ASTR 535 Lab notes

Jon Holtzman

Spring 2016

Time, coordinate systems, observability tools

Time Systems

Systems of time: see Naval observatory reference for a full listing of different types of time.

Solar Time

Time tied to position of Sun; based on amount of time it takes for the sun to return to the same position in the sky (aka days). Note the distinction between *mean* solar time (clock time) and *apparent* solar time (sundial, the "equation of time" and the analemma).

Most used solar time is Universal time. UT = local mean solar time at Greenwich = "Zulu". Tied to location of Sun, but average to "mean sun".

Local time: accounts for longitude of observer. For practicality, legal time is split into time zones.

In detail, official time is kept by atomic clocks (International Atomic Time, or TAI), and coordinated UT (UTC) is atomic time with leap seconds added to compensate for changes in earth's roation, where these are added to keep UTC within a second of solar time (UT1). See here for some details.

Sidereal time

Times based on position of stars, i.e. Earth's sidereal rotation period $\sim$ 23h 56m 4s. Local sidereal time is GMST (Greenwich mean sidereal time) minus longitude. At the vernal equinox (time in sky when Sun crosses the celestial equator as its declination is increasing), sidereal time = UT. Difference between UT and GMST is one rotation (day) over the course of a year, so about 2 hours per month.

Sideral is relevant for position of stars: stars come back to the same position every sidereal day. As we'll see below, a given star crosses the meridian when the local sidereal time equals the right ascension of the star.

Calendars

Standard calendar is Gregorian, with leap years, etc.

For astronomy, it is simpler to keep track of days rather then year/month/day. Most dates given by the Julian date (number of days since UT noon, Monday, January 1, 4713 BC). Variations include modified Julian data (JD - 2400000.5 fewer digits and starts at midnight), heliocentric Julian date (JD adjusted to the frame of reference of the Sun, so can differ by up to 8.3 minutes). Heliocentric JD is the amount of time it would take a pulse of light to arrive at the sun.

Note that repeating events are often described as an event *ephemeris:* $t_i(event) = t_0 + i(period)$.

The term *ephemeris* is also used to describe how the position of an object changes over time, e.g. planetary ephemerides.

Coordinate systems

LPL website on astronomical coordinate systems

Celestial coordinate systems

(diagram)

- RA-DEC: tied to Earth rotation, longitude and latitude. Zero RA at vernal equinox
- ecliptic: tied to plane of Earth rotation around the Sun. Zero ecliptic longitude tied to vernal equinox.
- galactic: tied to plane of the Milky Way

At vernal equinox, RA = 12h crosses the meridian at midnight.

Note that for a celestial coordinate system tied to the Earth's rotation, coordinates of an object change over time because of the changing direction of the Earth's axis: precession and nutation. Because of this, coordinates are always specified for some reference equinox: J2000/FK5, B1950, etc.; if using coordinates to point a telescope, you need to account for this (but generally, telescope software does this on its own). Note distinction between equinox and epoch, where the latter is relevant for objects that move (which everything does at some level).

Transformations between systems straightforward from spherical trigonometry.

Note the common usage of an Aitoff projection (equal areas) of the sky in celestial coordinates, with location of ecliptic and galactic plane. Software tools (Python, projection="aitoff" in subplot, IDL: aitoff and aitoff_grid in Astronomy users library).

Local coordinate systems

- Equatorial: HA-dec. $HA = LST - \alpha$. $LST = GMST - longitude$. Note normal convention for HA is to get larger to the west, i.e. opposite of RA. Objects at zenith have δ = latitude of observer.
- Horizon: alt-az or zd-az

Local coordinates are important for pointing telescopes. Note that there are various other effects that one has to consider for pointing a telescope at a source of known celestial position: proper motion, precession, nutation, "aberration of light", parallax, atmosphereic refraction.

Finding positions of celestial objects

- SIMBAD: look up coordinates of many objects outside solar system by name, etc., also provides much other reference information.
- VizieR catalog database Database of astronomical catalogs, with search and download possibilities.
- NED: NASA extragalactic database: galaxies, etc.
- solar system ephemerides: JPL HORIZONS

Orientations of objects in the sky

Usually specified by position angle: angle of object in degrees from NS line, measured counterclockwise.

An important observational position angle for spectroscopy: *parallactic angle*, the position angle of the line from zenith to horizon.

Observability

In general, one would like to observe objects through the shortest possible path through Earth's atmosphere, i.e., when they are *transiting* (crossing the meridian, HA=0). The more atmosphere the light goes through, the more losses due to atmospheric absorption/scattering (more severe at shorter wavelengths), and the more image degradation from atmospheric seeing. Of course, it doesn't make sense to wait for an object to transit if you don't have anything else to do in the meantime; efficient use of telescope time is the primary concern. One *airmass* is the amount of air directly above an observer. If you are looking at the zenith, you are looking through one airmass. Generally, most observers attempt to observe at airmasses less than 2, i.e. within 60 degrees of zenith. Once you hit an airmass of 3, the object is rapidly setting (except at very high declination). Of course, for some solar system objects (objects near the sun), one has no choice but to observe at high airmass.

Note that HA gives some indication of observability, but that higher declination objects can be observed to higher HA than lower declination objects. Roughly, at the celestial equator, an HA of 3 hours is about an airmass of 2, and in many cases, one doesn't want to go much lower in the sky.

Another issue with observability has to do with the Moon, since it is harder to see fainter objects when the sky is brighter. Moon brightness is related to its phase, and to a lesser extent, to distance from your object. Of course, if the Moon is below the horizon, it does not have an effect. So for planning observations of faint objects, one also has to consider Moon phase and rise/set times. Note that the sky brightness from the Moon is a function of wavelength, and at IR wavelengths, it is not a very significant contributor to the total sky brightness; so often, telescopes spend bright time working in the IR.

Tools

Here are some useful software tools to do tasks related to coordinate systems and observability, though there are others out there. Anything that accomplishes the desired tasks adequately is fine to use; just make sure you're not limited by the tools that you choose. These are available on the Astronomy Linux cluster; you can probably install them on your laptop, but they will probably not be there by default.

- skycalc/skycalendar: text based programs, installed on our Linux cluster (link is to source code if you wish to install on your laptop). skycalendar gives daily almanac, position of moon, etc. skycalc allows you to enter coordinates of an object and obtain observability information for any specified date. Other features included as well: coordinate transformation, position of planets.
- JSkyCalc: (java-jar /home/local/java/JSkyCalc.jar): JAVA implementation of skycalc, also installed on the Astronomy cluster (and available for download).
- WCSTOOLS: full set of useful coordinate system programs, e.g. coordinate system transformation (command skycoor). Largely useful for use with coordinate system information in image headers (more later). Installed on the astronomy cluster.
- Python: astropy.coordinates, IDL: euler in Astronomy users library.

Exercises

1. **Predict the RA crossing the meridian at midnight for the first of every month. Try the command skycalendar (on the cluster, unless you download it yourself for your laptop) - give yourself a wide terminal window first - to see how well you did**
2. **What time is it now? What is the sidereal time? What coordinates would it be most optimal to observe right now?**
3. **When are the dark (no moon above horizon) first half nights in first quarter ?**

4. **APO schedules the 3.5m in half-night blocks (A and B), split at midnight (or 1am during daylight savings). What are the best half-nights in the next year (month and half, e.g., Oct A, March B, etc.) to request to observe:**

   - **Virgo cluster of galaxies (note central galaxy is M87, look up the coordinates)**
   - **Galactic center (galactic coordinates are .... ask if you don't know!). You can use command skycoor (or Python or IDL tools) to convert galactic to equatorial (skycoor with no arguments gives syntax).**
   - **Jupiter (look up its position using JPL HORIZONS)**

5. **Run skycalc (choose observatory A for APO, ? gives list of command help, look at r, d, y, and h commands). For the galactic center, what is the maximum amount of time it can be observed at an airmass of less than 2.5? How about the Virgo cluster? Why are these different?**

6. **Run jskycalc. Play with all of the buttons! What planets will be visible spring 2016, and at what times of night? Note that you can load files with a list of coordinates, and you can make airmass observability charts for them.**

7. **Start to outline plan for an 3 half-night observing run during late March A halves, when we are taking our APO trip. Eventually, the plan should include a list of objects for each night with a tentative order of observation, taking into account how much time needs to be spent on each object. Our projects are still TBD, but will likely include observations with multiple instruments.**

   - **Determine the approximate range of RAs that we will be able to observe.**
   - **Given the NMSU 1st quarter proposals, which of them might we be able to make some observations for?**
   - **If you have other ideas for projects, start to tabulate them. (Sten/Diane stars for APOGEE calibration/neutron capture calibration, Triplespec RR Lyrae RV curves Drew Be stars)**
   - **Start to prepare a joint web page with the plan, including relevant information: coordinates of objects, finder images if necessary, links to tabulated spectra, instrument manuals, etc. etc.**

8. **Look up the catalog Globular Clusters in the Milky Way in VizieR and download it (make sure to get all of the rows).**

   - **Plot the locations in an Aitoff projection of equatorial coordinates. Can you detect Galactic structure?**
   - **What clusters would be possible to observe during our March run?**
   - **Convert coordinates to galactic coordinates and plot in an Aitoff projection.**

Friday, February 5

Image display and graphical file-based display tools

Image Display

Much astronomical data is in the form of 2D images. It is critical to understand how to display such data and be able to see all of the information it contains. This is an issue because in most cases, the data will contain more information that can be displayed on a screen at any one time. There are two issues: spatial resolution and, probably more importantly, *dynamic range*.

Spatial resolution

Note that many modern detectors have larger pixel dimensions than many computer displays. This means that it's not possible to see all of the pixels at one time; you can either see a subframe of the entire image at full spatial resolution, or the entire image at reduced spatial resolution; generally software does reduced spatial resolution by displaying every other, every 3rd, every 4th, etc. pixel value, so it is possible to miss features.

Dynamic range: brightness and contrast

Most image displays provide only 8-bits of display range in intensity, giving only 256 possible intensities; the human eye can't distinguish many more with any reliability. Most astronomical images can have up to 16-bits of dynamic range, 256 times more levels. Any image with more dynamic range must somehow be compressed into 8-bits before it can be displayed. This can be done by:

- sampling the true image coarsely (in intensity), which allows viewing of the whole dynamic range but can lead to the apparent loss of intensity detail
- fully sampling only a part of the true image range, which leads to the loss of ability to view detail outside the chosen range.

or by something in between. Most packages will use some default algorithm to make this choice automatically, so you have to be careful to understand what is being done, and what information might be lost in what you are looking at. Any decent display package will give you control over how to display the image, and you need to understand in detail how you can see different things in images when you display them in different ways. To be able to choose reasonable display parameters, you will need to know something about the intensity values in your images, so most display packages will allow you to directly see pixel intensities. This is also useful so you can make sure that the values are somewhere around the levels that you expect.

Image scaling parameters are generally specified by a low and a high data value (or a low value and a range) which give the limits in the true data which will be scaled into 8-bits. In old-imaging parlance, the *brightness* is set by the choice of value that will correspond to the darkest pixel, and the *contrast* is set by the difference between the darkest and lightest pixel.

Common choices for automatic scaling might be to display an image such that the pixel with the lowest data value in an image will appear black, and the pixel with the highest data value will appear white; this is sometimes called 100% scaling. However, many images can have defects which might appear as very low or very high data values, so often this choice will set display parameters non-optimally. Alternative autoscaling might be determined from the low and high data values of the middle 99% of the data values (i.e. exclude the 0.5% lowest and highest data values), or 98%, etc.

To change the display scaling factors, the data values must be rescaled and the image redisplayed. On modern machines, this is generally still quite fast. However, there is a faster way to *partially* get the same result, see below.

nonlinear scalings

Note you can also use a nonlinear scaling to sample a larger (or smaller) range. Example: logarithmic, square root scaling, asinh scaling.

color maps and pseudocolor

Once an intensity subsection is chosen, it can be displayed with any choice of "color map", which specifies the colors to be assigned to each of the display levels. These can be various shades of grey (greyscale) or some other color, or some arbitrary color scheme (pseudo-color). Note that most packages allow the user to manipulate the color table, allowing users to change the contrast and brightness of a displayed subsection; for this reason, it is usually reasonable to chose a range with a significantly larger range than 256 data values.

Most packages will allow the user to inspect individual data values based on a cursor location. Beware, however, of packages which give data readout based on scaling parameters and 8-bit display number only: these are unable to give correct values outside of the scaled region of the image.

The color map is implemented at a lower level and can generally be changed very rapidly. One use of this is to "stretch" or "roll" the color map to change the brightness and/or the contrast in the image.

true color images

True color images obviously require information about colors of the objects in the picture, so they cannot be made from an image taken through a single filter. Generally, three independent filters are used to create true color images, e.g. RGB images. The image in each individual filter must be properly scaled if one wants to make the true color image match what would be seen with the eye, i.e. correct white balance.

One can also use images in multiple filters to construct "pseudo-true" color images, e.g. emission line regions in one color, continuum in another, etc.

other display functions

Other useful display tools include zoom, blink, interactive image analysis (peak, valley, fwhm, etc), marking of objects, etc.

Quick introduction to astronomical image file format

FITS format. Two parts in one file: header plus data. Header contains ASCII information, data is in binary format. Be aware that headers must conform to specific lengths: don't use an editor on a FITS file. Headers have a small amount of required information, plus there are lots of possibilities for optional information.

Standalone display tools

- DS9: standalone display tool, but also most commonly used display tool with IRAF (an astronomical image processing package).
- XIMTOOL: another display tool that can be used with IRAF.
- GAIA: also includes image processing routines.

These are all installed on the Astronomy cluster; you should be able to install them on your laptop via the links above if you want to.

Of course, any image processing package will generally include a display tool as part of the package, and we will use these extensively. But, in dicussing principles of image display, perhaps it's best to start with standalone display tools. These can be very useful for quick-look analysis.

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

Exercises

Astronomical image processing: Introduction and basics

Friday, February 12, 2016

Various software packages have been developed for astronomical image processing, e.g.:

- IRAF. In particular, note PYRAF Python interface
- IDL (astronomy users library)
- XVISTA
- GAIA
- FIGARO
- MIDAS
- AIPS
- Add-on packages: STSDAS, PROS, DAOPHOT, ...

Pros and cons: availability, cost, GUI/command line, data handling (disk vs. memory), speed, ease of use (e.g., keywords vs. parm files), language and access to existing code, ability to add new code, scripts/procedures (internal control language).

Image processing package as a tool: tools can be incredibly useful, but sometimes significant investment in understanding/learning your tool really increases its utility. But also, in the long run, it's a tool, and you shouldn't be limited in what you choose to do by the tool you are comfortable with, so always keep open the possibility of other tools, or improving the capability of a tool.

What should you learn? These days, many instruments require rather involved tasks for reducing data. Often, the instrument team or observatory supplies routines (in some package) for doing these tasks. Generally, it is may be easier to use these routines rather than reprogram them using your favorite tool. So you are probably in the position of having to be comfortable with multiple tools, but you should also probably take the time to become an expert in at least one.

An alternative way to look at things is that to be at the forefront, you will likely be working with new instruments and/or new techniques. Using standard analysis may be unlikely to take the most advantage, or even work at all, with new data. So you want to be in the position of having the flexibility to develop tools yourself.

There are several programming environments that make it fairly simple to work with astronomical data. Here, we'll provide an introduction to two of the more popular environments in the US: Python (especially 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 iamges and basic CCD data reduction

CCD introduction and principles of operations

Photoelectric effect in a semiconductor. Photons excite photoelectrons, which are kept localized by electronics on the chip. Note that "input" is number of photons, "output" is number of electrons, which are related by the sensitivity (quantum efficiency) of the pixel.

Charge sensing (readout) by charge transfer through the array, first vertically to the serial register, then horizontally into the readout electronics.

Electronics: multiply input electrons by a gain factor (to optimize dynamic range), add a bias level (to avoid negative input), convert to digital via an A/D converter. "Input" is electrons, "output" is counts (also known as DN, or ADU).

Note that the bias level can vary with time/temperature, so in general, the bias level must be measured on *each individual exposure*. This is typically achieved by reading several "dummy" pixels after each row is read, where these "dummy" pixels act to record the current bias level. This leads to a set of columns of "dummy" pixels at the right-hand edge of every image, called the overscan. The overscan is used to derive the bias value for the frame. (Note in some cases, the bias level can actually vary during the course of the readout, in which case more sophisticated handling is required).

The physical architecture of CCDs leads to specific terminology: rows, columns, serial registers, overscan, underscan.

Note pixel-to-pixel sensitivity variations, and variation of sensitivity with wavelength.

Basic calibration

Basic calibration: bias level subtraction (subtract out bias level that was added), flat field division (compensates for pixel-to-pixel sensitivity variations).

Other possible calibration: bias pattern subtraction, dark subtraction, shutter shading division, fringe correction

Calibration Data

Calibration data: obtaining biases, darks, flats. Need for multiple exposures for noise reduction (biases and darks), outlier suppression (cosmic rays, stars in sky flats). Note issues with source of flat fields: how flat are they?

Creating calibration frames: combining images

creating superbias, superdark, flat field: combining images, including normalization for flats

The best estimator of parent population mean in a least-squares, maximum likelihood sense, is the sample mean. However, the sample mean is not especially robust in the case of outliers. Outliers occur in lots of astronomical contexts, e.g., cosmic rays, filtering of stars, or just bad data. Combining images while doing the best job of rejecting outliers is a critical part of many data reduction/analysis tasks.

Some more robust estimators: median, but it produces larger error of the mean (about 25% larger ($1.253\sigma/\sqrt{n}$ for normal distribution) than does the mean. Often can make use of a priori knowledge about outliers: e.g., stars and CRs are always positive. This leads to routines like maximum-pixel rejection. However, this leads to biases for all pixels without outliers. Hence, a better technique is min-max rejection; even this still leads to biases in pixels which have an outlier, and really throws away signal on others. Probably the best bet is to do n-sigma rejection, then recomputation of the mean. Problem here is that estimator of sigma can be very biased in the presence of outliers; may work better if you compute both mean and variance from the sample with the maximum value removed). Alternatively, apply using error model; for example, compute sigma from the median value and a noise model, use this to reject outliers, then average the remaining data points. Be aware of issues trying to reject stars, e.g., in twilight flats: there will always be some point in the profile at which your rejection will fail if done on a pixel-by-pixel basis.

If you don't have exposures at a common intensity, you need to normalize and potentially, consider the effect of different noise levels.

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)

IDL tools

Astronomical image processing packages: IRAF basics

Friday, February 26... maybe The Image Reduction and Analysis Facility (IRAF) is a suite of software developed by NOAO in the 1980's. It provides an environment for the reduction and analysis of astronomical data that is widely used, especially in the US astronomical community. However, there are certainly a number of astronomers who find the IRAF approach somewhat cumbersome or opaque, and who prefer to develop their own tools for data reduction. Nonetheless, some familiarity at least with IRAF tools is probably a very good idea.

IRAF has been incorporated into a more modern interface with the development of PYRAF, which is a Python front-end to the IRAF routines. In this day and age, use of IRAF through this interface, as opposed to the traditional CL interface, is *strongly* recommended.

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.

You can use pyraf through a front-end interpreter to emulate the original IRAF command-line interface, using the command: pyraf. This is convenient for previous users and for some tasks, but "hides" the Python interpreter and its power

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

IRAF contains many programs for astronomical analysis. These are grouped into *packages*, and individual commands are *tasks* within each package. Before a particular task can be run, the package within which it is located must be loaded, which is done by entering the package name. Several packages are loaded by default, and this can be customized in the login.cl file. If you know a task name, and need to find out what package it can be found in, try the *iraf.apropos('task')* command (*apropos task* in IRAF interpreter)

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

Most tasks have a set of adjustable values, or parameters, which govern the specifics of how the task operates. IRAF manages the values of these parameters by having an individual *parameter file* for each task. You can look at and modify the parameter values for a given task using the *iraf.epar('taskname')(epar taskname)* command, or *iraf.lpar('taskname')* to just print a list. This is useful for seeing what all of possible parameters are. If you modify parameters using *epar*, the modifications are saved and will be used in all future invocations of the task. Because of this behavior, it is useful, and perhaps even wise, to know how to reset parameters back to their default values: this can be accomplished using the *iraf.unlearn('taskname') (unlearn taskname)* command.

You can get detailed information about each task and its parameters using the *iraf.help('taskname') (help tasknamera)* command.

You can override a value from the parameter file by specifying it as a keyword in the commande, e.g. valname=xxxx. In this case, the parameter file is not permanently modified. This is convenient for scripting. On the command line, keywords are separated from the command and each other by spaces.

In the Python interface, parameters can also be accessed/modified as attributes of the task

IRAF is a **disk-based** system: commands that work with images require filenames of input images and filenames for output images.

You can issue operating system commands from within IRAF command language by starting the command lines with '!'

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. See plot window commands ('?')

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.