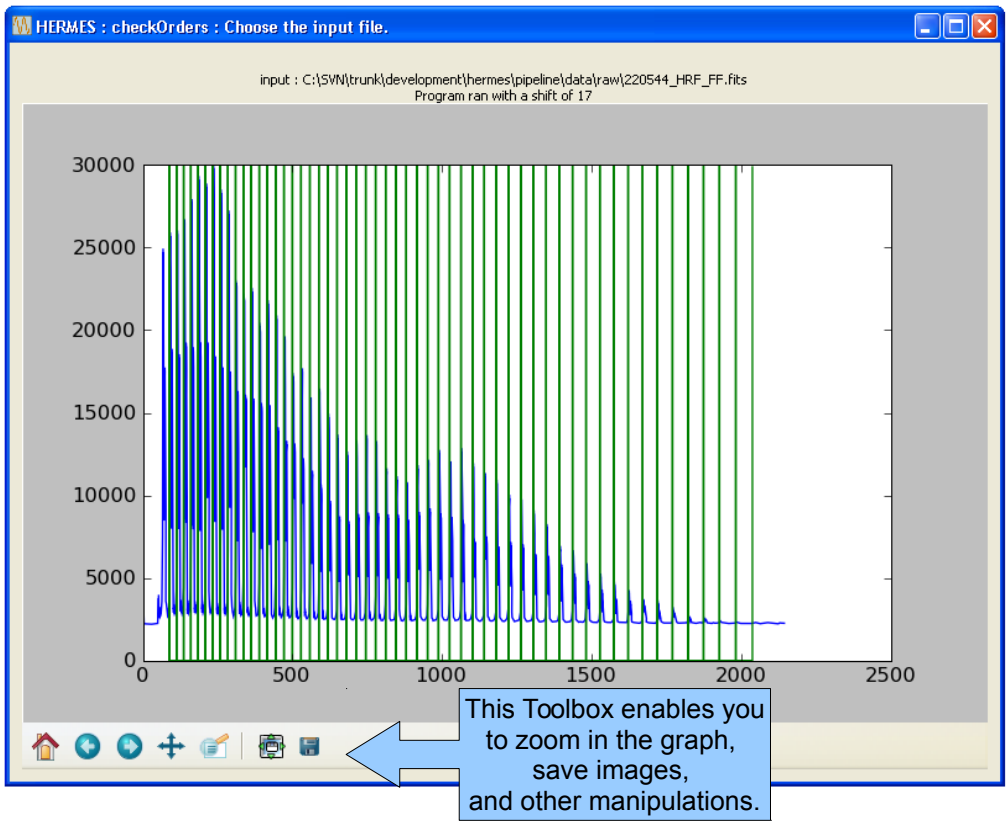
```
python checkOrders.py
```

The program will try to start the GTK engine used by our GUI. If it does not succeed, you will be prompted to give the full command-line arguments. Give the shift you want to use, and select the file using the multiple easy access ways present. By default, the program shows the inputPath as defined in the hermesConfig.xml file.





This is what appears for a while when the program runs.



When the computation is finished, the result is shown in a graph. Close the graph to end or run with another shift.

Log File

The log file is always situated in the “reduced” folder, and its name is always “hermes.log”.

The setup.py program has a sixth panel dedicated to the log file :