etrsitrs Documentation

Release 0.1

M.A. Brentjens

Aug 06, 2017

CONTENTS

Contents:

CHAPTER

ONE

README

`etrsitrs` is a Python package that implements datum transformations to and from the ETRF2000 reference frame of the ETRS89 system. It implements the transformations described in EUREF memo *Specifications for reference frame fixing in the analysis of a EUREF GPS campaign* by Boucher and Altamimi (2011).

1.1 Requirements

The following packages and tools are required:

numpy All computations.

nose Running the unit tests.

coverage Computing code coverage of the unit tests.

numpydoc Building documentation.

sphinx Building documentation.

pylint Static code analysis (optional).

1.2 Documentation

The documentation is built when running the test suite. Documentation is also provided pre-built in the following directories:

html doc/_build/html/index.html

PDF doc/_build/latex/etrsitrs.pdf

1.3 Test suite

The test suite is run with the command:

```
user@host: ~/etrs-itrs/$ ./run-tests.sh
```

This requires the `nose` and `coverage` packages, as well as the `nosetests` script. It also builds the documentation and does static code analysis if `pylint` is present.

CHAPTER

TWO

INSTALLATION

The etrsitrs package uses the standard Python distutils for building and installation. The package is written in Python version 2. Use:

```
user@host: ~/etrs-itrs/$ python2 setup.py install
```

for a default installation. Although on most systems as of 2012, python 2 is the default Python interpreter, use the python2 interpreter explicitly to ensure you don't accidentally use python version 3 on systems that have a dual Python installation. Use --prefix to install at an alternative location, for example:

```
user@host: ~/etrs-itrs/$ python2 setup.py install --prefix=/opt/python
```

2.1 setup.py help

```
Common commands: (see '--help-commands' for more)

  setup.py build      will build the package underneath 'build/'
  setup.py install    will install the package

Global options:
  --verbose (-v)  run verbosely (default)
  --quiet (-q)    run quietly (turns verbosity off)
  --dry-run (-n)  don't actually do anything
  --help (-h)     show detailed help message
  --no-user-cfg   ignore pydistutils.cfg in your home directory

Options for 'install' command:
  --prefix            installation prefix
  --exec-prefix       (Unix only) prefix for platform-specific files
  --home              (Unix only) home directory to install under
  --user              install in user site-package
                      '/home/brentjens/.local/lib/python2.7/site-packages'
  --install-base      base installation directory (instead of --prefix or --
                      home)
  --install-platbase  base installation directory for platform-specific files
                      (instead of --exec-prefix or --home)
  --root              install everything relative to this alternate root
                      directory
  --install-purelib   installation directory for pure Python module
                      distributions
  --install-platlib   installation directory for non-pure module distributions
  --install-lib       installation directory for all module distributions
                      (overrides --install-purelib and --install-platlib)
  --install-headers   installation directory for C/C++ headers
  --install-scripts   installation directory for Python scripts
  --install-data      installation directory for data files
  --compile (-c)      compile .py to .pyc [default]
  --no-compile         don't compile .py files
  --optimize (-O)      also compile with optimization: -O1 for "python -O", -O2
                       for "python -OO", and -O0 to disable [default: -O0]
  --force (-f)         force installation (overwrite any existing files)
  --skip-build         skip rebuilding everything (for testing/debugging)
  --record             filename in which to record list of installed files

usage: setup.py [global_opts] cmd1 [cmd1_opts] [cmd2 [cmd2_opts] ...]
   or: setup.py --help [cmd1 cmd2 ...]
   or: setup.py --help-commands
   or: setup.py cmd --help
```

CHAPTER

THREE

API DOCUMENTATION

3.1 etrsitrs

The `etrsitrs` Python module converts ITRS coordinates in various reference frames to ETRS89 coordinates in the ETRF2000 reference frame and vice versa. The conversions are described by 14 parameter transforms, consisting of seven parameters and their associated rates of change per year.

The transform and the coefficients are defined in the EUREF memo /Specifications for reference frame fixing in the analysis of a EUREF GPS campaign/ by Claude Boucher and Zuheir Altamimi. This module uses version 8 bis of this memo, published on 2011-May-18.

The seven parameter transform of coordinates from frame A to frame B is defined as

$$\begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} = \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} + \begin{pmatrix} T1 \\ T2 \\ T3 \end{pmatrix} + \begin{pmatrix} D & -R3 & R2 \\ R3 & D & -R1 \\ -R2 & R1 & D \end{pmatrix} \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix}$$

Table XX lists the parameters to transform from ITRFyyyy to ETRF2000.

The inverse transform, given the same parameters, is

$$\begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} = \begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} - \begin{pmatrix} T1 \\ T2 \\ T3 \end{pmatrix} - \begin{pmatrix} D & -R3 & R2 \\ R3 & D & -R1 \\ -R2 & R1 & D \end{pmatrix} \left[\begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} - \begin{pmatrix} T1 \\ T2 \\ T3 \end{pmatrix} \right].$$

Here, we used that the matrix $I + M$, with I the identity matrix, and M the matrix in the equations above, is a rotation matrix. Rotation matrices are unitary, hence their inverse is equal to their transpose.

The seven parameters Tn, D, and Rn are time dependent. To correctly perform the transformation, the parameter set P must first be propagated to the epoch at which the ITRF coordinates were observed, or at which the ITRF coordinates are desired, according to

$$P(t) = P(t_0) + \dot{P} \times (t - t_0),$$

where t_0 is the epoch at which the parameters are valid (2000.0 in this case), and t is the epoch at which the ITRF coordinates are observed or required. Both are in units of years.

3.1.1 Available sub modules

parameterset Organises the seven parameters necessary for the transforms.

datumtransformation The actual forward- and reverse transform math, and handling of annual change of the parameters.

main Contains the *convert_fn()* and *convert()* functions, as well as a table of predefined transforms *TRANSFORM_TABLE*, for supporting these functions.

3.2 Main

This module is home to the most important functions: *convert()* and *convert_fn()*. Use *convert()* if you are only interested in one or two coordinate conversions, but create a conversion function with the help of *convert_fn()* if you want to convert more coordinates.

This module contains a hardcoded table of predefined DatumTransformations, called *TRANSFORM_TABLE*. The function *find_transform()* searches this table.

`etrsitrs.main.`**`convert_fn`**(*from_frame*, *to_frame*, *epoch*)

Returns a function *convert_function(xyz_m)* that converts an XYZ vector from `from_frame` to `to_frame` at the given `epoch`.

Parameters

from_frame [string] Frame from which to ransform, e.g. 'ITRF2008'.

to_frame [string] Frame to which to transform, e.g. 'ETRF2000'.

epoch [number] Epoch at which the coordinates were observed, or are required, in years. Example: 2013.5.

Raises

KeyError If no appropriate transform is found

Returns

A function *f(xyz_m)* that returns a *numpy.array* of length 3.

Examples

```
>>> onsala_itrf2008 = numpy.array([3370658.542, 711877.138, 5349786.952])
>>> fn = convert_fn('ITRF2008', 'ETRF2000', 2005.0)
>>> print('%.3f, %.3f, %.3f' % tuple(fn(onsala_itrf2008)))
3370658.848, 711876.948, 5349786.770
```

`etrsitrs.main.`**`convert`**(*xyz_m*, *from_frame*, *to_frame*, *epoch*)

Converts the *xyz_m* vector from *from_frame* to *to_frame*. Use only if one has to convert one or two coordinates. Create a conversion function with *convert_fn()* if you have to convert a large number of coordinates.

Parameters

xyz_m [sequence of 3 floats] The coordinates to convert.

from_frame [string] Frame from which to ransform, e.g. 'ITRF2008'.

to_frame [string] Frame to which to transform, e.g. 'ETRF2000'.

epoch [number] Epoch at which the coordinates were observed, or are required, in years. Example: 2013.5.

Raises

KeyError If no appropriate transform is found.

Returns

A *numpy.array* of length 3.

Examples

```
>>> onsala_itrf2008 = numpy.array([3370658.542, 711877.138, 5349786.952])
>>> onsala_etrf2000 = convert(onsala_itrf2008, 'ITRF2008', 'ETRF2000', 2005.0)
>>> print('%.3f, %.3f, %.3f' % tuple(onsala_etrf2000))
3370658.848, 711876.948, 5349786.770
```

`class` `etrsitrs.main.`**`ETRF2000`**(*from_frame*, *parameters*, *rates*, *ref_epoch*)

ETRF2000 is a subclass of *DatumTransformation* that makes it possible to specify the 14 parameters from Boucher and Altamimi (2011) in the units used in their memo. That is, first the three translations in mm, then term D in units of 10^{-9} followed by the three rotations in mas. The *to_frame* is set to 'ETRF2000'.

The same order and units are used for the rates.

Parameters

from_frame [string] The frame from which the parameters transform, e.g. 'ITRF2008'.

parameters [sequence of 7 floats] The parameters [T1 (mm), T2 (mm), T3 (mm), D (1e-9), R1 (mas), R2 (mas), R3 (mas)].

rates: sequence of 7 floats The annual rates of change for the parameters [T1 (mm), T2 (mm), T3 (mm), D (1e-9), R1 (mas), R2 (mas), R3 (mas)].

ref_epoch [float] The reference epoch at which *parameters* are valid, e.g. 2000.0

Examples

```
>>> #        |'T1'  |'T2'  |'T3'   |'D'   |'R1'   |'R2'   |'R3'    |
>>> #        |(mm)  |(mm)  |(mm)   |x1e-9|(mas)  |(mas)  |(mas)   |
>>> ETRF2000('ITRF2008' ,
...          [52.1, 49.3, -58.5, 1.34, 0.891, 5.390, -8.712],
...          [ 0.1,  0.1,  -1.8, 0.08, 0.081, 0.490, -0.792],
...          2000.0)
ETRF2000(from_frame = 'ITRF2008', to_frame = 'ETRF2000',
        parameters = ParameterSet(translate_m = array([ 0.0521,  0.0493, -0.
↪0585]), term_d = 1.3400e-09, rotate_rad = array([  4.31968990e-09,   2.
↪61314574e-08,  -4.22369679e-08])),
        rates      = ParameterSet(translate_m = array([ 0.0001,  0.0001, -0.
↪0018]), term_d = 8.0000e-11, rotate_rad = array([  3.92699082e-10,   2.
↪37558704e-09,  -3.83972435e-09])),
        ref_epoch  = 2000.0)
```

etrsitrs.main.**find_transform**(*from_frame*, *to_frame*)

Finds the appropriate *DatumTransformation* in *TRANSFORM_TABLE*.

Parameters

from_frame [string] Frame from which to transform, e.g. 'ITRF2008'.

to_frame [string] Frame to which to transform, e.g. 'ETRF2000'

Returns

A *DatumTransformation* instance that can perform the requested conversion.

Raises

KeyError If no appropriate transform is found.

Examples

```
>>> find_transform('ETRF2000', 'ITRF2000')
ETRF2000(from_frame = 'ITRF2000', to_frame = 'ETRF2000',
        parameters = ParameterSet(translate_m = array([ 0.054,  0.051, -0.
↪048]), term_d = 0.0000e+00, rotate_rad = array([  4.31968990e-09,   2.
↪61314574e-08,  -4.22369679e-08])),
        rates      = ParameterSet(translate_m = array([ 0.,   0.,   0.]), term_d␣
↪= 0.0000e+00, rotate_rad = array([  3.92699082e-10,   2.37558704e-09,  -3.
↪83972435e-09])),
        ref_epoch  = 2000.0)
>>> find_transform('ITRF2008', 'ETRF2000')
ETRF2000(from_frame = 'ITRF2008', to_frame = 'ETRF2000',
        parameters = ParameterSet(translate_m = array([ 0.0521,  0.0493, -0.
↪0585]), term_d = 1.3400e-09, rotate_rad = array([  4.31968990e-09,   2.
↪61314574e-08,  -4.22369679e-08])),
        rates      = ParameterSet(translate_m = array([ 0.0001,  0.0001, -0.
↪0018]), term_d = 8.0000e-11, rotate_rad = array([  3.92699082e-10,   2.
↪37558704e-09,  -3.83972435e-09])),
        ref_epoch  = 2000.0)
```

```
>>> find_transform('ITRF1833', 'ETRF2000')
Traceback (most recent call last):
...
KeyError: "No 'ITRF1833' -> 'ETRF2000' in etrsitrs.main.TRANSFORM_TABLE."
```

3.3 Parameter sets

class `etrsitrs.parameterset.`**`ParameterSet`**(*translate_m*, *term_d*, *rotate_rad*)

A ParameterSet holds either the parameters Tn, D, and Rn, or their derivatives with respect to time, in units of meters for Tn, and radians for Rn.

Parameters

translate_m [sequence of floats] A vector containing the translation parameters T1, T2, and T3 in units of meters.

term_d [float] Term D from Boucher and Altamimi. It is the cosine of a tiny rotation, minus 1.

rotate_rad [sequence of floats] The rotation parameters R1, R2, and R3 in units of radians.

Examples

```
>>> ParameterSet((0.01, 0.02, 0.03), 3.14e-9, [-0.1, -0.2, -0.3])
ParameterSet(translate_m = array([ 0.01,  0.02,  0.03]), term_d = 3.1400e-09,␣
↪rotate_rad = array([-0.1, -0.2, -0.3]))
>>> ParameterSet((0.01, 0.02), 3.14e-9, [-0.1, -0.2, -0.3])
Traceback (most recent call last):
...
ValueError: translate_m((0.01, 0.02)) must be e sequence of 3 floats.
>>> ParameterSet((0.01, 0.02, 0.03), 3.14, [-0.1, -0.2, -0.3])
Traceback (most recent call last):
...
ValueError: term_d(3.14) must be a very small number.
>>> ParameterSet((0.01, 0.02, 0.03), 3.14e-9, 15.0)
Traceback (most recent call last):
...
TypeError: object of type 'float' has no len()
>>> ParameterSet((0.01, 0.02, 0.03), 3.14e-9, (15.0,14,13,12))
Traceback (most recent call last):
...
ValueError: rotate_rad((15.0, 14, 13, 12)) must be e sequence of 3 floats.
```

`ParameterSet` also supports multiplication by a number and addition of two `ParameterSet` s

```
>>> from math import pi
>>> mm  = 0.001
>>> mas = pi/(180.0*3600.0*1000.0)
>>> parameters = ParameterSet(array([52.1, 49.3, -58.5])*mm,
...                           1.34e-9,
...                           array([0.891, 5.390, -8.712])*mas)
>>> parameters
ParameterSet(translate_m = array([ 0.0521,  0.0493, -0.0585]), term_d = 1.
↪3400e-09, rotate_rad = array([  4.31968990e-09,   2.61314574e-08,  -4.
↪22369679e-08]))
>>> rates      = ParameterSet(array([0.1, 0.1, -1.8])*mm,
...                           0.08e-9,
...                           array([0.081, 0.490, -0.792])*mas)
>>> rates
ParameterSet(translate_m = array([ 0.0001,  0.0001, -0.0018]), term_d = 8.
↪0000e-11, rotate_rad = array([  3.92699082e-10,   2.37558704e-09,  -3.
↪83972435e-09]))
>>> ref_epoch = 2000.0
>>> parameters + rates*(2010.0 - ref_epoch)
ParameterSet(translate_m = array([ 0.0531,  0.0503, -0.0765]), term_d = 2.
↪1400e-09, rotate_rad = array([  8.24668072e-09,   4.98873278e-08,  -8.
↪06342114e-08]))
```

matrix()

Returns

The matrix

$$\begin{pmatrix} D & -R3 & R2 \\ R3 & D & -R1 \\ -R2 & R1 & D \end{pmatrix}$$

Examples

```
>>> ps = ParameterSet((0.01, 0.02, 0.03), 3.14e-9, [-0.1, -0.2, -0.3])
>>> ps.matrix()
array([[  3.14000000e-09,   3.00000000e-01,  -2.00000000e-01],
       [ -3.00000000e-01,   3.14000000e-09,   1.00000000e-01],
       [  2.00000000e-01,  -1.00000000e-01,   3.14000000e-09]])
```

3.4 Datum transformations

The *datumtransformation* module contains the actual transform math, implemented in the *forward_transform()* and *reverse_transform()* functions, as well as the *DatumTransformation* class, which manages rates of change of parameters and wraps the forward- and reverse- transforms.

etrsitrs.datumtransformation.**forward_transform**(*xyz_m*, *translate_m*, *rotation_matrix*)

Transform *xyz_m* given a translation vector and a rotation matrix. Only use *translate_m* and *matrix* from the *ParameterSet* returned by *propagate_parameters()*. Implements

$$\begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} = \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} + \begin{pmatrix} T1 \\ T2 \\ T3 \end{pmatrix} + \begin{pmatrix} D & -R3 & R2 \\ R3 & D & -R1 \\ -R2 & R1 & D \end{pmatrix} \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix}$$

Parameters

xyz_m [numpy.array of length 3] The coordinates to transform in meters.

translate_m [numpy.array of length 3] Propagated (T1, T2, T3).

rotation_matrix [numpy.array of shape (3, 3)] The rotation matrix obtained by calling the *ParameterSet.matrix()* method on the result of *propagate_parameters()*.

Returns

A numpy.array of length 3 with the transformed coordinates.

Examples

At epoch 2000.0:

```
>>> from numpy import array
>>> parameters = ParameterSet(translate_m = array([ 0.0521, 0.0493, -0.0585]),
...                           term_d = 1.3400e-09,
...                           rotate_rad = array([  4.31968990e-09,   2.
↪61314574e-08,  -4.22369679e-08]))
>>> onsala_itrf2008 = array([3370658.542, 711877.138, 5349786.952])
>>> onsala_etrf2000 = forward_transform(onsala_itrf2008,
...                                     parameters.translate_m,
...                                     parameters.matrix())
```

```
>>> print('%.3f, %.3f, %.3f' % tuple(onsala_etrf2000))
3370658.768, 711877.023, 5349786.816
```

etrsitrs.datumtransformation.**reverse_transform**(*xyz_m*, *translate_m*, *rotation_matrix*)

The opposite of *forward_transform()*. Transform xyz given a translation vector and a rotation matrix. Only use *translate_m* and *matrix* from the *ParameterSet* returned by *propagate_parameters()*. Implements

$$\begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} = \begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} - \begin{pmatrix} T1 \\ T2 \\ T3 \end{pmatrix} - \begin{pmatrix} D & -R3 & R2 \\ R3 & D & -R1 \\ -R2 & R1 & D \end{pmatrix} \left[\begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} - \begin{pmatrix} T1 \\ T2 \\ T3 \end{pmatrix} \right].$$

Parameters

xyz_m [numpy.array of length 3] The coordinates to transform in meters.

translate_m [numpy.array of length 3] Propagated (T1, T2, T3).

rotation_matrix [numpy.array of shape (3, 3)] The rotation matrix obtained by calling the *ParameterSet.matrix()* method on the result of *propagate_parameters()*.

Returns

A numpy.array of length 3 with the transformed coordinates.

Examples

Examples

At epoch 2000.0:

```
>>> from numpy import array
>>> parameters = ParameterSet(translate_m = array([ 0.0521, 0.0493, -0.0585]),
...                           term_d = 1.3400e-09,
...                           rotate_rad = array([  4.31968990e-09,   2.
↪61314574e-08,  -4.22369679e-08]))
>>> onsala_etrf2000 = array([3370658.768, 711877.023, 5349786.816])
>>> onsala_itrf2008 = reverse_transform(onsala_etrf2000,
...                                     parameters.translate_m,
...                                     parameters.matrix())
>>> print('%.3f, %.3f, %.3f' % tuple(onsala_itrf2008))
3370658.542, 711877.138, 5349786.952
```

class etrsitrs.datumtransformation.**DatumTransformation**(*from_frame*, *to_frame*, *parameters*, *rates*, *ref_epoch*)

A datum transformation is used to transform coordinates from reference frame A to reference frame B. It is defined by a frame *from* which to transform, a frame *to* which to transform, the transformation parameters at the reference epoch, and their rates of change.

Parameters

from_frame [string] The reference frame *from* which the parameters transform, for example 'ITRF2008'

to_frame [string] The reference frame *to* which the parameters transform, for example 'ETRF2000'

parameters [ParameterSet] The values of the transform parameters Tn, D, and Rn.

rates [ParameterSet] The annual rates of change for the parameters Tn, D, and Rn.

ref_epoch [float] The year to which the parameters are referenced. The parameters at `epoch` are `parameters` + `rates` * (epoch - ref_epoch)

Examples

```
>>> from numpy import array, pi
>>> mm  = 0.001
>>> mas = pi/(180.0*3600.0*1000.0)
>>> transform = DatumTransformation(
...     from_frame = 'ITRF2008', to_frame = 'ETRF2000',
...     parameters = ParameterSet(array([52.1, 49.3, -58.5])*mm,
...                               1.34e-9,
...                               array([0.891, 5.390, -8.712])*mas),
...     rates      = ParameterSet(array([0.1, 0.1, -1.8])*mm,
...                               0.08e-9,
...                               array([0.081, 0.490, -0.792])*mas),
...     ref_epoch  = 2000.0)
>>> transform
DatumTransformation(from_frame = 'ITRF2008', to_frame = 'ETRF2000',
        parameters = ParameterSet(translate_m = array([ 0.0521,  0.0493, -0.
↪0585]), term_d = 1.3400e-09, rotate_rad = array([  4.31968990e-09,   2.
↪61314574e-08,  -4.22369679e-08])),
        rates      = ParameterSet(translate_m = array([ 0.0001,  0.0001, -0.
↪0018]), term_d = 8.0000e-11, rotate_rad = array([  3.92699082e-10,   2.
↪37558704e-09,  -3.83972435e-09])),
        ref_epoch  = 2000.0)
```

The parameters are only valid for the reference epoch. If one needs to convert coordinates at any other epoch, the parameters must first be propagated to that epoch with the help of the rates of change:

```
>>> epoch = 2005.0
>>> par_2005 = transform.propagate_parameters(epoch)
>>> par_2005
ParameterSet(translate_m = array([ 0.0526,  0.0498, -0.0675]), term_d = 1.
↪7400e-09, rotate_rad = array([  6.28318531e-09,   3.80093926e-08,  -6.
↪14355897e-08]))
```

These propagated parameters can now be used to actually transform coordinates from the ITRF2008 frame to ETRF2000, at the epoch 2005.0. Here is an example for the Onsala Space Observatory, a EUREF class A station. According to the EUREF web site for this station, http://www.epncb.oma.be/_productsservices/coordinates/crd4station.php?station=ONSA, Onsala has the following coordinates:

Frame	Epoch (y)	X (m)	Y (m)	Z (m)
ETRF2000	2005.0	3370658.847 ± 0.001	711876.949 ± 0.001	5349786.771 ± 0.001
ITRF2008	2005.0	3370658.542 ± 0.001	711877.138 ± 0.001	5349786.952 ± 0.001

Let’s see how this works out:

```
>>> onsala_itrf2008 = array([3370658.542, 711877.138, 5349786.952])
>>> itrf_to_etrf = transform.convert_fn('ITRF2008', 'ETRF2000', epoch = 2005.0)
>>> onsala_etrf2000 = itrf_to_etrf(onsala_itrf2008)
>>> print('%.3f, %.3f, %.3f' % tuple(onsala_etrf2000))
3370658.848, 711876.948, 5349786.770
```

Not bad at all. We also have the reverse transform, from ETRF2000 to ITRF2008:

```
>>> onsala_etrf2000 = array([3370658.848, 711876.948, 5349786.770])
>>> etrf_to_itrf = transform.convert_fn('ETRF2000', 'ITRF2008', epoch = 2005.0)
>>> onsala_itrf2008 = etrf_to_itrf(onsala_etrf2000)
>>> print('%.3f, %.3f, %.3f' % tuple(onsala_itrf2008))
3370658.542, 711877.138, 5349786.952
```

For single use, one can call the *convert* method, which under the hood first creates a conversion function. Note that this is fairly wasteful in terms of cpu cycles:

```
>>> print('%.3f, %.3f, %.3f' %
...       tuple(transform.convert(onsala_itrf2008, 'ITRF2008', 'ETRF2000',␣
↪2005.0)))
```

```
3370658.848, 711876.948, 5349786.770
>>> print('%.3f, %.3f, %.3f' %
...       tuple(transform.convert(onsala_etrf2000, from_frame = 'ETRF2000', to_
↪frame = 'ITRF2008', epoch = 2005.0)))
3370658.542, 711877.138, 5349786.952
```

But be careful:

```
>>> transform.convert(onsala_etrf2000, from_frame = 'ETRF2000', to_frame =
↪'ITRF2005', epoch = 2005.0)
Traceback (most recent call last):
...
ValueError: No transform 'ETRF2000' -> 'ITRF2005' only 'ETRF2000' <-> 'ITRF2008
↪'.
```

propagate_parameters(*epoch*)

Propagate the parameter set to *epoch*. Only use parameters propagated with this method to *epoch* whenever you want to do a coordinate conversion.

Parameters

epoch [number] The year at which one desires the parameters, e.g. 2013.2.

Returns

A ParameterSet.

convert_fn(*from_frame*, *to_frame*, *epoch*)

Returns a function *convert(xyz_m)* that converts an XYZ vector from `from_frame` to `to_frame`. If `from_frame` is equal to `self.to_frame` and v.v., the function performs the inverse transform.

Parameters

from_frame [string] Frame from which to ransform, e.g. 'ITRF2008'.

to_frame [string] Frame to which to transform, e.g. 'ETRF2000'.

epoch [number] Epoch at which the coordinates were observed, or are required, in years. Example: 2013.5.

Raises

ValueError if *to_frame* or *from_frame* is not in *[self.to_frame, self.from_frame]*.

Returns

A function *f(xyz_m)* that returns a *numpy.array* of length 3.

convert(*xyz_m*, *from_frame*, *to_frame*, *epoch*)

Converts the *xyz_m* vector from *from_frame* to *to_frame*. If *from_frame* is equal to *self.to_frame* and v.v., the function performs the inverse transform. Use only if one has to convert one or two coordinates. Create a conversion function with *DatumTransformation.convert_fn()* if you have to convert a large number of coordinates.

Parameters

xyz_m [sequence of 3 floats] The coordinates to convert.

from_frame [string] Frame from which to ransform, e.g. 'ITRF2008'.

to_frame [string] Frame to which to transform, e.g. 'ETRF2000'.

epoch [number] Epoch at which the coordinates were observed, or are required, in years. Example: 2013.5.

Raises

ValueError if *to_frame* or *from_frame* is not in *[self.to_frame, self.from_frame]*.

Returns

A *numpy.array* of length 3.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX