Summary of IRAF tasks for spectroscopy

- Vikram Khaire

Notes:
Anything starting with $ is a command typed in a terminal
Blue highlighted text is for editing parameters on xgterm tasks
Yellow highlighted text are for trying out things on the screen on the interactive plots or ds9
The texts after # are comments on the command (don't type those on the terminal)

Basic packages IRAF should have:
`noao imred ccdred specred onedspec` (not in the same order)

The bias subtraction is the same as in the case of imaging. Flat fielding is different because one needs to take into account the wavelength-dependent response of pixels along the dispersion axis. The flats are also usually taken with halogen lamps.

Raw files checklist:
Bias files
Flat files (halogen lamp)
Object spectrum files
Lamp files (spectrum of standard lamp for wavelength calibration)
Standard star's spectrum files (for flux calibration)

Starting the IRAF (author's choice):

```
$ xgterm -fn 10x20 -sbr &
```

Then on xgterm:
(Remember to open xgterm or `$ cd` into xgterm to the directory where all raw files are given. Otherwise, you need to specify the path to files for all the operations. Later is a good practice if you are dealing with a large set of data scattered or arranged in different directories.)

```
$ ds9 &
```

You can start ds9 outside the xgterm as long as you don't want it to interact with xgterms (for the current IRAF version this is a better option since ds9 in xgterm shows only as a small portion of the CCD). For the spectroscopy data reduction, you don't need to interact with ds9.

Type cl (or ecl) on xgterm to login to IRAF:

```
$ cl
```

Bias subtraction (task zerocombine):

I) Creating a master bias

```
$ epar zerocombine
```

The zerocombine task is in `noao→ imred → ccdred`. To load these packges, just type `$ package_name` on terminal.

(Nore `$ lpar zerocombine` will list the parameters)

```
input = bias*.fits  # list of bias files
output = mbias.fits  # output bias files
combine = median
reject = avsigclip
ccdtype =  # keep it blank with a space bar unless your file has built in ccdtype
process = no
rdnoise = 4.0   # applicable for IGO data
gain = 1.5  # applicable for IGO data
```

Keep other parameters as default use **Ctrl + d** (save and exit), then execute using

```
$ zerocombine
```

The master bias file with the given output name (here; mbias.fits) would be created.

II) Subtracting master bias from all other files

Pixel-based operations on the files with the same size (same pixel rows and columns) can be easily performed by task imarith

To efficiently subtract the master bias from all other files, first, create a list of 'other' files. On xgterm (like linux) you can type

```
$ ls flat*.fits object*.fits lamp*.fits standard_star*.fits > other_than_bias.in
```

(Note: The structure of the above command is following
$ ls files_to_select > output_file_where_you_want_to_save_the_list)

Once the list of other files is created you can subtract the master bias from each file using the following command.

```
$ imarith @other_than_bias.in - mbias.fits
@list_of_output_files
```

Here, the list_of_output_files is a file with output file names (in the same order as other_than_bias.in file) where you want to store the bias subtracted files. It is easy to rewrite the files using the ist_of_output_files = other_than_bias.in. But if you choose to do that make sure that you have stored raw files somewhere else so that you can always redo the reduction if needed.

Now you have successfully subtracted master bias from every other file.

Flat fielding for spectroscopy (task flatcombine):

I) creating master flat

(same as in the case of imaging)

```
$ epar flatcombine
```

```
input = flat*.fits  # list of flat files (halogen lamp exposure)
output = mflat.fits  # output flat files
combine = median
reject = avsigclip
ccdtype =  # keep it blank with a space bar unless your file has built in ccdtype
process = no
rdnoise = 4.0   # applicable for IGO data
gain = 1.5  # applicable for IGO data
```

Keep other parameters as default use **Ctrl + d** (save and exit), then execute using

```
$ flatcombine
```

The master flat file with the given output name (here; mflat.fits) would be created.

II) normalizing the flat

(different from imaging)

First, you need to specify the dispersion axis (axis of ccd along which the spectrum is taken) so that you can calculate the response of the CCD along the dispersion axis. Specify that axis using following command. (Load `noao` → `onedespec` package on iraf before).

```
$ dispaxis = 1 #specific to IGO data
```

Here '1' is for the x-axis (horizontal row) and '2' is for the y-axis (vertical columns). The default value is '2' along y-axis. One can see the distribution along those axes (collapsing output along the respective perpendicular axis) using 'view' → 'horizontal graph / vertical graph' tabs on (top of) the ds9 display.

(Note that if you do not find `dispaxis` or `response` in the `onedspec` try using one at `twodspec`.)

Now to normalize flat you need to fit the response of pixels along the dispersion axis. This can be done using a task `response` (in package `specred`), as described below

```
$ epar response
```

calibrate = mflat.fits # applicable for IGO data
normaliz = mflat.fits[*,*] # use [*, *] to use whole file otherwise you need to specify ther range of pixels on x and y axis, something like mflat.fits [10:100, 20:400]
response = nflat.fits # output file

Keep other parameters the same but later with experience, you can choose a few of those by default.

Then execute using

```
$ response
```

It will ask if you want to fit interactively then say yes by entering. It will open a plotting IRAF interactive terminal. There are many functions (type ? to see various options on iraf command terminal; then use q and enter to go back on the interactive plot) that you can play around and fit the response with a n degree polynomial. (dashed line is a fit). Make a good fit. Measure of goodness is by eye only. Tha't how most people work with IRAF.

For example, see below for few simple things:

You can type d to delete points and press f to do a refit. You can change default parameters that were in epar response function interactively by typing : on the interactive terminal and writing out things.

For example, if you want to change the order of the polynomial to say 10 then type

: order 10
It will appear on the bottom left corner of IRAFterm interactive plot. Pay attention to that corner for such commands.

Then press enter and then f for refitting. When you refit IRAFterm will show the change in the order (top middle section of the plot).

Play around with various things till you are satisfied with the fit. Then hit q and polynomial will be applied to every pixel-row parallel to the dispersion axis.

III) dividing by the normalized flat

Same as in the case of bias subtraction use `imarith` task

Create a text file with a list of files other than bias and flat files i.e all object, standard-source, and lamp files.

$ ls lamp* standard* object* > divide.list

Now use this 'divide.list' file to divide and overwrite the files (you can always create other output name files to store flat fielded spectra, as described in the case of imaging).
.
$ imarith @divide.list / nflat.fits @divide.list

Now you have all files that are bias subtracted and flat-fielded!

Cosmic ray removal

If you look at any image/spectrum taken with high exposure time, many pixels will be exposed to cosmic rays. There are many ways to identify and remove those. Before extracting spectrum you can remove them using many functions in `imred → crutil`

For example, you can use `crmedian` function
Provide input and output image (here object and standard-star spectra) names or use a list of images (with '@' in front) as demonstrated previously.

You can display cosmic ray removed images with the original image on ds9 to see the difference. For every other task use the cosmic ray removed objects and standard source files (better to replace with original file names).

Tracing and extracting the spectrum

The spectrum falls on the specific range of pixels of the CCD. We need to trace these pixels along the dispersion axis so that the 1D spectrum can be extracted.

There are three steps: selecting aperture and background, tracing the spectrum and extracting spectrum along the trace.

I) select aperture and background windows

Use task `apall`

```
$ epar apall
input = "name_of_one_object_file.fits"    #List of input images
nfind = 1          #Number of apertures to be found automatically
(output = "")      #List of output spectra (you can keep it blanck)
(apertures = "1")          #Apertures
(format = "onedspec")  #Extracted spectra format ← most important
(background  = "fit")  # background to subtract
```

Keep other parameters the same but change read noise and gain for IGO data as:

```
(readnoise = "4")          #Read out noise sigma (photons)
(gain = "1.5")             #Photon gain (photons/data number)
```

Then execute by

```
$ apall
```

It will ask you many questions. Check if you want to change anything such as output name or just say ‘yes’ by entering.

It will open an IRAFterm interactive plot of counts vs pixels (perpendicular to dispersion axis). By default, it will select the center of the dispersion axis. In the case of IGO data, that is exactly where the CCD change happens. This can create a problem if the counts are not recorded or corrected (with our bias and flat-field corrections) properly. Sometimes if the center of CCD is where you have a strong absorption line, then you run into the same problem. To avoid or rectify these cases you can give a physical pixel number (check ds9 image of the spectrum) on the dispersion axis in `epar apall` at a location where `line` is written.

In such case (say at pixel 650 pixel), before running apall you can specify

```
(line    =                    650)  #Dispersion line
```

And then run `apall`.

The pixel (along the dispersion axis) vs count plot will automatically find the aperture (the horizontal bar on top of the peak) and background on the left and right of the peak (two horizontal bars). If you want you can change the size and location of aperture and background using simple on the plot commands. Use ? mark to see those.

For example, a few simple things are given here:

To change aperture size take your cursor and use l (for lower limit) and u (for upper limit) to increase the size of the aperture. A change in the horizontal bar will indicate the change in the aperture size.

To delete an aperture press d
To create a new aperture go to the center of the peak and press n

Type b to select the background region. It will replot with a zoom around the aperture. Then using z and cursor you can delete the default background region. Then use s twice to select a rightmost and leftmost point of your new background region.

Once the background region/window selection is done you can type q and it will take you back to the previous plot now showing a new background window.

II) Tracing the spectrum

Once the background and aperture windows are selected you can type q and it will ask you many basic confirmatory questions (read text on the leftmost bottom line) to which you can say yes by entering. Then it will redirect you to the plot with a trace (plus points) and a preliminary 1 order fit (dashed line). You can delete the outlier points by pressing d at the nearest point (it will put a cross sign on the data point to show the point is deleted; to undo delete press u) and then you can press f to see if fit improves. You can play with many things such as ‘order’ and get a good trace i.e a good dash line fit through the points. Look for RMS values (on the plot) to judge the fit. You can keep on improving till you get a subpixel RMS (i.e RMS < 1). The procedure is the same as fitting the response for the normalizing flat as described before.

Then press q and answer to questions (mostly yes) but may change the output name because it will not overwrite. Then it will show an extracted spectrum i.e counts vs pixels.

Also, note that the file extension 001 is added by default for the spectra. It indicates the number of apertures.

III) Extracting spectrum

The above step already extracts the spectrum. For any other object as well as lamp file you can use the same trace by adding the reference name of the object file for which the aperture has been already calculated as above in `apall` task, i,e:

```
input    =    list_of_object_and_spetrum  #List of input images
(output =                          ) List of output spectra
```

you can keep output blank It will create the extracted files with 0001 extensions since the aperture is one.

```
(referen=        name_of_ref_imgae) #List of aperture reference images.
```

Note that the aperture reference image is the one for which there is a file saved in the folder database. If you give a different output name in the previous task for extracting spectrum then search in the database for the file input name, you will see a file name starting with 'ap'. Remove the ap and copy the rest of the file name. This is the file name as the reference in the above line.

Next, you need to set all interactive prompts to 'no', except extracting spectrum. Keep other things as it is.

```
(interac=                      no) Run task interactively?
(find    =                     no) Find apertures?
(recente=                      no) Recenter apertures?
(resize =                      no) Resize apertures?
(edit    =                     no) Edit apertures?
(trace   =                     no) Trace apertures?
(fittrac=                      no) Fit the traced points
interactively?
(extract=                     yes) Extract spectra?
(extras =                      no) Extract sky, sigma, etc.?
(review =                      no) Review extractions?

(background = "fit")          #Background to subtract
```

Keep readnoise and gain as in the previous case.
And then run `apall` for all lamp object and standard source files.

For lamp frames do not subtract background. That is for extracting lamp spectrum make

```
(background = "none")          #Background to subtract
```

provide list of lamp files as input, don't change anything (since IRAF remembers previous steps) and run.

Note that to see what kind of keywords are accepted in any of these cases you can type ? (or any rubbish input) and enter in the field (i.e in the above case at the location of none type ? and hit enter). Then at the left bottom part of the terminal iraf will show acceptable keywords (starting with what ? choose: list of acceptable keywords).

Note that one lamp file is more than enough for identification. Observers take more lamps for contingency or they can combine and get one with better signal to noise for some sensitive measurements.

Now you have extracted spectrum and lamp. You can display those in IRAFterm by

```
$ splot name_of_the_fits_file
```

You can also plot this in python by reading the fits file. This is basically of plot counts (y-axis) vs pixels (x-axis). The next spet is wavelength calibration where you can convert pixels to wavelength.

Wavelength Calibration

<u>i) identify lines from the lamp and fit the dispersion</u>

One has to now identify the standard lines in the lamp spectrum. Use task `onedspec→ identify` for that

```
$ epar identify
images =    extracted_lamp_file_name   #Images containing
features to be identified

(coordlist = "linelists$idhenear.dat") #User coordinate list
```

The coordlist is important for automatic and easy identification. The standard wavelengths of the emission lines of standard lamps are stored in the IRAF. You can

```
$ cd linelists
```

to see the list of standard files. IGO has a helium neon lamp so you can use idhenear.dat (id's of helium neon argon lamp). The way to write this file as a path is to replace '/' in linux path by '$'. That is the path of linelists/idhenear.dat becomes linelists$idhenear.dat.

Keep everything else in identify the task to it's default and run it

```
$ identify
```

This will plot the lamp spectrum in the IRAFterm. Now open the line-list plot from the observatory and start identifying the lines with some pattern.

Identify a few lines and go to IRAFterm
Go to the center of the line and type m (stands for mark, I guess). It will mark the line and on the bottom left corner, it will ask the wavelength. Write wavelength there and enter.

If you misidentify then you can go back and press d to delete.

If you identify many lines then you can press l. It will try to identify most of the lines in the database. Some of them would be bad. You can delete those if you are sure or when you try to fit dispersion then you can remove those.

Type f for fitting the dispersion i.e fitting wavelength vs pixels. Create a good fit with sub-pixel RMS. You can delete the discrepant points/identifications.

Then press q and again q. It will write it into the database (a folder created by IRAF inside your main folder)

Note shift+x and shift+y will zoom in x and y directions
Type r will replot or reset the plot again.

ii) apply this reference

Use task onedspec→ refsepc

```
$ epar refspec
```

```
input = "extracted_object_File.fits"  #List of input spect
(references = "hene_s150_gr7.5769.2.0001")  #List of reference spectra
(sort = " ")              #Sort key
(group = " ")             #Group key
```

Give the reference of the lamp file from the database (without 'id' in front). Add space (i.e none) in group and sort and execute

```
$ refspec
```

Type ‘yes’ to defaults. This will just change the header of the object files (the last line of the header). Type ‘YES’ if you have list of files and just want ot use one reference.

iii) dispersion correction (final step)

Use task `onedspec→ dispcor`

```
$ epar dispcor
```

```
input = "j0840+3633_s150_gr7.5685.0.0001.0001.fits" #List of input spect
output = "output_name"          #List of output spectra
```

Now execute it

```
$ dispcor
```

That is all you need to do for getting the wavelength calibrated spectrum. The standard source will be then used for flux calibration.

You can calibrate all the spectra and sources. To combine different exposure of object (or standard-star) spectra you can use task `scombine` and provide the list of all wavelength calibrated object (standard-star) spectra.

Extra hacks and help:

i) IRAF

- You need not edit the parameters of a simple task (with command epar). You can just execute the task and when prompted provide filenames and other parameters.
- When you edit parameters you need not save the file and then execute. You can just use escape and type: go. This will save and execute the task directly.
- Because of edges, the wavelength calibrate spectrum does not look good. You can use task `scopy` and create another copy of the spectrum by just selecting different wavelength regions (central part excluding the edges). Then plot and display that using `splot`

- If you try to change the aspect ratio of IRAFterm plotting window with the mouse cursor, in Linux (ubuntu 18 or higher version), the whole window just becomes a line. It is very annoying. Rather type q and get out of the IRAFterm, then use mouse cursor on the IRAFterm (which is still showing previous plot) and change the aspect ratio. Now when you (or IRAF) plot anything on IRAFterm it will retain the aspect ratio. I usually just use full-screen IRAFterm.

- **Important:** To see how to use any task and kind of funcions it can have use help i.e use $ help name_of_the_task

ii) ds9

- If you open ds9 outside the xgterm then use file→ open for opening any file. Then for 2D ccd images use scale→ zscale to see colors (i.e z-axis by default in grey).

- To open another file in a new frame use frame→ new frame. In this way, you can open many images and blink (frame → blink) them if you want.

- To stop blinking use (frame → single)

- You can use view → horizontal graph OR vertical graph for visualizing x and y-axis (along row or column) counts.

- Always display images to see how various files look like. Check the header of the files for more information (file → header)

iii) Python fits reading

You can read fits files in many ways. I am listing one way in python:

```
# import functions
from astropy.io import fits
import numpy as np
import matplotlib.pyplot as plt

a = fits.open('final_spectrum.fits') # read wavelength calibrated file
flux = a[0].data # reading flux

# creating wavelength array
head =a[0].header # to read the header of the file. Check the header file!
wave = np.arange(len(flux))* head['CD1_1'] +  head['CRVAL1']
# here head['CRVAL1'] is starting wavelength
# and  head['CD1_1'] is constant wavelength separation

# plotting
plt.plot(wave, flux)
plt.show()
```

For correcting the bias files given in the exam:

Object files given in the exam have already been processed with some kind of processing. You can see that the edges of the files are removed as compared to the ones shown in the class. Also, the pixel values (i.e counts) are float and very small as compared to the pixels in bias files.

Therefore to use these bias files for bias correction you need to normalize the master bias. It would have been an easy task to just take the median (or mean) of all pixels of master bias and divide master bias with that median if we were dealing with just one CCD. Here we have two CCDs joined together. So you need to do this step for both CCDs.

There is a way to do that. You separate (split) master bias into two files. One for each CCD. Then divide them by their median (or mean) over pixels. Then combine them again.

(Note: You need to know the dimension of each file given to you. You can find it by reading the header of the files. A simple hack is to use the following command on xgterm IRAF :
`$ imhead *.fits`
It will print fits file names with a number in a square bracket. Something like:
bias_new.0.fits[2048,400]
It means the bias_new.0.fits has 2048 x 400 pixels, i.e 2048 pixels in the x-direction and 400 in y.)

To do this in IRAF follow the steps provided below.

1. Create a master bias file (mbias.fits)

2. Display it on ds9 and find at which pixel (integer number) two CCDs are joined. Use the mouse pointer to find the position approximately. Say that number is j

3. Then use task imcopy to copy two portions of mbias.fits
   `$ epar imcopy`
   input = "mbias.fits[1:j,*]" Input images
   output = "left_mbias.fits" Output images or directory
   After executing this you will get a file left_mbias.fits having x pixels ranging from first to jth pixel. Number of y pixels (since you wrote * above) will be the same as the input file.

   Similarly create another copy
   `$ epar imcopy`
   input = "mbias.fits[j+1:2048,*]" Input images
   output = "right_mbias.fits" Output images or directory

4. Now find mean by using taks imstat
   `$ imstat left_mbias.fits`
   To get median use $help imstat and read the description to see how to print median

5. Then divide by mean and replace file using imarith i.e
   `$imarith left_mbias.fits / mean left_mbias.fits`

6. Repeat step 4 and 5 for ohter mbias file (i.e right_mbias.fits)

7. Now combine these two normalized bias files using task imjoin
   `$ epar imjoin`
   input = "left_mbias.fits, right_mbias.fits" Input images or @file
   output = "new_mbias.fits" Output joined image
   (joindim = 1) Splice dimension (1=x, 2=y, 3=z, ...)
   (outtype = "2") Output datatype (defaults to highest intype) # 2 for 2d

This will create new_mbias.fits file by joining the above two files along the x-axis. Use this file for bias correction.

Acknowledgment: This document would not have been possible without the help of M. Vivek (IIA Bangalore) who kindly walked me through IRAF software and the whole reduction procedure. The data used in lectures and exams were also provided by him.
(People jokingly say that Vivek was the sole user of IGO. His Ph.D. thesis is based on a very large BAL quasar sample obtained from IGO. Unfortunately, IGO is currently not operational. It stopped working right around the time when Vivek defended his thesis.)