AstroLib Coords Documentation

Release 0.39 (2009-01-06)

Vicki Laidler

June 09, 2014

CONTENTS

Contents:

CHAPTER

ONE

README

1.1 Purpose

This package aims to provide much of the IDL "astron" functionality that pertains to coordinate manipulations in an OO framework. Our target user is a typical astronomer who needs to analyze data, work with catalogs, prepare observing proposals, and prepare for observing runs.

The initial version will provide simple functionality for working with positions in the same reference frame, without having to worry about units.

1.2 Dependencies

numpy

pytpm – a Python wrapper for the TPM library graciously contributed by Jeff Percival

1.3 Examples

```
>>> import astrolib.coords as C
>>> print C.__version__
0.39
```

Unit conversions

```
>>> ob = C.Position('12:34:45.34 -23:42:32.6')
>>> ob.hmsdms()
'12:34:45.340 -23:42:32.600'
>>> ob.dd()
(188.68891666666667, -23.709055555555555)
>>> ob.rad()
(3.2932428578545374, -0.41380108198269777)
```

Angular separations

```
>>> p1 = C.Position("01:23:45.300 +65:43:31.240")
>>> p2 = C.Position("01:23:45.62 +65:43:31.20")
>>> p1.angsep(p2)
0.000548 degrees
>>> delta = p1.angsep(p2)
>>> delta.arcsec()
```

```
1.973739377865491
>>> p1.within(p2, 3.0, units='arcsec')
True
>>> epsilon = C.AngSep(5.0)
>>> epsilon
5.000000 arcsec
>>> delta > epsilon
False
```

Astronomical Date specifications

```
>>> d = C.AstroDate('1997.3') # Defaults to Julian year; J or B prefix also ok
>>> d.year
1997.3
>>> d.jd
2450558.8250000002
>>> d.mjd
50558.325000000186
>>> d2 = C.AstroDate('MJD50658.25') # JD also ok for plain Julian Date
>>> d2.year
1997.5735797399041
>>> d2 < d
False
```

Coordinate conversions

```
>>> ob.j2000()
(188.68891666666667, -23.709055555555555)
>>> ob.b1950()
(188.03056480942405, -23.433637283819877)
>>> ob.galactic()
(298.01638938748795, 39.003358150874568)
>>> ob.ecliptic(timetag=C.AstroDate('J2000'))
(197.5848634558852, -18.293964120804738)
>>> p3 = C.Position("01:23:45 -65:43:21.4",equinox='J2000')
>>> p4 = C.Position("01:23:45 -65:43:21.4",equinox='B1950')
>>> p3.j2000()
(20.9375, -65.722611111111107)
>>> p4.j2000()
(21.356870704681981, -65.462921080444147)
>>> p3.angsep(p4)
0.312199 degrees
>>> p5 = C.Position((0.0,0.0),system='galactic')
>>> p5.j2000()
(266.40499571858879, -28.936169261309555)
```

Specify position in hmsdms

```
>>> polaris = C.Position("02:31:49.08 +89:15:50.8")
>>> polaris.dd()
(37.954500000000003, 89.264111111111106)
>>> polaris.hmsdms()
'02:31:49.080 +89:15:50.800'
>>> print polaris.details()
System: celestial
Equinox: j2000
```

Specify position in decimal degrees

```
>>> ob = C.Position((52.9860209, -27.7510006))
>>> ob.hmsdms()
'03:31:56.645 -27:45:03.602'
>>> ob.dd()
(52.9860209, -27.751000600000001)
```

Use as calculator without saving the intermediate object

```
>>> C.Position("12:34:45.4 -22:21:45.4").dd()
(188.68916666666667, -22.362611111111111)
```

1.4 TPM Citation

Investigators using this software for their research are requested to explicitly acknowledge "use of the TPM software library, by Jeffrey W. Percival" in any appropriate publications.

1.5 See Also

http://www.scipy.org/AstroLibCoordsHome

CHAPTER

TWO

COORDS.POSITION.POSITION

Position object to manage coordinate transformations.

class `astrolib.coords.position.`**`Position`**(*input*, *equinox='J2000'*, *system='celestial'*, *units='degrees'*)

The basic class in the coords library. The Position class is designed to permit users to define a position and then access many representations of it.

Attributes

input		The input used to create the Position.
units	{'degrees', 'radians'}	Unit in which the coords were specified.
equinox	string	Equinox at which the coordinates were specified.
system	{'celestial', 'galactic', 'ecliptic'}	
_tpmstate	integer	The TPM state of the position.
coord	`Coord`	A "smart" representation of the position.
_internal	(float, float)	The internal representation of the position (decimal degrees).

Parameters input : string (hh:mm:ss.ss +dd:mm:ss.sss) or tuple of numbers (dd.ddd, dd.ddd)

Coordinates of the position.

equinox : string

Equinox in which the coords are specified.

system : string

celestial, galactic, ecliptic, etc

units : {'degrees', 'radians'}

`angsep`(*other*)

Computes the angular separation (great circle distance) between two Positions.

Parameters other : another `Position`

Returns ans : `AngSep`

Angular separation.

`b1950`(*timetag=None*)

Return the position in Mean FK4 B1950 coordinates.

Parameters timetag : `AstroDate`

Timetag of returned coordinate.

Returns r, d : (float, float)

Tuple of RA and DEC in decimal degrees.

dd()

Returns _internal : (float, float)

Position in decimal degrees.

details()

Returns ans : string

Formatted `system` and `equinox` for printing.

ecliptic(*timetag=None*)

Return the position in IAU 1980 Ecliptic coordinates.

Parameters timetag : `AstroDate`

Timetag of returned coordinate.

Returns r, d : (float, float)

Tuple of (le,be) in decimal degrees.

galactic(*timetag=None*)

Return the position in IAU 1958 Galactic coordinates.

Parameters timetag : `AstroDate`

Timetag of returned coordinate.

Returns l, b : (float, float)

Tuple in decimal degrees.

hmsdms()

Returns value : string

Position in hms dms.

j2000(*timetag=None*)

Return the position in Mean FK5 J2000 coordinates.

Parameters timetag : `AstroDate`

Timetag of returned coordinate.

Returns r, d : (float, float)

Tuple of RA and DEC in decimal degrees.

rad()

Returns r1, r2 : (float, float)

Position in radians.

tpmstate(*endstate*, *epoch=None*, *equinox=None*, *timetag=None*)

This method allows the expert user to call the blackbox routine of the TPM library directly, for access to more state transitions than are supported in this interface. Little documentation is provided here because it is assumed you know what you are doing if you need this routine.

Parameters endstate : integer

As defined by the TPM state machine.

epoch : float

Epoch in Julian date; default is J2000.

equinox : float

Equinox in Julian date; default is `_tpmequinox`.

timetag : `AstroDate`

Timetag of returned coordinate.

Returns x2, y2 : (float, float)

Transformed coordinates in decimal degrees.

within (*other*, *epsilon*, *units='arcsec'*)

Check if `other` is within `self`.

Parameters other : another `Position`

epsilon : `AngSep` or number

Angular separation.

units : {'arcsec', 'degrees'}

Unit of `epsilon`, if it is specified as a number.

Returns ans : boolean

`True` if `other` is within `epsilon` of `self`.

2.1 Utility Functions

`astrolib.coords.position.`**dms** (*number*)

Convert from decimal to sexagesimal degrees,minutes,seconds.

Parameters number : number

Returns sign, dd, mm, ss : (string, int, int, float)

Sign, degrees, minutes, and seconds.

`astrolib.coords.position.`**gcdist** (*vec1*, *vec2*)

Input (ra,dec) vectors in radians; output great circle distance in radians.

Parameters vec1, vec2 : number

Position in radians.

Returns ans : number

Great circle distance in radians.

References

http://wiki.astrogrid.org/bin/view/Astrogrid/CelestialCoordinates

`astrolib.coords.position.`**hav** (*theta*)

haversine function, units = radians. Used in calculation of great circle distance.

Parameters theta : number

Angle in radians.

Returns ans : number

`astrolib.coords.position.`**`ahav`**`(`*x*`)`

archaversine function, units = radians. Used in calculation of great circle distance.

Parameters x : number

Returns ans : number

Angle in radians.

COORDS.POSITION.COORD

class `astrolib.coords.position.`**`Coord`**

General class for subclasses.

A Coord is distinct from a Position by being intrinsically expressed in a particular set of units.

Each Coord subclass knows how to parse its own input, and convert itself into the internal representation (decimal degrees) used by the package.

3.1 Subclasses

class `astrolib.coords.position.`**`Degrees`**(*input*)

Decimal degrees coord.

Attributes

a1, a2	float	Longitude and latitude in decimal degrees.

Parameters input : (float, float)

Coordinates in decimal degrees.

class `astrolib.coords.position.`**`Hmsdms`**(*input*)

Sexagesimal coord: longitude in hours of time (enforced).

Attributes

a1, a2	Numpy[int,int,float]	Longitude and latitude in hours, minutes, seconds

Parameters input : string

Coordinates as hh:mm:ss.sss +dd:mm:ss.sss (sign optional).

class `astrolib.coords.position.`**`Radians`**(*input*)

Radians coord.

Attributes

a1, a2	float	Longitude and latitude in radians.

Parameters input : (float, float)

Coordinates in radians.

CHAPTER

FOUR

COORDS.ANGSEP

Angular separation between two `Position`.

Can be defined by hand, or produced by "subtracting" two positions. Like `Position`, it will have an internal representation, and a variety of user-selected representations.

class `astrolib.coords.angsep.`**`AngSep`**(*value*, *units='arcsec'*)

The AngSep class lets the user compare the angular separation between different `Position` without having to think too much about units.

Each AngSep object is created with a particular length in a particular set of units. These are then converted into an internal representation which is used for all math and comparisons.

All unit checks are performed by checking the first few letters of the unit string, to provide more flexibility for the user. E.g., "arc", "arcs", "arcsec", and "arcseconds" will all evaluate to arcseconds.

All math and comparisons can be done either between two AngSep objects, or between an AngSep object and a number. In the latter case, the number is assumed to have the same units as the AngSep object.

Examples

Default units are arcsec

```
>>> a = angsep.AngSep(5)
>>> a
5.000000 arcsec
```

The usual arithmetic works

```
>>> b = angsep.AngSep(3)
>>> a+b
8.000000 arcsec
>>> a*3
15.000000 arcsec
```

Use AngSep together with `Position`

```
>>> p1 = P.Position('12:34:45.23 45:43:21.12')
>>> p2 = P.Position('12:34:47.34 45:43:23.0')
>>> sep = p1.angsep(p2)
>>> eps = angsep.AngSep(30,units='arcsec')
>>> p1.within(p2,eps)
True
>>> p2.within(p1,20)
False
```

Note: Angular Separations are inherently positive: negative separations have no physical meaning, and are forbidden.

Attributes

value	number	Magnitude of the angular separation.
units	string	Units in which the magnitude is expressed (arcsec, degrees, or radians).
_internal	number (degrees)	Internal representation of the separation.

Parameters value : number

Magnitude of the angular separation.

units : string

Arcsec or degrees.

Raises ValueError :

If value < 0. Negative separations are physically meaningless and thus forbidden.

approx (*other*, *epsilon*)

`True` if `self` and `other` are equal to within `epsilon`, which is considered to have the same units as `self`.

Note: This is not implemented as 'abs(self-other)<epsilon' because of the prohibition on negative separations.

Parameters other, epsilon : `AngSep` or number (units of `self`)

Returns ans : boolean

arcsec ()

Returns value : float

Separation in arcsec.

degrees ()

Returns value : float

Separation in degrees.

radians ()

Returns value : float

Separation in radians.

setunits (*units*)

Sets the units of the public representation, and converts the publically visible value to those units

Parameters units : {'radians', 'arcsec', 'degrees'}

CHAPTER

FIVE

COORDS.ASTRODATE

For more information about astronomical date specifications, consult a reference source such as this page provided by the US Naval Observatory.

Constants and formulae in this module were taken from the `times.h` include file of the `tpm` package by Jeff Percival, to ensure compatibility.

class `astrolib.coords.astrodate.`**`BesselDate`**(*datespec*)

Attributes

year	float	Decimal Besselian year
jd	float	Julian date
mjd	float	Modified Julian Date
datespec		Date specification as entered by the user

`jyear`()

Return the julian year using the already-converted julian date.

Returns ans : float

Decimal Julian year

class `astrolib.coords.astrodate.`**`JulianDate`**(*datespec*)

Attributes

year	float	Decimal Julian year.
jd	float	Julian date.
mjd	float	Modified Julian Date
datespec		Date specification as entered by the user

`byear`()

Return Besselian year based on previously calculated julian date.

Returns ans : float

Decimal Besselian year.

`astrolib.coords.astrodate.`**`AstroDate`**(*datespec=None*)

AstroDate can be used as a class for managing astronomical date specifications (despite the fact that it was implemented as a factory function) that returns either a BesselDate or a JulianDate, depending on the properties of the datespec.

AstroDate was originally conceived as a Helper class for the Position function for use with pytpm functionality, but also as a generally useful class for managing astronomical date specifications.

The philosophy is the same as Position: to enable the user to specify the date once and for all, and access it in a variety of styles.

Todo

1. Add math functions! Addition, subtraction.

2. Is there a need to support other date specifications? eg FITS-style dates?

Parameters datespec : string, float, integer, `datetime.datetime`, or `None`

Returns value : `JulianDate` or `BesselDate`

Date specification as entered by the user. Permissible specifications include:

- **Julian year**
  - 'J1997', 'J1997.325', 1997.325
  - Return a JulianDate
- **Besselian year**
  - 'B1950','B1958.432'
  - Return a BesselDate
- **Julian date**
  - 'JD2437241.81', '2437241.81', 2437241.81
  - Return a JulianDate
- **Modified Julian date**
  - 'MJD37241.31'
  - Return a JulianDate
- **`datetime.datetime` object**
  - Assume input time is UTC
  - Return a JulianDate
- **`None`**
  - Return the current time as a JulianDate

Raises ValueError :

Raises an exception if the date specification is a string, but begins with a letter that is not 'B','J','JD', or 'MJD' (case insensitive).

`astrolib.coords.astrodate.`**`byear2jd`**(*byear*)

Parameters byear : float

Decimal Besselian year.

Returns value : float

Julian date.

`astrolib.coords.astrodate.`**`jd2jyear`**(*jd*)

Parameters jd : float

Julian date.

Returns value : float

Decimal Julian year.

`astrolib.coords.astrodate.`**`jyear2jd`**(*jyear*)

Parameters jyear : float

Decimal Julian year.

Returns value : float

Julian date.

`astrolib.coords.astrodate.`**`utc2jd`**(*utc*)

Convert UTC to Julian date.

Conversion translated from TPM modules `utcnow.c` and `gcal2j.c`, which notes that the algorithm to convert from a gregorian proleptic calendar date onto a julian day number is taken from The Explanatory Supplement to the Astronomical Almanac (1992), section 12.92, equation 12.92-1, page 604.

Parameters utc : `datetime.datetime` object

UTC (Universal Civil Time).

Returns value : float

Julian date to the nearest second.

5.1 Global Variables

`astrolib.coords.astrodate.`**`B1950`** **= 2433282.42345905**

float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

`astrolib.coords.astrodate.`**`J2000`** **= 2451545.0**

float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

`astrolib.coords.astrodate.`**`CB`** **= 36524.21987817305**

float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

`astrolib.coords.astrodate.`**`CJ`** **= 36525.0**

float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

`astrolib.coords.astrodate.`**`MJD_0`** **= 2400000.5**

float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

CHAPTER

SIX

COORDS.PYTPM_WRAPPER

This routine wraps the `pytpm.blackbox` routine in order to apply the longitude convention preferred in coords. All `astrolib.coords` routines should call `blackbox` instead of `pytpm.blackbox`.

Since pytpm is itself a wrapper for the TPM library, the change could have been made there; but the modulo operator in C only works on integers, so it was simpler to do it in python. Also, this leaves pytpm itself as a more transparent wrapper for TPM.

`astrolib.coords.pytpm_wrapper.`**`blackbox`**(*x*, *y*, *instate*, *outstate*, *epoch*, *equinox*, *timetag=None*)

Parameters x, y : float

Position in decimal degrees.

instate, outstate : int

The TPM states of the position.

epoch : float

Epoch of the position.

equinox : float

Equinox of the position.

timetag : `AstroDate`

Timetag of returned coordinate.

Returns r, d : float

Converted coordinate.

6.1 Global Variables

`astrolib.coords.pytpm.`**`b1950`** **= 2433282.42345905**

float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

`astrolib.coords.pytpm.`**`j2000`** **= 2451545.0**

float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

`astrolib.coords.pytpm.`**CJ = 36525.0**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

`astrolib.coords.pytpm.`**CB = 36524.21987817305**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

`astrolib.coords.pytpm.`**s01 = 1**
int(x=0) -> int or long int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

`astrolib.coords.pytpm.`**s02 = 2**
int(x=0) -> int or long int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

`astrolib.coords.pytpm.`**s03 = 3**
int(x=0) -> int or long int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

`astrolib.coords.pytpm.`**s04 = 4**
int(x=0) -> int or long int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

`astrolib.coords.pytpm.`**s05 = 5**
int(x=0) -> int or long int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

`astrolib.coords.pytpm.`**`s06 = 6`**

int(x=0) -> int or long int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

a

PYTHON MODULE INDEX

a

INDEX