Data reduction of long-slit spectra using pyraf

Original version: Oct 24, 2016

Updated: November 6, 2016

1. Preliminaries

This document presents instructions for how to process long-slit spectra using `pyraf`. It was written for the PHY517/AST443 Astronomical Techniques class at Stony Brook University to guide data analysis taken with the DADOS spectrograph and the Meade 14-inch telescope.

`pyraf` is an environment within `python` to use the astronomical software package `iraf`. `iraf` itself dates back to when astronomers first started using CCDs - the first release was more than 30 years ago! Installing, setting up, and using `iraf` has traditionally been cumbersome; `pyraf` provides a more familiar and user-friendly interface. In particular, it is much easier to write scripts in `python/pyraf` than `iraf`. While other software packages provide good alternatives for working with imaging data, `iraf/pyraf` remains a standard method of working with spectroscopic data.

There are extensive help files available for spectroscopic `iraf` tasks. These can be accessed by typing

`help` *task*

on the `pyraf` command line, where *task* is the name of the command in question. The same help files are also available on the web, both at the Space Telescope Science Institute (STScI):

`http://stsdas.stsci.edu/cgi-bin/gethelp.cgi?noao.hlp`

and `iraf.net`:

`http://iraf.net/irafhelp.php?val=noao&help=Help+Page&pkg=1`

(Also google can be fairly helpful.)

2. Set-up

`pyraf` needs a few configuration files, which it expects to find in the same directory as the one it is started from. You can either create a new directory in a location of your choosing to keep all of your `pyraf` files (and always start `pyraf` there and then change to your data directory), or create / copy the set-up files to the directory with your data. We will assume the latter here.

To initialize a directory for `pyraf/iraf`, issue the command

```
mkiraf
```

(and confirm `xgterm` on the prompt). This creates a file `login.cl` and a directory `uparm`. Edit `login.cl` so that the variables imdir and cache point to the same place as `home`. Finally, also make a directory called `pyraf`:

```
mkdir pyraf
```

Note that these steps only have to be done once for a certain directory.

It is a good idea to open `ds9` before starting `pyraf`. `pyraf` can send images for viewing to `ds9`; also, you are testing that your window forwarding works.

```
ds9 &
```

Now you're ready to go! The easiest way to use `pyraf` is to type

```
pyraf
```

on the command line. (You can alternatively call it from a `python` session, with slightly different syntax.)

Similar to `python`, we start by loading the packages we will use. Issue the following on the `pyraf` command line to load the packages needed for spectroscopy:

```
noao
twodspec
onedspec
longslit
apextract
```

Note that these have to be loaded every time you start `pyraf`.

The following tells the `apextract` package that the spectral dispersion runs along rows (lines), and only has to be issued once inside your `pyraf` directory:

```
iraf.apextract.dispaxis=1
```

Alternatively, you can bring up the entire parameter file for `apextract` with the command `epar apextract`, and edit the respective line.

3. Outline

Prior to starting the work in `pyraf`, make sure you have dark-subtracted your science frames. Modify your `python` script from the imaging lab to do so.

For each wavelength setting of the spectrograph, you should have the following files:

- a flat-field taken with a continuum lamp (example: `flat.fits`)
- an image of an arc lamp emission line spectrum (`arc.fits`)
- an image of a calibration star with known spectral type and brightness (`star.fits`)
- one or more science exposures (`science_1.fits, science_2.fits`, etc.)

The file names here are only examples, but note that the ending for each image has to be "`.fits`"; "`.FIT`" will not be recognized by `pyraf`. You can use the `unix` task `rename` to batch-rename your data, if necessary.

4. Flat-fielding and cutting the data

The flat-field is used to correct small-scale pixel sensitivity variations, as well as slit-width variations. It is NOT used to calibrate the sensitivity as a function of wavelength, since we usually do not know the intrinsic spectrum of the calibration lamp (the standard star observations are used for this).

Start by plotting a cut across the image:

```
implot flat.fits
```

This shows a cut along a row ("line" in `iraf` language), i.e. along the spectral direction. Hit "c" in the display window to get a view along a column. You should clearly see the three spectra corresponding to the three entrance slits of the DADOS spectrograph. Identify the slit that you are working with and measure its boundaries (press space to identify the current cursor position). In this example, the limits of our target spectrum are rows 35 and 95.[1] Type "q" to exit.

We then fit a spline function to the flat-field of our target slit:

```
response flat.fits flat.fits[*,35:95] flat.N.fits
```

Answer "yes" to the prompt:

> `Fit the normalization spectrum for flat.fits interactively (yes):` yes

You should now see a plot of a spectrum. The second argument in the command above specifies to extract a spectrum from all columns, but only rows 35-95 of `flat.fits` - this is the spectrum shown in the display window. Adjust the order of the fit by typing "`:order` *n*" (*n* being the desired order) into the display window, and hitting "f" to re-fit. Make sure the fit is a good description of the data, but avoid too much ringing at the endpoints of the spectrum. Type "q" when you're done. The output image, `flat.N.fits`, is every row of the input image, `flat.fits`, divided by this fit. This is the flat-field by which we correct all other data.

To apply the flat-field, as well as to cut the images and only retain the information from our target slit, we use the `imarith` task, which provides basic math operations on images, e.g.

[1]Note that we assume here that the CCD is well aligned with the spectrograph, meaning that the spectra run parallel to the x-axis. Come see me if your spectra have a significant tilt.

```
imarith science_1.fits[*,35:95] / flat.N.fits[*,35:95] science_1.CF.fits
```

Note how the section of each image is specified in the two operands. This step has to be done on every image (arcs, standard stars, and science images). If you have a lot of input files, you can work with lists of input images. E.g. you could make a text file "`in.txt`", where each line is an input argument such as `arc.fits[*,35:95]`. Similarly, make a file to hold the output names, "`out.txt`". You can then batch-process the images by running

```
imarith @in.txt / flat.fits[*,35:95] @out.txt
```

To examine 2d images, such as the output image here, you can use the task `imexamine`, which opens the image in `ds9`. Type "q" to exit, as usual.

5. Extracting the spectra

"Extracting" the spectra means to sum up the flux of the target object from those rows with object flux to create a 1-dimensional spectrum. In the same step, we will estimate and subtract the background flux from the part of the slit that observed "empty" sky. The task `apall` does all of these at the same time:

```
apall science_1.CF.fits line=360
```

`apall` shows a view along a column, i.e. at a fixed wavelength - it shows the spatial information along the slit. In the example, we have specified to show the data centered at row 360; specifying a position is useful e.g. when viewing an emission-line object. Answer the following prompts with their defaults:

```
Find apertures for science_1.CF? ('yes'):    yes
Number of apertures to be found automatically (1):    1
Resize apertures for science_1.CF? ('yes'): yes
Edit apertures for science_1.CF? ('yes'):    yes
```

You should now see a profile of the flux along the slit in the display window. Make sure you see the flux of the object; if not, choose a different central line. You should see an automatically placed aperture (ideally on the object), as well as an aperture to place the background. You will probably have to adjust both of these to capture most of the object flux, as well as to select a wide background region that is free of object flux. To adjust the object aperture, move the curser to where you want to place the lower aperture limit and hit "l". Repeat for the upper limit with "u". To place the background region, first hit "b". Rescale the window to show the entire graph by typing "w a". To delete the existing background region, hit "z". Hit "s" to place one limit of the background region, and "s" again to place the other limit. Hit "q" when you're done to return to the object aperture extraction mode, and "q" again to finish the task. It will then ask you:

```
Trace apertures for science_1.CF?
```

"Tracing" the aperture means to fit the same object and background aperture at multiple columns (wavelengths) along the spectrum. If you have a continuum source that's well defined over the entire width of your spectrum (e.g. a star), answer "yes". If you only have a few emission lines, type "no".

Answer the following prompts with their default values:

`Write apertures for science_1.CF to database?` yes

`Extract aperture spectra for science_1.CF?` yes

`Review extracted spectra from science_1.CF?` yes

`Review extracted spectrum for aperture 1 from science_1.CF?` yes

You now have a 1D, background-subtracted spectrum of your target, saved in a file called `science_1.CF.ms.fits`. You can visualize the result with

```
  splot science_1.CF.ms.fits
```

Note that the x-coordinate is still in pixels, not in wavelength units. Repeat `apall` for all your science images, as well as for your standard star observations (for the standard star, you do want to trace the aperture).

6. Wavelength calibration

To calibrate the wavelength identification of the spectra, first run `apall` on the arc image, `arc.CF.fits`. The aperture should be the entire slit, and you should NOT subtract a background (add `background=none` when starting `apall`).

To identify the emission lines, and determine the dispersion relation, i.e. the relation between position along the x-axis and wavelength, use the `identify` task:

```
  identify arc.CF.ms.fits
```

which brings up your arclamp spectrum. First, mark 2-3 lines by placing the cursor on them, hitting "m" and entering the wavelength of the line. Hit "f" to fit for the dispersion solution; type "q" to exit the fit display and return to your spectrum - note how the x-axis has changed to Angstroms. Type "l" to automatically identify more lines. CHECK that these are correct! Delete any identification that are just noise ("d"). Re-fit ("f"). The default plot shown after fitting shows the residuals as function of wavelength; hit "h", "i", "j", "k", "l" to visualize various other relations. Note that the underlying task here is `icfit`. When you're happy with the fit, type "q" to exit and write the solution to the database.

Side note: if you are determining a wavelength solution that is similar to one that you found before (e.g. when you tried to re-set a wavelength setting), you can try the task `reidentify`:

```
  reidentify arc_1.CF.ms.fits arc_2.CF.ms.fits interactive=yes
```

This will attempt to transfer the dispersion solution from `arc_1.CF.ms.fits` to `arc_2.CF.ms.fits`.

In this case, you can try hitting "g" instead of "f", which fits only for a zero-point shift of the solution. Check that the results are ok, and that the lines are correctly identified!

To transfer the dispersion solution found on an arc lamp, we add its information to the header file of the science images:

```
hedit science_1.CF.ms.fits REFSPEC1 "arc_1.CF.ms.fits" add=yes
```

This command writes a new keyword "`REFSPEC1`" with a value pointing to the arc file with the dispersion solution into the `science_1.CF.ms.fits` file. To apply the dispersion solution and calibrate the wavelength axis, use the `dispcor` command:

```
dispcor science_1.CF.ms.fits science_1.CF.ms.L.fits
```

7. Flux calibration

To calibrate the sensitivity as a function of wavelength, we compare the observed spectrum of a star to its known reference spectrum. To do so, first extract the standard star spectrum and apply the appropriate wavelength calibration as above. Then, run the `standard` task:

```
standard star.CF.ms.L.fits stdspec caldir=onedstds$blackbody/ observatory=oro
airmass=1.1 star_name=V
```

In the prompts, enter the magnitude of the star and the band it was measured in, as well as the star's spectral type (or temperature). (Look these up in Simbad.):

```
Magnitude of star ():
Magnitude type (V|J|H|K|L|Lprime|M) ():
Effective temperature or spectral type ():
star.CF.ms.L.fits[1]:  Edit bandpasses?  (no|yes|NO|YES|NO!|YES!) ('yes'): no
```

The output is a textfile "`stdspec`" which lists the star's reference and observed fluxes in a number of basspands across the observed spectrum.

To determine the sensitivity function (conversion of counts to energy flux as function of wavelength), use the `sensfunc` task:

```
sensfunc stdspec sfunc
```

which brings up a fitting routine, as before. Adjust the order of the fit to make sure it captures the variability along the spectrum, but try to avoid too much ringing at the edges (you can add fake datapoints with "a", if necessary).

To apply the sensitivity function to your data, use `calibrate`:

```
calibrate science_1.CF.ms.L.fits.fits science_1.cal.fits sensitivity="sfunc"
extinct=no
```

which produces the final, flux-calibrated spectrum.

Alternative: these steps may in fact be easier to write in `python` using `numpy`. A reference spectrum for the spectrophotometric standard star can be modelled as a black-body spectrum with the star's temperature, which is implemented in `astropy`:
`http://docs.astropy.org/en/stable/analytic_functions/`

8. Measuring line fluxes

The `splot` task, which we have used for plotting spectra before, is a convenient tool for measuring line fluxes. After bringing up a spectrum in `splot`, use the window keystrokes (`w y` to zoom in by a factor of 2 in y; `w x` to zoom in by a factor of 2 in x) to zoom in on a line / a set of lines.

To measure a single line, hit "k" on the left side of the line at the position where you want to set the background; hit "k" again on the right side (you might have to hit "k" three times to tell it that you want to fit a Gaussian). `Splot` will return the line center, line flux, and equivalent width. Try this a few times on the same line to estimate the uncertainty.

If lines are blended or nearby, you can use "d" to fit multiple lines at the same time. Again, hit "d" on the left side and on the right side of the area you are trying to fit. Then, mark the lines to be fit with "g". Hit "q" to indicate you are done with marking lines. Answer the following prompts. By default, you should try to fit all positions and widths, but keep the background fixed (it is determined by the first two "d" points). `splot` reports the measured fluxes one-by-one; use the "+" and "-" keys to move to the next line. Again, do this a few times to estimate the uncertainty.

Note that `splot` can smooth the spectrum (keystroke "s"), which may help bring out faint lines. Also note that to revert to the unsmoothed spectrum, you have to exit `splot` ("q").

9. Exporting the spectra to ASCII files

You can use the task `wspectext` to output the 1D spectra to ASCII files:

```
wspectext final.fits final.txt header=no
```