ASTR 535 Lab notes

Jon Holtzman

Spring 2016

Basic display operation

- image display with DS9: DS9 can be used as a standalone display tool
  - start: *ds9*
  - select a file to display using File/Open menu
  - manipulate display scaling using Scale button, note no manual scaling option on main menu, but see Scale/Scale Parameters
  - manipulate color map using mouse motions with right button
  - manipulate region to display using Zoom button
- image display with GAIA
  - Type the alias *starsetup* to set up environment variable, paths, etc. (this is a local NMSU defined alias).
  - Enter *gaia* to start gaia.
  - Note the help window, available from the button at the top right
  - load images using the File menu. You can also start gaia with an image file name on the command line. You can open a new window using the File menu and display another image there, etc.
  - image display: color tables (Color Map), automatic scaling algorithms (Auto Cut, Intensity Map), manual scaling
  - display region: note zoom buttons and zoom and pan windows
  - image histograms: View/Cut levels
  - image slices: View/Slice

Astronomical image processing: Introduction and basics

Friday, February 12, 2016 useful in conjuction with PyRAF) and IDL. Working in one of these environments allows you to script the use of existing routines, and also to develop your own routines. Also extremely important to have tools to be able to explore data.

Getting started with Python

Basics

- Start python using `ipython -matplotlib`
- Python works with *objects*. All objects have different attributes and methods.
- Get information
  - `type(var)` gives type of variable.
  - `var?` gives information on variable (iPython only).
  - `var.<tab>` gives information on variable attributes and methods.
- Python as a language
  - conditionals via `if/elif/else`
  - looping via `for, while`

File I/O with astropy

- FITS: header/data, data types, HDUList, etc.
  - `from astropy.io import fits`
  - `hd = fits.open(filename)` returns HDULIST
  - `hd[0].data` is the data from initial HDU
  - `hd[0].header` is the header from initial HDU
- ASCII:
  - `from astropy.io import ascii`
  - `a = ascii.read(filename)` returns Table with columns.

Image statistics

- numpy array methods, e.g.:
  - `data.sum()` total
  - `data.mean()` mean
  - `data.std` standard deviation
- subsections: `data[y1:y2, x1:x2`

Image display

- primitive display via imshow
  - `plt.imshow(hd[0].data,vmin=min,vmax=max)`
- display using pyds9
  - `from pyds9 import *`
  - `d = DS9()` opens a DS9 window, associates with object `d`.
  - `d.set("fits_filename")` display from file
  - `d.set_pyfits(hd)` display from HDULIST
  - `d.set_np2arr(hd[0].data)` display from numpy array
  - `d.set("scale limits 400 500)` sets display range
  - command list
- display with tv
  - `import os`
  - `os.environ["PYTHONPATH"] = /home/holtz/python`
  - `from tv.tv import *`
  - `t=TV()`
  - `t.tv(hd[0],min=400,max=500)`
  - `t.tv(hd[0].data)`
  - zoom, pan, colorbar
  - blinking image buffers with +/-

Plotting

- `plt.figure`
- `plt.plot(hd[0].data[:,100]` along column 100
- `plt.plot(hd[0].data[500,:]` along row 500

Histogram

- `plt.hist(data.flatten(),[bins=n],[bins=np.arange(min,max,delta)],[log=True])`

HDU (Header Data Unit) consists of a header (array of character strings) and data (2D array of numbers).

Getting started with IDL

Exercises

Introduction to CCD images and basic CCD data reduction

Python tools

Basic techniques for image reduction in IDL are just image arithmetic and statistics (e.g. mean() and std() methods of numpy arrays). To median images, stack them into a *data cube*, then use numpy.median(cube,axis=0) to median them together.

- You can create a cube "in advance", using, e.g., cube=numpy.zeros(nim,nrow,ncol), and load using: cube[0,:,:]=im1, cube[1,:,:]=im2, etc
- You can create a cube "on the fly", using, e.g., cube=numpy.array([im1,im2,im3])
- med=numpy.median(cube,axis=0)

Astronomical image processing packages: IRAF basics

IRAF/DS9 basic operation

One time only for each directory: run `mkiraf`, which creates `login.cl`, and starts a uparm/subdirectory for parameter files. This file can be customized at a later time if you have settings you want to start with every time. To enable a larger frame buffer for display, uncomment and modify line: `stdimage = imt2048`

The preferred method of running IRAF in the modern era is using the PYTHON interface, `pyraf`. You can use pyraf via a normal python interface:

`from pyraf import iraf` or `from pyraf.iraf import *`

for the former, you'll need to precede tasknames with iraf. You will then need to use standard PYTHON styntax, rather than the old IRAF `cl` syntax.

For displaying images, start an image display tool, i.e. DS9, in the background: `ds9 &`. Be aware that the stdimage that is set in the login.cl file may limit the maximum size of the image that will be displayed.

Help

- there is an internal `.help` command.
- IRAF help
- tutorials

Basics

- e.g., to load noao package via Python interpreter: iraf.noao()
- e.g., to load noao package via PYRAF interpreter: noao

image display with DS9 through IRAF: the *display* task

The default mode is to autoscale the image (zscale=yes, zrange=yes). You can manually set the display range using z1=low and z2=high, but you must also turn off autoscaling (zs- zr-) for the manual values to take effect

Note that the data value display in the display window is derived from the display pixel value, so you won't see actual data values that are below z1 or above z2, the value display will just say ¡ z1 or ¿ z2 - rather annoying.

If using IRAF to control the ds9 display, the ds9 scaling option will not be available.

If image buffer isn't set correctly, you can reset using, e.g.,

```
iraf.set(stdimage='imt2048')
```

Other

image cross sections: the *implot* task.

Image histogram: imhist. Look at the parameter file. Note that you can specify image subsections using filename[x1:x2,y1:y2]

Image statistics: imstat. Look at the parameter file. Note you can specify image subsections as above.

Image arithmetic: imarith. You can do arithmetic with images and constants, or with multiple images. For example: imarith file1.fits - 363 will subtract a constant of 363 from the image, imarith file1.fits / file2.fits will divide file1 by file2 (on a pixel-by-pixel basis).

Inspection of stellar images: imexam. Note 'a', 'r', and 'm' keys, '?' for help (note you have to exit help to get interactive cursor), 'q' for quit.

For many tasks that require an input file, it is possible to specify a list of input files if the same action is to be taken on each. This is accomplished by creating a file (e.g., files.lis) that has a list of all of the files that you wish to run the task on, each on a separate line. Then, instead of giving an image name to the task, you give the list file name than contains the image names, preceded by an @ sign, e.g. @files.lis. Remember when you are proccessing images that IRAF wants to write the output files; if you don't have write permission in the input directory, you'll need to also supply an @output.lis file with the names for the output images.

Quit pyraf using the `.exit` command

IRAF user guides

See http://iraf.noao.edu/docs/

IRAF data reduction

IRAF package imred/ccdred: zerocombine, darkcombine, flatcombine, ccdproc (note need for header cards)

lower level: imcombine, image arithmetic

Note that you may have to do a iraf.setinst first, have to pay attention to ccdproc parameters!

IRAF file list specification: comma-separated string

IRAF simple stellar photometry

phot/apphot

Exercises

*Astronomical image processing/reduction: Basic tools Friday, April 1, 2016

When observing, a bare minimum requirement is the ability to look at your data. In many cases, however, it is preferable to have tools to do some quick image manipulation and analysis, and these will be required for image reduction/analysis. It's best if these are easily available so that you are likely to encounter them in most computing situations, and ideally, could access them on your laptop if you have one.

In the current computing climate, I would recommend using Python tools wherever possible. For some analysis, IRAF routines provide a lot of developed routines, so if IRAF installed, these can be useful; I would recommend using them from a Python environment to be able to take advantage of native Python features.

For image display, `ds9` is probably the best choice, although there may be alternatives.

Our goal is to work towards reduction of all of our APO images.

Getting started

- Start ds9 in the background

  ```
  ds9 &
  ```

- Start an iPython session

  ```
  ipython --matplotlib
  ```

- Import standard Python packages

  ```
  import numpy as np
  import matplotlib.pyplot as plt
  import pyds9
  ```

  (note that you can put these in a `/.ipython/profile_defa/startup/00startup.py` script to load every time you start ipython.)

- Import useful astropy routines

  ```
  from astropy.io import fits
  ```

- Create `login.cl` file If IRAF is available, make sure you have a login.cl file. If you don't:

  ```
  mkiraf # note this is a UNIX command, not a python command
  ```

  and edit the `login.cl` file to set `stdimage=imt2048`, or copy a `login.cl` file from a previous directory.

  ```
  from pyraf import iraf
  ```

  in which case, you will need to call iraf routines using `iraf.routine_name()` which makes it clear that they are IRAF routines. If you want to enter the routine names without the `iraf.` prefix, type

  ```
  from pyraf.iraf import *
  ```

Reading images

- Read image into variable:

  ```
  im=fits.open(filename)[0]
  ```

  Note that this reads the first extension ([0]) into a HDU object, with im.header containing the header, and im.data containing the data

- For convenience, you might want to:
  - Set up a variable with the directory name for the images, to avoid having to retype it:

    ```
    imdir='/pathtoimage directory/'
    im=fits.open(imdir+'nameoffile')[0]
    ```

  - Set up a symbolic link to the directory with the images, to avoid having to retype it:

    ```
    %ln -s /pathtoimage directory/ raw    # UNIX command
    im=fits.open('raw/nameoffile')[0]
    ```

- IDL: im=mrdfits('filename')

Displaying images

- Direct from memory (variable):

  ```
  d=DS9()   # to open display
  hd=fits.open(filename)   # puts HDUlist of file into hd
  d.set_pyfits(hd)  #  display from HDUList variable
  d.set("scale limits 400 500")  (sets display range)
  ```

  If you are doing image arithmetic and want to display a numpy array, you can do so with:

  ```
  d.set_np2arr(hd[0].data)  # display from numpy array
  ```

  You might want to write yourself a simple Python function to display and scale with a single simple command.

- Direct from disk, using IRAF display:

  ```
  iraf.display(imdir+'nameoffile')
  ```

  If you wish to control display parameters (recommended):

  ```
  iraf.display(imdir+'nameoffile',zrange='No',scale='No',z1=low,z2=high)
  ```

  where low, high are the values you want for color mapping. If you want to have your values set by default, you can:

  ```
  iraf.epar('display')
  ```

  and set zrange and scale to 'No', or alternatively:

  ```
  iraf.display.setParam('zrange=no')
  iraf.display.setParam('zscale=no')
  ```

- IDL: atv,im,[min=min,max=max]

Image inspection

- image cross sections:
  - Python:

    ```
    plt.plot(im.data[:,500])  # plots row 500
    plt.plot(im.data[500,:)  # plots column 500
    ```

  - IRAF: *implot* task.
    - See plot window commands ('?')
    - 'l' and 'c' for line (row) and column plots, as determined by cursor location
  - IDL: plot,im[*,500]
- Image histogram:
  - Python:

    ```
    plt.hist(im.data.flatten(),bins=....)
    ```

  - IRAF *imhist*. Look at the parameter file for options. Note that you can specify image subsections using `filename[x1:x2,y1:y2]`
  - IDL: plothist,im
- Image statistics:
  - Python: use numpy array methods: mean, sum and std, e.g.,

    ```
    mean=im.data[400:600,400:600].mean()
    tot=im.data[400:600,400:600].sum()
    sig=im.data[400:600,400:600].std()
    ```

  - IRAF imstat. Look at the parameter file. Note you can specify image subsections as above.
  - IDL: MEAN(), STDEV() functions
- Image arithmetic:
  - Python: just use normal arithmetic, e.g.:

    ```
    a=im1.data-bias
    b=im1.data=im2.data
    ```

  - IRAF imarith: file based. You can do arithmetic with images and constants, or with multiple images. For example: *imarith file1.fits - 363* will subtract a constant of 363 from the image, *imarith file1.fits / file2.fits* will divide file1 by file2 (on a pixel-by-pixel basis).
  - IDL: normal array arithmetic
- Interactive inspection of stellar images:
  - Python: someone needs to write some tools!
  - IRAF imexam: need to display image with iraf.display() first. Note 'a', 'r', and 'm' keys, '?' for help (note you have to exit help to get interactive cursor!), 'q' for quit.
  - IDL: atv

Basic data reduction

Overscan subtraction

- Determine overscan region location
- Determine whether constant overscan (subtraction of a single value) is appropriate, or if not, consider possibilities:
  - Fit to overscan as a function of row
  - Median overscan as a function of raw
- Remove overscan
  - Using image arithmetic
  - Using IRAF: ccdproc (note overscan options)

Superbias (zero) frame construction

- Inspect overscan-subtracted bias frames. If there is repeatable structure in these, construct a superbias frame by combining overscan-subtracted bias frames:
  - Using image arithmetic
  - Using IRAF: zerocombine
  - Note that there are multiple options for combining stacks of frames, to avoid contamination by outliers, resulting biases, noise minimization, etc: mean, median, max-reject, min-max reject, sigma clipping, etc. Median is a simple algorithm that is fairly robust if not perfectly optimal.
- Note that any noise in your superbias frame will be propagated to every image you reduce, hence the desire to combine many individual bias frames, and only to use a superbias if there is repeatable structure to subtract!

Flat field construction

- You will need to construct separate flat fields for each filter/configuration that you use
- Flat fields should be normalized before combining to account for variations in lamp/sky brightness
- Final flat fields should be normalized such that dividing by them does not change the overall mean level significantly, so that noise can still be calculated using the observed number of counts
- Making flats:
  - Using image arithmetic
  - Using IRAF: flatcombine
  - Again, there are many frame combination options.

Exercises

Data reduction

Friday, April 15, 2016

Our goal is to understand all of the steps and issues involved with data reduction and how they may be dealt with when people reduce data, and to try to avoid, as much as possible, "black-box" recipes for reducing data.

To be able to capture the process, it is best if data reduction efforts always be scripted, so that you have a record of what you did, and a resource to look back on the next time you have to do it again!

Your goal is to deliver basic data reduction scripts for the standard stars observed with ARCTIC and DIS

Basic data reduction

- Overscan subtraction
  - Determine overscan region location
  - Determine whether constant overscan (subtraction of a single value) is appropriate, or if not, consider possibilities:
    - Fit to overscan as a function of row
    - Median overscan as a function of raw
  - Remove overscan
    - Using image arithmetic
    - Using IRAF: ccdproc (note overscan options)
- Superbias (zero) frame construction
  - Inspect overscan-subtracted bias frames. If there is repeatable structure in these, construct a superbias frame by combining overscan-subtracted bias frames:
    - Using image arithmetic
    - Using IRAF: zerocombine
    - Note that there are multiple options for combining stacks of frames, to avoid contamination by outliers, resulting biases, noise minimization, etc: mean, median, max-reject, min-max reject, sigma clipping, etc. Median is a simple algorithm that is fairly robust if not perfectly optimal.
  - Note that any noise in your superbias frame will be propagated to every image you reduce, hence the desire to combine many individual bias frames, and only to use a superbias if there is repeatable structure to subtract!
- Flat field construction
  - You will need to construct separate flat fields for each filter/configuration that you use
  - Flat fields should be normalized before combining to account for variations in lamp/sky brightness
  - Final flat fields should be normalized such that dividing by them does not change the overall mean level significantly, so that noise can still be calculated using the observed number of counts. Don't want to change numbers much because want to measure uncertainty on brightness later
  - Making flats:
    - Using image arithmetic
    - Using IRAF: flatcombine
    - Again, there are many frame combination options.

Basic spectroscopic calibration

1. normal CCD processing: overscan, (bias, dark). (Note that Triplespec is not a CCD, so requires normal IR detector processing: dark/bias subtraction).
2. flat fielding. Note problem that dome flats have spectral energy distribution of light source. "Flatten" the flats in the wavelength direction to preserve error analysis, i.e. remove the large scale wavelength dependence, but preserve the pixel-to-pixel response variations. In the spatial direction, flat fielding is like imaging, but often the requirements on accuracy are less stringent. An extra spatial component in the flats comes from variation of slit width.
3. wavelength calibration. Use arc lamps with known lines. Identify lines, determine line centers (centroid or fitting), and fit function to centers vs. wavelength.
4. flux calibration: correction for throughput as a function of wavelength. Not always required, e.g. if measuring strengths relative to nearby continuum. Spectrophotometric standards, e.g. Massey et al. ApJ 328, 315 (1988). If fluxing is performed, usually also want to correct for atmospheric extinction as a function of wavelength and airmass: use of mean extinction coefficients.
5. Object reduction: extracting object spectrum ("tracing" the object) and sky spectrum. Aperture extraction vs. optimal extraction. Caveats: spectral curvature.
6. Advanced topics: nod and shuffle, atmospheric feature correction (esp in IR).

IRAF utilities

IRAF: response and doslit

- load specred package:

```
iraf.imred()
iraf.specred()
```

- response takes out the observed flat field response in the wavelength direction (which is a combination of the flat field SED and the spectrograph response)
- doslit is the "meta" task that does wavelength calibration, flux calibration, and object extraction for point sources
  - Images must be run through CCDPROC first (or have CCDPROC flag in header).
  - For the arc list, be aware that the `.fits` should not be included in the file name, it is automatically added (with imtype = fits)

Individual commands (instead of doslit doing the whole thing):

- apall: marks apertures and does the extraction
- for arc: apall arc ref=object (from above marked) inter- backg- recen- trace-
- identify: m to mark 2 lines, f to quick fit, q, l to identify more lintes, f to refit (:func cheb :order 3 to change function), d to delete lines. Reidentify can be used to id lines on subsequent spectra with similar wavelength calibration
- refspec on each object file, reference=arcname (may need to remove sort key)
- dispcor: applies wavelength solution to extracted spectrum, linearizes if requested

Scripting issues

Different people/packages have different preferences for handling issues involved with scripting data reduction. In particular, a set of images taken on a given night is generally divided among different types: flat field frames (in different filters/configurations), bias frames, wavelength calibration frames, object frames (in different filters/configurations), etc., and these need to be handlede differently.

One way of handling this is to try to extract all of the relevant information from file headers. This requires that the data acquisition software put the appropriate information there, and that the user specifies things in such a way to guarantee the information is correct, or subsequently edits it so that it is.

IRAF: instrument files, setinst command, hselect comment, hedit command

Alternatively, one might just prepare some standard input files that list frames of a given type.

Finally, one might just build into a script the appropriate files to use for each step of the reduction process.

Exercises