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 $\delta =$ 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

Quick introduction to astronomical image file format

Standalone display tools

Basic display operation

Exercises

Friday, February 26

Astronomical image processing packages: Introduction and IRAF basics

IRAF/DS9 basic operation

- One time only: run `mkiraf`, which creates `login.cl`. 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`
    You will then need to use standard PYTHON styntax, rather than the old IRAF `cl` syntax.
  - `pyraf` actually runs a front-end interpreter to emulate the original IRAF command-line interface. 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

1 Astronomical image processing: Introduction and basics

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

- IRAF (links and such)

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.

1.1 Getting started with Python

1.1.1 Basics

- Start python using `ipython -matplotlib`
- Python works with *objects*. All objects can have 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

1.1.2 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 data from initial HDU
  - `hd[0].header` is header from initial HDU
- ASCII:
  - `from astropy.io import ascii`
  - `a=ascii.read(`*`filename`*`)` returns Table with columns.

1.1.3 Image statistics

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

1.1.4 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])` for a plot along column 100
- `plt.plot(hd[0].data[500,:])` for a plot along row 500

Histogram

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

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_default/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_np2arr(hd[0].data)  # display from numpy array
  d.set("scale limits 400 500")  (sets display range)
  ```

  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

Exercises