NavPy Documentation

Release 0.1

NavPy Team

February 10, 2017

Contents

This will be NavPy's documentation site. (under construction). Visit later!

Contents:

Module Documenation

This is the documentation for using the individual module functions.

1.1 Coordinate Transformations

1.1.1 Direction Cosine Matrices

navpy.**angle2dcm**(*rotAngle1*, *rotAngle2*, *rotAngle3*, *input_unit='rad'*, *rotation_sequence='ZYX'*, *output_type='ndarray'*)

This function converts Euler Angle into Direction Cosine Matrix (DCM). The DCM is described by three sucessive rotation rotAngle1, rotAngle2, and rotAngle3 about the axis described by the rotation_sequence.

The default rotation_sequence='ZYX' is the aerospace sequence and rotAngle1 is the yaw angle, rotAngle2 is the pitch angle, and rotAngle3 is the roll angle. In this case DCM transforms a vector from the locally level coordinate frame (i.e. the NED frame) to the body frame.

This function can batch process a series of rotations (e.g., time series of Euler angles).

Parameters **rotAngle1, rotAngle2, rotAngle3** : angles {(N,), (N,1), or (1,N)}

They are a sequence of angles about successive axes described by rotation_sequence.

input_unit : {'rad', 'deg'}, optional

Rotation angles. Default is 'rad'.

rotation_sequence : {'ZYX'}, optional

Rotation sequences. Default is 'ZYX'.

output_type : {'ndarray','matrix'}, optional

Output type. Default is 'ndarray'.

Returns **C** : {3x3} Direction Cosine Matrix

Notes

Programmer: Adhika Lie Created: May 03, 2011 Last Modified: January 12, 2016

navpy.**dcm2angle**(*C*, *output_unit='rad'*, *rotation_sequence='ZYX'*)

This function converts a Direction Cosine Matrix (DCM) into the three rotation angles. The DCM is described by three sucessive rotation rotAngle1, rotAngle2, and rotAngle3 about the axis described by the rotation_sequence.

The default rotation_sequence='ZYX' is the aerospace sequence and rotAngle1 is the yaw angle, rotAngle2 is the pitch angle, and rotAngle3 is the roll angle. In this case DCM transforms a vector from the locally level coordinate frame (i.e. the NED frame) to the body frame.

This function can batch process a series of rotations (e.g., time series of direction cosine matrices).

Parameters **C** : {(3,3), (N,3,3), or (3,3,N)}

direction consine matrix that rotates the vector from the first frame to the second frame according to the specified rotation_sequence.

output_unit : {'rad', 'deg'}, optional

Rotation angles. Default is 'rad'.

rotation_sequence : {'ZYX'}, optional

Rotation sequences. Default is 'ZYX'.

Returns **rotAngle1, rotAngle2, rotAngle3** : angles

They are a sequence of angles about successive axes described by rotation_sequence.

Notes

The returned rotAngle1 and 3 will be between +/- 180 deg (+/- pi rad). In contrast, rotAngle2 will be in the interval +/- 90 deg (+/- pi/2 rad).

In the 'ZYX' or '321' aerospace sequence, that means the pitch angle returned will always be inside the closed interval +/- 90 deg (+/- pi/2 rad). Applications where pitch angles near or larger than 90 degrees in magnitude are expected should used alternate attitude parameterizations like quaternions.

1.1.2 ECEF and NED

navpy.**ecef2ned**(*ecef*, *lat_ref*, *lon_ref*, *alt_ref*, *latlon_unit='deg'*, *alt_unit='m'*, *model='wgs84'*)

Transform a vector resolved in ECEF coordinate to its resolution in the NED coordinate. The center of the NED coordiante is given by lat_ref, lon_ref, and alt_ref.

Parameters **ecef** : {(N,3)} input vector expressed in the ECEF frame

lat_ref : Reference latitude, unit specified by latlon_unit, default in deg

lon_ref : Reference longitude, unit specified by latlon_unit, default in deg

alt : Reference altitude, unit specified by alt_unit, default in m

Returns **ned** : {(N,3)} array like ecef position, unit is the same as alt_unit

Examples

```
>>> import numpy as np
>>> from navpy import ecef2ned
>>> lat
```

navpy.**ned2ecef**(*ned*, *lat_ref*, *lon_ref*, *alt_ref*, *latlon_unit='deg'*, *alt_unit='m'*, *model='wgs84'*)

Transform a vector resolved in NED (origin given by lat_ref, lon_ref, and alt_ref) coordinates to its ECEF representation.

Parameters **ned** : {(N,3)} input array, units of meters

lat_ref : Reference latitude, unit specified by latlon_unit, default in deg

lon_ref : Reference longitude, unit specified by latlon_unit, default in deg

alt_ref : Reference altitude, unit specified by alt_unit, default in m

Returns **ecef** : {(N,3)} array like ned vector, in the ECEF frame, units of meters

Notes

The NED vector is treated as a relative vector, and hence the ECEF representation returned is NOT converted into an absolute coordinate. This means that the magnitude of *ned* and *ecef* will be the same (bar numerical differences).

Examples

```
>>> import navpy
>>> ned = [0, 0, 1]
>>> lat_ref, lon_ref, alt_ref = 45.0, -93.0, 250.0 # deg, meters
>>> ecef = navpy.ned2ecef(ned, lat_ref, lon_ref, alt_ref)
>>> print("NED:", ned)
>>> print("ECEF:", ecef)
>>> print("Notice that 'down' is not same as 'ecef-z' coordinate.")
```

1.1.3 ECEF and LLA

navpy.**ecef2lla**(*ecef*, *latlon_unit='deg'*)

Calculate the Latitude, Longitude and Altitude of a point located on earth given the ECEF Coordinates.

Parameters **ecef** : {(N,3)} array like input of ECEF coordinate in X, Y, and Z column, unit is meters

latlon_unit : {('deg','rad')} specifies the output latitude and longitude unit

Returns **lat** : {(N,)} array like latitude in unit specified by latlon_unit

lon : {(N,)} array like longitude in unit specified by latlon_unit

alt : {(N,)} array like altitude in meters

References

[R1]

navpy.**lla2ecef**(*lat*, *lon*, *alt*, *latlon_unit='deg'*, *alt_unit='m'*, *model='wgs84'*)

Convert Latitude, Longitude, Altitude, to ECEF position

Parameters **lat** : {(N,)} array like latitude, unit specified by latlon_unit, default in deg

lon : {(N,)} array like longitude, unit specified by latlon_unit, default in deg

alt : {(N,)} array like altitude, unit specified by alt_unit, default in m

Returns **ecef** : {(N,3)} array like ecef position, unit is the same as alt_unit navpy.**lla2ned**(*lat*, *lon*, *alt*, *lat_ref*, *lon_ref*, *alt_ref*, *latlon_unit='deg'*, *alt_unit='m'*, *model='wgs84'*)

Convert Latitude, Longitude, Altitude to its resolution in the NED coordinate. The center of the NED coordiante is given by lat_ref, lon_ref, and alt_ref.

For example, this can be used to convert GPS data to a local NED frame.

Parameters **lat** : {(N,)} array like latitude, unit specified by latlon_unit, default in deg

lon : {(N,)} array like longitude, unit specified by latlon_unit, default in deg

alt : {(N,)} array like altitude, unit specified by alt_unit, default in m

lat_ref : Reference latitude, unit specified by latlon_unit, default in deg

lon_ref : Reference longitude, unit specified by latlon_unit, default in deg

alt : Reference altitude, unit specified by alt_unit, default in m

Returns **ned** : {(N,3)} array like ecef position, unit is the same as alt_unit

1.1.4 NED and LLA

navpy.**ned2lla**(*ned*, *lat_ref*, *lon_ref*, *alt_ref*, *latlon_unit='deg'*, *alt_unit='m'*, *model='wgs84'*)

Calculate the Latitude, Longitude and Altitude of points given by NED coordinates where NED origin given by lat_ref, lon_ref, and alt_ref.

Parameters **ned** : {(N,3)} array like input of NED coordinate in N, E, and D column, unit is meters

lat_ref : Reference latitude, unit specified by latlon_unit, default in deg

lon_ref : Reference longitude, unit specified by latlon_unit, default in deg

alt_ref : Reference altitude, unit specified by alt_unit, default in m

latlon_unit : {('deg','rad')} specifies the output latitude and longitude unit

Returns **lat** : {(N,)} array like latitude in unit specified by latlon_unit

lon : {(N,)} array like longitude in unit specified by latlon_unit

alt : {(N,)} array like altitude in meters

navpy.**lla2ned**(*lat*, *lon*, *alt*, *lat_ref*, *lon_ref*, *alt_ref*, *latlon_unit='deg'*, *alt_unit='m'*, *model='wgs84'*)

Convert Latitude, Longitude, Altitude to its resolution in the NED coordinate. The center of the NED coordiante is given by lat_ref, lon_ref, and alt_ref.

For example, this can be used to convert GPS data to a local NED frame.

Parameters **lat** : {(N,)} array like latitude, unit specified by latlon_unit, default in deg

lon : {(N,)} array like longitude, unit specified by latlon_unit, default in deg

alt : {(N,)} array like altitude, unit specified by alt_unit, default in m

lat_ref : Reference latitude, unit specified by latlon_unit, default in deg

lon_ref : Reference longitude, unit specified by latlon_unit, default in deg

alt : Reference altitude, unit specified by alt_unit, default in m

Returns **ned** : {(N,3)} array like ecef position, unit is the same as alt_unit

1.2 Attitude Transformations

`navpy.angle2quat`(*rotAngle1*, *rotAngle2*, *rotAngle3*, *input_unit='rad'*, *rotation_sequence='ZYX'*)

Convert a sequence of rotation angles to an equivalent unit quaternion

This function can take inputs in either degree or radians, and can also batch process a series of rotations (e.g., time series of Euler angles). By default this function assumes aerospace rotation sequence but can be changed using the `rotation_sequence` keyword argument.

Parameters **rotAngle1, rotAngle2, rotAngle3** : {(N,), (N,1), or (1,N)}

They are a sequence of angles about successive axes described by rotation_sequence.

input_unit : {'rad', 'deg'}, optional

Rotation angles. Default is 'rad'.

rotation_sequence : {'ZYX'}, optional

Rotation sequences. Default is 'ZYX'.

Returns **q0** : {(N,)} array like scalar componenet of the quaternion

qvec : {(N,3)} array like vector component of the quaternion

Notes

Convert rotation angles to unit quaternion that transfroms a vector in F1 to F2 according to

$v_q^{F2} = q^{-1} \otimes v_q^{F1} \otimes q$

where $\otimes$ indicates the quaternion multiplcation and v_q^F is a pure quaternion representation of the vector v_q^F. The scalar componenet of v_q^F is zero. For aerospace sequence ('ZYX'): rotAngle1 = psi, rotAngle2 = the, and rotAngle3 = phi

Examples

```
>>> import numpy as np
>>> from navpy import angle2quat
>>> psi = 0
>>> theta = np.pi/4.0
>>> phi = np.pi/3.0
>>> q0, qvec = angle2quat(psi,theta,phi)
>>> q0
0.80010314519126557
>>> qvec
array([ 0.46193977,  0.33141357, -0.19134172])
```

```
>>> psi = [10, 20, 30]
>>> theta = [30, 40, 50]
>>> phi = [0, 5, 10]
>>> q0, qvec = angle2quat(psi,theta,phi,input_unit = 'deg')
>>> q0
array([ 0.96225019,  0.92712639,  0.88162808])
>>> qvec
array([[-0.02255757,  0.25783416,  0.08418598],
       [-0.01896854,  0.34362114,  0.14832854],
       [-0.03266701,  0.4271086 ,  0.19809857]])
```

navpy.**dcm2quat**(*C*, *rotation_sequence='ZYX'*)

Convert a DCM to a unit quaternion

Parameters **C** : direction consine matrix that rotates the vector from the first frame

to the second frame according to the specified rotation_sequence. rotation_sequence: {'ZYX'}, optional. Rotation sequences. Default is 'ZYX'.

Returns **q0** : {(N,)} array_like

Scalar componenet of the quaternion

qvec : {(N,3)} array_like

Vector component of the quaternion

Examples

```
>>> import numpy as np
>>> from navpy import dcm2quat
>>> C = np.array([[  9.25570440e-01,   3.36869440e-01,  -1.73581360e-01],
                  [ -3.42051760e-01,   9.39837700e-01,   5.75800000e-05],
                  [  1.63132160e-01,   5.93160200e-02,   9.84972420e-01]])
>>> q0,qvec = dcm2quat(C)
>>> q0
0.98111933015306552
>>> qvec
array([-0.0150997 ,  0.08579831,  0.17299659])
```

navpy.**qmult**(*p0*, *pvec*, *q0*, *qvec*)

Quaternion Multiplications r = p x q

Parameters **p0, q0** : {(N,)} array_like

Scalar componenet of the quaternion

pvec, qvec : {(N,3)} array_like

Vector component of the quaternion

Returns **r0** : {(N,)} array like scalar componenet of the quaternion

rvec : {(N,3)} array like vector component of the quaternion

Examples

```
>>> import numpy as np
>>> from navpy import qmult
>>> p0, pvec = 0.701057, np.array([-0.69034553,  0.15304592,  0.09229596])
>>> q0, qvec = 0.987228, np.array([ 0.12613659,  0.09199968,  0.03171637])
>>> qmult(q0,qvec,p0,pvec)
(0.76217346258977192, array([-0.58946236,  0.18205109,  0.1961684 ]))
>>> s0, svec = 0.99879, np.array([ 0.02270747,  0.03430854, -0.02691584])
>>> t0, tvec = 0.84285, np.array([ 0.19424161, -0.18023625, -0.46837843])
>>> qmult(s0,svec,t0,tvec)
(0.83099625967941704, array([ 0.19222498, -0.1456937 , -0.50125456]))
>>> qmult([p0, s0],[pvec, svec],[q0, t0], [qvec, tvec])
(array([ 0.76217346,  0.83099626]), array([[-0.59673664,  0.24912539,  0.03053588], [ 0.19222498
```

navpy.**quat2angle**(*q0*, *qvec*, *output_unit='rad'*, *rotation_sequence='ZYX'*)

Convert a unit quaternion to the equivalent sequence of angles of rotation about the rotation_sequence axes.

This function can take inputs in either degree or radians, and can also batch process a series of rotations (e.g., time series of quaternions). By default this function assumes aerospace rotation sequence but can be changed using the `rotation_sequence` keyword argument.

Parameters **q0** : {(N,), (N,1), or (1,N)} array_like

Scalar componenet of the quaternion

qvec : {(N,3),(3,N)} array_like

Vector component of the quaternion

rotation_sequence : {'ZYX'}, optional

Rotation sequences. Default is 'ZYX'.

Returns **rotAngle1, rotAngle2, rotAngle3** : {(N,), (N,1), or (1,N)} array_like

They are a sequence of angles about successive axes described by rotation_sequence.

output_unit : {'rad', 'deg'}, optional

Rotation angles. Default is 'rad'.

Notes

Convert rotation angles to unit quaternion that transfroms a vector in F1 to F2 according to

$v_q^{F2} = q^{-1} \otimes v_q^{F1} \otimes q$

where $\otimes$ indicates the quaternion multiplcation and v_q^F is a pure quaternion representation of the vector v_q^F. The scalar componenet of v_q^F is zero. For aerospace sequence ('ZYX'): rotAngle1 = psi, rotAngle2 = the, and rotAngle3 = phi

Examples

```
>>> import numpy as np
>>> from navpy import quat2angle
>>> q0 = 0.800103145191266
>>> qvec = np.array([0.4619398,0.3314136,-0.1913417])
>>> psi, theta, phi = quat2angle(q0,qvec)
>>> psi
1.0217702360987295e-07
>>> theta
0.7853982192745731
>>> phi
1.0471976051067484
```

```
>>> psi, theta, phi = quat2angle(q0,qvec,output_unit='deg')
>>> psi
5.8543122160542875e-06
>>> theta
45.00000320152342
>>> phi
60.000003088824108
```

```
>>> q0 = [ 0.96225019,  0.92712639,  0.88162808]
>>> qvec = np.array([[-0.02255757,  0.25783416,  0.08418598],                    [-0.018968
>>> psi, theta, phi = quat2angle(q0,qvec,output_unit='deg')
>>> psi
array([  9.99999941,  19.99999997,  29.9999993 ])
>>> theta
array([ 30.00000008,  39.99999971,  50.00000025])
>>> phi
array([ -6.06200867e-07,   5.00000036e+00,   1.00000001e+01])
```

navpy.**quat2dcm**(*q0*, *qvec*, *rotation_sequence='ZYX'*, *output_type='ndarray'*)

Convert a single unit quaternion to one DCM

Parameters **q0** : {(N,), (N,1), or (1,N)} array_like

Scalar componenet of the quaternion

qvec : {(N,3),(3,N)} array_like

Vector component of the quaternion

rotation_sequence : {'ZYX'}, optional

Rotation sequences. Default is 'ZYX'.

output_type : {'ndarray','matrix'}, optional

Output is either numpy array (default) or numpy matrix.

Returns **C_N2B** : direction consine matrix that rotates the vector from the first frame

to the second frame according to the specified rotation_sequence.

Examples

```
>>> import numpy as np
>>> from navpy import quat2dcm
>>> q0 = 1
>>> qvec = [0, 0, 0]
>>> C = quat2dcm(q0,qvec)
>>> C
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

```
>>> q0 = 0.9811
>>> qvec = np.array([-0.0151, 0.0858, 0.1730])
>>> C = quat2dcm(q0,qvec,output_type='matrix')
>>> C
matrix([[  9.25570440e-01,   3.36869440e-01,  -1.73581360e-01],
        [ -3.42051760e-01,   9.39837700e-01,   5.75800000e-05],
        [  1.63132160e-01,   5.93160200e-02,   9.84972420e-01]])
```

1.2.1 Utilities

navpy.**wrapToPi**(*e*)

Wraping angle to [-pi,pi] interval

navpy.**omega2rates**(*pitch*, *roll*, *input_unit='rad'*, *euler_angles_order='roll_pitch_yaw'*, *output_type='ndarray'*)

This function is used to create the transformation matrix to go from: [p, q, r] –> [roll_rate, pitch_rate, yaw_rate]

where pqr are xyz body rotation-rate measurements expressed in body frame. Yaw, pitch, and roll are the Euler angles. We assume the Euler angles are 3-2-1 (i.e Yaw -> Pitch -> Roll) transformations that go from navigation-frame to body-frame.

Parameters pitch : pitch angle, units of input_unit.

roll : roll angle , units of input_unit.

input_unit : units for input angles {'rad', 'deg'}, optional

euler_angles_order : {'roll_pitch_yaw', 'yaw_pitch_roll'}, optional

Assumed order of Euler Angles attitude state vector (see `Notes`).

output_type : {'ndarray' or 'matrix'}, optional

Numpy array (default) or matrix

Returns R : transformation matrix, from xyz body-rate to Euler angle-rates

numpy 'output_type' 3x3 (Note: default return variable is an ARRAY, not a matrix)

Notes

Since the returned transformation matrix is used to transform one vector to another, the assumed attitude variables order matters. The `euler_angles_order` parameter can be used to specify the assumed order.

The difference is demonstrated by example:

By default euler_angles_order='roll_pitch_yaw' R = omega2rates(pitch, roll) [roll_rate] [omega_x] [pitch_rate] = dot(R,[omega_y]) [yaw_rate] [omega_z]

Now assume our attitude state is [yaw, pitch, roll].T R = omega2rates(pitch, roll, euler_angles_order='yaw_pitch_roll') [yaw_rate] [omega_x] [pitch_rate] = dot(R,[omega_y]) [roll_rate] [omega_z]

References

[1] Equation 2.74, Aided Navigation: GPS with High Rate Sensors, Jay A. Farrel 2008

[2] omega2rates.m function at: http://www.gnssapplications.org/downloads/chapter7/Chapter7_GNSS_INS_Functions.tar.gz

1.3 Earth Modeling

navpy.**llarate**(*VN*, *VE*, *VD*, *lat*, *alt*, *lat_unit='deg'*, *alt_unit='m'*)

Calculate Latitude, Longitude, Altitude Rate given locally tangent velocity

Parameters VN : {(N,)} array like earth relative velocity in the North direction, m/s

VE : {(N,)} array like earth relative velocity in the East direction, m/s

VD : {(N,)} array like earth relative velocity in the Down direction, m/s

lat : {(N,)} array like latitudes, unit specified in lat_unit, default deg

alt : {(N,)} array like altitudes, unit specified in alt_unit, default m

Returns **lla_dot** : {(N,3)} np.array of latitude rate, longitude rate, altitude rate.

The unit of latitude and longitude rate will be the same as the unit specified by lat_unit and the unit of altitude rate will be the same as alt_unit

See also:

earthrad called by this method

Examples

```
>>> import numpy as np
>>> from navpy import llarate
>>> llarate(100,0,0,45.0,0) # Moving North at 100 m/s, location is at N45.0
array([ 0.00089983,  0.        ,  0.        ])
>>> # That output was in deg/sec
>>> lat = [np.pi/4, -np.pi/6]
>>> alt = [100.0, 50]
>>> VN = [100, 0]
>>> VE = [0, 100]
>>> VD = [0, -5]
>>> llarate(VN,VE,VD,lat,alt,lat_unit='rad')
array([[  1.57047955e-05,   0.00000000e+00,   0.00000000e+00],          [  0.00000000e+00,   1.
>>> # That output was in rad/sec
```

navpy.**llarate**(*VN*, *VE*, *VD*, *lat*, *alt*, *lat_unit='deg'*, *alt_unit='m'*)

Calculate Latitude, Longitude, Altitude Rate given locally tangent velocity

Parameters **VN** : {(N,)} array like earth relative velocity in the North direction, m/s

VE : {(N,)} array like earth relative velocity in the East direction, m/s

VD : {(N,)} array like earth relative velocity in the Down direction, m/s

lat : {(N,)} array like latitudes, unit specified in lat_unit, default deg

alt : {(N,)} array like altitudes, unit specified in alt_unit, default m

Returns **lla_dot** : {(N,3)} np.array of latitude rate, longitude rate, altitude rate.

The unit of latitude and longitude rate will be the same as the unit specified by lat_unit and the unit of altitude rate will be the same as alt_unit

See also:

earthrad called by this method

Examples

```
>>> import numpy as np
>>> from navpy import llarate
>>> llarate(100,0,0,45.0,0) # Moving North at 100 m/s, location is at N45.0
array([ 0.00089983,  0.        ,  0.        ])
>>> # That output was in deg/sec
>>> lat = [np.pi/4, -np.pi/6]
>>> alt = [100.0, 50]
>>> VN = [100, 0]
```

```
>>> VE = [0, 100]
>>> VD = [0, -5]
>>> llarate(VN,VE,VD,lat,alt,lat_unit='rad')
array([[   1.57047955e-05,    0.00000000e+00,    0.00000000e+00],           [   0.00000000e+00,    1.
>>> # That output was in rad/sec
```

`navpy.earthrad`(*lat*, *lat_unit='deg'*, *model='wgs84'*)

Calculate radius of curvature in the prime vertical (East-West) and meridian (North-South) at a given latitude.

Parameters **lat** : {(N,)} array like latitude, unit specified by lat_unit, default in deg

Returns **R_N** : {(N,)} array like, radius of curvature in the prime vertical (East-West)

R_M : {(N,)} array like, radius of curvature in the meridian (North-South)

Examples

```
>>> import numpy as np
>>> from navpy import earthrad
>>> lat = 0
>>> Rtransverse, Rmeridian = earthrad(lat)
>>> Rtransverse
6378137.0
>>> Rmeridian
6335439.3272928288
>>> lat = [0, np.pi/2]
>>> Rtransverse, Rmeridian = earthrad(lat,lat_unit='rad')
>>> Rtransverse
array([ 6378137.        ,  6399593.62575849])
>>> Rmeridian
array([ 6335439.32729283,  6399593.62575849])
```

`navpy.earthrad`(*lat*, *lat_unit='deg'*, *model='wgs84'*)

Calculate radius of curvature in the prime vertical (East-West) and meridian (North-South) at a given latitude.

Parameters **lat** : {(N,)} array like latitude, unit specified by lat_unit, default in deg

Returns **R_N** : {(N,)} array like, radius of curvature in the prime vertical (East-West)

R_M : {(N,)} array like, radius of curvature in the meridian (North-South)

Examples

```
>>> import numpy as np
>>> from navpy import earthrad
>>> lat = 0
>>> Rtransverse, Rmeridian = earthrad(lat)
>>> Rtransverse
6378137.0
>>> Rmeridian
6335439.3272928288
>>> lat = [0, np.pi/2]
>>> Rtransverse, Rmeridian = earthrad(lat,lat_unit='rad')
>>> Rtransverse
array([ 6378137.        ,  6399593.62575849])
>>> Rmeridian
array([ 6335439.32729283,  6399593.62575849])
```

`navpy.earthrate(lat, lat_unit='deg', model='wgs84')`

Calculate the earth rotation rate resolved on NED axis given VN, VE, VD, lat, and alt.

Paul Groves's Notation: ω_{IE}^{N}, Eq. (2.75), Ch. 2.3, pp. 44

Parameters **lat** : {(N,)} array like latitudes, unit specified in lat_unit, default deg

Returns **e** : {(N,3)} np.array of the earth's rotation rate

The unit is in rad/seconds.

References

[1] P. Groves, GNSS, Inertial, and Integrated Navigation Systems, Artech House, 2008

1.4 Miscellaneous Utilities

`navpy.skew(w, output_type='ndarray')`

Make a skew symmetric 2-D array

Parameters **w** : {(3,)} array_like

Returns **C** : {(3,3)} skew symmetric representation of w

Examples

```
>>> import numpy as np
>>> from navpy import skew
>>> w = [1, 2, 3]
>>> skew(w)
array([[ 0., -3.,  2.],
       [ 3.,  0., -1.],
       [-2.,  1.,  0.]])
```

CHAPTER 2

User Guide

This is the user guide demonstrating the utility of NavPy.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Bibliography

[R1] Jekeli, C.,"Inertial Navigation Systems With Geodetic Applications", Walter de Gruyter, New York, 2001, pp. 24