ai.cs Documentation

Release

Alexey Isavnin

Dec 18, 2017

Contents

AI.CS is the coordinates transformation package in Python. It offers functionality for converting data between geometrical coordinates (cartesian, spherical and cylindrical) as well as between geocentric and heliocentric coordinate systems typically used in spacecraft measurements. The package currently also supports rotations of data by means of rotation matrices. Transformations between spacecraft coordinate systems are implemented as a Python binding to the CXFORM library.

CHAPTER 1

Getting started

This tutorial will guide you through basic usage of AI.CS.

1.1 Installation

AI.CS is developed for Python 3, so make sure that you have a working isntallation of it. The package is distributed together with C portion of CXFORM library, which is compiled automatically during installation. Thus make sure that you have a functioning compiler in your system, for instance, gcc.

Assuming the above requirements are satisfied install the package with Python package manager:

```
$ pip install ai.cs
```

1.2 Geometrical coordinates

AI.CS ships with functions for conversion between cartesian and spherical coordinates and between cartesian and cylindrical coordinates:

```
import numpy as np
from ai import cs

# cartesian to spherical
r, theta, phi = cs.cart2sp(x=1, y=1, z=1)

# spherical to cartesian
x, y, z = cs.sp2cart(r=1, theta=np.pi/4, phi=np.pi/4)

# cartesian to cylindrical
r, phi, z = cs.cart2cyl(x=1, y=1, z=1)
```

```
# cylindrical to cartesian
x, y, z = cs.cyl2cart(r=1, phi=np.pi/2, z=1)
```

Most of the functions support both scalars and numpy arrays as input:

```
import numpy as np
from ai import cs

# converting spherical spiral from spherical to cartesian coordinates
x, y, z = cs.sp2cart(
    r=np.ones(100),
    theta=np.linspace(-np.pi/2, np.pi/2, 100),
    phi=np.linspace(0, np.pi*6, 100)
)
```

1.3 Spacecraft coordinates

AI.CS provides Python bindings to CXFORM library for conversion between various geocentric and heliocentric cartesian coordinate systems. For example, the code below performs transformation of data from GSE to HEEQ coordinate system:

```
from datetime import datetime
from astropy import units as u
from ai import cs

# converting (0.5, 0.5, 0.5) AU location from GSE to HEEQ at current time
x, y, z = cs.cxform(
    'GSE',
    'HEEQ',
    datetime.now(),
    x=u.au.to(u.m, 0.5),
    y=u.au.to(u.m, 0.5),
    z=u.au.to(u.m, 0.5)
)
```

Both scalars and numpy arrays are supported as input:

```
from datetime import datetime, timedelta
from astropy import units as u
from ai import cs

# converting circular orbit at 1 AU from cylindrical to cartesian coordinates
r = np.ones(365)*u.au.to(u.m, 1)
phi = np.linspace(0, np.pi*2, 365)
z = np.zeros(365)
x_HEE, y_HEE, z_HEE = cs.cyl2cart(r, phi, z)

# converting HEE to HEEQ
x_HEEQ, y_HEEQ, z_HEEQ = cs.cxform(
    'HEE',
    'HEEQ',
    [datetime(2016, 1, 1)+timedelta(days=d) for d in range(365)],
    x=x_HEE,
    y=y_HEE,
```

```
    z=z_HEE
)
```

1.4 Geometrical transformations

Currently AI.CS offers only one type of geometrical transformations - rotations. Rotation is executed by means of 3D transformation matrices for right-handed rotations around X, Y and Z axes:

```
import numpy as np
from ai import cs

# get (3x3) rotation matrix for rotation by pi/4 around X axis
Tx = cs.mx_rot_x(gamma=np.pi/4)
# get (3x3) rotation matrix for rotation by -pi/4 around Y axis
Ty = cs.mx_rot_y(theta=-np.pi/4)
# get (3x3) rotation matrix for rotation by pi/2 around Z axis
Tz = cs.mx_rot_z(phi=np.pi/2)
```

Is is also possible to construct rotation matrices for compound rotations in one shot:

```
import numpy as np
from ai import cs

# get matrix for right-handed rotation around X, Y and Z axes (exactly in this order)
T = cs.mx_rot(theta=np.pi/4, phi=np.pi/4, gamma=np.pi/4)

# get matrix for right-handed rotation around Z, Y and X axes (exactly in this order)
T_reverse = cs.mx_rot_reverse(theta=np.pi/4, phi=-np.pi/4, gamma=-np.pi/4)
# T_reverse effectively reverses the transformation described by T in this case
```

Rotation matrices can be applied to data in cartesian coordinates in the following way:

```
import numpy as np
from ai import cs

# a cube with the side length 2
x = np.array([1, 1, 1, 1, -1, -1, -1, -1])
y = np.array([1, 1, -1, -1, 1, 1, -1, -1])
z = np.array([1, -1, 1, -1, 1, 1, -1, -1])

# rotate cube by pi/4 around each axis
T = cs.mx_rot(theta=np.pi/4, phi=np.pi/4, gamma=np.pi/4)
x, y, z = cs.mx_apply(T, x, y, z)
```

CHAPTER 2

Supported spacecraft coordinates

AI.CS supports the same selection of spacecraft coordinate systems as the original CXFORM library does:

GEI Geocentric Equatorial Inertial, also known as True Equator and True Equinox of Date, True of Date (TOD), ECI, or GCI

J2000 Geocentric Equatorial Inertial for epoch J2000.0 (GEI2000), also known as Mean Equator and Mean Equinox of J2000.0

GEO Geographic, also known as Greenwich Rotating Coordinates (GRC), or Earth-fixed Greenwich (EFG)

MAG Geomagnetic

GSE Geocentric Solar Ecliptic

GSM Geocentric Solar Magnetospheric

SM Solar Magnetic

RTN Radial Tangential Normal (Earth-centered)

GSEQ Geocentric Solar Equatorial

HEE Heliocentric Earth Ecliptic

HAE Heliocentric Aries Ecliptic

HEEQ Heliocentric Earth Equatorial

CHAPTER 3

API documentation

3.1 ai.cs

The module defines various coordinate transformations and related functions: transformations between cartesian, cylindrical and spherical coordinates; rotational matrices; transformations between various geocentric and heliocentric coordinates (Python wrapper for cxform library).

ai.cs.**cart2cyl**(*x*, *y*, *z*)
Converts data in cartesian coordinates into cylyndrical.

Parameters

- **x** (*scalar or array_like*) – X-component of data.
- **y** (*scalar or array_like*) – Y-component of data.
- **z** (*scalar or array_like*) – Z-component of data.

Returns Tuple (r, phi, z) of data in cylindrical coordinates.

ai.cs.**cart2sp**(*x*, *y*, *z*)
Converts data from cartesian coordinates into spherical.

Parameters

- **x** (*scalar or array_like*) – X-component of data.
- **y** (*scalar or array_like*) – Y-component of data.
- **z** (*scalar or array_like*) – Z-component of data.

Returns Tuple (r, theta, phi) of data in spherical coordinates.

ai.cs.**cxform**(*cs_from*, *cs_to*, *dt*, *x*, *y*, *z*)
Performs conversion between various geocentric and heliocentric coordinate systems.

Parameters

- **cs_from** (*str*) – Indentifier of the source coordinate system. Can be one of 'GEI', 'J2000', 'GEO', 'MAG', 'GSE', 'GSM', 'SM', 'RTN', 'GSEQ', 'HEE', 'HAE', 'HEEQ'.

- **cs_to** – Identifier of target coordinate system. Can be one of 'GEI', 'J2000', 'GEO', 'MAG', 'GSE', 'GSM', 'SM', 'RTN', 'GSEQ', 'HEE', 'HAE', 'HEEQ'.
- **dt** (*datetime or array_like of datetime*) – Datetime of the conversion.
- **x** (*scalar or array_like*) – X-component of data.
- **y** (*scalar or array_like*) – Y-component of data.
- **z** (*scalar or array_like*) – Z-component of data.

Returns Tuple (x, y, z) of data in target coordinate system.

ai.cs.**cyl2cart**(*r*, *phi*, *z*)
Converts data in cylindrical coordinates into cartesian.

Parameters

- **r** (*scalar or array_like*) – R-component of data.
- **phi** (*scalar or array_like*) – Phi-component of data.
- **z** (*scalar or array_like*) – Z-component of data.

Returns Tuple (x, y, z) of data in cartesian coordinates.

ai.cs.**mx_apply**(*T*, *x*, *y*, *z*)
Applies rotation to data using rotational matrix.

Parameters

- **T** (*numpy.matrix*) – Rotational matrix.
- **x** (*scalar or array_like*) – X-component of data.
- **y** (*scalar or array_like*) – Y-component of data.
- **z** (*scalar or array_like*) – Z-component of data.

Returns Tuple (x, y, z) of data in cartesian coordinates.

ai.cs.**mx_rot**(*theta*, *phi*, *gamma*)
Returns rotational matrix for compound rotation around X, Y and Z axes. The order of rotation is X-Y-Z.

Parameters

- **theta** (*scalar*) – Rotation angle around Y in radians.
- **phi** (*scalar*) – Rotational angle around in Z radians.
- **gamma** (*scalar*) – Rotational angle around X in radians.

Returns Numpy rotational matrix.

ai.cs.**mx_rot_reverse**(*theta*, *phi*, *gamma*)
Returns rotational matrix for compound rotations around X, Y and Z axes. The order of rotation is Z-Y-X.

Parameters

- **theta** (*scalar*) – Rotational angle around Y in radians.
- **phi** (*scalar*) – Rotational angle around in Z radians.
- **gamma** (*scalar*) – Rotational angle around X in radians.

Returns Numpy rotational matrix.

ai.cs.**mx_rot_x**(*gamma*)
Returns rotational matrix for right-handed rotation around X axis.

Parameters **gamma** (*scalar*) – Rotation angle around X in radians.

Returns Numpy rotational matrix.

ai.cs.**mx_rot_y**(*theta*)
Returns rotational matrix for right-handed rotation around Y axis.

Parameters **theta** (*scalar*) – Rotation angle around Y in radians.

Returns Numpy rotational matrix.

ai.cs.**mx_rot_z**(*phi*)
Returns rotational matrix for right-handed rotation around Z axis.

Parameters **phi** (*scalar*) – Rotation angle around Z in radians.

Returns Numpy rotational matrix.

ai.cs.**sp2cart**(*r*, *theta*, *phi*)
Converts data in spherical coordinates into cartesian.

Parameters

- **r** (*scalar or array_like*) – R-component of data.
- **theta** (*scalar or array_like*) – Theta-component of data.
- **phi** (*scalar or array_like*) – Phi-component of data.

Returns Tuple (x, y, z) of data in cartesian coordinates.

Python Module Index

a

Index