Use Cases of Anaconda

SURP 2023 Python Bootcamp

Ohio State Astronomy

Slides by: Joy Bhattacharyya

scipy

- Optimization (scipy.optimize)
- Integration (scipy.integrate)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fft)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Spatial data structures and algorithms (scipy.spatial)
- Statistics (scipy.stats)
- Multidimensional image processing (scipy.ndimage)

astropy

- Units and Quantities (astropy.units)
- Constants (astropy.constants)
- Data Tables (astropy.table)
- Time Series (astropy.timeseries)
- Astronomical Coordinate Systems (astropy.coordinates)
- World Coordinate System (astropy.wcs)
- FITS File Handling (astropy.io.fits)
- Cosmological Calculations (astropy.cosmology)
- Astrostatistics Tools (astropy.stats)

astropy.units

- handles defining, converting between, and performing arithmetic with physical quantities
- also handles logarithmic units such as magnitude and decibel.
- Quantity objects: the combination of a value and a unit.

```
>>> from astropy import units as u
>>> 42.0 * u.meter
<Quantity  42. m>
>>> [1., 2., 3.] * u.m
<Quantity [1., 2., 3.] m>
>>> import numpy as np
>>> np.array([1., 2., 3.]) * u.m
<Quantity [1., 2., 3.] m>
```

```
>>> q = 42.0 * u.meter
>>> q.value
42.0
>>> q.unit
Unit("m")
```

```
>>> x = 1.0 * u.parsec
>>> x.to(u.km)
<Quantity 30856775814671.914 km>
```

```
>>> u.m / u.m
Unit(dimensionless)
```

Unit	Description
adu	adu
AU	astronomical unit
beam	beam
bin	bin
chan	chan
ct	count (ct)
DN	dn (DN)
earthMass	Earth mass
earthRad	Earth radius
electron	Number of electrons
jupiterMass	Jupiter mass
jupiterRad	Jupiter radius
Jy	Jansky: spectral flux density
lsec	Light second
lyr	Light year
pc	parsec: approximately 3.26 light-years.
ph	photon (ph)
R	Rayleigh: photon flux
Ry	Rydberg: Energy of a photon whose wavenumber is the Rydberg constant
solLum	Solar luminance
solMass	Solar mass
solRad	Solar radius

G	6.6743e-11	**m3 / (kg s2)**	GRAVITATIONAL CONSTANT
N_A	6.02214076e+23	**1 / (mol)**	AVOGADRO’S NUMBER
R	8.31446262	**J / (K mol)**	GAS CONSTANT
Ryd	10973731.6	**1 / (m)**	RYDBERG CONSTANT
a0	5.29177211e-11	**m**	BOHR RADIUS
alpha	0.00729735257		FINE-STRUCTURE CONSTANT
atm	101325	**Pa**	STANDARD ATMOSPHERE
b_wien	0.00289777196	**m K**	WIEN WAVELENGTH DISPLACEMENT LAW CONSTANT
c	299792458	**m / (s)**	SPEED OF LIGHT IN VACUUM
e	1.60217663e-19	**C**	ELECTRON CHARGE
eps0	8.85418781e-12	**F/m**	VACUUM ELECTRIC PERMITTIVITY
g0	9.80665	**m / s2**	STANDARD ACCELERATION OF GRAVITY
h	6.62607015e-34	**J s**	PLANCK CONSTANT
hbar	1.05457182e-34	**J s**	REDUCED PLANCK CONSTANT
k_B	1.380649e-23	**J / (K)**	BOLTZMANN CONSTANT
m_e	9.1093837e-31	**kg**	ELECTRON MASS
m_n	1.6749275e-27	**kg**	NEUTRON MASS
m_p	1.67262192e-27	**kg**	PROTON MASS
mu0	1.25663706e-06	**N/A2**	VACUUM MAGNETIC PERMEABILITY
muB	9.27401008e-24	**J/T**	BOHR MAGNETON
sigma_T	6.65245873e-29	**m2**	THOMSON SCATTERING CROSS-SECTION
sigma_sb	5.67037442e-08	**W / (K4 m2)**	STEFAN-BOLTZMANN CONSTANT
u	1.66053907e-27	**kg**	ATOMIC MASS

GM_earth	3.986004e+14	**m3 / (s2)**	NOMINAL EARTH MASS PARAMETER
GM_jup	1.2668653e+17	**m3 / (s2)**	NOMINAL JUPITER MASS PARAMETER
GM_sun	1.3271244e+20	**m3 / (s2)**	NOMINAL SOLAR MASS PARAMETER
L_bol0	3.0128e+28	**W**	LUMINOSITY FOR ABSOLUTE BOLOMETRIC MAGNITUDE 0
L_sun	3.828e+26	**W**	NOMINAL SOLAR LUMINOSITY
M_earth	5.97216787e+24	**kg**	EARTH MASS
M_jup	1.8981246e+27	**kg**	JUPITER MASS
M_sun	1.98840987e+30	**kg**	SOLAR MASS
R_earth	6378100	**m**	NOMINAL EARTH EQUATORIAL RADIUS
R_jup	71492000	**m**	NOMINAL JUPITER EQUATORIAL RADIUS
R_sun	695700000	**m**	NOMINAL SOLAR RADIUS
au	1.49597871e+11	**m**	ASTRONOMICAL UNIT
kpc	3.08567758e+19	**m**	KILOPARSEC
pc	3.08567758e+16	**m**	PARSEC

astropy.coordinates

- classes for representing a variety of celestial/spatial coordinates and their velocity components
- SkyCoord objects are instantiated by passing in positions with specified units and a coordinate frame (default: ICRS)

```
>>> from astropy import units as u
>>> from astropy.coordinates import SkyCoord
>>> c = SkyCoord(ra=10.625*u.degree, dec=41.2*u.degree, frame='icrs')
```

```
>>> c = SkyCoord(ra=10.68458*u.degree, dec=41.26917*u.degree)
>>> c.ra
<Longitude 10.68458 deg>
>>> c.ra.hour
0.7123053333333335
>>> c.ra.hms
hms_tuple(h=0.0, m=42.0, s=44.299200000000525)
>>> c.dec
<Latitude 41.26917 deg>
>>> c.dec.degree
41.26917
>>> c.dec.radian
0.7202828960652683
```

- tools for converting between systems in a uniform way.
- on-sky and 3D separations between two coordinates can be computed

```
>>> c_icrs = SkyCoord(ra=10.68458*u.degree, dec=41.26917*u.degree, frame='icrs')
>>> c_icrs.galactic
<SkyCoord (Galactic): (l, b) in deg
    (121.17424181, -21.57288557)>
```

```
>>> c1 = SkyCoord(ra=10*u.degree, dec=9*u.degree, distance=10*u.pc, frame='icrs')
>>> c2 = SkyCoord(ra=11*u.degree, dec=10*u.degree, distance=11.5*u.pc, frame='icrs')
>>> c1.separation_3d(c2)
<Distance 1.52286024 pc>
```

```
>>> c1 = SkyCoord(ra=10*u.degree, dec=9*u.degree, frame='icrs')
>>> c2 = SkyCoord(ra=11*u.degree, dec=10*u.degree, frame='fk5')
>>> c1.separation(c2)  # Differing frames handled correctly
<Angle 1.40453359 deg>
```

astropy.wcs

Convert between astronomical and pixel coordinates

```
>>> from astropy.io import fits
>>> from astropy.wcs import WCS
>>> from astropy.utils.data import get_pkg_data_filename
>>> fn = get_pkg_data_filename('data/j94f05bgq_flt.fits', package='astropy.wcs.tests')
>>> f = fits.open(fn)
>>> w = WCS(f[1].header)
>>> sky = w.pixel_to_world(30, 40)
>>> print(sky)
<SkyCoord (ICRS): (ra, dec) in deg
    (5.52844243, -72.05207809)>
>>> f.close()
```

astropy.table

- store and manipulate heterogeneous tables of data
- specify a description, units, and output formatting for columns.
- perform Table Operations like database joins, concatenation, and binning.

```
>>> from astropy.table import QTable
>>> import astropy.units as u
>>> import numpy as np

>>> a = np.array([1, 4, 5], dtype=np.int32)
>>> b = [2.0, 5.0, 8.5]
>>> c = ['x', 'y', 'z']
>>> d = [10, 20, 30] * u.m / u.s

>>> t = QTable([a, b, c, d],
...            names=('a', 'b', 'c', 'd'),
...            meta={'name': 'first table'})
```

```
>>> t
<QTable length=3>
  a      b     c      d
                    m / s
int32 float64 str1 float64
```

```
>>> t.info
<QTable length=3>
name  dtype   unit  class
---- ------- ----- --------
   a   int32         Column
   b float64         Column
   c    str1         Column
   d float64 m / s Quantity
```

astropy.io.fits

- provides access to FITS files
- inspect and modify header
- read image and spectrum files as numpy arrays
- read and edit tables

```
>>> from astropy.io import fits
>>> fits_image_filename = fits.util.get_testdata_filepath('test0.fits')

>>> hdul = fits.open(fits_image_filename)
```

```
>>> hdul.info()
Filename: ...test0.fits
No.    Name         Ver    Type        Cards   Dimensions   Format
  0  PRIMARY          1 PrimaryHDU      138   ()
  1  SCI              1 ImageHDU         61   (40, 40)    int16
  2  SCI              2 ImageHDU         61   (40, 40)    int16
  3  SCI              3 ImageHDU         61   (40, 40)    int16
  4  SCI              4 ImageHDU         61   (40, 40)    int16
```

```
>>> hdr = hdul[0].header
>>> hdr['targname'] = ('NGC121-a', 'the observation target')
>>> hdr['targname']
'NGC121-a'
>>> hdr.comments['targname']
'the observation target'
```

```
>>> data = hdul[1].data
```

```
>>> data.shape
(40, 40)
>>> data.dtype.name
'int16'
```

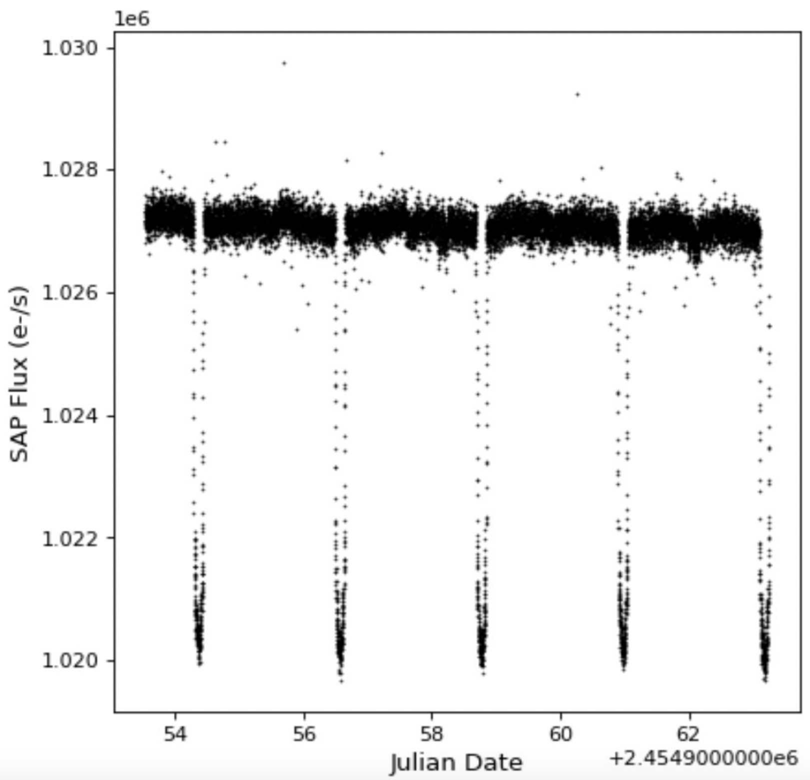
astropy.timeseries

- QTable subclasses that have special columns to represent times using the Time class.
- provide time series-specific functionality above and beyond Qtable- binning, folding and periodogram.

```
>>> from astropy.timeseries import TimeSeries
>>> ts = TimeSeries.read(filename, format='kepler.fits')
```

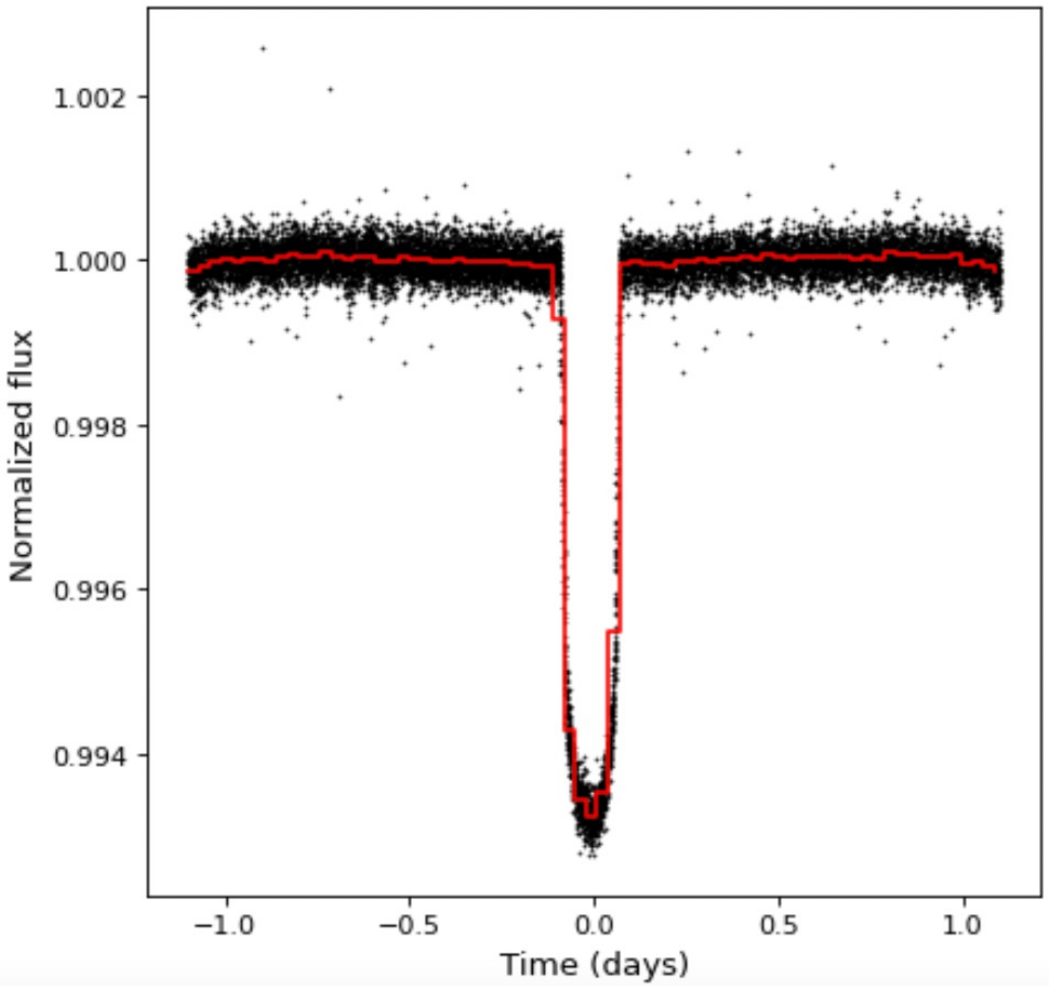
```
import matplotlib.pyplot as plt
plt.plot(ts.time.jd, ts['sap_flux'], 'k.', markersize=1)
plt.xlabel('Julian Date')
plt.ylabel('SAP Flux (e-/s)')
```

(png, svg, pdf)



```
plt.plot(ts_folded.time.jd, ts_folded['sap_flux_norm'], 'k.', markersize=1)
plt.plot(ts_binned.time_bin_start.jd, ts_binned['sap_flux_norm'], 'r-', drawstyle='steps
plt.xlabel('Time (days)')
plt.ylabel('Normalized flux')
```

(png, svg, pdf)



astropy.cosmology

- predefined available cosmologies-

Name	Source	H0	Om	Flat
WMAP1	Spergel et al. 2003	72.0	0.257	Yes
WMAP3	Spergel et al. 2007	70.1	0.276	Yes
WMAP5	Komatsu et al. 2009	70.2	0.277	Yes
WMAP7	Komatsu et al. 2011	70.4	0.272	Yes
WMAP9	Hinshaw et al. 2013	69.3	0.287	Yes
Planck13	Planck Collab 2013, Paper XVI	67.8	0.307	Yes
Planck15	Planck Collab 2015, Paper XIII	67.7	0.307	Yes
Planck18	Planck Collab 2018, Paper VI	67.7	0.310	Yes

- User defined FLRW-like cosmology can be created
- distances, ages, and lookback times corresponding to a measured redshift
- Finding the redshift at a given value of a cosmological quantity

```
>>> from astropy.cosmology import WMAP9 as cosmo
>>> cosmo.H(0)
<Quantity 69.32 km / (Mpc s)>
```

```
>>> cosmo.comoving_distance(np.array([0.5, 1.0, 1.5]))
<Quantity [1916.06941724, 3363.07062107, 4451.7475201 ] Mpc>
```

```
>>> from astropy.cosmology import FlatLambdaCDM
>>> cosmo = FlatLambdaCDM(H0=70, Om0=0.3, Tcmb0=2.725)
>>> cosmo
FlatLambdaCDM(H0=70.0 km / (Mpc s), Om0=0.3, Tcmb0=2.725 K,
              Neff=3.04, m_nu=[0. 0. 0.] eV, Ob0=None)
```

```
>>> cosmo.luminosity_distance(4)
<Quantity 35842.353618623194 Mpc>
```

```
>>> import astropy.units as u
>>> from astropy.cosmology import Planck13, z_at_value
>>> z_at_value(Planck13.age, 2 * u.Gyr)
<Quantity 3.19812061 redshift>
```

astropy.stats

binom_conf_interval	Binomial proportion confidence interval given k successes, n trials.
poisson_conf_interval	Poisson parameter confidence interval given observed counts
median_absolute_deviation	Calculate the median absolute deviation (MAD).
signal_to_noise_oir_ccd	Computes the signal to noise ratio for source being observed in the optical/IR using a CCD.
bootstrap	Performs bootstrap resampling on numpy arrays.
cdf_from_intervals	Construct a callable piecewise-linear CDF from a pair of arrays.
interval_overlap_length	Compute the length of overlap of two intervals.
histogram_intervals	Histogram of a piecewise-constant weight function.
fold_intervals	Fold the weighted intervals to the interval (0,1).
sigma_clip	Perform sigma-clipping on the provided data.
sigma_clipped_stats	Calculate sigma-clipped statistics on the provided data.
jackknife_resampling	Performs jackknife resampling on numpy arrays.
jackknife_stats	Performs jackknife estimation on the basis of jackknife resamples.
rayleightest	Performs the Rayleigh test of uniformity.
vonmisesmle	Computes the Maximum Likelihood Estimator (MLE) for the parameters of the von Mises distribution.
bayesian_blocks	Compute optimal segmentation of data with Scargle's Bayesian Blocks
histogram	Enhanced histogram function, providing adaptive binnings
calculate_bin_edges	Calculate histogram bin edges like numpy.histogram_bin_edges.
bayesian_info_criterion	Computes the Bayesian Information Criterion (BIC) given the log of the likelihood function evaluated at the estimated parameters, the number of parameters, and the number of samples.
akaike_info_criterion	Computes the Akaike Information Criterion (AIC)