The SWIFT IRAF package

Ryan Houghton

Contents

List of Figures

Chapter 1

Introduction

SWIFT (Short Wavelength Integral Field specTrograph) is an integral field unit based (currently) at the Palomar 5m Telescope. This document is intended to describe the IRAF data reduction package for SWIFT (also called Swift). The principles of the data reduction mostly follow those for longslit spectroscopy; that is to say, each "slitlet" of the IFU is treated independently as though it were a single slit.

There are currently two ways to reduce SWIFT data using this pipeline. There are the original (and well tested) routines (wrapped in a user-friendly task called *swiftredMS*, see §3) which reduce the data with multiple interpolations (currently 3 separate interpolation steps to make a cube: extraction and tracing; wavelength calibration and removal of spectral curvature; horizontal line correction, which is optional).

There is also a set of routines in development (most with the original names as before, but ending in a "1") which can create a cube using just a single interpolation. However, these advanced routines are not the subject of this manual.

This document is split into the following sections:

Introduction The current section you are reading.

Installation A brief guide to installing the Swift IRAF package.

Quickstart A brief summary of *swiftred*, the main pipeline procedure in the Swift package. This section will tell you how to make a cube with the minimum effort (albeit not of publication quality).

Advanced data reduction A more in depth look at some of the features available in *swiftred* with details on how to reduce your data to publication quality.

Swift Procedures A detailed description of the individual procedures which are called in the *swiftred* procedure

Distributed Calibration Files Information on the calibration files that are distributed with the pipeline.

Tips and Tricks A few useful hints on how to get the best of the pipeline.

Troubleshooting If the pipeline has a tantrum, spank it with this.

Limitations and improvements Information on known problems with the pipeline and possible future improvements.

Chapter 2

Installation

2.1 Preliminaries

Before getting started, it is essential that the IRAF package has already been installed on your system together with the following packages:

- twodspec
- longslit
- apextract
- stsdas
- analysis
- fourier
- dither
- fitsutil
- proto
- imred
- bias
- ccdred

I'm afraid if you don't have these installed and don't know how to install them, you're going to need to read up on it. You'll also need a *home* IRAF directory (most likely ∼**/iraf/**) which should contain **login.cl** and **loginuser.cl** files. If you don't have these and don't know what they are, Google "begginers guide to iraf".

The pipeline (hereafter referred to as Swift[1]) also occasionally makes use of certain standard command line utilities such as *grep*, *awk*, *rm*, *ls*, *nl*, *wc*, *sdiff*, *ln*. Although the core of the pipeline has been designed to avoid using shell commands, they should be made available in the shell environment which IRAF is launched from (i.e. accessible with an escape (!) sequence). In addition to these standard linux utilities, it may help you to install and use the ESO shell tools *dfits* and *fitsort*. Finally, those familiar with IFU data analysis will most likely have QFitsView installed; because of its unique abilities for analysing IFU data, it can be called from the IRAF command line once Swift has been loaded *if* it is available to the shell from which IRAF was launched[2].

2.2 Getting started

Change directory to your IRAF home directory (where you launch **cl** from).

cd $home/iraf

Next, make a directory/ies for the location where you wish to install the Swift pipeline.

mkdir -p packages/swift

Now you need to untar the Swift package into the directory you just created. Normally, the tar archive unpacks to a directory called **swift**. So in the current example, you need to copy the tar archive into **packages** and unpack it there.

cp some_location/swift.iraf.X.X.X.tgz $home/iraf/packages

cd $home/iraf/packages

tar -xvfz swift.iraf.X.X.X.tgz

... <files unpack to swift dir> ...

It's now a good idea to check that everything is where it should be

ls $home/iraf/packages/swift

You should see a lot of files prefixed with **sw_** and nearly all ending in **.cl**.

Now you need to edit the **loginuser.cl** IRAF file (found in your IRAF directory), defining both the location of the new Swift package and the package itself, together with the package help files. In your **loginuser.cl** file, add the following before the last

[1]note the difference between the instrument SWIFT and the reduction package Swift

[2]Note to Mac OS X users: *alias* and *open -a /Applications/QFitsView* may be your friends here

keep statement:

```
# swift
reset swift = home$packages/swift/
task $swift = swift$swift.cl
printf ("reset helpdb=%s,swift$lib/helpdb.mip\nkeep\n",
        envget("helpdb")) | cl
flpr
```

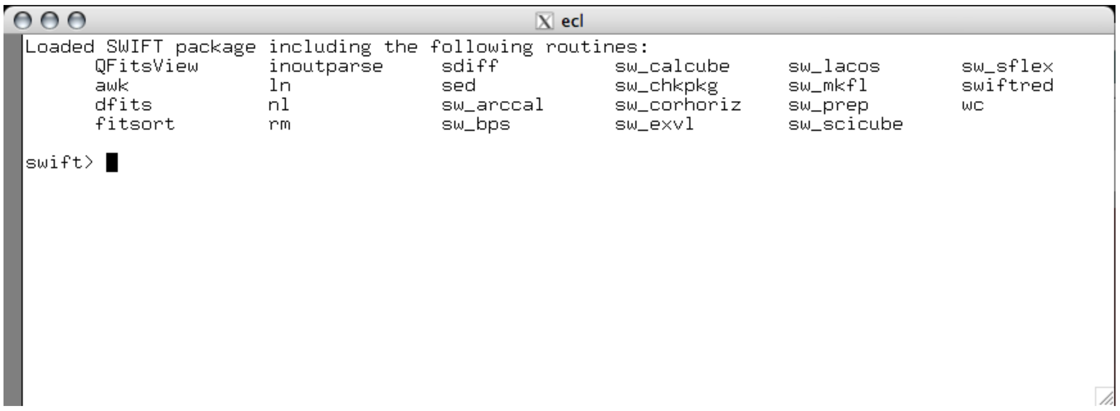
The first line sets a system variable to the location of the Swift package; the second defines the package; the third adds the help files to the IRAF help command and the last statement clears the cache.

If you now start IRAF in the usual way, you can load the Swift package by typing at the cl prompt:

swift

You should see the package load and a list of the new commands / procedures available to you (see Fig. 2.1).

This manual's latex source is available in the root Swift directory or in **doc/**. You can also browse the online help documents: typing **help** will show a brief list of all the available procedures in the Swift package; you can also be more specific by following **help** with the name of one of the procedures, such as **help swiftredMS**.



```
Loaded SWIFT package including the following routines:
      QFitsView      inoutparse     sdiff          sw_calcube     sw_lacos       sw_sflex
      awk            ln             sed            sw_chkpkg      sw_mkfl        swiftred
      dfits          nl             sw_arccal      sw_corhoriz    sw_prep        wc
      fitsort        rm             sw_bps         sw_exvl        sw_scicube

swift> 
```

Figure 2.1: The start screen of the Swift package, detailing the available command line routines. Note that the screen may appear differently in pyraf.

Chapter 3

Quickstart: using *swiftredMS.cl*

We start by presuming that the user is familiar with the basics of IRAF and longslit spectroscopy reduction; once you have installed and loaded up the Swift package you may *epar* the main reduction procedure for SWIFT data: *swiftredMS*. You will find that there are a lot of variables to tweak. However, fear not, as you can leave most of these at the default setting if you want to just make a quick cube to inspect the quality of the data.

You should know that SWIFT is made of two spectrographs: master and slave. Each spectrograph produces one half of the instrument field of view. You need to reduce the data from both of these spectrographs and combine the output; this is done by *swiftredMS*.

The first variable is **sci_input**. This is where you specify the (three digit) exposure *number(s)* of the science data you wish to reduce. E.g. files 'master001.fits' and 'slave001.fits' are represented by '001'. You can specify multiple files with commas, or in text files (prefixed with @ as is the usual way in IRAF). Wildcards cannot be used here. These input files will NOT be combined; they will be processed one-by-one so the user can combine them at the end, with the relevant offsets (to be determined by the user).

The next variable is **prefix_sci**. This describes the state of the input science files. If they are completely unprocessed, use **-**. If they have been 'prepped' (see later), use **^p**; if they have been bad pixel / cosmic corrected and prepped, use **^bp**.

The third variable is **vl_frame**. Here you should specify the file numbers of the calibration frames taken with the vertical line mask in the focal plane and the halogen lamp on. This file is *compulsory* and reduction is not possible without it. It should preferably have been taken on the same night as the science data. You can also specify more than one frame here (three is recommended) using commas or a file list prefixed with @: the frames will be bad pixel corrected and combined to remove cosmic ray strikes.

The fourth variable is **al_frame**. You should specify the file numbers of the calibration frames which were taken with the arc lamps on and no mask in the focal plane ('open'). This file is again *compulsory* as it will provide a wavelength solution for all spectra of the IFU; data reduction cannot proceed without it. As with **vl_frame**, you can specify more than one frame here (three is recommended) and they will be bad pixel corrected and combined to remove any cosmic ray strikes.

The fifth variable is **lamp_frame**. You can specify the file numbers of the flat field calibration frames (with the halogen lamp on and no mask in the focal plane). These files are not compulsory to create cubes, but are required if you wish to flat field pixels and/or attempt to correct for illumination (see **S_FI** below).

The sixth variable is **hl_frame**. You can specify the file numbers of the calibration frames which were taken with the horizontal line mask in the focal plane and the halogen lamp on. These files are not compulsory to create cubes, but are required to correct for the slight warp in the instrument field of view (see **S_HL** below).

The next six variables (all ending in 'name') are just suffixes for the reduced calibration frames; you do not need to edit them (and are advised not to).

The next 10 variables are all prefixed with **S_** (all in CAPITALS) and activate/deactivate certain *stages* of the calibration and reduction process (each corresponding to a different sub routine, usually). Some of these stages must be completed before you can reduce your science cubes, while others can be skipped if you want to have a quick look at the data in cube format. You can activate/complete these stages one at a time, or all in one go.

The first of these stages is *S_PREP* and needs to be activated (i.e. set to **yes**) to calibrate and reduce raw data frames. This stage must be run on all input frames once. This stage subtracts the bias, clips the overscan around the edge of the actual detector pixels and stitches the data from each amplifier together. It also combines multiple frames into one in the case of calibration frames. Make sure you have specified valid (unprep'd) input files/lists for the *sci_frame*, *vtrace_frame* and *arc_frame* otherwise you'll get an error! If *S_PREP* is set to yes, then **sci_prefix** should be set to **-**. Once you have completed this stage successfully, you can deactivate it and set **sci_prefix** to **ˆp**.

After this comes **S_EXVL** which must be activated and performed once to reduce any data. In this calibration stage, vertical lines are traced in the the vertical line mask frame to locate and extract the slitlets (44 in total in SWIFT: 22 in master, 22 in the slave).

The next parameter, **S_ARCCAL** should also be set to **yes**. In this calibration stage, the arc lines in each slitlet are compared to a line list (see the parameter **al_list**) in order to find a wavelength solution for all the spectra. This stage is essential to make reduced cubes and must be set to **yes**. If the procedure has problems automatically identifying the lines you may need to identify them manually; see §5.3.

The next calibration stage is **S_FI** and should be set to **yes** if you want to create flat field and/or illumination calibration data. Flat fielding involves dividing each input frame by a normalised high S/N frame to remove pixel-to-pixel quantum efficiency variations, as well as correct for dust on the detector. Illumination correction involves using the halogen flat fields to correct the science frames for illumination effects (both in the field of view / focal plane and spectroscopically). Using halogen lamps to perform this correction is only first order approximation: it's known (by me) that the halogen lamp flats have a different illumination to the *dome* or *twilight* flats, both of which are progressively closer approximations to the illumination pattern in the actual science data. It is possible to correct the illumination effects using the dome or twilight flats: see §4 and §**??** for more details. If you're using the smaller spaxel scales (e.g. 16mas), you may want to investigate taking and using moon flats.

The next parameter, **S_CALCUBES** should be set to **yes**: this calibration stage takes the vertical line frame and arc frame and makes cubes of them. To see if the data reduction is accurate, you should inspect these calibrations and ensure that the vertical lines are all aligned in the field of view and the arc lines all line up in the spectral direction; this then implies that the reduction is correct and you can trust the subsequent reduction of the science data. This parameter must also be set to **yes** for another parameter (**ainspect**) to be activated (discussed very soon).

The parameters **S_HL**, **S_BPS** and **S_SFLEX** can all be set to **no** for just a quick look at your data; we will deal with these calibration stages later in this document (§4); needless to say, they are not essential for the creation of basic cubes, but they may be important for generating publication quality data.

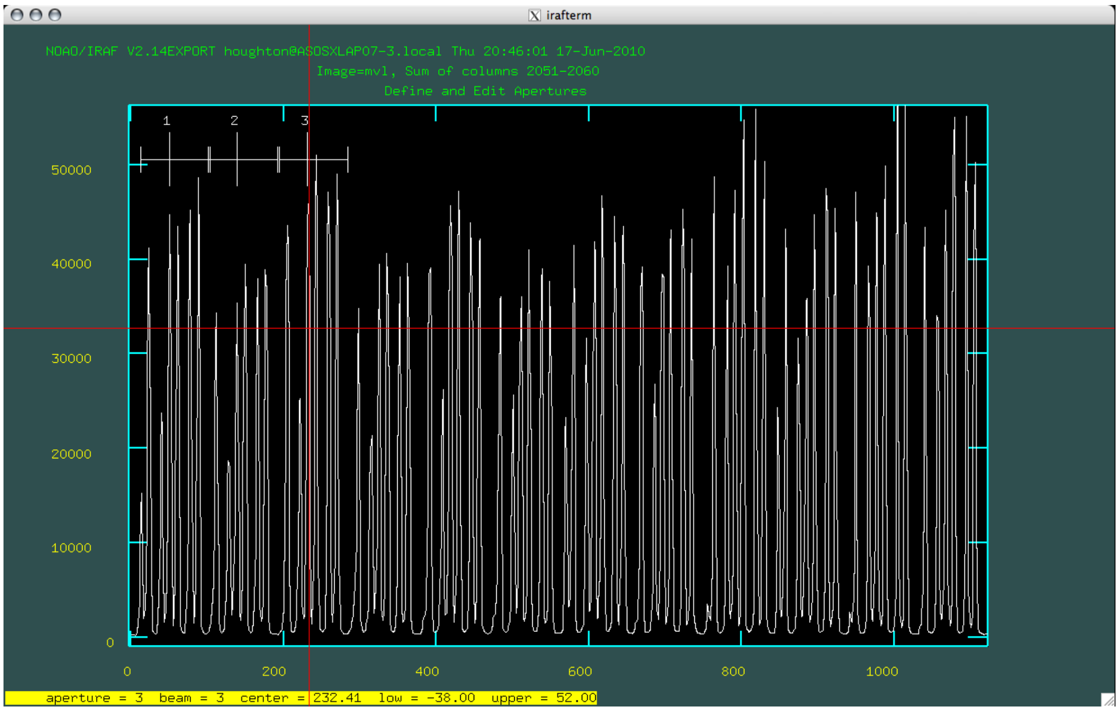
The last parameters, **S_SCICUBE** and **S_MSMERGE** should be set to **yes** if you want to reduce any science data. The first uses the (reduced) calibrations created by the previous stages to process the supplied science frames; the latter merges the master and slave spectrograph outputs into a single fits cube, called *ms???.cube.sci.fits*, where ??? refers to the exposure number you gave in **sci_input** above.

The next five parameters all start with *p_* and are used to tell the **S_PREP** stage which files to prep. For normal quick reductions (default settings), they should all be set to **yes**, except **p_htrace** (if you want a really quick reduction, you can set **p_flat** to **no** too, so long as **S_FI** is also set to **no**). For more info, see §4.

The seven parameters after these all start with **do_** and all refer to certain options in the previous stages. For the moment, **do_b** and **do_trim** should be set to **yes** but **do_lacos**, **do_thresh** and **do_skylines** should be set to **no**. If you want to flat field your data, set **do_flat** to **yes**; similarly if you want to illumination correct your data, set **do_illum** to **yes**. The use will be explained in detail in §4.

That covers the essential parameters. Before you run the *swiftredMS* procedure, you need to check a few more things. Firstly, which arc lamps were on when the arc frame was taken? If both Argon and Neon were on, check that the parameter **line_list** is set to **swift$ArNe.dat**; likewise, if just Ar was on, set it to **swift$Ar.dat**, etc. You should also make sure that three "interaction" parameters (**interfit**, **reidans** and **interact**) are set to **no** unless you know what you're doing. For now, let the procedure handle the finer details. However, you should make sure **ainspect** is set to **yes**. You should also make sure that the **verbose** parameter is set to **yes** - it's a good idea to see what's going on and watch out for warnings or errors (which you can then forward to an IRAF guru if things go wrong). What spaxel scale were your calibrations and data taken with? Input the value in mas (16, 80 or 235) in the **scale** parameter (just before the stage **S_** parameters).

You are now ready to run the pipeline! It's not all automated so there is some



```
NOAO/IRAF V2.14EXPORT houghton@ASOSXLAP07-3.local Thu 20:46:01 17-Jun-2010
Image=mvl, Sum of columns 2051-2060
Define and Edit Apertures
```

```
aperture = 3  beam = 3  center = 232.41  low = -38.00  upper = 52.00
```

Figure 3.1: An example of the coarse vertical line frame in the 235mas spaxel scale being labelled with extraction apertures (centred on the middle (of 7) traces). Note that apertures (or slitlets) are strictly numbered from left to right in chronological order.

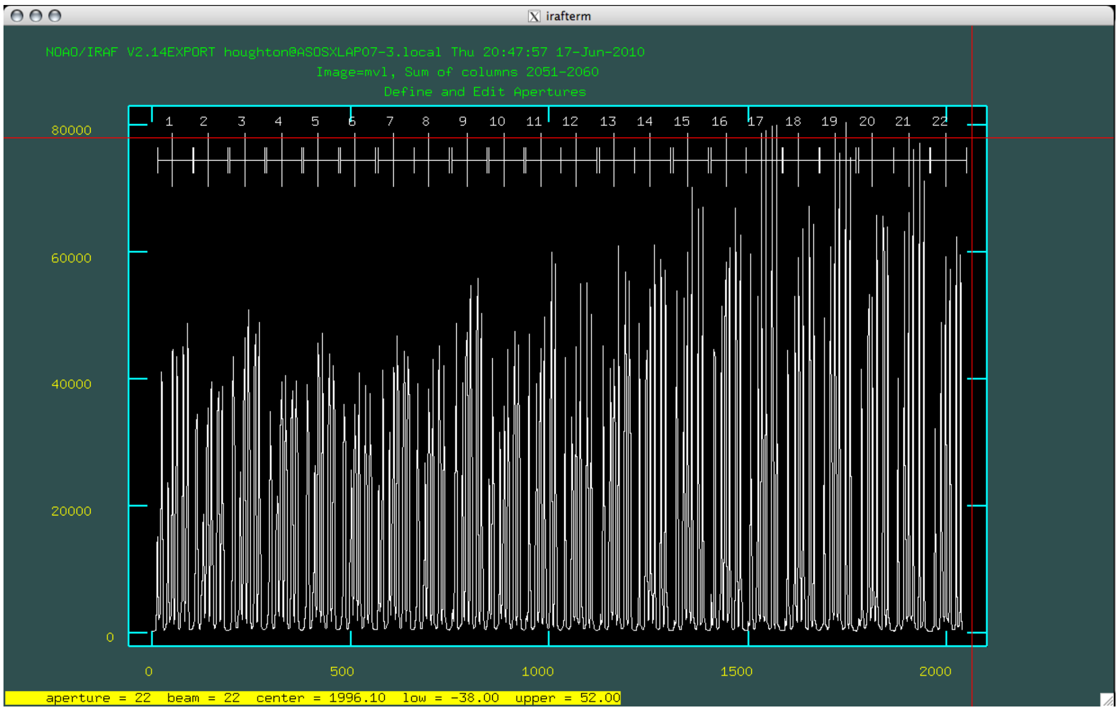
interaction required, particularly for the **S_EXVL** stage. We'll take you through that now.

3.1 exvl: extracting vertical line traces

When you activate the **S_EXVL** stage and run *swiftredMS*, you will be asked:

Edit apertures for [vertical line frame]? (yes):

You should answer **yes** to this. A window will pop up which should look something like Fig. 3.1 which shows a cut through all the vertical lines. For example, in the 235mas scale with the coarse vertical lines, each slitlet should have 7 vertical lines running along the wavelength axis, grouped in a two, then a three, then another two (the first line may be just off the edge of some slitlets). For other scales, using the coarse or fine vertical lines, this pattern changes. You now need to mark the central trace in each of the slitlets (i.e. in the 235mas scale with the coarse vertical lines, this is the middle line in the grouping of three); at this point you may find it easier to zoom in using the **w** and **e** keys. To do this, move the mouse (and the red cross hairs) over the


```
NOAO/IRAF V2.14EXPORT houghton@ASOSXLAP07-3.local Thu 20:47:57 17-Jun-2010
Image=mvl, Sum of columns 2051-2060
Define and Edit Apertures
```

```
aperture = 22  beam = 22  center = 1996.10  low = -38.00  upper = 52.00
```

Figure 3.2: A coarse vertical line frame observed in the 235mas spaxel scale. All the 22 traces have been marked and are shown above the relevant areas.

middle trace in the first (far left) slitlet and press **m** (for **m**ark). A white line should appear above the trace showing the width of the slitlet and the number of the aperture (or in this case, slitlet). You need to repeat this another 22 times, moving from the left to the right in strict order (see Fig. 3.1). Finally you should be left with 22 marked apertures to trace and extract (as in Fig. 3.2). You can delete traces and remark them if you make a mistake, but the ordering is strictly with aperture number ascending from left to right as shown in Fig. 3.2.

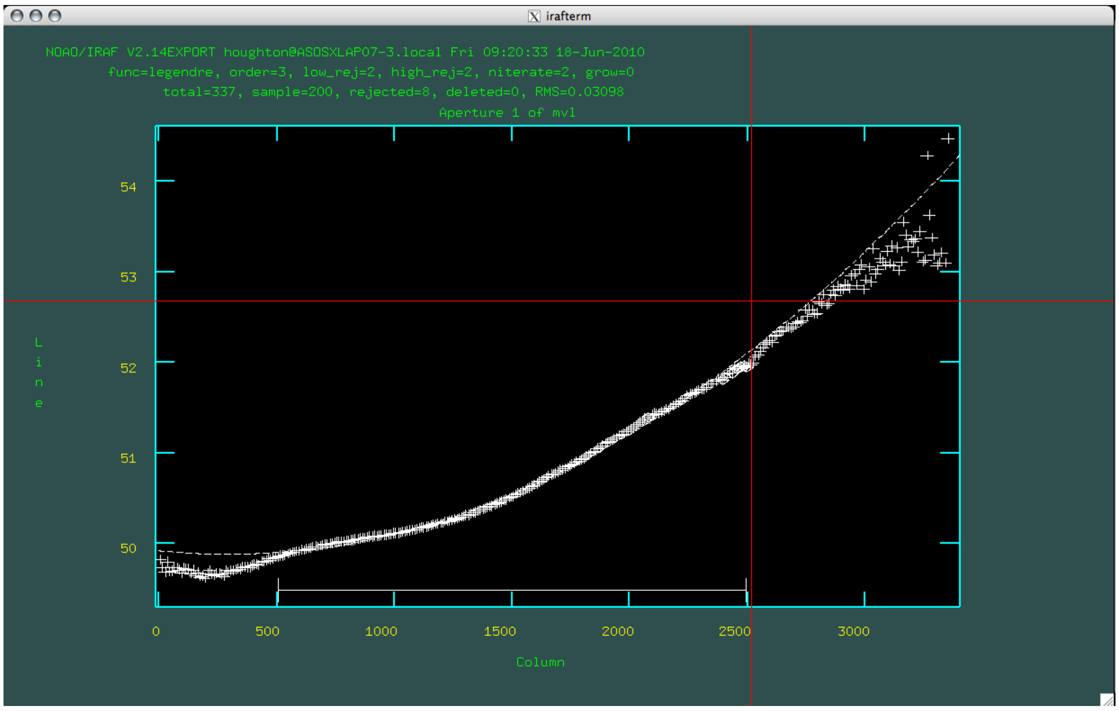
Once you have marked the apertures, quit by pressing **q** in the plotting window. After that, you will be asked:

Trace apertures for [vertical line frame]? (yes):

You must reply **yes** to this question; failure to do so will result in failure to reduce any data. You will then be asked:

Fit traced positions for [vertical line frame] interactively? (yes):

At this point, it's best to reply **yes** although if you're in a hurry you might risk trusting the automated tracing (it's not bad) and reply **no**; if you choose the latter option, be sure to create the cubes of the calibration data (by setting **S_CALCUBES**



```
NOAO/IRAF V2.14EXPORT houghton@ASOSXLAP07-3.local Fri 09:20:33 18-Jun-2010
        func=legendre, order=3, low_rej=2, high_rej=2, niterate=2, grow=0
               total=337, sample=200, rejected=8, deleted=0, RMS=0.03098
                                              Aperture 1 of mvl
```

Figure 3.3: The *swiftredMS* routine follows the maxima of the selected vertical line. At the very short and long wavelengths, it can sometimes have difficulty due to problems with the SWIFT PSF in these regions, particularly if the data was taken before 2010; thus is is best to fit the low scatter regions in the centre (where the PSF is good) and use a low order polynomial to extrapolate into the affected regions.

to **yes**) and inspect the cube created by the vertical line frame (in something like QFitsView, an external application) to make sure all the vertical lines are aligned and... vertical.

If you choose to trace the lines interactively, you will be asked:

Fit curve to aperture 1 of [vertical line frame] interactively? (yes):

To which you should of course reply **yes**. You will then see something along the lines of the trace shown in Fig. 3.3. Note that the white band at the bottom shows the **sample** which is being fitted. Press **?** to learn more about how you can change this fit. Try to fit only the regions where the trace was stable: you can clearly see that at short and long wavelengths the scatter increases dramatically (due to a PSF problem at short and long wavelengths - this data was taken before 2010) and these regions should not be fit by the polynomial (as is the case in Fig. 3.3). Furthermore, keep the order of the fit low (around 3-5) so a reliable extrapolation can be made into the poor PSF regions.

Once you're happy with the fit, press **q** to quit: you will then be asked if you want to fit the trace for aperture 2 interactively... if you do, reply **yes**. This process will repeat for all 22 slitlets. At the end you will be asked:

Extract apertures for [vertical line frame] ? (yes):

to which you must reply **yes**. The routine then extracts 22 2D specra from the vertical line frame, following the traces you just fitted and correcting for the curvature (via linear interpolation).

You just calibrated the vertical lines in the master spectrograph! But you now need to calibrate them in the slave spectrograph. So repeat the process, first marking the apertures (slit lets), then tracing and extracting them.

After calibrating the vertical lines, the rest is normally automatic: it bags up the 22 2D slitlets into a single MEF (mutli-extension fits) file with suffix *.strip.fits* for both the master and slave spectrographs.

3.2 arc_cal

This stage (activated with **S_ARCCAL**) starts by extracting the slitlets from the arc frames using the traces calculated in the previous section. It then tries to automatically identify the arc lines in individual slit lets (which it's usually quite good at if the correct line list has been given) and does this multiple times across each slitlet to calculate the slit curvature (*reidentify*). It then uses *fitcoords* to find a polynomial solution to the wavelength solutions and slit curvature. Of course, this has to be done for both the master and slave spectrograph inputs. Sometimes, the automatic identification of lines fails; in this case, you'll have to manually enter the position of the arc lines. However, there is a trick to avoid manually identifying all the arc lines; just identify 3 lines at 0.6402um, 0.7635um and 0.9123um. Then hit **l**, **f**, **q** a few times to let the automatic finding algorithm do the rest. You should then get a good fit (as described in the data reduction manual).

3.3 calcube

After the previous stages it's always wise to create and inspect the calibration cubes (the arc frame and the vertical line frame); but you have to create them first. With **S_CALCUBE** set to **yes**, *swiftredMS* interpolates the arc and vertical line traces onto a regular grid (saving the 22 interpolated strips from each spectrograph in MEFs with a suffix *.tstrip.fits*) and stacks them along the second spatial dimension to make cubes (with suffixes *.cube.fits*).

Because it's quite tricky to automatically identify all the arc lines correctly in the wavelength calibration stage (*S_ARCCAL*), this stage displays the transformed (wavelength calibrated) arc frame in DS9. By default, **ainspect** is set to **yes**, so after creating a cube of the transformed arc frame, a 2D image is created from this cube and then displayed in the first buffer of DS9 (that's why DS9 must be open when running the

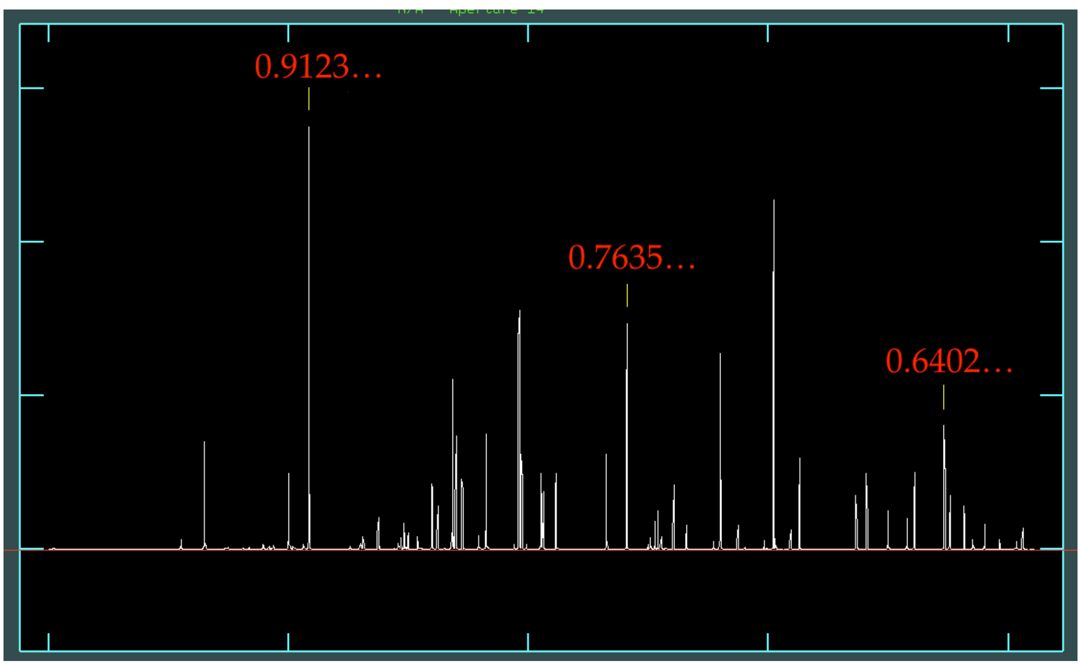


Figure 3.4: The three key SWIFT arc lines to identify manually

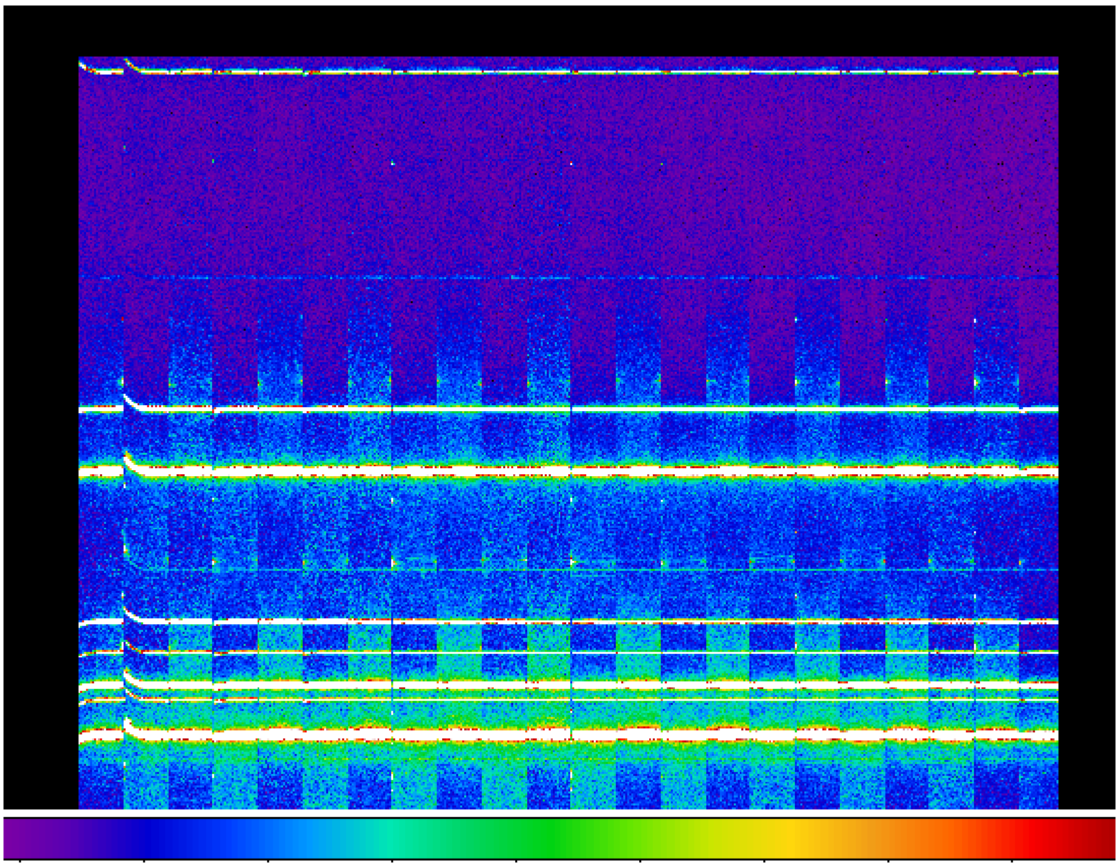
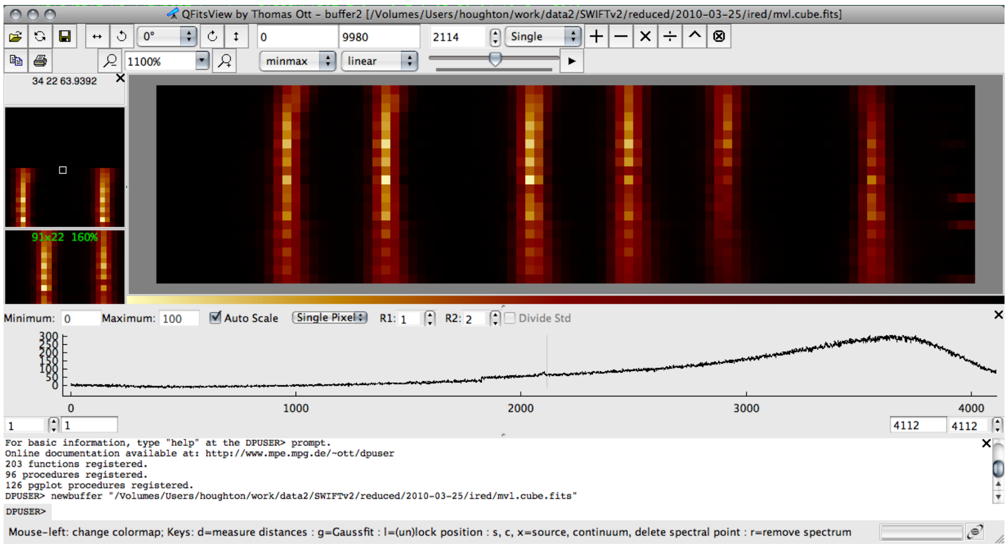


Figure 3.5: An example of what can go wrong with the automatic wavelength calibration: here we see that the first few slitlets aren't quite right; we're going to need to repeat the *S_ARCCAL* process being very careful with the *reidentify* stages. It's always a good idea to inspect the transformed arc frame, even if things appeared to work fine from the written output (as in this case).

pipeline). Hopefully, all the arc lines will be well aligned throughout the frame (showing both the master and slave outputs merged into one frame). However, they might not be (such as in Fig. 3.5) in which case, you probably need to get some help before carrying on with your reductions, or attempt an interactive arc calibration with the **reidans** parameter (which can be time consuming).

If you set **inspect** to **yes**, a wavelength collapsed image of the master and slave vertical line mask will be displayed in DS9. The lines should appear vertical and be well aligned across the master and slave (top and bottom 22slices) spectrographs (to better than a pixel accuracy, at least where the PSF is good) as shown in Fig. 3.6. To be absolutely confident of the quality of the vertical line calibration, you should open the *VL.cube.fits* file in QFitsView and inspect how the vertical lines behave across the different wavelength channels.



```
For basic information, type "help" at the DPUSER> prompt.
Online documentation available at: http://www.mpe.mpg.de/~ott/dpuser
203 functions registered.
96 procedures registered.
126 pgplot procedures registered.
DPUSER> newbuffer "/Volumes/Users/houghton/work/data2/SWIFTv2/reduced/2010-03-25/ired/mvl.cube.fits"
DPUSER>
```

Mouse-left: change colormap; Keys: d=measure distances : g=Gaussfit : l=(un)lock position : s, c, x=source, continuum, delete spectral point : r=remove spectrum

Figure 3.6: Using QFitsView we can see if the vertical traces have been extracted and aligned correctly: in this figure, they appear aligned to better than a pixel width. Note that due to slight misalignment with the mask, the right-most vertical line is not seen.

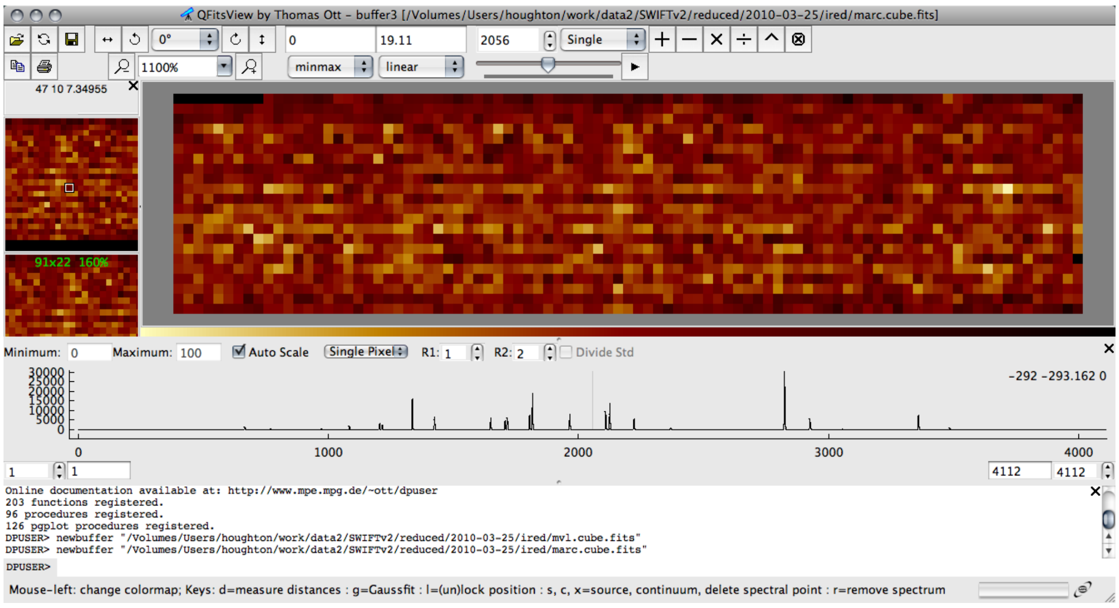
3.4 scicube

If you supplied science frames and asked the pipeline to reduce them (with the **S_SCICUBE** & **S_MSMERGE** parameters set to **yes**), *swiftredMS* will extract the slitlets using the vertical line traces you just fitted and then transform the slitlets to a regular grid using the polynomial fits made to the arc lines; finally it will put them all together in a cube (with suffix *sci.cube.fits*). Note that the intermediate stages are also available for inspection (with suffixes *.strip.fits* and *.tstrip.fits*).

From now on, any data taken on the same night as these calibrations (vertical line and arc frame) can be reduced very quickly: set **S_EXVL**, **S_ARCCAL** and **S_CALCUBES** to **no**. Give the new science files in **sci_input** and then set **S_PREP**, **S_SCICUBE**, **S_MSMERGE** to **yes**. You should also check which files will be prep'd: if you're supplying only science frames, you should activate **p_sci** and deactivate all other **p_** parameters. Run the script and it should output the reduced cubes without any intervention; you can use this reduction software as a genuine pipeline.

You should create and work in a new directory for each night's data you wish to reduce (so as not to get the IRAF *databases* muddled). In principle, for a single run, you could use the calibrations from the first night to reduce the data from the second night, but there's no guarantee the cube will be correctly formatted (although it might give you what you want - just a quick look at the data).

Although you now have cubes of your science data, there are several potential problems with them: you haven't corrected for flexure in the spatial or spectral direction; you haven't identified or corrected for bad pixels or cosmics; you haven't corrected for



```
Online documentation available at: http://www.mpe.mpg.de/~ott/dpuser
203 functions registered.
96 procedures registered.
126 pgplot procedures registered.
DPUSER> newbuffer "/Volumes/Users/houghton/work/data2/SWIFTv2/reduced/2010-03-25/ired/mvl.cube.fits"
DPUSER> newbuffer "/Volumes/Users/houghton/work/data2/SWIFTv2/reduced/2010-03-25/ired/marc.cube.fits"
DPUSER>
```

Mouse-left: change colormap; Keys: d=measure distances : g=Gaussfit : l=(un)lock position : s, c, x=source, continuum, delete spectral point : r=remove spectrum

Figure 3.7: Although not much to look at in the spatial dimensions, using QFitsView you can at least inspect that all the arc lines are aligned in every spectrum by moving the cursor over each spaxel: good evidence that the science exposures will be wavelength calibrated correctly? Don't be too sure: you should really do this with *arcinspect*, to spot subtle errors like that in Fig. 3.5.

the warped field of view using the horizontal line frame; you may not have corrected for flat field variation or illumination effects. However, don't give up just yet; most of these problems can be solved using *swiftredMS*, but you need to know more about how to use it: that's the subject of the next section.

Chapter 4

Advanced data reduction

It's probably best at this point to go through each of the individual parameters in the *swiftredMS* procedure; that way, you'll know what each one does and will be familiar with the procedure's capabilities before we go through an in depth example.

4.1 *swiftredMS* variables

For the case of each parameter listed below, we specify the variable name, which values (if any) are forced in parentheses and then on the next line, finer details about the use of the variable.

The following options allow the user to specify input files. This is done using the exposure numbers (in 3 digit format, separated by commas) or in text files (using 3 digit numbers, one on each line).

sci_input ()
: You should give exposure *number(s)* of the science data you wish to reduce (three digits separated by commas). E.g. files 'master001.fits' and 'slave001.fits' are represented by '001'. You can specify multiple files with commas, or in text files (prefixed with @ as is the usual way in IRAF). Wildcards cannot be used here. These input files will NOT be combined; they will be processed one-by-one so the user can combine them at the end, with the relevant offsets (to be determined by the user).

prefix_sci (- | ∧p | ∧bp)
: This is a prefix used to skip to a further stage in the pipeline. If you are starting from scratch, give **-** here. If you have already prepped all the input science frames, enter ∧**p**. If you have prepped and bad pixel / cosmic ray corrected the frames, enter ∧**bp**.

vl_frame ()
: You should give the file numbers for the vertical line frames here (halogen lamp on

with the vertical mask in place). As described in §3.1, this frame is used to trace and extract the slitlets. It MUST be specified to reduce any data (even if you've run the pipeline through in this directory before). You can also specify more than one frame here with commas (e.g. "001,002,003") or a file list prefixed with @: the median of all frames will be used to help remove any cosmic ray strikes (3 recommended).

al_frame ()
Give the file numbers of the arc line frames here. They must be specified and processed (§3.2) to wavelength calibrate the science data. You can also specify more than one frame here using either commas (e.g. "001,002,003") or a list prefixed with @: the median of all frames will be used to help remove any cosmic ray strikes (3 recommended).

lamp_frame ()
Give the file numbers of the halogen lamp frames. These files will be used to flat field and illumination correct the science data; they aren't essential to create cubes, but are recommended. You can also specify more than one frame here using either commas (e.g. "001,002,003") or a list prefixed with @: the median of all frames will be used to help remove any cosmic ray strikes (5 recommended).

hl_frame ()
Give the file numbers of the horizontal line frames here (halogen lamp on with the horizontal mask in place). These files are used to correct a slight warping in the IFU field of view, but are not essential for most science cases; if mosaicing or if your science target fills the field of view, you should provide and process these frames (see §5.5). You can also specify more than one frame here using either commas (e.g. "001,002,003") or a list prefixed with @: the median of all frames will be used to help remove any cosmic ray strikes (3 recommended).

The following options define 'tags' (names) to label certain files. You don't need to change any of them.

vlname (VL)
This is a tag used to identify the reduced vertical line frame. Please leave this as it is.

alname (ARC)
This is a tag used to identify the reduced arc line frame. Please leave this as it is.

lampname (LAMP)
This is a tag used to identify the reduced halogen lamp frame. Please leave this as it is.

flatname (FLAT)
This is a tag used to identify the reduced flat field frame. Please leave this as it is.

illumname (ILLUM)
This is a tag used to identify the reduced illumination frame. Please leave this as it is unless you want to use a different illumination cube, e.g. if you took twilight flats, you may wish to use them to illumination correct your data. See §**??** for how to do this.

hlname (HL)
This is a tag used to identify the reduced horizontal line frames. Please leave it as it is.

This next one is important and should be changed depending on your data and instrument setup.

scale (0.235)
This defines the plate scale of the observations, in arc seconds. SWIFT is capable of observing in three scales: 235mas (seeing mode), 80mas and 16mas; the latter two are usually exclusively used with PALM3K (adaptive optics).

The next few options all begin with 'S_' and turn on/off certain stages of the pipeline. Most of them are calibration stages, while the last two apply the calibrations to the science frames to create output cubes.

S_PREP (yes | no)
You should know from §3 that this stage prepares the input frames for data reduction (i.e. this stage performs bias subtraction, trimming, and combination of multiple frames).

S_EXVL (yes | no)
You should know from §3.1 that to extract any slitlets, this parameter must be set to yes and the stage completed at least once.

S_ARCCAL (yes | no)
Again, you should know that this wavelength calibrates the slitlets using the supplied arc frame; it is an essential calibration stage.

S_FI (yes | no)
If you want to calibrate the halogen lamp frames to flat field pixel-to-pixel variations and/or perform illumination correction on the science data, say *yes* here. You must also then provide the names of the halogen lamp frames in *lamp_frame*.

S_HL (yes | no)
If you want to calibrate the horizontal trace correction to correct a slight warp in the IFU FOV, say *yes* here. This stage will identify and trace horizontal lines in order to correct for the warped FOV. Note that halogen illuminated horizontal mask frames must also be given in *hl_frame*. This stage is not required to create cubes of the data, but users should be warned that the FOV of the instrument will not be perfectly rectangular without it (see §5.5). In practice you should only

calibrate this stage if you're mosaicing or observing fine details across the full field of view (e.g. large local planets).

S_CALCUBES (yes | no)
If you want to check that your calibrations are correct (wise), then you can check that vertical line data appears vertical when collapsed across all wavelengths, and that the calibrated arc lines are straight and aligned to the same row in all slitlets (as in Fig. 3.7). However, this stage is not required to reduce data.

S_BPS (yes | no)
Attach bad pix mask (BPM) to every science input image? Static bad pixel masks (mBPM.dat and sBPM.dat) can be found and edited in the directory *swift$*. If this stage is performed, BPMs are attached to every science input frame (0=good, 1=bad) and propagated through the reduction process in parallel to the data; bad pixels in the data frame are also replaced prior to (2D) interpolation to minimise bleeding effects during the reduction process. With attached masks, it is possible to see which pixels have been contaminated with bad data (values > 0 in the mask) and to what degree. It is then up to the user to decide on a cut above which (>0 being most pessimistic; > 0.1 being reasonable) they should mask out data pixels from any further analysis. At the end of the reduction stage, files with the suffix **.bpm.cube.fits** are the corresponding BPMs for the science cubes (with suffix**.sci.cube.fits**).

S_SFLEX (yes | no)
If the instrument has strong flexure (in the spatial dimension) then the slitlet apertures created from the vertical line mask may not align with the science data. This stage attempts to perform a cross-correlation between the arc frame (summing spectrally over *arc_sec* to isolate an arc line in alternating slitlets) and the input science frame (summing spatially over *sky_sec* to isolate a skyline in the same alternating slitlets) to calculate the (spatial) shift; it then applies this shift to the nearest pixel. The estimated shift is attached to the header of the input frame for future reference, but users should note that decimal places should be rounded to the nearest pixel if manually reversing the procedure. **NOTE:** EXPOSURES LESS THAN 300s WILL SHOW NO SKY LINES AND THIS ROUTINE WILL FAIL FOR THEM.

S_SCICUBE (yes | no)
Reduce the science frames using these calibrations? If this stage is applied, the input science frames are processed using the calibrations defined in the database directory and cubes (with suffixes **.sci.cube.fits**) are created for every input frame. Note that the extracted slitlets are stored with suffix **.strip.fits**, the (arc) wavelength calibrated slitlets with suffix **.tstrip.fits** and the (skyline) wavelength calibrated frames with suffix **.ttstrip.fits**; none of these files are required once the cubes have been created (and can be deleted).

S_MSMERGE (yes | no)
This stage, if activated, will combine the master and slave spectrograph outputs into a single science frame, prefixed with **ms** and suffixed with **.sci.cube.fits**, with the exposure number in-between. E.g. ms035.sci.cube.fits is the reduced output of files master035.fits and slave035.fits.

The next few stages start with 'p_' and turn on/off the prepping of various input files.

p_flat (yes | no)
Prep the halogen (flat) input files? If you're starting a new reduction from scratch, this should be set to **yes**. However, if you're repeating a reduction, the prepped files probably already exist (masterFLAT.fits and slaveFLAT.fits) and this stage can be skipped by setting it to **no**.

p_vtrace (yes | no)
Prep the vertical line input files? If you're starting a new reduction from scratch, this should be set to **yes**. However, if you're repeating a reduction, the prepped files probably already exist (masterVL.fits and slaveVL.fits) and this stage can be skipped by setting it to **no**.

p_arc (yes | no)
Prep the arc input files? If you're starting a new reduction from scratch, this should be set to **yes**. However, if you're repeating a reduction, the prepped files probably already exist (masterARC.fits and slaveARC.fits) and this stage can be skipped by setting it to **no**.

p_sci (yes | no)
Prep the science input files? If you're starting a new reduction from scratch, this should be set to **yes**. However, if you're repeating a reduction, the prepped files probably already exist (pmaster###.fits and pslave###.fits) and this stage can be skipped by setting it to **no**. **NOTE:** you must also change *prefix_sci* to ∧**p** in this case.

p_htrace (yes | no)
Prep the horizontal line input files? If you're starting a new reduction from scratch, this should be set to **yes**. However, if you're repeating a reduction, the prepped files probably already exist (masterHL.fits and slaveHL.fits) and this stage can be skipped by setting it to **no**.

The next few parameters define which options are included in the reduction of the data.

do_b (yes | no)
If you want to perform the bias subtraction in the *S_PREP* stage, set this parameter to **yes**. If *S_PREP* is set to **no**, this parameter has no effect.

do_trim (yes | no)
: If you want to trim the input data down (to just those pixels that receive light from the instrument) in the *S_PREP* stage, set this parameter to **yes**. If *S_PREP* is set to **no**, this parameter has no effect.

do_flat (yes | no)
: If you want to apply the flat fields (masterFLAT.fits and slaveFLAT.fits) to the science data, set this flag to **yes**. If you want to achieve signal-to-noise ratios > 20, you must flat field.

do_illum (yes | no)
: If you want to apply the illumination cubes (masterILLUM.fits and slaveILLUM.fits) to the science data, set this flag to **yes**.

do_lacos (yes | no)
: Perform *La Cosmic* (cosmic ray identification) on input files? This is only performed if the parameter **S_BPS=yes** and will use the La Cosmic algorithm (van Dokkum 2001) on individual slitlets to identify cosmic rays hits; these cosmics are then added to the BPMs (which are reduced in parallel to the data).

do_thresh (yes | no)
: Perform the extra thresholding stage? This stage is performed only if **do_lacos=yes**, which in turn is only performed if **S_BPS=yes**. La Cosmic finds most cosmic hits, but a few escape its powers. These are usually very round and circular hits rather than streaks. This stage takes the model (galaxy+sky) created by La Cosmic and subtracts it off the input science frame; any pixels greater than *threshold* in this image are then marked as bad. For most science cases (faint galaxies, faint emission line objects) this method works well and mops up the remaining cosmics; however, care should be taken when reducing bright (flux/telluric/kinematic standard) stars, where residuals (even shot noise) are likely to be higher than *threshold*.

do_skylines (yes | no)
: Use the skylines to tweak wavelength solution (spectral flexure correction)? If this stage is selected, we attempt to wavelength calibrate (a second time) using skylines present in long exposures. **NOTE:** this stage will FAIL if no skylines are present in the data (i.e. the exposure time was < 5 mins). This stage also adds an extra interpolation to the final science cubes.

do_hl (yes | no)
: Apply the horizontal line correction to the science cubes (de-warp the field of view)? Use of this stage will add an interpolation to the reduction process.

The following options allow detailed changes to certain stages of the pipeline. They are recommended for expert use only.

vl_b_sample ()
Here you can specify the background region when fitting the vertical lines. This may be necessary if you're using the coarse vertical lines to calibrate the 80mas or 16mas scales (e.g. "[-15:-10,10:15]").

vl_width ()
Here you can specify the predicted width of the vertical lines. As above, you may need to specify a large number if using the coarse vertical line mask with the smaller 80mas or 16mas scales (e.g. "10").

master_a1_gain ()
In the *S_PREP* stage, the data is scaled to remove the gain of the CCD amplifiers (converting the counts back to electrons rather than ADU). This is the gain (in e/ADU) of the first amp in the master.

master_a2_gain ()
This is the gain (in e/ADU) of the second amp in the master.

slave_a1_gain ()
This is the gain (in e/ADU) of the first amp in the slave.

slave_a2_gain ()
This is the gain (in e/ADU) of the second amp in the slave.

msgainratio ()
Usually the difference in gains between the master and slave chips is taken out by the illumination correction. But you can give a manual correction here.

masterbpfile (swift$mBPM.dat | swift$mBPM2.dat)
Give static bad pixel data file for the master chip (NOT FITS). BPMs are stored in the *swift$* directory and can be edited there.

slavebpfile (swift$sBPM.dat | swift$sBPM2.dat)
Give static bad pixel data file for the slave chip (NOT FITS). BPMs are stored in the *swift$* directory and can be edited there.

line_list (swift$ArNe.dat | swift$Ar.dat)
Line lists are available (to edit) in the directory *swift$* and one must be specified here.

sky_list ()
If the user plans to wavelength calibrate using skylines, they need to specifiy a line list for the skylines here. At present, this is fixed to a file in the directory *swift$* (which may be edited there).

a_sec ()
If using the *S_SFLEX* option to correct for spatial flexure, a suitable range to

isolate an arc line in odd slitlets must be given here. However, you can also specify **auto** (recommended) to allow the pipeline to specify the region.

sky_sec ()
If using the *S_SFLEX* option to correct for spatial flexure, a suitable range to isolate a sky line in odd slitlets must be given here. However, you can also specify **auto** (recommended) to allow the pipeline to specify the region.

max_shift ()
When using the *S_SFLEX* option, the user can give a maximum pixel shift to search for here.

cut_low ()
When extracting the slitlets with information gained in the *S_EXVL* stage, the number of pixels to cut left of central vertical trace must be given here. **NOTE:** this parameter is different pre-2010 and post-2010 and for different scales. Post-2010 and for the 235mas scale, it is set to -45.1 (the 0.1 helps prevent a bug in apall which occasionally rounds 45.0 to 44.0).

cut_high ()
When extracting the slitlets with information gained in the *S_EXVL* stage, the number of pixels to cut right of central vertical trace must be given here. **NOTE:** this parameter is different pre-2010 and post-2010 and for different scales. Post-2010 and for the 235mas scale, it is set to 45.1 (the 0.1 helps prevent a bug in apall which occasionally rounds 45.0 to 44.0).

xorder ()
When fitting the arc frame, this is the x-order for the polynomial fit.

yorder ()
When fitting the arc frame, this is the y-order for the polynomial fit.

skyxorder ()
When using the *do_skylines* option to correct for spectral flexure, the user can specify the polynomial degree to fit here.

nlost ()
When using the *do_skylines* option to correct for spectral flexure, the user can give the maximum number of lines to drop in the reidentify stage here; note that it's quite easy to drop skylines and still get a good overall correction because there are lots of skylines and the correction is in 1D only (curvature in the skylines from flexure is not corrected for, as it has not been observed).

lam0 ()
When creating an output cube, interpolations are required. The user can specify the output starting wavelegth onto which the data is interpolated here.

dlam ()
When creating an output cube, interpolations are required. The user can specify the output wavelegth interval (pixel scale) onto which the data is interpolated here.

nlam ()
When creating an output cube, interpolations are required. The user can specify the number of output pixels here.

loglam ()
When extracting kinematics (velocities, dispersions etc), data is often required on a log-lambda (wavelength) grid. To avoid having to interpolate the data again, the user can specify that the initial interpolation be on a log-lambda grid here.

x1 ()
This is the initial x coordinate for the output science cube (in arc seconds, master and slave combined), which is usually half of the longest dimension (-11" for the 235mas scale). **NOTE:** this will need to be changed for the 80mas and 16mas scales.

y1 ()
This is the initial y coordinate for the output science cube (in arc seconds, master and slave combined), which is usually half of the shortest dimension (-4.4" for the 235mas scale). **NOTE:** this will need to be changed for the 80mas and 16mas scales.

onx ()
Give the number of spaxels along the x dimension for the output science cubes (master and slave combined).

ony ()
Give the number of spaxels along the y dimension for the output science cubes (master and slave combined).

threshold ()
When using the *do_thresh* stage, the user can specify the threshold value here, above which all pixels are classified as cosmics (after model subtraction).

interp ()
If using the *S_SFLEX* option to correct for spatial flexure, the user can specify the method of interpolation here (nearest is STRONGLY recommended; there's no advantage to linear or better, but there is the disadvantage of an extra interplolation on the data, smoothing out the noise statistics).

trans_interp ()
When interpolating the data onto a regular grid, the user can specify the type of interpolation for the transform here: nearest is NOT recommended as the data is not sampled finely enough; instead, linear interpolation (or better) is recommended.

vt_sample ()
: In the *S_EXVL* stage, when one traces the vertical lines, the traces can go sub-optimal at long and short wavelengths (where the PSF of the instrument degrades). To avoid this, the user can specify a (good PSF) pixel range to fit traces to here, thus cutting out the short and long lambda points automatically. With data taken before 2010, you should probably use "500:2500" here; however, post 2010, you can use the full range (e.g. "*").

sflip (yes | no)
: The SWIFT spectra are reversed to the normal direction in IRAF; as such, if you use the *S_PREP* stage to prepare the frames, you also need to have this flag set to **yes**. However, if you used the *old python pipeline* to prep the input frames, you can use this pipeline to reduce them if this parameter is set to **no** (and, naturally, *S_PREP* must also be set to **no**; see §7).

The next few options allow more interaction from the pipeline. WARNING: if you turn these stages on, you will need to understand how the pipeline works to know what to do with the various options which are presented; very little explanation is given.

interfit (yes | no)
: Make the FITCOORDS stage interactive? If you want to make sure the pipeline is fitting the arc frames correctly, you can make the FITCOORDS stage interactive and monitor the results. WARNING: this takes some time and patience.

reidans (yes | YES | no | NO)
: Make the reidentify stage interactive? Before FITCOORDS, the arc lines must be identified across each slitlet. This can sometimes go wrong; if this is the case, or if you like pain, you can monitor and ammend the results by setting this parameter to yes or YES. Note that CAPITALS stop REIDENTIFY from asking you repeatedly (for every slitlet) if you want to fit interactively.

interact ()
: Do you want to make nearly EVERY OTHER REDUCTION STAGE interactive? If something bad is happening or the script is crashing somewhere or the cubes look crazy, you may well want to do this. My advice: go get a cup of tea and a biscuit before you start; phone ahead and say you'll be late back.

arcinspect (yes | no)
: Select this option to display the transformed ARC frame in DS9 to check that all the ARC lines are aligned along the entire wavelength range across all slitlets. See Fig. 3.5 for an example of what to look out for.

verbose (yes | no)
: Does the pipeline pause for long periods of time without printing anything? Does this worry you? Well now you can revel in the rapture that is information overload.

Chapter 5

Swift procedures

We'll now go through the individual procedures that *swiftred* makes use of to reduce the data. This may be of interest to you if you want to write your own reduction script which makes use of these functions.

5.1 sw_prep

This procedure prepares the input data frames for further reduction: it will bias subtract, trim and combine frames with various scaling / clipping options.

5.1.1 Inputs

input ()
: Give input frames here (single, REGEXP or @file)

output ()
: Give standard output file name(s), **%in%out%other** or ∧prefix notation. Note that after you've run *sw_prep*, this parameter changes to the name of a filelist containing the output file names. This helps in scripting the prep stages.

s_b (yes | no)
: Perform bias subtraction?

cb_order ()
: The order of the polynomial fitted to the overscan region.

cb_loreject ()
: The low rejection factor (times σ) when fitting the polynomial to the bias region.

cb_hireject ()
: The high rejection factor (times σ) when fitting the polynomial to the bias region.

cb_niter ()
: The number of rejection iterations when fitting the polynomial to the bias region.

combine (yes | no)
: Combine all input frames?

coutput ()
: Filename for combined output

ctype (sum | average | median)
: Type of image combination

scale (none | mode | median | mean | exposure)
: Type of image scaling before combining.

reject (none | minmax | ccdclip | crreject | sigclip | avsigclip | pclip)
: Type of pixel rejection to apply when combining

nlow ()
: If using minmax rejection, this is the number of low pixels to reject

nhigh ()
: If using minmax rejection, this is the number of high pixels to reject

nkeep ()
: If using minmax rejection, this is the min number of pixels to keep (+ve) or max number to reject (-ve)

mclip (yes | no)
: Clip from median rather than mean?

lsigma ()
: If using sigclip rejection, this is the low limit for sigma clipping

hsigma ()
: If using sigclip rejectionm this is the high limit for sigma clipping

btype (column | line | none)
: Type of bias subtraction (only **column** is supported at present).

a1_gain ()
: Give the gain (e/ADU) for amplifier 1

a2_gain ()
: Give the gain (e/ADU) for amplifier 2

trim (yes | no)
: Trim images (overscan, bias section etc)?

bpm ()
: Give the bad pixel mask (BPM) here to correct combined image (FITS, PL or DAT; good=0, bad=1)

interact (yes | no)
Make all stages as interactive as possible?

verbose (yes | no)
Verbose output?

5.1.2 Description

NB: At present, only the *SP1* and *SP2* (2 Amp.) readout modes are supported; as most of the SWIFT data upto now has been readout using 2 amplifiers, this shouldn't be a problem for the majority of cases. If a 2 Amp. readout was not used, this procedure should quit gracefully.

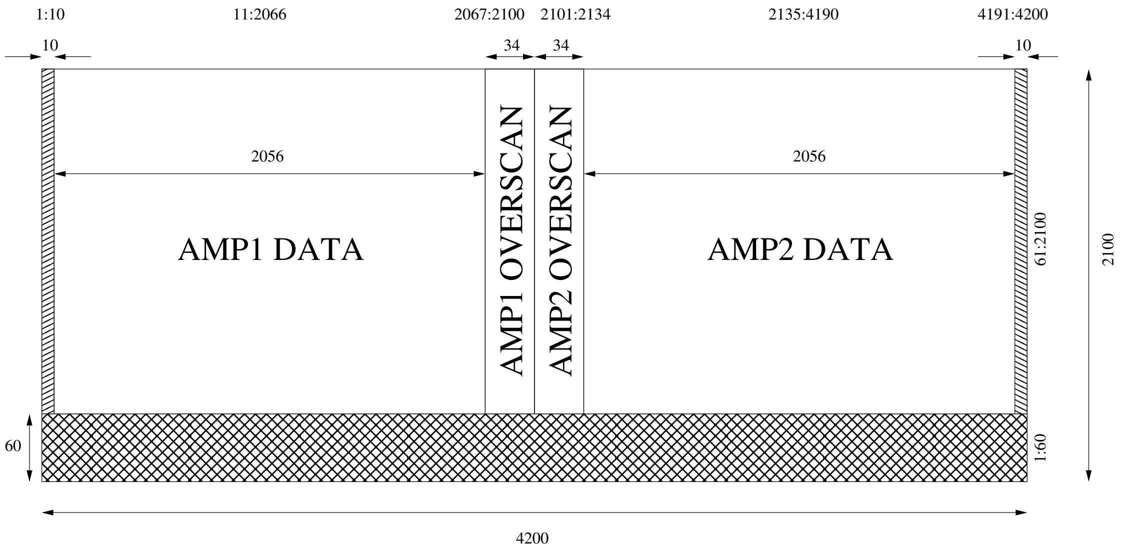


Figure 5.1: The pixel format for the 2 amp. *SP2* readout mode (master).

The data that comes off the SWIFT CCD is read out of the CCDs is not as one might naively expect. Rather than rows of pretty spectra, depending on the number of A2D amplifiers used to read out the data, it is necessary to clip and subtract a bias level from each section (from each amp.) and then stitch the sections together. SWIFT is capable of reading out data with: just one amp. (*U1_AMP* mode); two amps (*SP1* and *SP2* modes); and 4 amps. (not yet implemented). The advantage of more amps. is a much shorter readout time.

Figs. 5.1 and 5.2 illustrate the raw formats for SP1 (slave) and SP2 (master) data. There are two data sections (from each amp.) and two bias sections (again, from each amp.). Then there are a series of excess regions (shaded in Figs. 5.1 and 5.2) which are not important for data reduction (consult Tim Goodsall for more information about these regions).

The *sw_prep* routine starts by subtracting the relevant bias from each section (unless told not to) using the assigned bias regions to calculate the values. Note that the bias

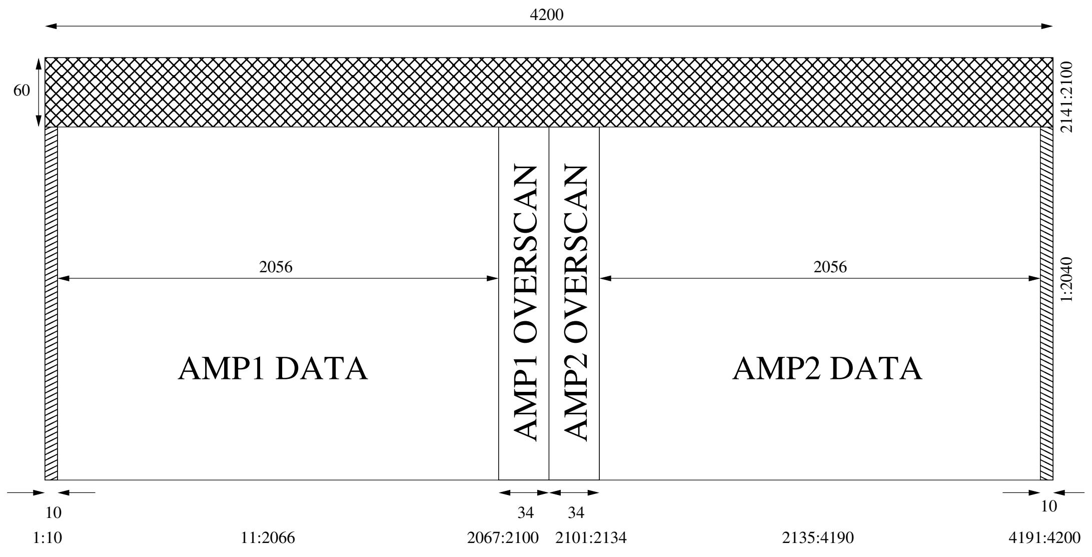


Figure 5.2: The pixel format for the 2 amp. *SP1* readout mode (slave).

value is not a constant in the case of SWIFT but a low order polynomial is fit to the bias regions and then subtracted off the data regions; the user can change the order of this fit using the parameter *cb_order*; it them trims off the overscan regions (if *s_trim=yes*) and merges all the data into one contiguous array.

5.2 sw_exvl

vtrace_frame ()
Give the name of the prep'd vertical trace frame here.

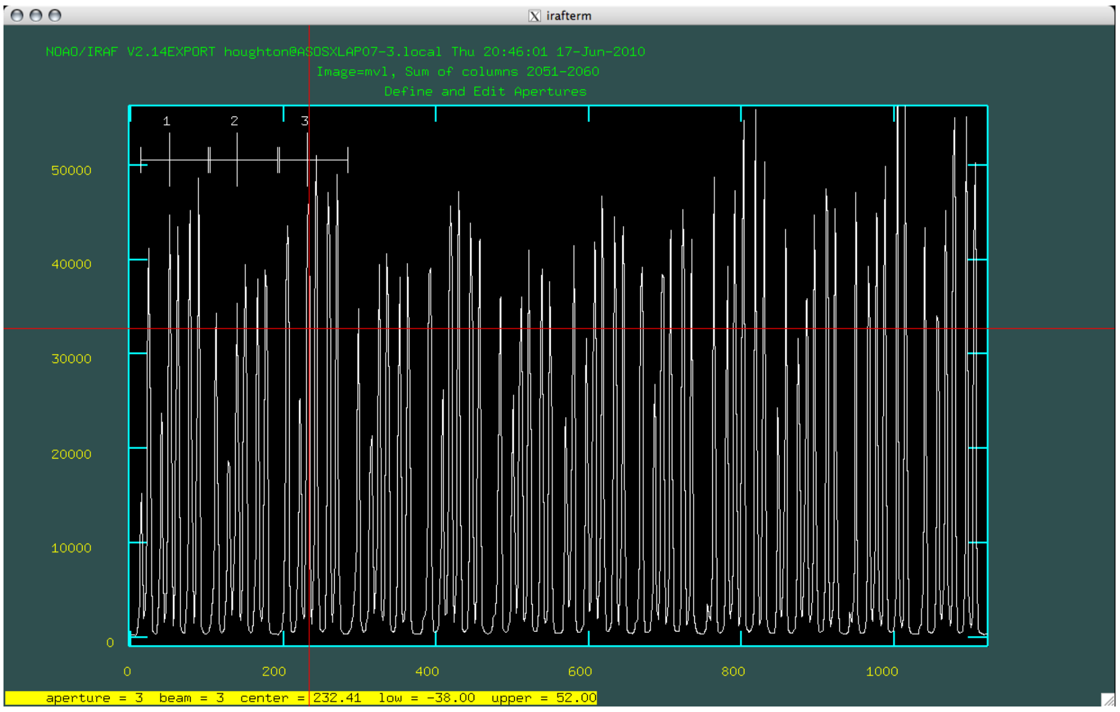
cut_low ()
Number of pixels to cut left of central vertical trace. Together with *cut_high* below, these two numbers, added together, make up the size (width) of each slitlet.

cut_high ()
Number of pixels to cut right of central vertical trace.

b_sample ()
The background sample region.

width ()
The width of the vertical lines (in pixels).

vtrace_sample ()
The pixel range over which to trace / fit a polynomial to the position of the vertical line in each slitlet. Note that this should only include the region where the spatial



```
NOAO/IRAF V2.14EXPORT houghton@ASOSXLAP07-3.local Thu 20:46:01 17-Jun-2010
Image=mvl, Sum of columns 2051-2060
Define and Edit Apertures
```

```
aperture = 3  beam = 3  center = 232.41  low = -38.00  upper = 52.00
```

Figure 5.3: An example of the vertical trace frame being labelled with extraction apertures (centered on the middle (of 7) traces). Note that apertures (or slitlets) are strictly numbered from left to right in chronological order.

and spectral PSFs are well behaved and is not able to account for any staggering (in the spectral direction) between the slitlets.

aptab ()
: The aperture table for slitlet ordering. Leave blank for SWIFT data.

verbose (yes | no)
: Print out a verbose output?

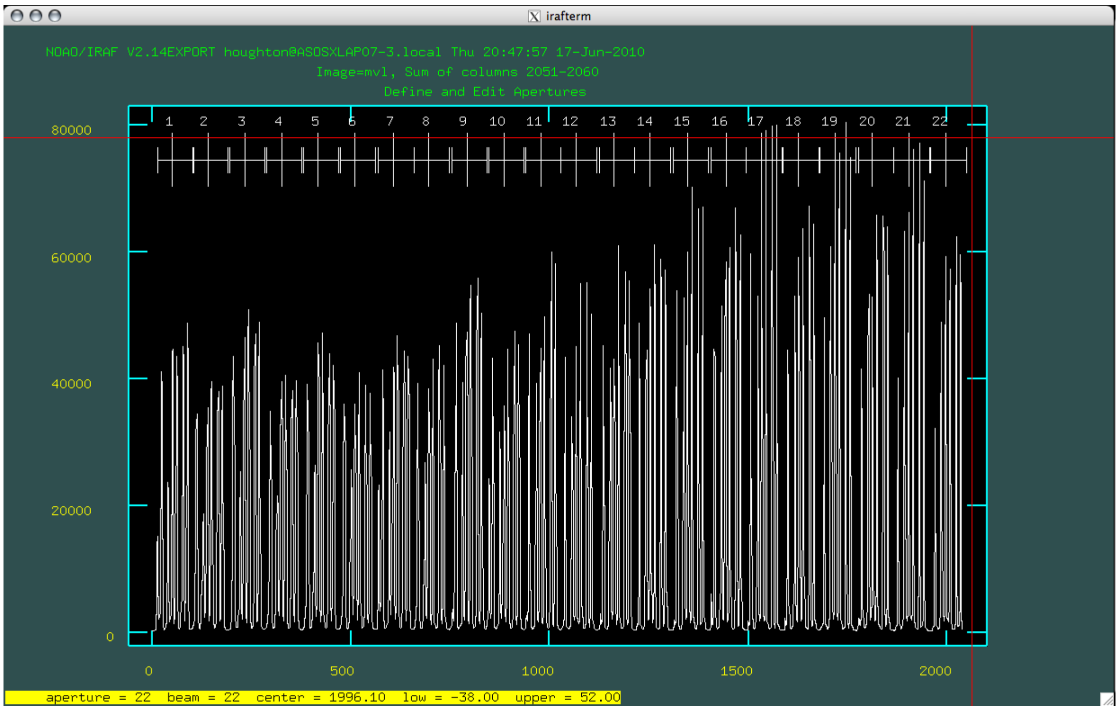
5.2.1 Description

As described previously in §3.1, this stage is responsible for identifying the 22 slitlets, tracing the vertical lines from the vertical mask to find the curvature in each slitlet and then extracting the slitlets to create 22 2D spectra.

Although already discussed, here we reiterate the actions required by the user in the *sw_exvl* stage.

When you first run *sw_exvl* on your data, you will be asked:

Edit apertures for [vertical line frame]? (yes):



```
NOAO/IRAF V2.14EXPORT houghton@ASOSXLAP07-3.local Thu 20:47:57 17-Jun-2010
Image=mvl, Sum of columns 2051-2060
Define and Edit Apertures
```

```
aperture = 22  beam = 22  center = 1996.10  low = -38.00  upper = 52.00
```

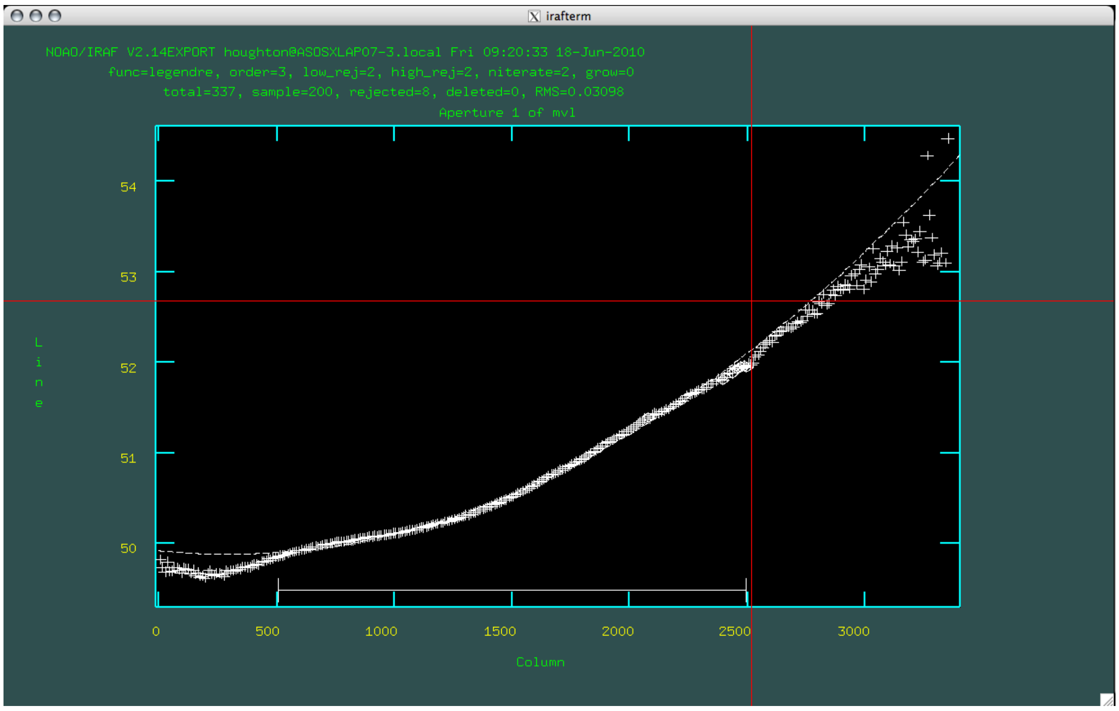
Figure 5.4: All the 22 traces have been marked and are shown above the relevant areas.

You should answer **yes** to this. A window will pop up which should look something like Fig. 3.1 which shows a cut through all the vertical lines. Each slitlet should have 7 vertical lines running along the wavelength axis, grouped in a two, then a three, then another two (but often the first line is just off the edge of the slitlets). You now need to mark the central trace in each of the slitlets (i.e. the middle one in the grouping of three); at this point you may find it easier to zoom in using the **w** and **e** keys. To do this, move the mouse (and the red cross hairs) over the middle trace in the first (far left) slitlet and press **m** (for **m**ark). A white line should appear above the trace showing the width of the slitlet and the number of the aperture (or in this case, slitlet). You need to repeat this another 22 times, moving from the left to the right in strict order (see Fig. 5.3). Finally you should be left with 22 marked apertures to trace and extract (as in Fig. 5.4). You can delete traces and remark them if you make a mistake, but the ordering is strictly with aperture number ascending from left to right as shown in Fig. 5.4.

Once you have marked the apertures, quit by pressing **q** in the plotting window. You will be asked:

Trace apertures for [vertical line frame]? (yes):

You must reply **yes** to this question; failure to do so will result in failure to re-



```
NOAO/IRAF V2.14EXPORT houghton@ASOSXLAP07-3.local Fri 09:20:33 18-Jun-2010
      func=legendre, order=3, low_rej=2, high_rej=2, niterate=2, grow=0
           total=337, sample=200, rejected=8, deleted=0, RMS=0.03098
                                  Aperture 1 of mvl
```

Figure 5.5: The swiftred routine follows the maxima of the selected vertical line, but at the short and long wavelengths, it can sometimes have difficulty due to problems with the SWIFT PSF in these regions; thus is is best to fit the low scatter regions in the centre (where the PSF is good) and use a low order polynomial to extrapolate into the affected regions.

duce any data. You will then be asked:

Fit traced positions for [vertical line frame] interactively? (yes):

At this point, it's best to reply **yes** although if you're in a hurry you might risk trusting the automated tracing (it's not bad) and reply **no**; if you choose the latter option, be sure to create the cubes of the calibration data (by setting **S_CALCUBES** to **yes**) and inspect the cube created by the vertical line frame (in something like QFitsView, an external application) to make sure all the vertical lines are aligned and... vertical.

If you choose to trace the lines interactively, you will be asked:

Fir curve to aperture 1 of [vertical line frame] interactively? (yes):

To which you should of course reply **yes**. You will then see something along the lines of the trace shown in Fig. 5.5. Note that the white band at the bottom shows the

sample which is being fitted. Press **?** to learn more about how you can change this fit. Try to fit only the regions where the trace was stable: you can clearly see that at short and long wavelengths the scatter increases dramatically (due to a PSF problem at short and long wavelengths) and these regions should not be fit by the polynomial (as is the case in Fig. 5.5). Furthermore, keep the order of the fit low (around 3) so a reliable extrapolation can be made into the poor PSF regions.

Once you're happy with the fit, press **q** to quit: you will then be asked if you want to fit the trace for aperture 2 interactively... if you do, reply **yes**. This process will repeat for all 22 slitlets. At the end you will be asked:

Extract apertures for [vertical line frame] ? (yes):

to which you must reply **yes**. The routine then extracts 22 2D specra from the vertical line frame, following the traces you just fitted and correcting for the curvature (via linear interpolation).

After this stage, the rest of the reduction is normally automatic: it bags up the 22 2D slitlets into a single MEF (mutli-extension fits) file with suffix *.strip.fits.*

5.3 sw_arccal

arc_frame ()
Give a single prep'd arc frame here.

vtrace_frame ()
Give a single prep'd vertical trace frame here; not that you must have already run *sw_exvl* on this frame prior to running *sw_arccal.*

cut_low ()
Number of pixels to cut left of central vertical trace; this must be the same value that was used while running *sw_exvl.*

cut_high ()
Number of pixels to cut right of central vertical trace; this must be the same value that was used while running *sw_exvl.*

arc_list (ArNe.dat | Ar.dat | Ne.dat | userlinelist.dat)
Give the arc line list here. There are only four choices here, all prefixed with the *Swift* home directory: **ArNe.dat**, **Ar.dat**, **Ne.dat** and **userlinelist.dat**. The names should be self explanatory: the first includes lines for both the Argon and Neon lamps and should be used when both lamps were switched on to take the arc frame; the second contains lines just for Argon and is particularly useful if the Neon lamp failed during the observing run; the third contains just the Neon lines and generally should be required unless the Argon lamp failed. Finally, the last line list does not exist: it is for the user to create and place in the *Swift* home

directory; this option allows for an alternative line list, which may be required if trying to optimise the line list, or if using the pipeline for a different instrument. Note that if you specify the wrong line list for the arc frame, the identification of lines will fail and no solution will be found; data reduction cannot continue in such circumstances.

arcfilelist ()
This variable needs to be set to a filename that will (after this stage) contain a list of extracted arc strips. The user need not worry too much about this variable, although it will be used later on and needs to be passed to later reduction stages. For the aficionados, this text file lists the names of the wavelength solutions stored in the **database**.

xorder ()
Specify the x-order for the *fitcoords* stage. The arc lines appear curved in the slitlets, and this curvature needs to be removed so that the spectra can be interpolated (at a later stage) onto a regular grid (in space and wavelength). This is the order of the polynomial fitted to the arc lines along the x-axis

yorder ()
As above, this is the y-order used in *fitcoords*.

nlost ()
This is the maximum number of lines to drop in the *reidentify* stage.

sflip (yes | no)
Usually spectra run from low lambda on the left to large lambda on the right. However, this isn't the case for the extracted SWIFT slitlets (unless the files were prep'd with the old pipeline; see §7). Rather than wasting time transforming the files, it's easier to tell the (auto)identify (and reidentify) stages that the spectral axis is flipped. The axis is then reverted to the normal orientation when the data is transformed onto a regular grid at no extra time cost (at a later stage).

interfit (yes | no)
This option will make the *fitcoords* stage interactive.

reidans (yes | YES | no | NO)
This option will make the *reidentify* stage interactive. A simple **yes** or **no** causes the routine to ask if you want to run *reidentify* interactive every time; however, setting this to **YES** or **NO** forces the reply for every slitlet such that the routine does not ask again.

interact (yes | no)
This option makes nearly all the other stages (i.e. excluding *reidnetify* and *fitcoords*) interactive. It slows things down a little, but it's useful for debugging.

verbose (yes | no)
Set to **yes** for a verbose output to the screen.

5.3.1 Description

As discussed previously in §3.2, this stage starts by extracting 22 2D slitlets from the arc frame using the traces calculated in §5.2. It then tries to automatically identify the arc lines in the arc frame (which it's usually quite good at if the correct line list has been given) and does this multiple times across each slit to calculate the slit curvature (*reidentify*). It then uses *fitcoords* to find a polynomial solution to the wavelength solutions and slit curvature (which in turn is used by *transform* in either the *sw_calcubes* or *sw_scicubes* stages to transform the slitlets onto an orthogonal x-λ grid).

If you have trouble automatically identifying arc lines, you may need to identify them manually. For that reason, the plots of the arc lines shown in Figs. 5.6, 5.7, 5.8, 5.9, 5.10, 5.11 may prove useful.

5.4 sw_calcube

inputfile ()
Give the input file name here (i.e. the file you want to make a cube of). You can use REGEXPs or file lists.

arcfilelist ()
This variable should be the same as was given in the *sw_arccal* stage: it's a list of filenames corresponding to the slitlets that were extracted (and calibarted) from the arc frame. This file allows the procedure to find the right information in the **database**.

outprefix ()
The prefix added onto the output cubes.

cubesuffix ()
The suffix added onto the output cubes.

scale ()
The spaxel scale in arc seconds.

lam0 ()
The input data is transformed to a regular wavelength grid: this is the starting wavelength for that grid.

dlam ()
This is the wavelength interval between pixels along the spectral direction.

nlam ()
This is the total number of pixels along the wavelength axis after transformation.

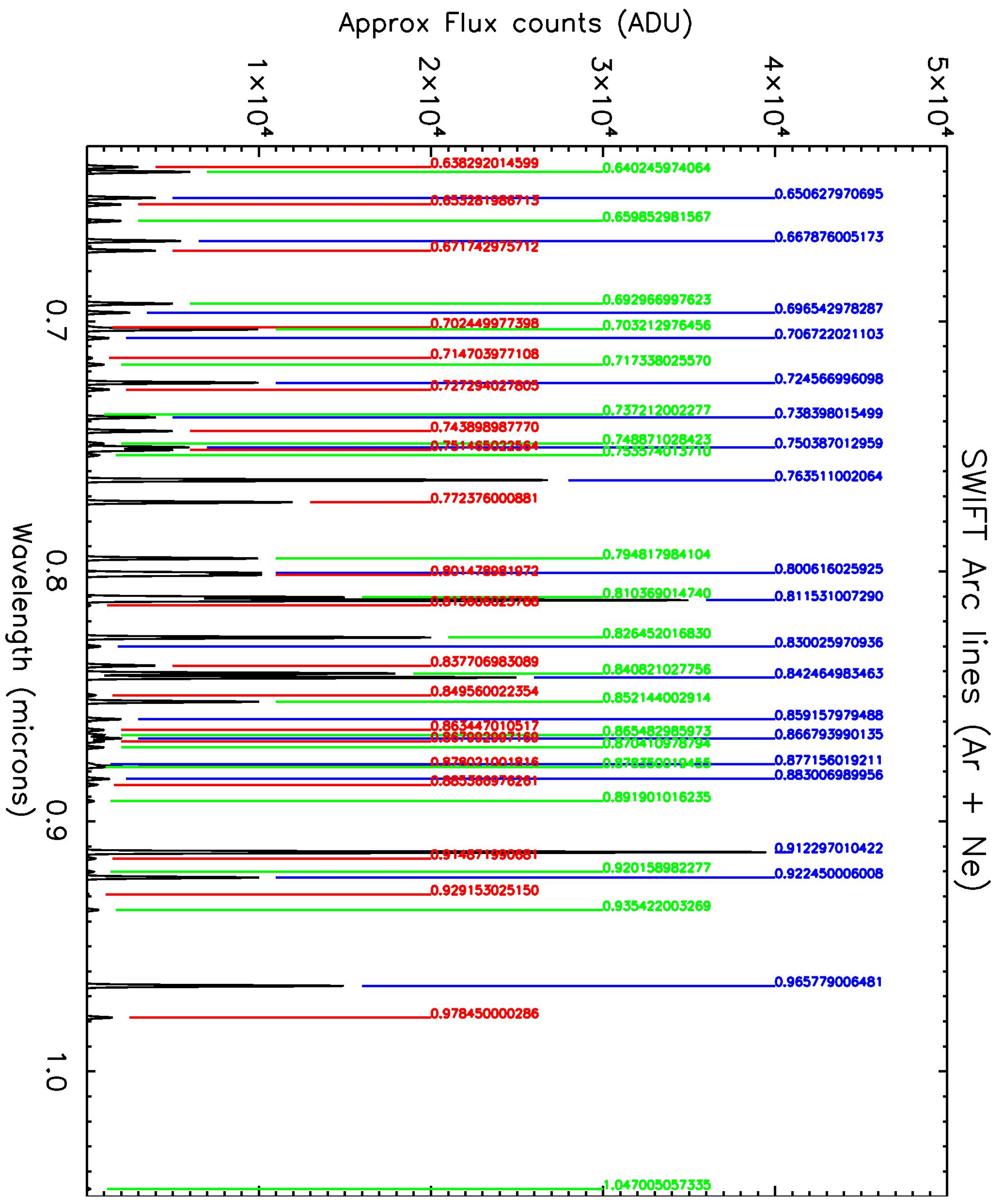


Figure 5.6: A plot of the Argon and Neon arc lines, shown with relative amplitudes and wavelengths. The colours of the lines are just there to aid the eye.

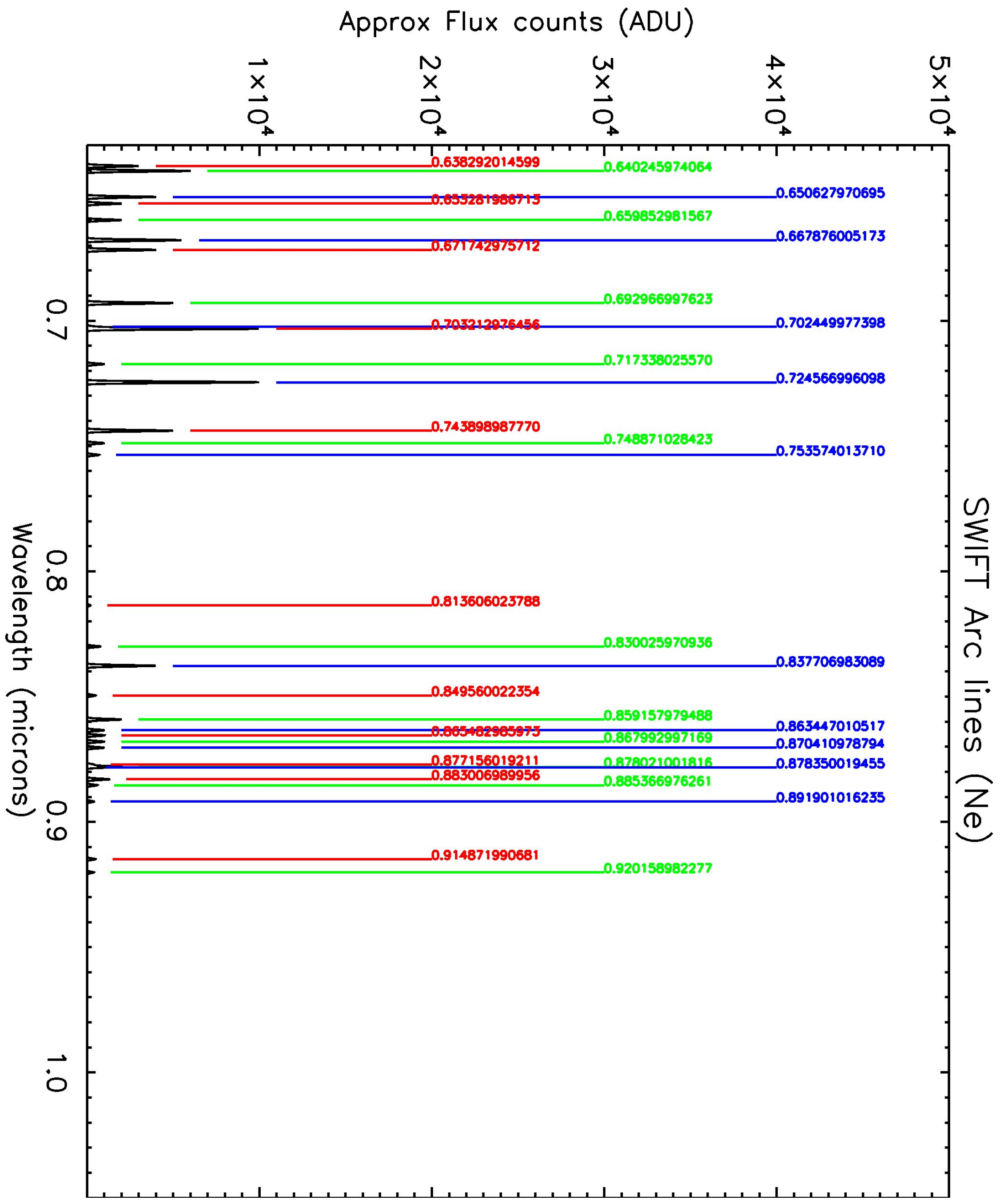


Figure 5.7: Like Fig. 5.6, but just showing the Neon lines.

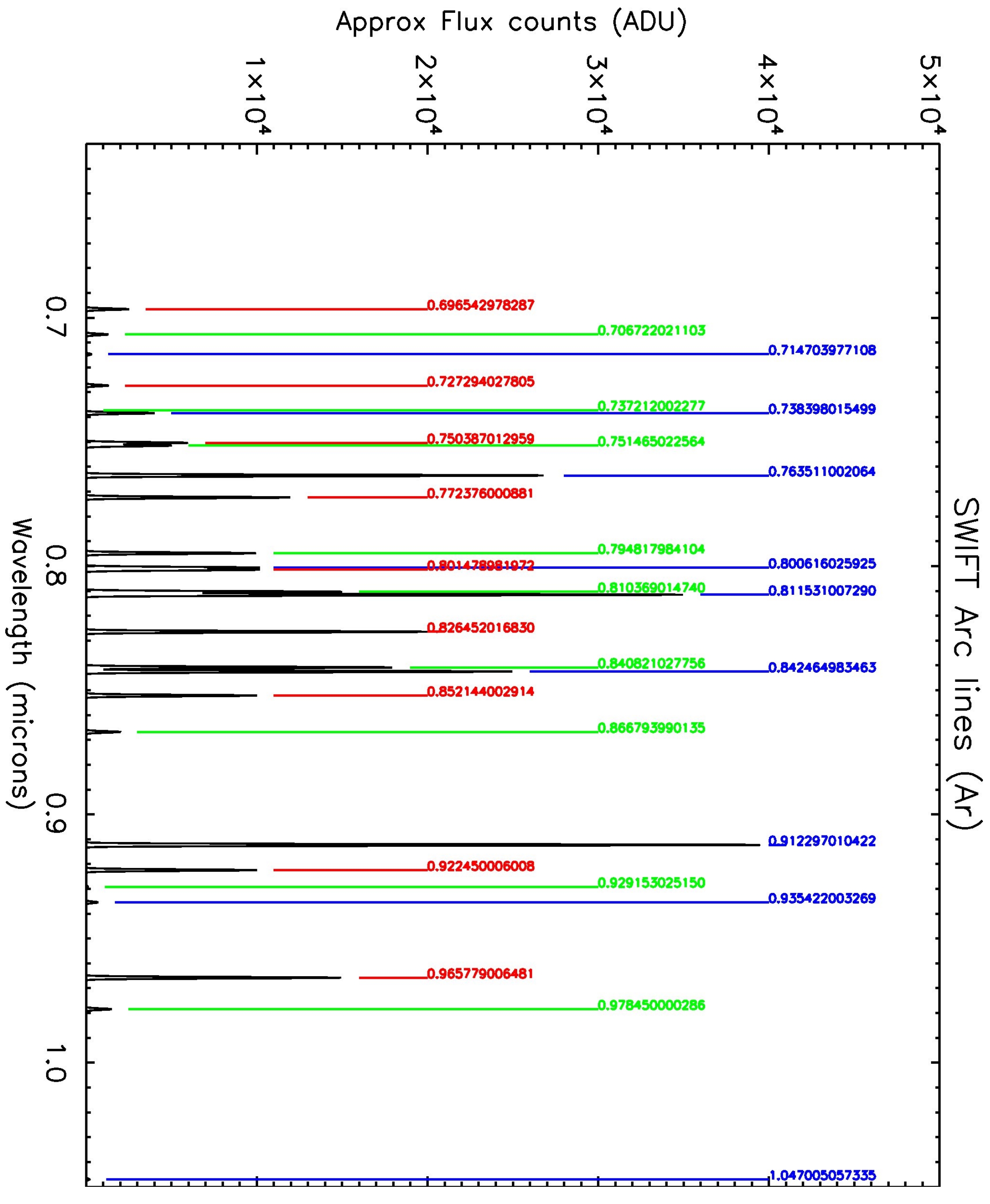


Figure 5.8: Like Fig. 5.6, but just showing the Argon lines.

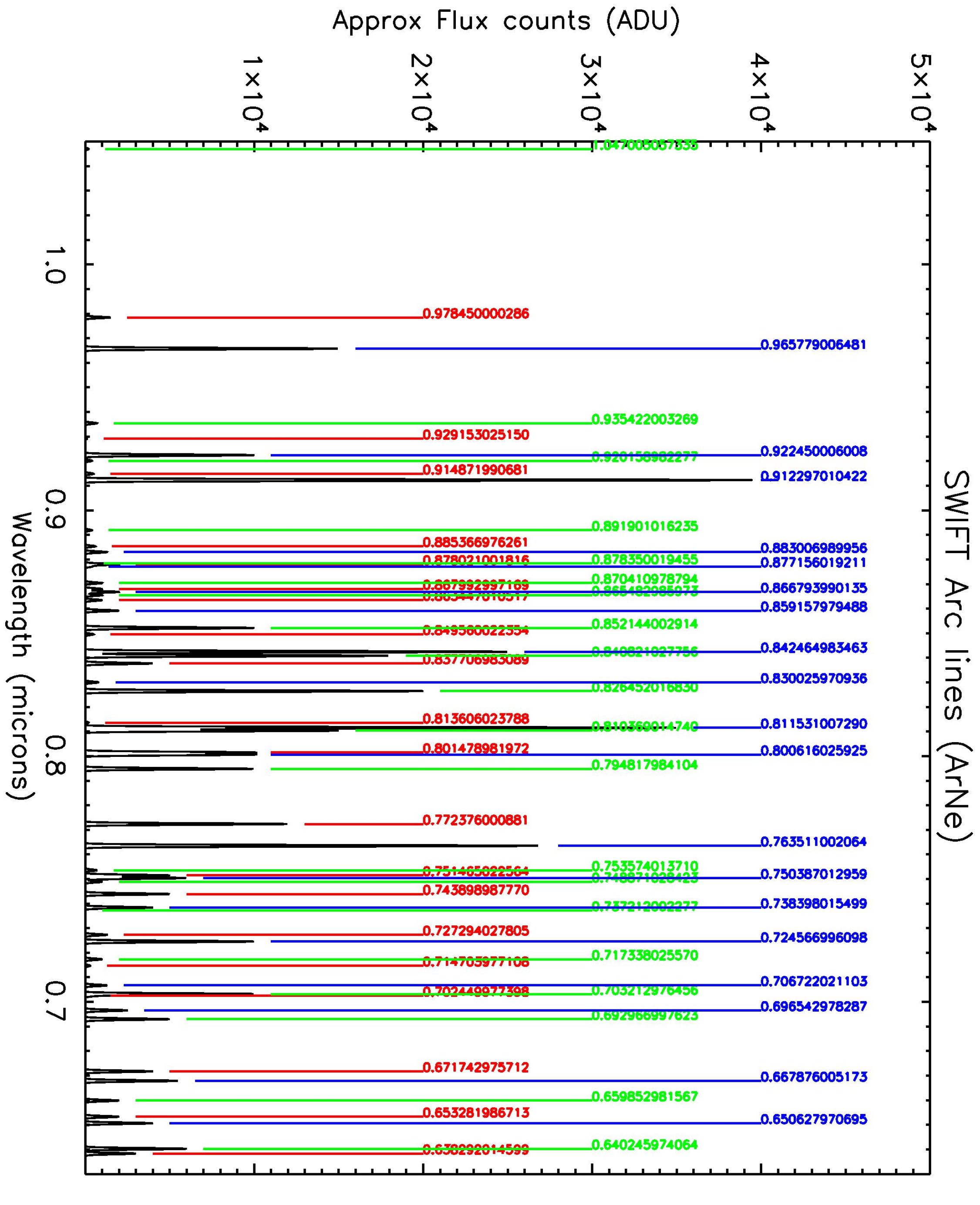


Figure 5.9: Like Fig. 5.6, but with a reversed wavelength axis.

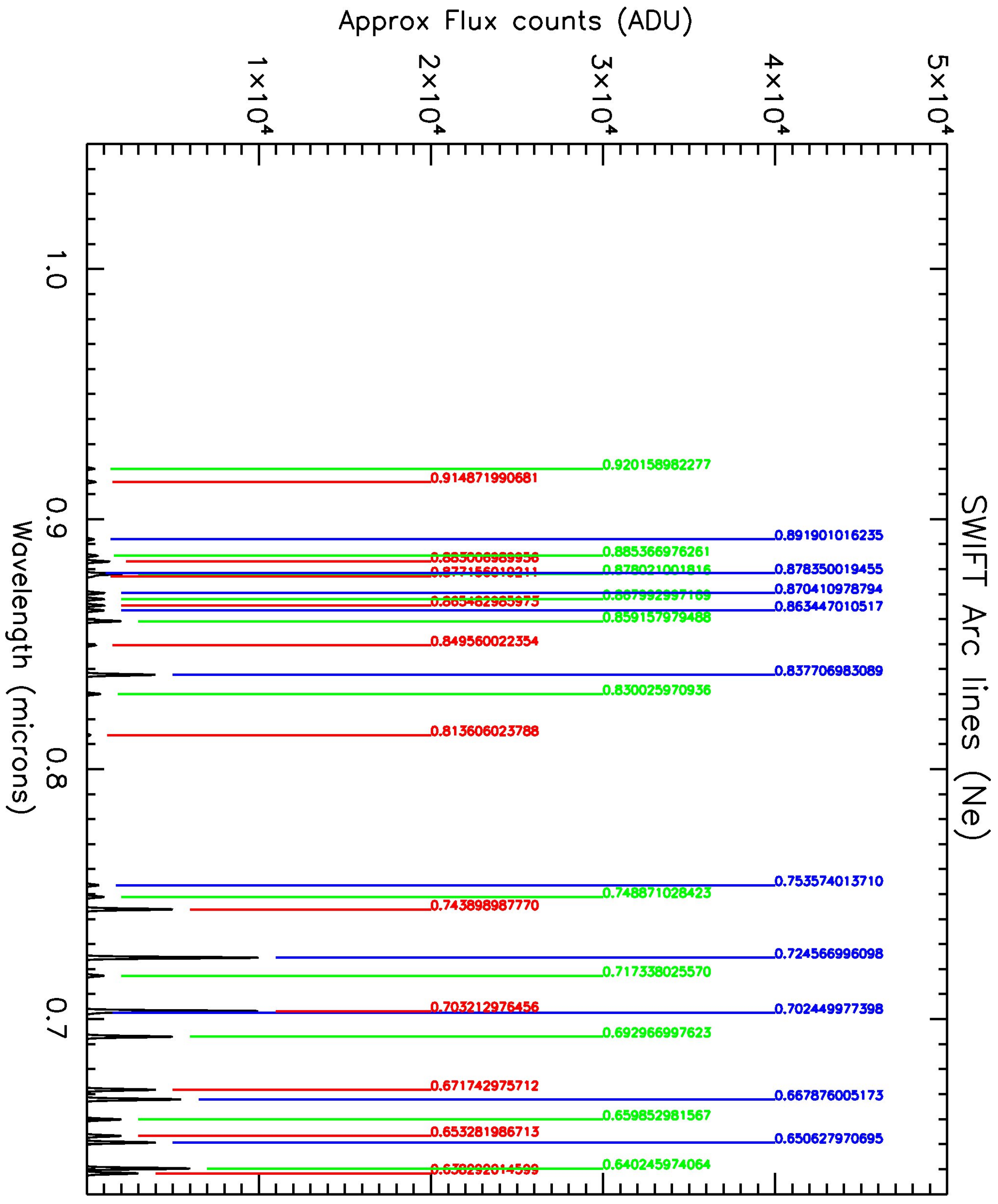


Figure 5.10: Like Fig. 5.7, but with a reversed wavelength axis.

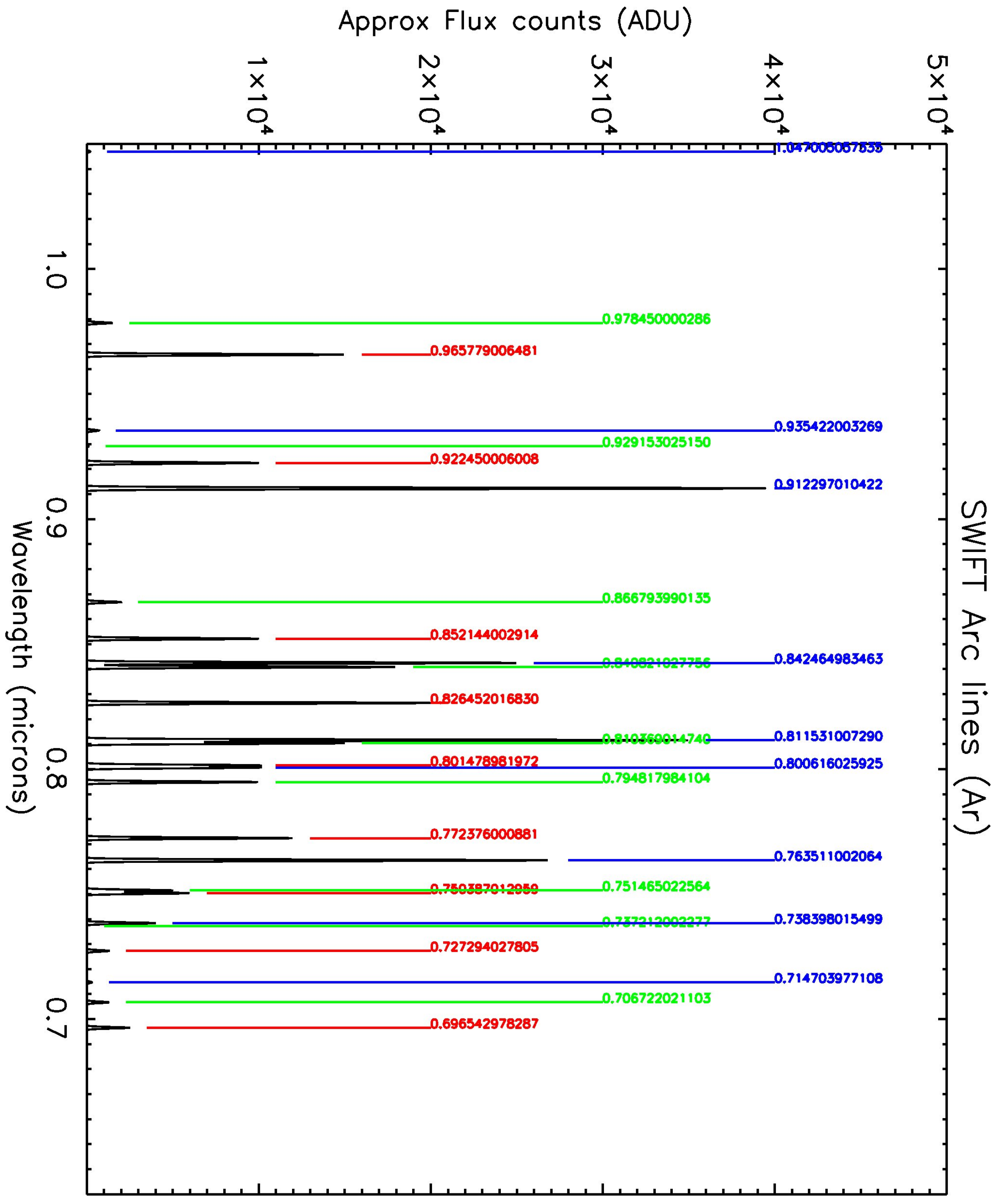


Figure 5.11: Like Fig. 5.8, but with a reversed wavelength axis.

loglam ()
: If you're going to extract kinematics from the spectra, you will need to rebin the spectra onto a logλ wavelength axis; why not save yourself some time (and an extra interpolation stage) by insisting the data is transformed onto a logλ grid here?

trans_interp (nearest | linear | poly3 | poly5 | spline3)
: Here you can specify the type of interpolation for the *transform.* Only certain methods are accepted.

verbose (yes | no)
: Select this option if you want a verbose output printed to the screen.

5.4.1 Description

Again, as discussed previously, it's always wise to create and inspect the calibration cubes (the vertical line frame and the arc frame) using something like QFitsView. But you have to create them first. With **S_CALCUBES** set to **yes**, *swiftred* interpolates the arc and vertical line traces onto a regular grid (saving the 22 interpolated strips in MEFs with a suffix *.tstrip.fits*) and stacks them along the second spatial dimension to make cubes (with suffixes *.cube.fits*).

When you inspect the vertical trace cube, the vertical lines should appear vertical and be well aligned (to better than a pixel accuracy, at least where the PSF is good) as shown in Fig. 5.12. When you inspect the arc frame, all the arc lines in each spaxel should be aligned to much better than a pixel (actually, better than a fifth of a pixel).

5.5 sw_corhoriz

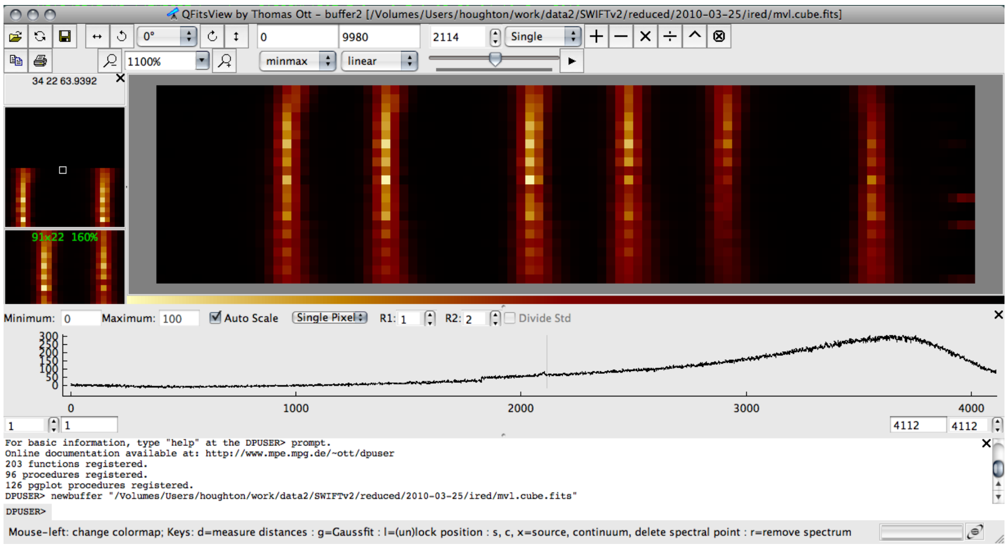
htrace_frame ()
: Give the prep'd horizontal trace frame here.

vtrace_frame ()
: Give the prep'd vertical trace frame here. You should have already run *sw_exvl* on this frame.

HLcoordlist ()
: A list of coordinates (arcsec, one each line) for the horizontal lines (defines the spatial scale along y). **NOTE:** this will need to be changed for the 80mas and 16mas scales.

VLcoordlist ()
: A list of coordinates (arcsec, one each line) for the vertical lines (defines the spatial scale along x). **NOTE:** this will need to be changed for the 80mas and 16mas scales.

arcfilelist ()
: This is passed straight from *sw_arccal*. See the entry there (§5.3) for more info.



```
For basic information, type "help" at the DPUSER> prompt.
Online documentation available at: http://www.mpe.mpg.de/~ott/dpuser
203 functions registered.
96 procedures registered.
126 pgplot procedures registered.
DPUSER> newbuffer "/Volumes/Users/houghton/work/data2/SWIFTv2/reduced/2010-03-25/ired/mvl.cube.fits"
DPUSER>
```

Mouse-left: change colormap; Keys: d=measure distances : g=Gaussfit : l=(un)lock position : s, c, x=source, continuum, delete spectral point : r=remove spectrum

Figure 5.12: Using QFitsView we can see if the vertical traces have been extracted and aligned correctly: in this figure, they all look good (aligned to better than a pixel width). Note that due to slight misalignment with the mask, the right-most vertical line is not seen.

cut_low ()
: This is the number of pixels to cut left of central vertical trace (as described in §5.2, and should be the same as the value given there).

cut_high ()
: This is the number of pixels to cut right of central vertical trace (as described in §5.2, and should be the same as the value given there).

scale ()
: The spaxel scale in arc seconds. **NOTE:** this will need to be changed for the 80mas and 16mas scales.

lam0 ()
: The input data is transformed to a regular wavelength grid: this is the starting wavelength for that grid.

dlam ()
: This is the wavelength interval between pixels along the spectral direction.

nlam ()
: This is the total number of pixels along the wavelength axis after transformation.

loglam ()
: Select this option to output the wavelength axis in logλ units.

sflip ()
: For SWIFT data, the spectral axis is flipped (long wavelengths on the left). Hence this flag is usually set to **yes**.

x1 ()
: This is the initial x coordinate for the output science cube (in arc seconds, master and slave combined), which is usually half of the longest dimension (-11" for the 235mas scale). **NOTE:** this will need to be changed for the 80mas and 16mas scales.

y1 ()
: This is the initial y coordinate for the output science cube (in arc seconds, master and slave combined), which is usually half of the shortest dimension (-4.4" for the 235mas scale). **NOTE:** this will need to be changed for the 80mas and 16mas scales.

onx ()
: Give the number of spaxels along the x dimension for the output science cubes (master and slave combined).

ony ()
: Give the number of spaxels along the y dimension for the output science cubes (master and slave combined).

objcubesuffix ()
: The suffix for the object cubes.

bpmcubesuffix ()
: The suffix for the bad pixel cubes.

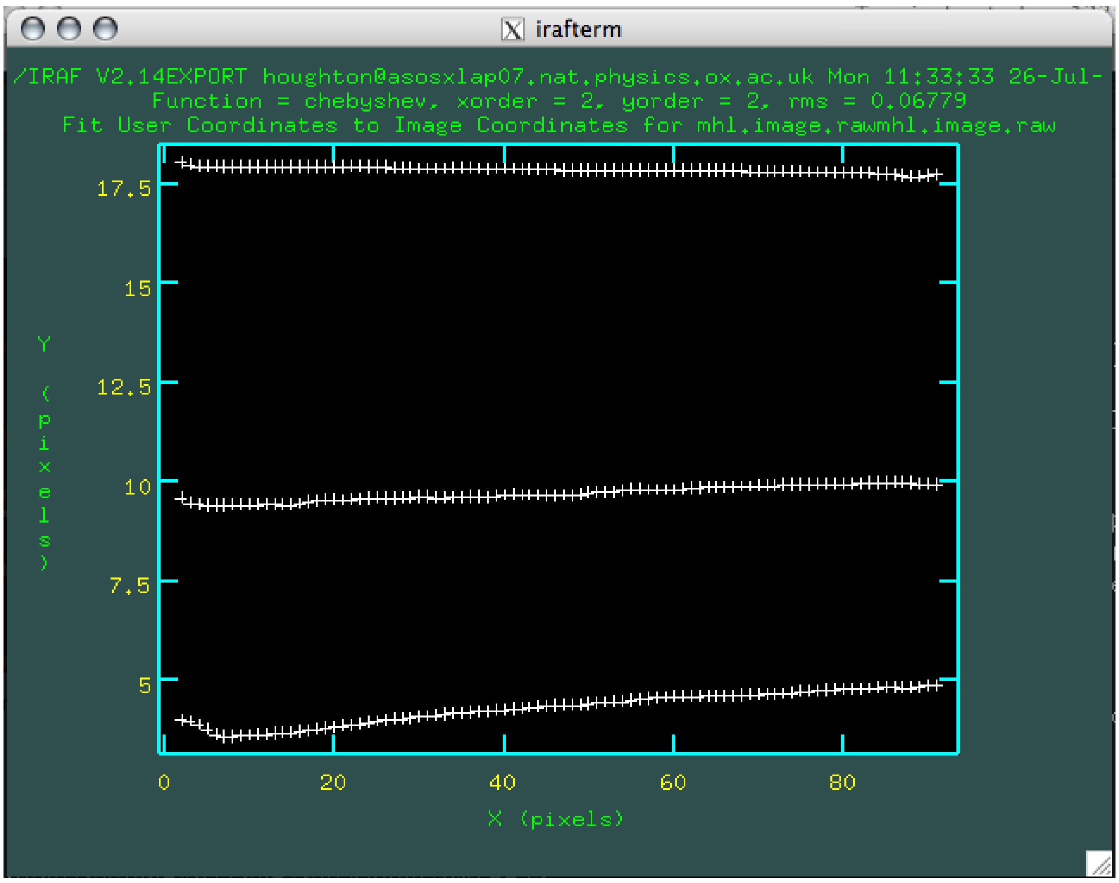
trans_interp (nearest | linear | poly3 | poly5 | spline3)
: Here you can specify the type of interpolation for the *transform*. Only certain methods are accepted.

interfit (yes | no)
: Select this to make the *fitcoords* stage interactive.

interact (yes | no)
: If selected, nearly all the other stages involved in this procedure that can be interactive, will be interactive.

5.5.1 Description

This procedure corrects the SWIFT field of view (FoV) for its non-rectangular shape. Using frames taken with the horizontal mask in place (to make horizontal lines), this procedure reduces (extracts the slitlets, wavelength calibrates and creates a cube) the horizontal line frame and then collapses along the wavelength axis to create an image



```
/IRAF V2.14EXPORT houghton@asosxlap07.nat.physics.ox.ac.uk Mon 11:33:33 26-Jul-
          Function = chebyshev, xorder = 2, yorder = 2, rms = 0.06779
    Fit User Coordinates to Image Coordinates for mhl.image.rawmhl.image.raw
```

Figure 5.13: An image of the horizontal lines (in the master chip), as located and traced by *reidentify*.

Figure 5.14: An image of a (master) frame created using the horizontal mask and having been collapsed along the wavelength direction; note that the horizontal lines are not exactly horizontal (the field of view is not rectangular, but is warped); this is particularly evident at the bottom of the frame.

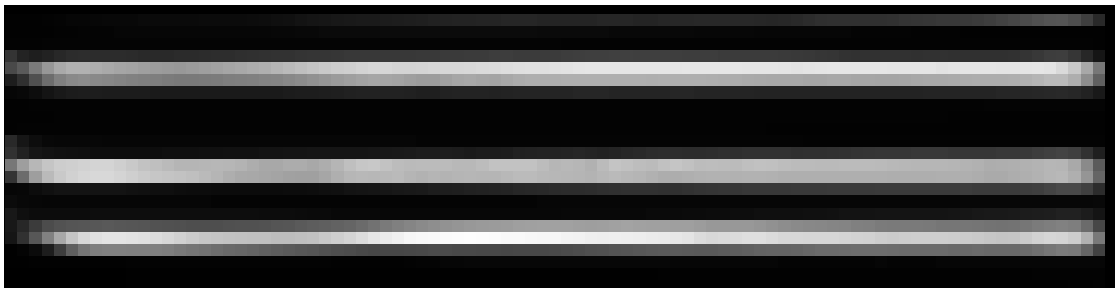


Figure 5.15: This image is again of the horizontal line frame, like in Fig. 5.14 (collapsed along the wavelength axis) but each plane in wavelength had been transformed onto a regular grid to make the field of view rectangular and unwarped.

of the horizontal lines. It then uses *identify*, and *reidentify* to trace the horizontal lines and applies *fitcoords* to fit a low order polynomial to the resulting trace locations; this can be seen in Fig. 5.13 which shows the points identified along the horizontal lines. Finally the *transform* procedure is used to correct, or unwarp the FoV onto a regular grid; comparing Figs. 5.14 and 5.15, one can see the effects of the transform on the horizontal line cube.

At present, it is necessary to manually identify the horizontal traces by moving the mouse over them, clicking on **m**; you will then need to enter the location of the line - use the pixel coordinate at the bottom left of the screen. In the near future, this step will be automated.

5.6 sw_sflex

arc_frame ()

Please specify a single prep'd arc frame here.

sci_input ()

Please give a single/REGEXP/filelist of input science file(s) here.

sci_output ()
Please give single/%a%b%c/filelist of output file(s) here. Note the files must be different to the input.

arc_sec ()
Specify a suitable range to isolate a strong arc line in odd slitlets. If you display the arc frame, this region should highlight the majority of arc lines in alternating (odd) slitlets. See Fig. 5.16 for an example.

sky_sec ()
A suitable range to isolate a sky line in odd slits

max_shift ()
Give maximum pixel shift to search for

interp ()
Interpolation type for (spatial) flexure correction

rmobject (yes | no)
Fit Legendre polynomials to remove the object spectra? (yes/no)

order ()
Give the polynomial order for fitting and removing object spectra ($1 < \text{order} < 4$ recommended).

medslice (yes | no)
Median filter the summed slices to remove cosmics / residuals? (yes/no)

xwin ()
The median window to remove cosmic ray hits and make the cross-correlation cleaner.

arclam ()
If *arc_sec* is set to **auto**, then you need to give the wavelength of a suitably exposed, bright arc line here. $\lambda = 0.9658$ is usually good.

skylam ()
If *sky_sec* is set to **auto**, then you need to give the wavelength of a suitably exposed, bright sky line here. $\lambda = 0.8830$ is usually good.

lam0 ()
Give the wavelength of the first (spectral) pixels here.

dlam ()
Give the wavelength increment per pixel.

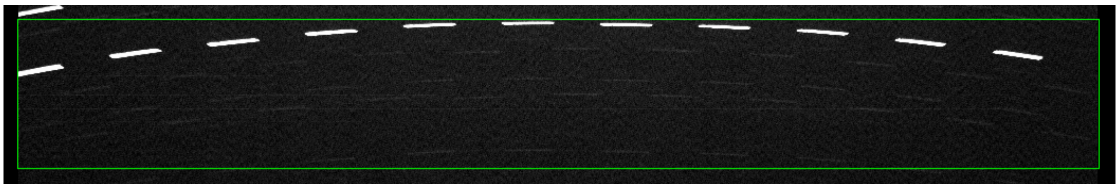


Figure 5.16: An example of a suitable *arc_sec* is shown in green: see how a bright arc emission line has been isolated in alternating (odd) slitlets.

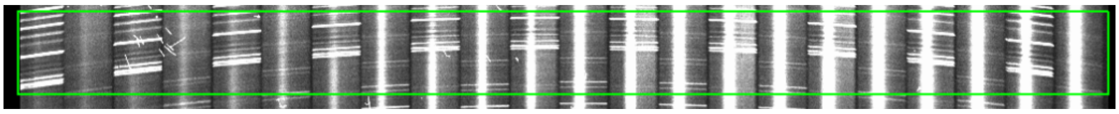


Figure 5.17: Like Fig. 5.16, we highlight a region which isolates emission lines in alternate slitlets; however, this time, the emission lines are sky lines. Note it is difficult to clearlyisolate a single skyline in alternate slitlets, but isolation of a group of bright lines is feasible. The bright vertical line passing through the slitlets is the object spectrum.

nlam ()
: Give the total number of spectral pixels in the spectra.

sflip (yes | no)
: If you prep'd the files with the new pipeline, the spectral axis will be reversed compared the the norm, so set this to **yes**; if you used the old pipeline, set this to **no**.

verbose (yes | no)
: Print verbose output?

5.6.1 Description

This procedure attempts to remove the spatial component of the flexure in SWIFT. When the telescope moves across the sky, stresses in the instrument create flexure of components and cause the image on the detector to move by a few pixels; this motion can be in the spectral direction or in the spatial direction with the end result that subsequent exposures (lasting > 5 mins) may not align with each other on the detector. This causes problems for sky subtraction (see §5.8) and spatial alignment. To combat the latter, here we attempt to cross-correlate a signal from the science exposure (created from sky lines) with one from the arc exposure (created from arc lines which were taken while the telescope is pointing at zenith, like all calibrations). By doing so, we can estimate the shift in the spatial direction and subsequently correct for it with a nearest neighbour interpolation, avoiding smoothing the data (although the user can specify the type of interpolation with *interp*, **nearest** neighbour is recommended for the aforementioned reason).

The first stage in the process is to isolate a signal (emission lines) in alternating slitlets; by then summing over the spectral direction, we get a 1D image of the slitlet edges in both the arc and the science frame (which resemble histograms where alternate bins have alternate high and low levels).

This routine now has the ability to automatically select a box in the raw frames which brackets a single emission line, like in Fig. 5.16 by using *sw_sellinereg* (§5.14). Set *arc_sec* and *sky_sec* to **auto** if you want to use this ability (recommended, because it adjusts the boxes for every input image using the wavelength calibrations). But make sure you set *arclam* and *skylam* to sensible values, or leave them at the defaults. You also need to give *lam0*, *dlam* and *nlam* and *sflip*.

Prior to summing over the spectral dimension, it may help to fit and remove any object spectrum in the science slitlets: this is achieved with *rmobject* which fits a low order polynomial along each column in the spectral direction and subtracts it; emission lines are not subtracted because those pixels are rejected in the fitting phase.

We then supersample (by a factor of 10) the two 1D slit images and cross-correlate them. Prior to this stage, cosmic ray hits have not been removed and so can cause difficulty in the cross-correlation. For this reason we allow the 1D slit images to be median filtered prior to supersampling; this aids in removing cosmics and forcing the edges of the slitlets to be sharp; the width of the box filter can be changed with *xwin*. The reason for supersampling is that the cross-correlation peak can then be read to subpixel accuracy.

Finally, we look at the cross-correlation peak (there is more than one, but we isolate ourselves to look for shifts smaller than *max_shift*). Rather than fitting a Gaussian, we just locate the maximum value and use the location of this maximum to calculate the relative shift between the frames; the science frame is then shifted to match the arc frame (and therefore all other calibration frames that were taken at zenith).

5.7 sw_bps (including sw_lacos)

input ()
Please give a single/REGEXP/filelist of input science file(s) here.

output ()
Please give single/%a%b%c/filelist of output file(s) here. Note the files must be different to the input. This file will hold the original data in the first fits extension (having been corrected for static bad pixels and cosmic rays, if desired) and the associated bad pixel mask (BPM) in the second extension (including static bad pixels and cosmic hits if desired).

outmask ()
The output cosmic ray mask.

model ()

The object model subtracted from the frames before La Placian detection of cosmics.

gain ()
: Specify the detector gain here; this is used by La Cosmic but for most cases should be set to 1.0 because the Swift data is scaled by the gain during the *sw_prep* stage.

readn ()
: Specify the read out noise (RON) of the detector here; again, this is used by La Cosmic.

xorder ()
: Specify the order of the polynomial fitted along the x-axis. Before La Cosmic attempts to identify cosmic rays, it needs to subtract off the sky emission lines. To do this it fits a polynomial along every row in the x direction and subtracting the result.

yorder ()
: Specify the order of the polynomial fitted along the y-axis. Before La Cosmic attempts to identify the cosmic rays, it needs to subtract off the object spectra (roughly). It does this by fitting a polynomial along every spaxel in each slitlet and subtracting the result.

sigclip ()
: Specify the detection limit for cosmic rays; this is a La Cosmic parameter, please see information on that algorithm for further information.

sigfrac ()
: Specify the fractional detection limit for neighbouring pixels; this is a La Cosmic parameter, please see information on that algorithm for further information.

objlim ()
: Specify the contrast limit between a cosmic ray and the underlying (object) pixels; this is a La Cosmic parameter, please see information on that algorithm for further information.

niter ()
: This is the maximum number of iterations of the La Cosmic algorithm performed on each slitlet; for further details, please refer to the La Cosmic algorithm.

axis ()
: The dispersion axis.

threshold ()
: If you selected *s_thresh*, you may specify the threshold value (above which all pixels are identified as cosmics) here.

interact (yes | no)
: Select this option to make as many stages as possible interactive.

verbose (yes | no)
: Select if you want a verbose output sent to the screen.

5.7.1 Description

This procedure firstly uses the given static bad pixel mask (BPM) given in *bp_file* to interpolate over known bad pixels; this is to avoid bleeding of these bad pixels into their neighbours in future interpolations. It also attaches the BPM to the output file as an extra extension - thus hereafter, each file contains two extensions: SCI for the data, and BPM for the bad pixel mask.

If asked to use the La Cosmic algorithm, the procedure first splits the input image into slitlets (using information from the *sw_exvl* stage stored in the **database**) and then calls another procedure, *sw_laco* (which is a slightly modified version of the IRAF spectral La Cosmic algorithm available online) on each of the extracted slitlets. Note that the extraction of slitlets is done without interpolation, so as not to blend cosmics into neighbouring pixels. After cosmics have been identified, it corrects those pixels and adds the locations to the BPM (attached as the second fits extension).

Finally, if the user set the *s_thresh* option, the procedure uses the object and sky subtracted frame from La Cosmic and thresholds all pixels above a limit defined by the parameter *threshold*; those pixels are corrected (interpolated over) in the data frame and the locations added to the BPM.

5.8 sw_scicube

inputfile ()
: Give the input file name here (i.e. the file you want to make a cube of). You can use REGEXPs or file lists.

arcfilelist ()
: This variable should be the same as was given in the *sw_arccal* stage: it's a list of filenames corresponding to the slitlets that were extracted (and calibarted) from the arc frame. This file allows the procedure to find the right information in the **database**.

vtrace_frame ()
: Give the vertical trace frame here; you must have run *sw_exvl* on this frame previously (to populate the **database** directory with information on the slitlet locations).

s_flat ()
: Flag to perform flat fielding.

s_illum ()
Flag to perform illumination correction.

s_skylines ()
Select this option to use the skylines to tweak the wavelength solutions. If selected, the procedure will use *identify*, *reidentify* and *fitcoords* to wavelength calibrate each input frame and then transform the data onto a regular grid. Currently, this option adds another interpolation into the reduction process, although in the future it is hoped that this stage will be combined with the initial transformation using information from the arc frames during *sw_arccal*.

makeskyframe ()
Flag to save model of sky emission.

flat_frame ()
Tag for flat calibrations.

illum_frame ()
Tag for illumination cubes.

skyref_frame ()
The name of a file to use for spectral flexure correction instead of the file being processed (should have already been reduced in this did).

sky_list ()
If using the *s_skylines* option to correct for spatial flexure, you should give the skyline list here (used to identify the skylines).

cut_low ()
Number of pixels to cut left of central vertical trace. Together with *cut_high* below, these two numbers, added together, make up the size (width) of each slitlet. This should be set to the same value used in *sw_exvl*.

cut_high ()
Number of pixels to cut right of central vertical trace. This should be set to the same value used in *sw_exvl*.

x1 ()
The starting spaxel for the transform (units of pixels).

lam0 ()
The input data is transformed to a regular wavelength grid: this is the starting wavelength for that grid.

dlam ()
This is the wavelength interval between pixels along the spectral direction.

nlam ()
: This is the total number of pixels along the wavelength axis after transformation.

loglam ()
: When extracting kinematics (velocities, dispersions etc), data is often required on a log-lambda (wavelength) grid. To avoid having to interpolate the data again, the user can specify that the initial interpolation be on a log-lambda grid here.

skyxorder ()
: If the *s_skylines* stage is selected, you can specify the order used in *fitcoords* along the wavelength axis here. Note that the y-order is not ammendable; this is because flexure does not alter the y-order determined from the arc frames in *sw_arccal* and to minimise the number of free parameters (thereby better constraining the fit), we keep the y-order to 1.

scale ()
: This is the spaxel scale of the input (and all calibration) data. This is used to create a suitable world coordinate system in the cube headers. In the future, it may be possible to calculate this value from the vertical mask frame in *sw_corhoriz*.

objcubesuffix ()
: The suffix given to output object cubes.

bpmcubesuffix ()
: The suffix given to output bad pixel / cosmic cubes.

sflex_interp ()
: The type of interpolation for the SPATIAL flexure correction.

trans_interp (nearest | linear | poly3 | poly5 | spline3)
: Specify the type of interpolation for all *transform* stages.

sflip (yes | no)
: The SWIFT spectra are reversed to the normal direction in IRAF; as such, if you use the *S_PREP* stage to prepare the frames, you also need to have this flag set to **yes**. However, if you used the *old python pipeline* to prep the input frames, you can use this pipeline to reduce them if this parameter is set to **no** (and, naturally, you should not have prep'd the files with *sw_prep*; see §7).

CASS_DELTA ()
: The difference (in degrees) between the Cassegrain ring angle and the SWIFT position angle (longest axis along the east-west, master on the north side). This is used to define the world cordite system of the output cubes. Note this number changed after the 2010 intervention; post-2010, it is **-100.0**

instrument ()
: For SWIFT data, leave this set to **1**.

interact (yes | no)
Use this option to make all possible stages interactive.

verbose (yes | no)
Select to print verbose output to the screen.

5.8.1 Description

This procedure is the bread and butter of the Swift pipeline. Once all calibrations have been processed with the earlier routines (*sw_exvl* and *sw_arccal* are essential; *sw_corhoriz* , *sw_sflex* and *sw_bps* are optional), this procedure uses the information obtained from the calibrations to create 3D cubes from the 2D input images.

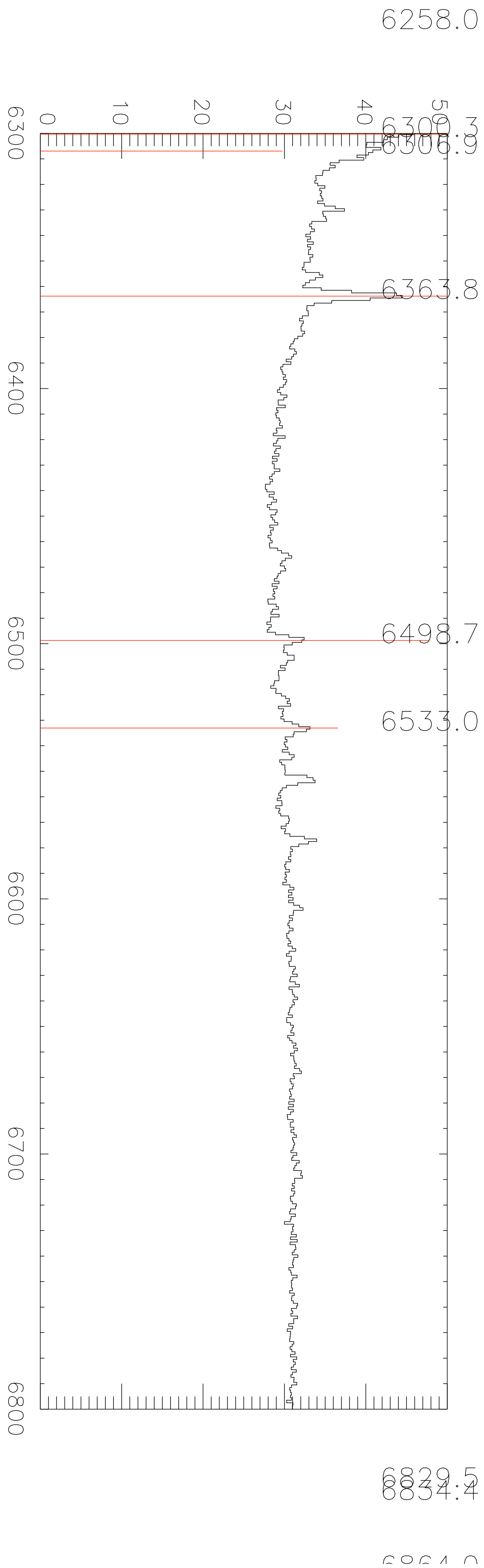
The procedure starts by extracting the slitlets using the information gained in *sw_exvl*; note that if you used *sw_bps* on the input frames, it also extracted slitlets from the BPM. It then transforms the data (and BPMs) onto a regular grid using the information from *sw_arccal.* Next, if the *s_skylines* option was selected, the procedure uses the linelist given in *sky_list* to identify the skylines in the input frames. Note that if there are no skylines present in the data (e.g. the exposure time was <5 mins) then this stage will fail. Rather than identifying the skylines in the actual data, the procedure fits a low order polynomial along the spatial axis at every wavelength (for every slitlet) and then identifies lines in that frame; the advantage is the removal of noise and object spectra. It then uses *reidentify*, *fitcoords* and *transform* in the usual manner to remove the spectral flexure (the user can specify the order of the fit along the wavelength axis used in *fitcoords* with *skyxorder*).

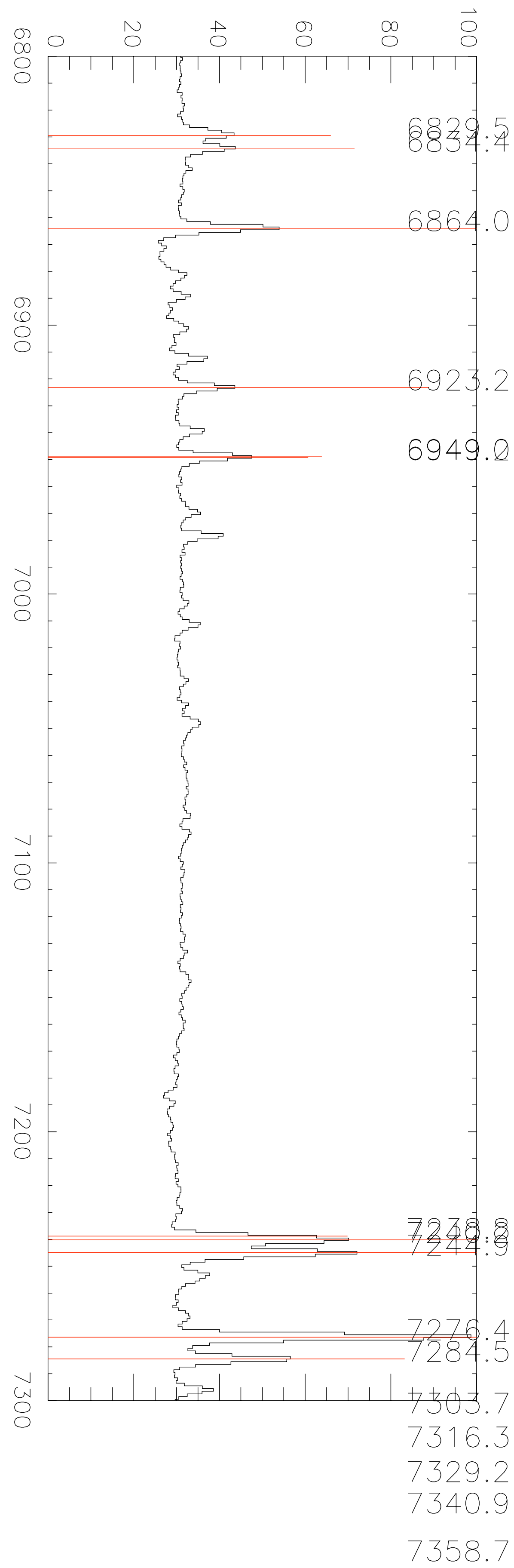
Next, we create cubes of the (transformed) data (and associated BPMs). Then, if the user specified a horizontal line frame in *htrace_frame*, the procedure will use the information in the **database** to correct the input data (and BPMs) for the warped FoV in SWIFT (see §5.5). Note that you must have already run *sw_corhoriz* on the horizontal line frame otherwise this option will fail (as will the whole *sw_scicubes* procedure).

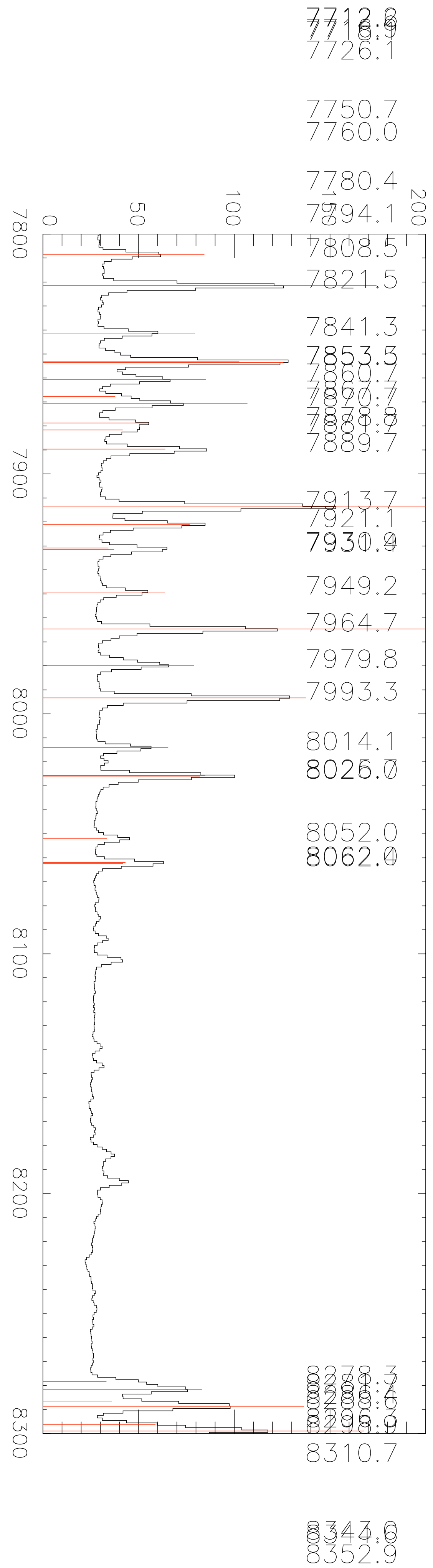
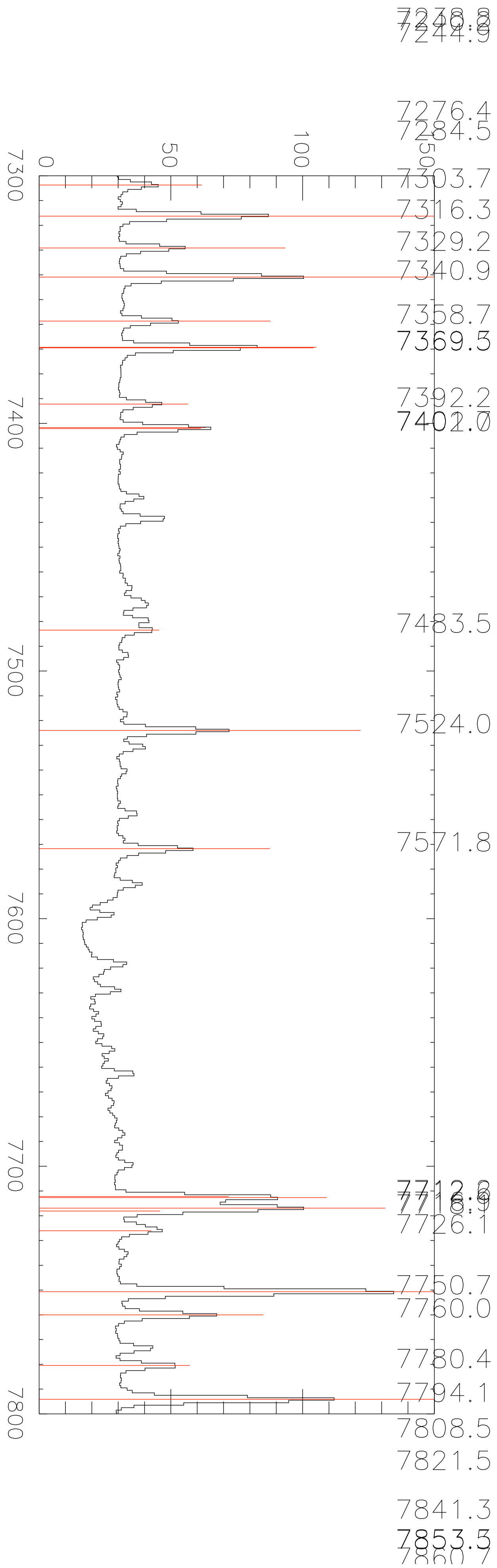
Finally, the output cube is saved to disk; it's named after the input frame, but the suffix **.sci.cube** is added just before the **.fits**. Similarly, if the input frame had been processed with *sw_bps*, the BPM cube is given the same name as the input frame but with the suffix **.bpm.cube**.

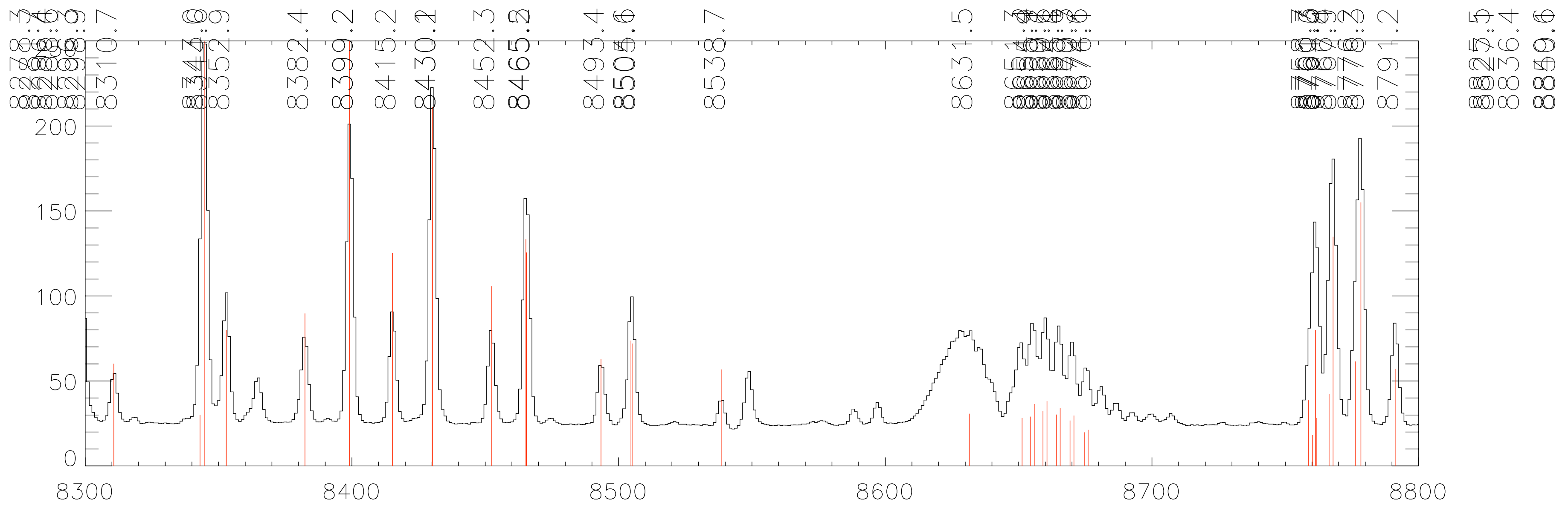
5.8.2 Skyline identifications

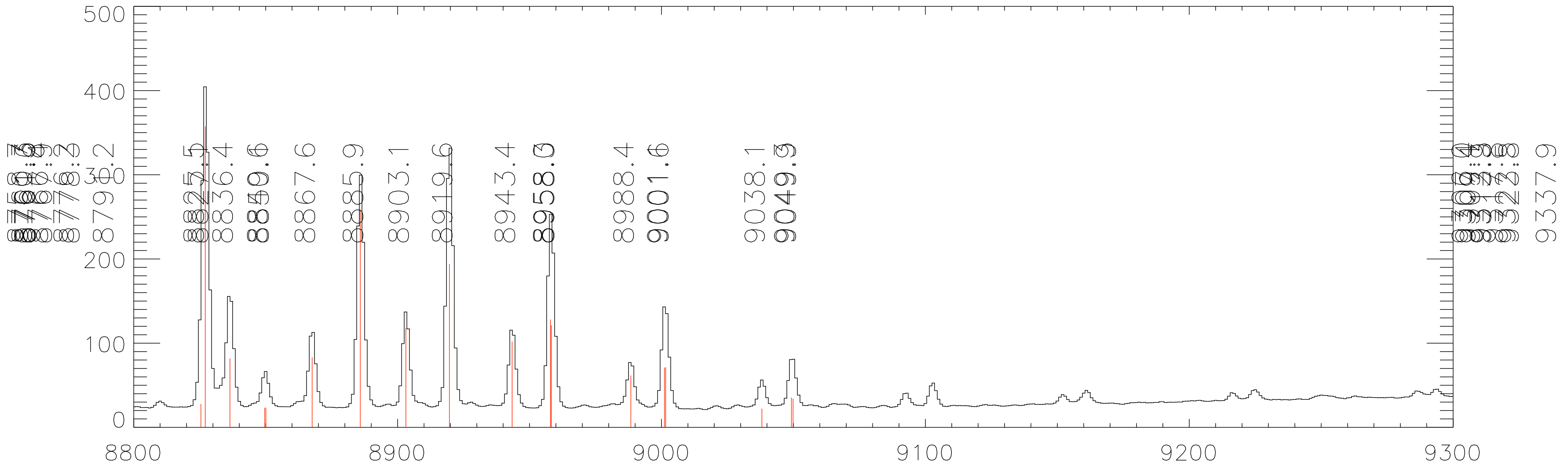
If you have trouble automatically identifying the sky lines, you could try tweaking *sw_lskyident*, described in §5.13, or you could identify some of the lines manually using the following plots kindly generated by Niranjan Thatte; no need to use them all, about 5 to 10 should do if you spread them over the full wavelength range.

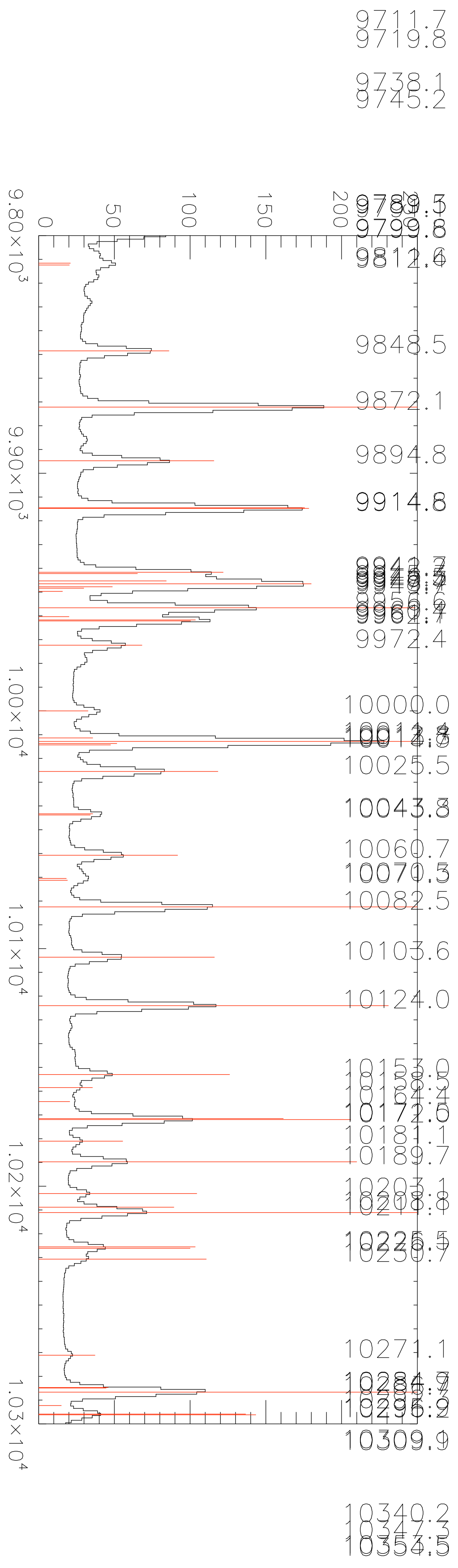
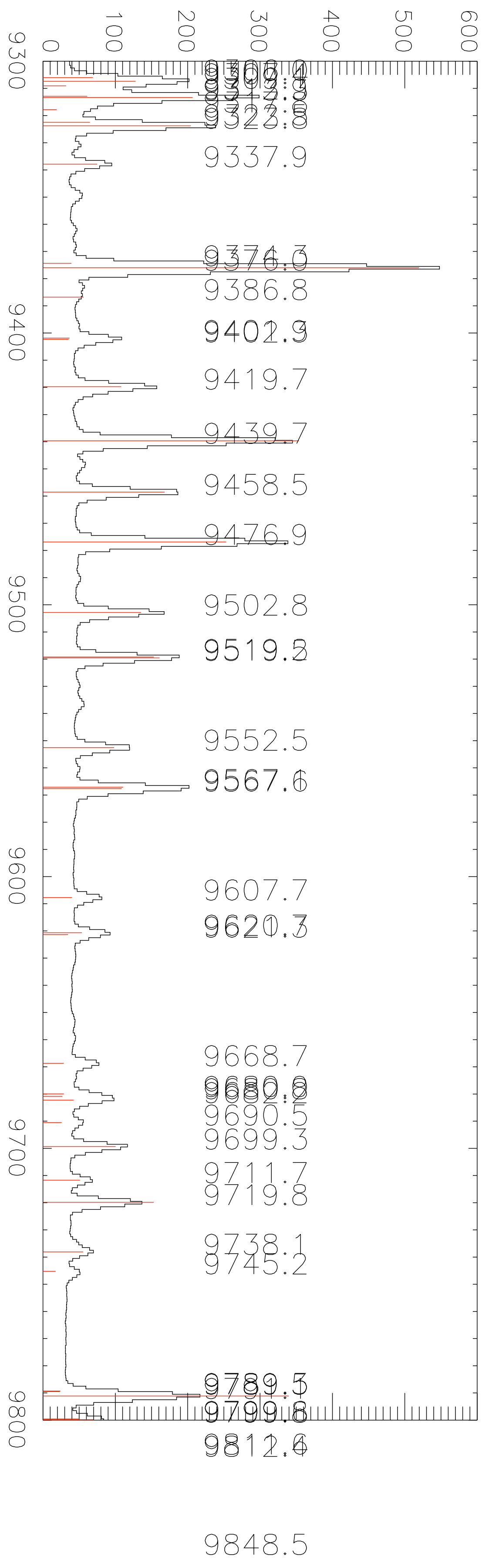

6258.0
6300.3
6306.9
6363.8
6498.7
6533.0
6829.5
6834.4
6300
6400
6500
6600
6700
6800
0
10
20
30
40
50


6829.5
6834.4
6864.0
6923.2
6949.0
7238.8
7240.0
7244.9
7276.4
7284.5
7303.7
7316.3
7329.2
7340.9
7358.7
6800
6900
7000
7100
7200
7300
0
20
40
60
80
100


7800
7900
8000
8100
8200
8300
0
50
100
150
200
7712.6
7716.9
7726.1
7750.7
7760.0
7780.4
7794.1
7808.5
7821.5
7841.3
7853.5
7860.7
7870.7
7889.7
7913.7
7921.1
7931.4
7949.2
7964.7
7979.8
7993.3
8014.1
8026.7
8052.0
8062.4
8279.3
8310.7
8343.0
8352.9

7300
7400
7500
7600
7700
7800
0
50
100
150
7238.8
7244.9
7276.4
7284.5
7303.7
7316.3
7329.2
7340.9
7358.7
7369.5
7392.2
7401.7
7483.5
7524.0
7571.8
7712.6
7716.9
7726.1
7750.7
7760.0
7780.4
7794.1
7808.5
7821.5
7841.3
7853.5
7860.7


8310.7
8343.6
8352.9
8382.4
8399.2
8415.2
8430.2
8452.3
8465.2
8493.4
8505.6
8538.7
8631.5
8791.2
8825.5
8836.4
8849.6
0
50
100
150
200
8300
8400
8500
8600
8700
8800


8791.2
8825.5
8836.4
8849.6
8867.6
8885.9
8903.1
8919.6
8943.4
8958.0
8988.4
9001.6
9038.1
9049.3
9337.9
0
100
200
300
400
500
8800
8900
9000
9100
9200
9300


9711.7
9719.8
9738.1
9745.2
9789.5
9799.8
9812.6
9848.5
9872.1
9894.8
9914.8
9972.4
10000.0
10013.4
10025.5
10043.8
10060.7
10070.3
10082.5
10103.6
10124.0
10153.0
10172.6
10181.1
10189.7
10203.1
10218.8
10226.5
10250.7
10271.1
10284.7
10296.9
10309.1
10340.2
10347.3
10354.5
9.80×10^3
9.90×10^3
1.00×10^4
1.01×10^4
1.02×10^4
1.03×10^4
0
50
100
150
200

9305.4
9322.8
9337.9
9374.3
9386.8
9401.9
9419.7
9439.7
9458.5
9476.9
9502.8
9519.5
9552.5
9567.6
9607.7
9620.7
9668.7
9680.8
9690.5
9699.3
9711.7
9719.8
9738.1
9745.2
9789.5
9799.8
9812.6
9848.5
9300
9400
9500
9600
9700
9800
0
100
200
300
400
500
600

5.9 sw_mkfl

input_frame ()
: Provide a single prep'd flat frame (lamp, dome, twilight or even sky) here.

outillum_frame ()
: Specify the name for the output illumination frame.

vtrace_frame ()
: Give the vertical line frame name here (usually **VL.fits**) that you have already calibrated (see §5.2).

outflat_frame ()
: Specify the name for the output pixel-to-pixel flat field frame; note that if you gave a dome or twilight flat for *input_frame*, you should leave this blank ("").

masterthro ()
: Give the master throughput curve (interpolated onto the same grid as the science data) here. Usually this will be **swift$master_thro.dat** (see §6)

slavethro ()
: Give the slave throughput curve (interpolated onto the output wavelength grid) here. Usually **swift$slave_thro.dat**

chiptype ()
: Specify if all the frames are from the **master** or **slave** chip. This will then be used to apply the relevant throughput correction.

illumspec2d ()
: A file name for the model of the spectrum (used to infer the illumination).

arcfilelist ()
: This is a list of the arc filenames. This is nearly always **arcfiles.txt** and should already exist (it's created by *sw_arccal*).

directfit (yes | no)
: If you want to try and fit the illumination *before* interpolating the input onto a regular grid (e.g. you have a lamp flat as input and you want a pixel-to-pixel flat frame as output you should set this to **yes**). Note that if you have emission or absorption lines in the spectrum of the input frame (e.g. in twilight, dome or sky flats), then this method doesn't work well; it's really only useful for smooth halogen lamp spectra.

medfit ()
: Normally we try and fit the illumination pattern along each spaxel with a high order polynomial or spline. However, you *might* like to try a median filter instead; it can produce quite nice results if the input was a raw twilight or sky exposure.

s_skylines ()
: Do you want to fit skylines in the input to correct for spectral flexure? If you input was taken at zenith (e.g. lamp flat, dome flat and most twilight flats) you won't want to do this (there are no sky lines in a lamp or dome flat anyway!); but if you're trying to create an illumination frame from a sky exposure, you *will* want to set this to **yes**.

sky_list ()
: Give the sky line list here if *s_skylines* is set to **yes**. See §6

cut_low ()
: When fitting the illumination with a high order spline or polynomial, we need to reject sky residuals / cosmics / bad pixels etc. This is the lower limit for sigma-clipping.

cut_high ()
: Like the above, this is the upper limit for sigma-clipping.

lam0 ()
: This is the wavelength of the first pixel in the interpolated (illumination) cube; it should match that of your data.

dlam ()
: This is the wavelength interval in the interpolated (illumination) cube; it should match your data.

nlam ()
: This is the number of pixels in the interpolated (illumination) cube spaxels; it should match that of your data.

loglam ()
: Do you want to interpolate onto a logλ grid? If you did this for your science data, you should do it here. Otherwise, set this to **no**

scale ()
: Give the spaxel space here; it's used to make a header for the output cubes.

funct (spline1 | spline3 | legendre)
: Here you can define the type of function used to fit the illumination pattern along each spectrum. I personally like **spline3**.

order ()
: Specify the order of the polynomial/spline fit here. Orders of 20 to 40 are not silly, but recommended.

sample ()
: Specify the range (in pixels along the wavelength axis) over which the illumination

pattern will be fitted with polynomials. This helps avoid the spectrum ends which can throw the fit off what it should be; this is particularly true if you used the 750 dichroic but are trying to fit below 0.75 microns.

low ()
Specify the lower limit for sigma clipping in the polynomial/spline fit.

high ()
Specify the higher limit for sigma clipping in the polynomial/spline fit.

niterate ()
Specify the number of sigma clipping iterations here.

grow ()
If a pixel is rejected in the sigma clipping stage, neighbouring pixels can also be rejected and this parameter gives the number of neighbouring pixels that are rejected.

trans_interp ()
Give the type of interpolation here. Use the same method that you used for the science data.

sflip ()
Were the input frames prep'd with this new pipeline? If so, set this to **yes**; if not, set it to **no**.

interact (yes | no)
Use this parameter to make all possible stages (the spline/polynomial fitting) interactive.

verbose ()
Activate this parameter to receive a verbose output to the screen.

5.9.1 Description

Before reading this (and afterwards too), read all the information about the parameter inputs and outputs above; not all the info there is repeated here.

This procedure is the only one that is not included by default in the *swiftred* pipeline. That's because it's very tricky to do, let alone get right, so it's best if the user does this manually (at least for the moment). There are two distinct problems that this procedure aims to combat: flat field correction and illumination correction. Each one is solved by a different input and output, but this procedure can produce both flat field and/or illumination frames to correct science data. Note that some people split this problem into three parts: pixel-to-pixel variations (AKA flat field variations), field-of-view illumination variations and spectral illumination variations (which I group together and call illumination variations).

The basic principals are as follows:

1. To create a pixel-to-pixel flat field frame, we divide out the spectral shape along each spaxel of a (halogen) LAMP flat by fitting and dividing out a high order polynomial or spline (with rejection). The resulting variations (of around 5% will correct for the pixel-to-pixel variations). The smooth fit to the spaxels can also be used to create an illumination cube, but see below.

2. To correct for illumination, we create an illumination cube which must be applied to the science cubes. The usual principal is as follows: first wavelength calibrate the data and make a cube, then divide out the average spectrum in the cube (i.e. a spectrum of the twilight sky, or a spectrum of the illuminated dome surface) and then fit (with a polynomial or spline) the result as this represents the differences in illumination, both across the field-of-view and spectrally. Although I'd love to be able to create a 2D illumination FRAME, rather than an illumination cube, this just isn't possible in IRAF because you need to divide out the average spectrum and to do that, you need to wavelength calibrate (i.e. transform) your data.

We now look at these options in more detail.

Pixel-to-pixel (Flat field) correction

The pixels in a CCD detector do not all respond equally to the same incident amount of flux; each has a slightly different efficiency. In the SWIFT CCDs these variations are at the level of around 5%; thus without correcting for this affect, no matter how long you integrate for you will only ever manage a maximum signal-to-noise (S/N) of 20. If you need to do better than this, you need to divide out these pixel-to-pixel variations using a **flat field** frame; this will then allow to you achieve a S/N $\lesssim$ 300 (the noise limit on the corrected pixel-to-pixel variations). To create a flat field frame, you need (halogen) lamp flats.

If you pass a prep'd (halogen) lamp flat to the *sw_mkfl* procedure (in *input_frame*) and ask for a pixel-to-pixel flat frame (i.e. give a filename for *outflat_frame*), *directfit* is automatically selected to fit and remove (i.e. divide out) the illumination before any interpolation is performed; this is because the pixel-to-pixel flat field needs to be created without any interpolation. The procedure will fit a high order polynomial (or spline) along each spaxel (with sigma clipping) to fit the overall spectral shape (including the illumination). Once the fit to the spectrum has been divided out, you're left with a flat field frame that provides the fractional variations in efficiency of each pixel in the CCD; this frame is returned with the name given in *outflat_frame.*

However, the fit we made to the spaxels is (to some degree) a measure of the illumination (multiplied by the halogen lamp spectrum). Thus we interpolate this fit into a cube and divide out an average lamp spectrum to create an illumination cube, which is returned in *outillum_frame* (see next section).

Thus to perform a "flat field" correction, you should first prep the science frames, then divide all your prep'd science frames by the pixel-to-pixel flat field frame (given by *outflat_frame*) and then make cubes out of these prep'd and flat fielded science frames.

Illumination correction from a (Halogen) Lamp flat

Creating the pixel-to-pixel flat field above has an added side effect: if you specify a filename for *outillum_frame*, then an illumination cube will be created with that name. This illumination cube will also be corrected for throughput differences between the master and slave chips; see the next section for more info.

If you want to try and remove the illumination effects from your science data using the illumination cube created from this lamp flat, you should divide the science cubes by the illumination cube given by *outillum_frame*. However, there may be several complications with this approach, which you need to look out for. Firstly, because of spatial flexture, the illumination pattern in the science data may not be aligned with the illumination pattern in created from the lamp flat; it's up to you to solve this problem. Secondly, the ilumination cube created by the lamp flat (in my optionion) doesn't match that created in the data; I think that dome or twilight flats produce better illumination cubes (see below). But you're free to try whatever you like.

Illumination correction using Dome flats, Twilight flats or even night sky frames

To create an illumination frame similar to that seen in science observations we start by observing a source which will illuminate the FoV uniformly (i.e. one at, or "near" to infinity); the twilight sky is the best option, but dome flats (where the inside of the dome is illuminated with lamps) can also be used as can simple sky exposures. Twilight flats probably produce the best results because the source is bright (unlike the night sky exposures) and nearer infinity than the dome flats (so matching the science illumination better). However, night sky exposures have the advantage that the flexure (and so position of the illumination pattern) is likely to be very similar to your science exposures.

If you pass the prep'd (twilight/dome/sky) frame to this procedure in *input_frame*, it will first create a cube of the data and then divide out the average spectrum in the cube. It will then fit a high order polynomial (or spline) to every spaxel which is then used as a measure of the illumination along that spaxel; the fitting is best done with quite severe (sigma clipping) rejection with *low* and *high* set to around 2, particularly if the input frame was a twilight or dome flat because the polynomial needs to be fitted to the continuum and reject the residuals from emission/absorption features.

This polynomial fit will then be corrected for throughput differences between the master and slave chips; we scale to the average of both the throughput curves given in *masterthro* and *slavethro*. There are a few important points regarding throughput corrections. It is vital that you specify the correct chip in *chiptype* which should be the same chip (master or slave) that corresponds to the input frames and the science frames you with to correct. Furthermore, the throughput curves need to be interpolated onto the same grid as the science data (specified here by *lam0*, *dlam*, *nlam*, *loglam*, *sflip*) before being passed to this procedure.

Finally, the throughput corrected illumination cube is returned with the name given in *outillum_frame* and you can divide your data through by it to remove the illumination pattern. However, you should be aware that flexure may move the illumination pattern in your science data compared to that in the twilight flat; it's up to you to solve this problem.

5.10 sw_flattencube

input ()
: Give the input cube here (single file, REGEXP or filelist). Make sure the data is indeed a 3D cube.

output ()
: Give an output file name here (single file, %in%out%remainder, or ∧prefix).

verbose (yes | no)
: Activate this parameter to see a verbose output printed to the screen.

5.11 sw_unflattencube

input ()
: Give the input cube here (single file, REGEXP or filelist). Make sure the data is indeed a 2D image made with *sw_flattencube*.

output ()
: Give an output file name here for the 3D output (single file, %in%out%remainder, or ∧prefix).

nx ()
: The number of spaxels along x in the output cube.

ny ()
: The number of spaxels along y in the output cube.

verbose (yes | no)
: Activate this parameter to see a verbose output printed to the screen.

5.11.1 Description

This routine is the reverse of *sw_flattencube*; it converts a 2D image (created by *sw_flattencube*) into a 3D cube. Note that the 2D→3D conversion should be the direct inverse of the flattening process in *sw_flattencube*, but is not the same as *sw_scicube* or *sw_calcube*, which calibrate, interpolate and resample *RAW* data into cubes.

5.12 sw_larcident

sflip (yes | no)
If you prep'd the files with this pipeline, leave this as **yes**; else if you prep'd with the old pipeline, set this to **no** (and see §7).

arc_list (ArNe.dat | Ar.dat | Ne.dat | userlinelist.dat)
Give the arc line list here.

crval ()
Give the coordinate reference value here (wavelength at CRPIX). Usually you don't need to edit this.

cdelt ()
Give the coordinate delta value here (wavelength change per pixel).Usually you don't need to edit this.

refspec ()
You can give a wavelength calibrated arc spectrum here (transformed). The manual says this will help identify the right lines; I say it makes no difference.

order (> 2)
Give the order for the autoidentification routine. Note that this should stay low (4 or 5) to minimise the freedom available for the line finding algorithm; once an approximate solution is found, then a higher order fit is performed afterwards (order=6).

5.12.1 Description

This routine sets up *autoidentify* and *aidpars* which together are used to automatically identify arc lines and then wavelength calibrate the spectra. It does not have any direct effect on data and is called in *sw_arccal*. However, if you find you're having trouble with automatically identifying the arc lines, you could try modifying *crval* and/or *cdelt*.

In the future, this routine may be updated for compatibility with other instruments (e.g. SINFONI).

5.13 sw_lskyident

sky_list (skylines.dat | skylines_cut.dat | userskylinelist.dat)
Give the arc line list here.

crval ()
Give the coordinate reference value here (wavelength at CRPIX). Usually you don't need to edit this.

cdelt ()
: Give the coordinate delta value here (wavelength change per pixel).Usually you don't need to edit this.

5.13.1 Description

This routine sets up *autoidentify* and *aidpars* which together are used to automatically identify sky lines and then wavelength calibrate the spectra (to combat the spectral component of flexure). It does not have any direct effect on data and is called in *sw_scicube*. However, if you find you're having trouble with automatically identifying the sky lines, you could try modifying *crval* and/or *cdelt*.

In the future, this routine may be updated for compatibility with other instruments (e.g. SINFONI).

5.14 sw_sellinereg

arcframe ()
: Give the name of the raw arc frame here.

searchlam ()
: Give the wavelength of the emission line to bracket here.

skyframe ()
: Give the name of the raw frame with (calibrated) skyspectra here (this is optional and is not a requirement).

database ()
: Give the database directory here; usually this is just **database**

lam0 ()
: Give the wavelength of the first spectral pixel(s) here.

dlam ()
: Give the wavelength increment per pixel here.

nlam ()
: Give the total number of wavelength pixels here.

searchwidth ()
: Give the width of the emission line here; in reality, this should be ∼5 x FWHM.

recovpix1 ()
: This parameter returns the first pixel (along the spectral axis) to bracket the chosen wavelength.

recovpix2 ()
: This parameter returns the second pixel (along the specral axis) to bracket the chosen wavelength

sflip (yes | no)
: If the data was prep'd with this new pipeline, the spectral axis is reversed to the norm, so set this parameter to **yes**; otherwise set it to **no**.

verbose (yes | no)
: Print verbose output?

5.14.1 Description

In the *sw_sflex* stage (§5.6), it's necessary to specify a region which brackets an emission line in alternate slitlets; you have to do this for an arc and a sky line. This routine traces which pixels get transformed where (during the wavelength calibration) and backtracks them, knowing the wavelength they were transformed to. This then allows one to know the (approx) wavelength of pixels in the raw frame. The *arcframe* is obligatory for this process, but you don't need to specify the *skyframe* if you don't want to (because you're trying to find a pixel range in the arc frame or you didn't wavelength calibrate the science frame using skylines). However, note that if you're trying to bracket a sky line and you don't give the *skyframe*, the routine won't be able to correct for spectral flexure and the result could well be wrong. If you give a skyframe that doesn't exist in the database (i.e. you haven't already made a cube of it using the sky line wavelength calibration option) then you'll get an error.

Good wavelengths to choose are: 0.9658 for the Arc frame and 0.8830 for the sky frame. A good *seachwidth* is 0.011.

5.15 inoutparse

input ()
: Please give a single/REGEXP/filelist of input science file(s) here.

output ()
: Please give single/%a%b%c/filelist of output file(s) here. Note the files must be different to the input.

chip (master | slave)
: The chip type.

inlist ()
: After running the procedure, this variable will contain the name of a text file which contains the input file names, one per line (AKA a filelist).

outlist ()
After running the procedure, this variable will contain the name of a text file which contains the output file names, one per line (AKA a filelist).

verbose ()
Select to print verbose output to the screen.

5.15.1 Description

Although not really a Swift procedure, *inoutparse* is used by the Swift procedures and so it's useful to describe its purpose here. Essentially, it takes the various possible forms of input and output files that users may give and generates filelists (text files containing the names of files, one per line) which can then be used in the procedure. Various input methods are available to users of the Swift procedures: they can specify a single fits file (with or without the **.fits** extension); use wildcards (such as ***.fits**, **master0??.fits**, **slave0[0-1][0-5].fits**) to specify a number of files quickly, or use a filelist, as described previously by prefixing the name of the filelist with **@**. Similarly, the user may specify the output files in many ways: they may choose a single file name; us the special IRAF syntax with fildcard inputs which replaces leading text with different text, such as **%add_this%remove_this%rest_of_filename_or_wildcard**, they may choose to define a filelist of the output file names, or they may choose to use the special Swift prefix syntax, such as $\wedge$**prefix**. In order to handle and merge all these possible input and output options, *inoutparse* was developed. The inner workings are very tedious, and won't be documented in detail. Suffice to say, if you give it a sensible input and output, it will generate an input and output filelists which can be found in the parameters *inlist* and *outlist*. To access these, you may use the **myvar=inoutparse.inlist** syntax.

5.16 sw_chkpkg

verbose (yes | no)
Select to print verbose output to the screen.

5.16.1 Description

Again, this is not really a Swift procedure, but it's used by the procedures. All it does is check that the relevant packages required by Swift have been loaded. If not, it tells you which ones need to be loaded (and quits politely).

Chapter 6

Distributed Calibration Files

There are a number of **dat** files that are distributed with the pipeline in the **swift** directory. Here we describe what they're used for and when to use them.

Arc line lists (Ar.dat, ArNe.dat)
: These files are line lists for the arc lamps. You need to specify one of these in the *arc_list* parameter in *swiftred* or the *arcfilelist* in *sw_arccal*. Common sense should prevail, if both the Argon and Neon lamps were on when you took the arc exposures, use **ArNe.dat**; if just the Argon lamp was on, use *Ar.dat*; if just the Neon lamp was on, panic (or make your own Neon line list from the other two).

Static bad pixel masks (BPMs) (mBPM.dat, mBPM_2100x4200.dat, mBPM2.dat, mBPM_4112x2040.dat, sBPM.dat, sBPM_2100x4200.dat, sBPM2.dat, sBPM_4112x2040.dat)
: These files are text files listing the bad pixels in the master (prefixed with **m**) and slave (prefixed with **s**) CCDs. Most of these files are the same as each other, so to simplify matters, you should know:
(m,s)BPM.dat=(m,s)BPM_2100x4200.dat;
(m,s)BPM2.dat=(m,s)BPM_4112x2040.dat.
For most purposes, you should use the (m,s)BPM2.dat files. These are for use with files prep'd (and trimmed) with *sw_prep*. If however you prep'd all input files with the old pipeline, you should use the (m,s)BPM.dat files (and see §7). If at any point you're unsure which to use, check the dimensions of your prep'd image with *catfits* and then use the BPM file with the matching name, e.g. if your master image is 2100x4200 pixels in size, you should use mBPM_2100x4200.dat as the BPM.

Sky line lists (skylines.dat, skylines_cut.dat)
: These two files are used in *sw_scicube* to identify skylines in the input data and correct for the spectral flexure: *skylines.dat* is the original line list, but it contains too many lines to be much use; *skylines_cut.dat* is a subset of the former line list and works well with most SWIFT data. Therefore, you should nearly always use *skylines_cut.dat*.

Throughput Curves (master_throughput.dat, slave_throughput.dat)

These files give the throughputs for the master and slave chips, as determined by Niranjan Thatte (**epsilon.dat**, available in the Swift directory and on the sharepoint). I made **master_throughput.dat** and **slave_throughtput.dat** from **epsilon.dat** by interpolating them onto a regular wavelength grid (from 0.63 microns to 1.05 microns in 1Å steps, just like the default wavelength setup for swift). Note that if you resample to a logλ grid, you will need to resample thesd curves to the new logλ grid before giving them to *sw_mkfl*. To correct the throughput in *sw_mkfl* (§5.9), I first average the two curves and then divide by the {master/slave} to get the {master/slave} correction factors; thus we correct to an average, not to a standard and you'll still need to flux calibrate your data.

Chapter 7

Tips and Tricks

Here's a list of useful ideas to get the most of the pipeline.

7.1 Using Swift as a true pipeline

Once you've run all the calibration stages, you can run *sw_scicubes* or similarly, *swiftred* with just the *S_SCICUBE* stage activated as many times as you like on different science files (be sure to set the *p_...* stages correctly). All the calibration infomation is stored in the *database* directory and you only need to specify the correct calibration frames each time for the data cubes to be created (without user interaction, if desired). Thus, a genuine *pipeline* can be setup which will take input frames and spit out cubes (although you might want to consider running *sw_bps* on the frames to correct for static bad pixels/cosmics and create BPM cubes too).

7.2 Using Swift with files prep'd with the previous python/c code

If you've been using the old pipeline, but you've now swapped to this new one, all your files might already have been prep'd with the old pipeline. Fear not, this pipeline should be able to use these files. Firstly, you need to make sure *sflip* is set to **no** and toggle the IRAF shell variable *apaxis* from 1 to 2 with the command:
cl> reset apaxis=2
This sets up the pipeline to deal with the orientation of images prep'd with the old pipeline. However, you also need to change the BPMs (see §6) and if you plan to use the *S_SFLEX* stage in *swiftred* (or the stand-alone *SW_SFLEX* routine), you'll need to redefine *arc_sec* and *sky_sec*; see §5.6 for more details.

7.3 Adding frames

Don't add science frames together in the *sw_prep* stage; add them once you've made cubes of them. Although *imcombine* can add cubes with BPM rejection, you're probably best converting the relevant bad pixels to Nans in IDL and then using the old pipeline's *swiftCombineCubesBySlice.py* – *imcombine* takes FOREVER for some reason (memory?).

7.4 Aligning multiple exposures

To align different cubes, you could try collapsing them over a certain wavelength range with *blkav* and then fitting Gaussians to the science object; then you can use these shifts with the old pipeline's *swiftCombineCubesBySlice*. If you can't see the science object in the collapsed images, it might be worth looking at the shifts that the telescope made between exposures and shifting the cubes accordingly.

7.5 Pyraf

This pipeline is fully compatible with *Pyraf*.

7.6 Old prep Vs New prep

If you compare data prep'd with the old pipeline and data prep'd with the new pipeline, you'll notice that there's a slight shift (1 pixel) along the wavelength axis. This doesn't really have any effect on the reduction process or the data, but it was a bug in the old pipeline; the new pipeline does things correctly.

7.7 Converting BPMs from the old to new pipeline

If you ever want to convert the BPMs for data prep'd by the old pipeline (e.g. mBPM.dat) into BPMs for data prep'd with the new pipeline (e.g. mBPM2.dat), the following *awk* command may come in useful:

```
> awk 'NR==1 {print $0}; NR > 1 {printf("%4d %4d %4d %4d \n",
4200-$4-68-10+1,4200-$3-68-10+1,$1, $2)}' mBPM.dat
```

Likewise, if you want to convert BPMs for data prep'd with this new pipeline back to BPMs for data preped with the old pipeline, the following becomes useful:

```
> awk 'NR==1 {print $0}; NR > 1 {printf("%4d %4d %4d %4d \n", $3,$4,"
",4200-$2-68-10+1,4200-$1-68-10+1)}' mBPM2.dat
```

Chapter 8

Troubleshooting

Here's an arsenal of ammunition to throw at a badly behaved insurgent pipeline.

8.1 The automatic identification of arc lines fails!

This could be caused by a number of things. Firstly, did the pipeline fail at the first attempt of the first slitlet? If so, it's probably one of the following: Check what is the *arclinelist* set to. Does this match which lamp(s) were working on the night? Can you see the correct lines in the input frame? Did you prep the files with the old pipeline? If so, what is *sflip* set to? What is *apaxis* set to? If, however, the arc identification fails midway through the slitlets then the problem might be more serious. Here are your options:

1. You can tinker with the params in *sw_larcident* (which sets up *autoidentify* and *aidpars*) and then run *sw_larcident* followed by *autoidentify* on the slitlet in question, i.e.
   ```
   cl> unlearn autoidentify aidpars sw_larcident
   cl> sw_larcident
   cl> autoidentify ARC.0010.fits
   ```
   and see if you get a successful result; once you have (for all slitlets), you can try running the pipeline again and it should work. Tinkering with *autoidentify* and *aidpars* via *sw_larcident* is fustrating and doesn't usually yield results quickly; you can also make it work for this data set and break it for a subsequent one. On the whole, this approach is not recommended as it's generally slow and annoying.

2. Perform a manual identification on the lines using Figs. 5.6, 5.7, 5.8, 5.9, 5.10 and 5.11. This may seem the long way round, but believe me it's not. Usually if you mark three or 4 lines spanning the wavelength range, you can get the routine to find the rest automatically and then fit it as usual.

3. You could email me.... but I'm unlikely to have much spare time. Chances are I'll get back to you in over a month. So this is best avoided unless you think it's a

genuine bug in the pipeline that's causing problems.

8.2 There's still bad pixels/columns in my data

Double check you're using the right BPM file: mBPM.dat for the master chip and sBPM.dat for the slave chip; BPM2.dat if you prep'd the data in this pipeline, BPM.dat if you prep'd the data in the old pipeline. If it looks like just one column, then it might be one I missed - try adding it to the BPMs and email me the changes. You should also make sure it isn't just a cosmic that was missed (you can try playing with the La Cosmic parameters if you like) and that you asked the pipeline to remove bad pixels and cosmics in the first place (*S_BPS* and *s_lacos* must be set to yes, and it might help to have *s_thresh* set too; play with the*theshold* value too).

8.3 Careful with .fits extensions

Every input file can be given with or without the **.fits** extension; as such, if there are two files in the same working directory, one with **.fits** and one without, this will cause an error or may even lead to both files being processes. Don't do this.

Chapter 9

Limitations and Improvements

1. I plan to have a look at different ways of combining cubes in IRAF. Imcombine takes forever and it may be due to memory issues. Combining 'by slice' or at least in smaller blocks may be be way forward.

2. Flat fielding and illumination correction are very difficult in IRAF and have not been included in the *swiftred* pipeline; however, the *sw_mkfl* procedure does attempt to address these issues, although problems remain at present (see §5.9)

3. I haven't tested how well *sw_corhoriz* matches the master and slave cubes; it should be ok, but it hasn't been tested. There are good reasons for performing *sw_corhoriz* seperately on the master and slave frames, although it should be possible to modify it to act on a combined master and slave collapsed image of the horizontal lines; naturally, the correction must then be applied on a combined master and slave cube, which isn't currently created.

4. I hope to make a script which reduced the master and slave data simultaneously and stiches it together.

5. Niranjan has come up with a alternate method of wavelength calibrating the cubes: he skips the *sw_arccal* stage and instead does all the wavelength calibration (and de-warping) with the skylines. The advantage of this is the removal of an interpolation stage, but the drawback is that there may not be enough lines at low wavelength ($<$700nm) to really constrain this region.

6. Matthias has found a way to extract the slitlets without interpolation; naturally the vertical trace correction isn't performed either, but this can be done using *identify*, *reidentify*, *fitcoords* and *transform*, and the latter (interpolation) stage can be combined with the *sw_arccal transform* stage, thus removing an interpolation stage.

7. I've been experimenting with the files created by *fitcoords* and I've managed to hack the arc fitcoords file to include the spectral flexure component found in the sky lines

fitcoords file. In principal, this would allow me to extract slitlets and wavelength calibrate (incl. spectral flexure correction) in one interpolation.... watch this space.

8. Performing the *sw_corhoriz* stage will probably always add an interpolation to the reduction; however, it may be possible to add a complicated wold coordinate system (WCS) in the cube headers to correctly give the pixel positions without having corrected for the warped FoV.

9. The gemini people have created an interesting suite of cube creation and manipulation software... I plan to see if any of it is useful to us at some point in the future.