Astroutils user manual

Toni Sagristà Sellés
Astronomisches Rechen-Institut
Zentrum für Astronomie Heidelberg
UNIVERSITÄT HEIDELBERG
Heidelberg, Baden-Württemberg, Deutschland

March 17, 2014

Abstract

The Python astroutils package provides a set of tools useful for astronomy programs, such as coordinate and reference system conversions, IO of astronomical data from and to different sources and targets, etc. This document thoroughly describes the package along with its strengths and drawbacks. It also provides some simple examples on how to use its functionalities.

1 Introduction

The original aim of this package was to aid in the preparation of data from ESA's Gaia space mission for their visualisation in planetarium domes. It started as a straightforward module which loaded some data from a database using the TAP client in GAVO's VOTable python package, transformed it and wrote it to an output file. Then, in order to make it a bit more flexible it grew larger and larger until one could basically specify the desired output columns in any desired units and the module would do all the conversions and transformations for you. In the process, the astroutils package was created. It is a collection of utilities and functions that do common astronomical calculations, conversions and transformations.

2 Package description

This section of the document will go through all the submodules and explain their functionality with several comprehensive examples. If you are looking for a complete API reference you can always use `pydoc` in the following fashion:

```
> pydoc astroutils.units
```

```
Help on module astroutils.units in astroutils:

NAME
    astroutils.units

FILE
    /home/tsagrista/Workspaces/workspace/VisualisationData/astroutils/units.py

DESCRIPTION
    Small submodule that provides convenient unit conversion interface.
    :Author:
        Toni Sagrista Selles
    :Organization:
        Astronomisches Rechen-Institut - Zentrum fur Astronomie Heidelberg - UNIVERSITAT HEIDELBERG
    :Version:
        0.1

    Requirements
    ------------
    * `Astropy 0.3+ <http://www.astropy.org>`_

FUNCTIONS
    convert(unit0, unit1, value)
        Converts the given value from unit0 to uni1 using astropy
```

Let us now go over all the modules in the astroutils package in a bit of detail.

Core coordinates algorithms (astroutils.coordinatescore)

The *astroutils.coordiantescore* module contains the core functions to transform between coordinates and between reference systems. It specifically has the functionality to transform from cartesian to spherical polar coordinates and vice-versa, for both positions and proper motions. Also, it has methods to transform between the commonly used astronomical reference systems, namely equatorial, ecliptic and galactic.

To use the core algorithms for coordinate transformations you can import the module as follows.

```
>>> import astroutils.coordinatescore as cc
```

Let's begin with some basic functionality to transform from cartesian coordinates to spherical polar coordinates.

```
>>> x, y, z = 10, 0, 0
>>> cc.cartesian_to_spherical([x, y, z])
[0.0, 0.0, 10.0]
```

In this example we transform a point at (10, 0, 0) in cartesian coordinates and get it back in spherical polar coordinates; the longitude $\varphi = 0\,rad$, the latitutde $\theta = 0\,rad$ and the distance $r = 10$. Please note that in our cartesian reference system the x axis points to the origin of polar coordinates.

```
>>> cartesian = [0, 10, 0]
>>> spherical = cc.cartesian_to_spherical(cartesian)
>>> print spherical
[1.5707963267948966, 0.0, 10.0]
```

Here we see that the y axis is orthogonal to x, for we can see its direction has a longitude of a $\frac{\pi}{2}$, 90 degrees and no latitude.

```
>>> cartesian = [0, 0, 10]
>>> spherical = cc.cartesian_to_spherical(cartesian)
>>> print spherical
[0.0, 1.5707963267948966, 10.0]
```

Finally, we see that the z axis points north in an orthogonal direction of the xy plane. -z points, then, south.

Let's now see some other examples.

```
>>> cartesian = [12, 33, -5]
>>> spherical = cc.cartesian_to_spherical(cartesian)
>>> print spherical
[1.2220253232109897, -0.14144210916329877, 35.4682957019364]
>>> cartesian = cc.spherical_to_cartesian(cc.cartesian_to_spherical(cartesian))
>>> print cartesian
[12.0, 33.0, -5.0]
```

Here we go from cartesian to spherical and then back to cartesian. We'll se now that we can also convert velocities to proper motions and back.

```
>>> cartesian = [5, 2, 9]
>>> cart_vel = [1, -3, -2.2]
>>> sph = cc.cartesian_to_spherical_pm(cartesian + cart_vel)
>>> print "Spherical coordinates: %s" % sph[:3]
Spherical coordinates: [0.3805063771123649, 1.031589733535445,
    10.488088481701515]
>>> print "Spherical proper motion: %s" % sph[3:]
Spherical proper motion: [-0.5862068965517243, -0.0709018563948743,
    -1.9832021856308324]
```

Here we go from an object at position (5, 2, 9) with the cart_vel velocity vector to its position in spherical coordinates, proper motions in longitude and latitude and radial velocity.

Now let's see how we can convert between astronomical reference systems. We can get the transformation matrices from one referencystem to another just by invoking the functions with the signature refys1_to_refys2().

```
>>> print cc.galactic_to_equatorial()
[[-0.06451124 0.48891344 -0.86994364 0.     ]
 [-0.87221785 -0.45117528 -0.18888328 0.     ]
 [-0.48484464 0.74659528 0.45554491 0.       ]
 [ 0.         0.         0.         1.       ]]
>>> print cc.equatorial_to_ecliptic()
[[ 1.         0.         0.         0.       ]
```

```
 [ 0.          0.91748213 0.39777699 0.        ]
 [ 0.         -0.39777699 0.91748213 0.        ]
 [ 0.          0.         0.         1.        ]]
```

Astroutils implements all the transformations between the equatorial, the ecliptic and the galactic systems. If you need to transform your cartesian coordinates from one system to another, you need to use the methods with the _v suffix, as indicated in the following example.

```
>>> eq = [10, 10, 10]
>>> gal = cc.equatorial_to_galactic_v(eq)
>>> print gal
[-14.215737256691948, 7.843334430945757, -6.032820174101103]
```

In the same fashion, using the _sph suffix, one can convert spherical coorinates, for example we can get the galactic l and b from the equatorial α and δ.

```
>>> alpha=266.41
>>> delta=-28.94
>>> import math
>>> gal = cc.equatorial_to_galactic_sph([math.radians(alpha),
    math.radians(delta)])
>>> l, b = gal
>>> print "l=%f, b=%f" %(l, b)
l=0.005997, b=-0.007792
```

In this last example we have entered the approximate equatorial coordinates of the galactic center ($\alpha = 266.41°, \delta = -28.94°$) and we have transformed them to galactic coordinates. Obviously, the galactic longitude l and latitude b are very very close to zero, for the galactic center is the origin of polar coordinates in the galactic reference system.

Astronomical coordinate systems wrapper (astroutils.coordinates)

We have included for the sake of simplicity a high-level wrapper of the *coordinatescore* module that is much easier to use for the end user. it is the *astroutils.coordinates* module. The idea is similar to the *astropy.coordinates* package (http://docs.astropy.org/en/stable/coordinates/index.html) but it is not as flexible from the user perspective, even though it includes more functionality, such as ecliptic coordinates and proper motions and cartesian velocities handling. To import the package, just do as follows.

```
>>> import astroutils.coordinates as coord
```

You can of course give it whatever name you want. Now, we have the coordinates as class instances and we can initialize them with either cartesian or spherical positions. Also, we can transform from one class to the other very easily.

```
>>> eq = coord.Equatorial(ra=266.41, dec=-28.94, distance=10, unit=('deg',
    'deg', 'pc'))
```

```
>>> eq._cart
[-0.5479723599091653, -8.734096330837025, -4.838934549715959]
>>> eq._sph
[4.649731660238094, -0.505098285527159, 10]
>>> eq.get(x='km')
{'x': -16908660262353.826}
>>> eq.get(alpha='rad')
{'alpha': 4.649731660238094}
>>> eq.get(alpha='mas')
{'alpha': 959075999.9999999}
>>> eq.get(x='km', y='pc', z='lyr')
{'y': -8.734096330837025, 'x': -16908660262353.826, 'z': -15.782493647313924}
>>> eq.get_basic('alpha', 'rad')
4.649731660238094
```

Now, transforming coordinates between reference systems is as easy as invoking the transform_to() method passing a string with the desired reference system, being it 'galactic', 'ecliptic' or 'equatorial'.

```
>>> ecl = eq.transform_to('ecliptic')
>>> print ecl
ecliptic (longitude=4.657 rad, latitude=-0.097 rad, dist=10.000 pc)
>>> print ecl.to_string(coord._cartesian)
ecliptic (x=-0.548 pc, y=-9.938 pc, z=-0.965 pc)
```

And there's more. We can also drop in velocities and proper motions. When we do so, the calls get a bit larger.

```
>>> gal = coord.Galactic(l=10, b=60, dist=30, mul=24.3e15, mub=1.3e-6,
    radialvelocity=-13, unit=('deg','deg','pc','mas/a','rad/s','km/s'))
>>> print gal
galactic (longitude=0.175 rad, latitude=1.047 rad, dist=30.000 pc)
  vel (xdot=3.73316490 rad/s, ydot=0.00000130 rad/s, zdot=-13.00000000 km/s)
>>> print gal.to_string(coord._cartesian)
galactic (x=14.772 pc, y=2.605 pc, z=25.981 pc)
  vel (xdot=-16.12514287 km/s, ydot=54.01802697 km/s, zdot=-11.25831075 km/s)
```

And of course we can also convert it to the other reference systems like we did before.

```
>>> eq = gal.transform_to('equatorial')
>>> print eq
equatorial (longitude=3.847 rad, latitude=0.222 rad, dist=30.000 pc)
  vel (xdot=1.03794025 rad/s, ydot=1.48931041 rad/s, zdot=-13.00000000 km/s)
```

units

This package is just a wrapper around the unit conversion functionality of Astropy. I started with my own implementation supporting some basic units. It worked pretty well, but then I stumbled upon Astropy.units package which supports a lot more units than I could never have dreamt of. Some use examples follow.

```
>>> import astroutils.units as u
>>> u.convert('km', 'pc', 1e23)
3240779289.4697552
>>> u.convert('pc/s', 'm/s', 1)
30856775814671916.0
>>> u.convert('rad/a', 'deg/s', 1)
1.8155936925837933e-06
>>> u.convert('rad', 'km', 1)
ERROR: UnitsException: 'rad' (angle) and 'km' (length) are not convertible
    [astropy.units.core]
```

3 References and acknowledgements

The astroutils package uses a set of third-party libraries that have proven very reliable and useful.

- Christoph Gohlke's transformations library. A library providing matrix transformations and quaternions handling. http://www.lfd.uci.edu/gohlke/.
- Astropy's unit conversion. http://www.astropy.org.
- Numpy 1.7 http://www.numpy.org.
- VOTable by GAVO http://vo.ari.uni-heidelberg.de/soft/subpkgs