xyz_calcs_update

January 18, 2024

[1]:
```
%load_ext Cython
```

[11]:
```
from matplotlib import pyplot as plt

from astropy import units
from astropy.visualization import quantity_support
from pyuvdata import utils as uvutils
import numpy as np

# from Cython.Compiler.Options import get_directive_defaults
# directive_defaults = get_directive_defaults()
# directive_defaults['linetrace'] = True
# directive_defaults['binding'] = True
```

[4]:
```
quantity_support();
```

[5]:
```
# cython: profile=True
# cython: linetrace=True
# cython: binding=True
# distutils: define_macros=CYTHON_TRACE_NOGIL=1
# distutils: define_macros=CYTHON_TRACE=1
```

[52]:
```
def XYZ_from_LatLonAlt_new(latitude, longitude, altitude):
    latitude = np.ascontiguousarray(latitude, dtype=np.float64)
    longitude = np.ascontiguousarray(longitude, dtype=np.float64)
    altitude = np.ascontiguousarray(altitude,  dtype=np.float64)

    n_pts = latitude.size

    if longitude.size != n_pts:
        raise ValueError(
            "latitude, longitude and altitude must all have the same length"
        )
    if altitude.size != n_pts:
        raise ValueError(
            "latitude, longitude and altitude must all have the same length"
        )
```

```
#       = np.zeros((3, n_pts), dtype=np.float64, order="C")

    xyz = xyz_from_latlonalt_new_cy(latitude, longitude, altitude, Body.Earth.
 ↪value)
    xyz = xyz.T
    if n_pts == 1:
        return xyz[0]

    return xyz


def LatLonAlt_from_XYZ_new(xyz, check_acceptability=True):
    # convert to a numpy array
    xyz = np.asarray(xyz)
    if xyz.ndim > 1 and xyz.shape[1] != 3:
        raise ValueError("The expected shape of ECEF xyz array is (Npts, 3).")

    squeeze = xyz.ndim == 1

    if squeeze:
        xyz = xyz[np.newaxis, :]

    xyz = np.ascontiguousarray(xyz.T, dtype=np.float64)
#     n_pts = xyz.shape[1]

#     lla = np.empty((3, n_pts), dtype=np.float64, order="C")

    # checking for acceptable values
    if check_acceptability:
        norms = np.linalg.norm(xyz, axis=1)
        if not all(
            np.logical_and(
               norms >= 6.35e6,
               norms <= 6.39e6
            )
        ):
            raise ValueError("xyz values should be ECEF x, y, z coordinates in␣
 ↪meters")

    lla = lla_from_xyz_new_cy(xyz, Body.Earth.value)

    if squeeze:
        return lla[0, 0], lla[1, 0], lla[2, 0]
    return lla[0], lla[1], lla[2]

def ECEF_from_ENU_new(enu, latitude, longitude, altitude):
```

```
    """
    Calculate ECEF coordinates from local ENU (east, north, up) coordinates.

    Parameters
    ----------
    enu : ndarray of float
        numpy array, shape (Npts, 3), with local ENU coordinates.
    latitude : float
        Latitude of center of ENU coordinates in radians.
    longitude : float
        Longitude of center of ENU coordinates in radians.
    altitude : float
        Altitude of center of ENU coordinates in radians.


    Returns
    -------
    xyz : ndarray of float
        numpy array, shape (Npts, 3), with ECEF x,y,z coordinates.

    """
    enu = np.asarray(enu)
    if enu.ndim > 1 and enu.shape[1] != 3:
        raise ValueError("The expected shape of the ENU array is (Npts, 3).")
    squeeze = False

    if enu.ndim == 1:
        squeeze = True
        enu = enu[np.newaxis, :]
    enu = np.ascontiguousarray(enu.T, dtype=np.float64)


    xyz = ECEF_from_ENU_new_cy(
        enu,
        np.ascontiguousarray(latitude, dtype=np.float64),
        np.ascontiguousarray(longitude, dtype=np.float64),
        np.ascontiguousarray(altitude, dtype=np.float64),
        Body.Earth.value
    )
    xyz = xyz.T
    if squeeze:
        xyz = np.squeeze(xyz)

    return xyz

def ENU_from_ECEF_new(xyz, latitude, longitude, altitude):
    """
```

```
    Calculate local ENU (east, north, up) coordinates from ECEF coordinates.

    Parameters
    ----------
    xyz : ndarray of float
        numpy array, shape (Npts, 3), with ECEF x,y,z coordinates.
    latitude : float
        Latitude of center of ENU coordinates in radians.
    longitude : float
        Longitude of center of ENU coordinates in radians.
    altitude : float
        Altitude of center of ENU coordinates in radians.

    Returns
    -------
    ndarray of float
        numpy array, shape (Npts, 3), with local ENU coordinates

    """
    xyz = np.asarray(xyz)
    if xyz.ndim > 1 and xyz.shape[1] != 3:
        raise ValueError("The expected shape of ECEF xyz array is (Npts, 3).")

    squeeze = False
    if xyz.ndim == 1:
        squeeze = True
        xyz = xyz[np.newaxis, :]
    xyz = np.ascontiguousarray(xyz.T, dtype=np.float64)

    # check that these are sensible ECEF values -- their magnitudes need to be
    # on the order of Earth's radius
    ecef_magnitudes = np.linalg.norm(xyz, axis=0)
    sensible_radius_range = (6.35e6, 6.39e6)
    if np.any(ecef_magnitudes <= sensible_radius_range[0]) or np.any(
        ecef_magnitudes >= sensible_radius_range[1]
    ):
        raise ValueError(
            f"{frame} vector magnitudes must be on the order"
            f" of the radius of the {world}"
        )

    # the cython utility expects (3, Npts) for faster manipulation
    # transpose after we get the array back to match the expected shape
    enu = ENU_from_ECEF_new_cy(
        xyz,
        np.ascontiguousarray(latitude, dtype=np.float64),
        np.ascontiguousarray(longitude, dtype=np.float64),
```

```
        np.ascontiguousarray(altitude, dtype=np.float64),
        # we have already forced the frame to conform to our options
        # and if we  don't have moon we have already errored.
        Body.Earth.value,
    )
    enu = enu.T

    if squeeze:
        enu = np.squeeze(enu)

    return enu
```

[43]:
```
%%cython -a -3 --compile-args=-O3 --compile-args=-fopenmp --link-args=-fopenmp␣
↪--force
# -*- mode: python; coding: utf-8 -*-
# Copyright (c) 2020 Radio Astronomy Software Group
# Licensed under the 2-clause BSD License

# distutils: language = c
# cython: linetrace=True

# python imports
import warnings
import enum

# cython imports

cimport cython
cimport numpy
from libc.math cimport atan2, cos, sin, sqrt
# This initializes the numpy 1.7 c-api.
# cython 3.0 will do this by default.
# We may be able to just remove this then.
numpy.import_array()

cdef class Ellipsoid:
  cdef readonly numpy.float64_t gps_a, gps_b, e_squared, e_prime_squared,␣
↪b_div_a2

  @cython.cdivision
  def __init__(self, numpy.float64_t gps_a, numpy.float64_t gps_b):
    self.gps_a = gps_a
    self.gps_b = gps_b
    self.b_div_a2 = (self.gps_b / self.gps_a)**2
    self.e_squared = (1 - self.b_div_a2)
    self.e_prime_squared = (self.b_div_a2**-1 - 1)
```

```
# A python interface for different celestial bodies
class Body(enum.Enum):
  Earth = Ellipsoid(6378137, 6356752.31424518)
  # moon data taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/
 ↪moonfact.html
  # with radius from spice_utils
  Moon = Ellipsoid(1737.1e3, 1737.1e3 * (1 - 0.0012))


# expose up to python
# in order to not have circular dependencies
# define transformation parameters here
# parameters for transforming between xyz & lat/lon/alt
# keep for consistent API though these really shouldn't be used anymore
gps_a = Body.Earth.value.gps_a
gps_b =  Body.Earth.value.gps_b
e_squared =  Body.Earth.value.e_squared
e_prime_squared =  Body.Earth.value.e_prime_squared

@cython.boundscheck(False)
@cython.wraparound(False)
@cython.cdivision(True)
cpdef numpy.ndarray[dtype=numpy.float64_t, ndim=2] lla_from_xyz_new_cy(
  numpy.float64_t[:, ::1] xyz,
  Ellipsoid body,
):
  cdef Py_ssize_t ind
  cdef int ndim = 2
  cdef int n_pts = xyz.shape[1]
  cdef numpy.npy_intp * dims = [3, <numpy.npy_intp>n_pts]

  cdef numpy.ndarray[dtype=numpy.float64_t, ndim=2] lla = numpy.
 ↪PyArray_EMPTY(ndim, dims, numpy.NPY_FLOAT64, 0)
  cdef numpy.float64_t[:, ::1] _lla = lla

  cdef numpy.float64_t gps_p, gps_theta

  # see wikipedia geodetic_datum and Datum transformations of
  # GPS positions PDF in docs/references folder
  for ind in range(n_pts):
    gps_p = sqrt(xyz[0, ind] ** 2 + xyz[1, ind] ** 2)
    gps_theta = atan2(xyz[2, ind] * body.gps_a, gps_p * body.gps_b)

    _lla[0, ind] = atan2(
      xyz[2, ind] + body.e_prime_squared * body.gps_b * sin(gps_theta) ** 3,
      gps_p - body.e_squared * body.gps_a * cos(gps_theta) ** 3,
```

```
    )

    _lla[1, ind] = atan2(xyz[1, ind], xyz[0, ind])

    _lla[2, ind] = (gps_p / cos(lla[0, ind])) - body.gps_a / sqrt(1.0 - body.
 ↪e_squared * sin(lla[0, ind]) ** 2)

  return lla

@cython.boundscheck(False)
@cython.wraparound(False)
@cython.cdivision(True)
cpdef numpy.ndarray[dtype=numpy.float64_t, ndim=2] xyz_from_latlonalt_new_cy(
  numpy.float64_t[::1] _lat,
  numpy.float64_t[::1] _lon,
  numpy.float64_t[::1] _alt,
  Ellipsoid body,
):
  cdef Py_ssize_t i
  cdef int ndim = 2
  cdef int n_pts = _lat.shape[0]
  cdef numpy.npy_intp * dims = [3, <numpy.npy_intp>n_pts]

  cdef numpy.ndarray[dtype=numpy.float64_t, ndim=2] xyz = numpy.
 ↪PyArray_EMPTY(ndim, dims, numpy.NPY_FLOAT64, 0)
  cdef numpy.float64_t[:, ::1] _xyz = xyz

  cdef numpy.float64_t  sin_lat, cos_lat, sin_lon, cos_lon, gps_n

  for ind in range(n_pts):
    sin_lat = sin(_lat[ind])
    sin_lon = sin(_lon[ind])

    cos_lat = cos(_lat[ind])
    cos_lon = cos(_lon[ind])

    gps_n = body.gps_a / sqrt(1.0 - body.e_squared * sin_lat ** 2)

    _xyz[0, ind] = (gps_n + _alt[ind]) * cos_lat * cos_lon
    _xyz[1, ind] = (gps_n + _alt[ind]) * cos_lat * sin_lon

    _xyz[2, ind] = (body.b_div_a2 * gps_n + _alt[ind]) * sin_lat
  return xyz

# this function takes memoryviews as inputs
# that is why _lat, _lon, and _alt are indexed below to get the 0th entry
@cython.boundscheck(False)
```

```
@cython.wraparound(False)
cpdef numpy.ndarray[numpy.float64_t, ndim=2] ENU_from_ECEF_new_cy(
  numpy.float64_t[:, ::1] xyz,
  numpy.float64_t[::1] _lat,
  numpy.float64_t[::1] _lon,
  numpy.float64_t[::1] _alt,
  Ellipsoid body,
):
  cdef Py_ssize_t i
  cdef int ndim = 2
  cdef int nblts = xyz.shape[1]
  cdef numpy.npy_intp * dims =  [3, <numpy.npy_intp> nblts]
  cdef numpy.float64_t xyz_use[3]

  cdef numpy.float64_t sin_lat, cos_lat, sin_lon, cos_lon

  # we want a memoryview of the xyz of the center
  # this looks a little silly but we don't have to define 2 different things
  cdef numpy.float64_t[:] xyz_center = xyz_from_latlonalt_new_cy(_lat, _lon,␣
 ↪_alt, body).T[0]

  cdef numpy.ndarray[numpy.float64_t, ndim=2] _enu = numpy.PyArray_EMPTY(ndim,␣
 ↪dims, numpy.NPY_FLOAT64, 0)
  cdef numpy.float64_t[:, ::1] enu = _enu

  sin_lat = sin(_lat[0])
  cos_lat = cos(_lat[0])

  sin_lon = sin(_lon[0])
  cos_lon = cos(_lon[0])

  for i in range(nblts):
    xyz_use[0] = xyz[0, i] - xyz_center[0]
    xyz_use[1] = xyz[1, i] - xyz_center[1]
    xyz_use[2] = xyz[2, i] - xyz_center[2]

    enu[0, i] = -sin_lon * xyz_use[0] + cos_lon * xyz_use[1]
    enu[1, i] = (
      - sin_lat * cos_lon * xyz_use[0]
      - sin_lat * sin_lon * xyz_use[1]
      + cos_lat * xyz_use[2]
    )
    enu[2, i] = (
      cos_lat * cos_lon * xyz_use[0]
      + cos_lat * sin_lon * xyz_use[1]
      + sin_lat * xyz_use[2]
    )
```

```
  return _enu

# this function takes memoryviews as inputs
# that is why _lat, _lon, and _alt are indexed below to get the 0th entry
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef numpy.ndarray[dtype=numpy.float64_t] ECEF_from_ENU_new_cy(
  numpy.float64_t[:, ::1] enu,
  numpy.float64_t[::1] _lat,
  numpy.float64_t[::1] _lon,
  numpy.float64_t[::1] _alt,
  Ellipsoid body,
):
  cdef Py_ssize_t i
  cdef int ndim = 2
  cdef int nblts = enu.shape[1]
  cdef numpy.npy_intp * dims = [3, <numpy.npy_intp>nblts]
  cdef numpy.float64_t sin_lat, cos_lat, sin_lon, cos_lon

  # allocate memory then make memory view for faster access
  cdef numpy.ndarray[dtype=numpy.float64_t, ndim=2] _xyz = numpy.
 ↪PyArray_EMPTY(ndim, dims, numpy.NPY_FLOAT64, 0)
  cdef numpy.float64_t[:, ::1] xyz = _xyz

  # we want a memoryview of the xyz of the center
  # this looks a little silly but we don't have to define 2 different things
  cdef numpy.float64_t[:] xyz_center = xyz_from_latlonalt_new_cy(_lat, _lon,␣
 ↪_alt, body).T[0]

  sin_lat = sin(_lat[0])
  cos_lat = cos(_lat[0])

  sin_lon = sin(_lon[0])
  cos_lon = cos(_lon[0])

  for i in range(nblts):
    xyz[0, i] = (
      - sin_lat * cos_lon * enu[1, i]
      - sin_lon * enu[0, i]
      + cos_lat * cos_lon * enu[2, i]
      + xyz_center[0]
    )
    xyz[1, i] = (
      - sin_lat * sin_lon * enu[1, i]
      + cos_lon * enu[0, i]
      + cos_lat * sin_lon * enu[2, i]
```

```
      + xyz_center[1]
    )
    xyz[2, i] = cos_lat * enu[1, i] + sin_lat * enu[2, i] + xyz_center[2]

  return _xyz
```

[43]: <IPython.core.display.HTML object>

[33]:
```
for size in [1, 10, 100, 500]:#, 1000, 10000, 100000]:
    center_lat = -30.721
    center_lon = 21.428
    center_alt = 1052.5
    lats = center_lat  + np.random.normal(0, .0005, size=size)
    lons = center_lon+ np.random.normal(0, .0005, size=size)
    alts = np.random.uniform(1051, 1054, size=size)
    lats *= np.pi/180.
    lons *= np.pi/180.
    xyz = uvutils.XYZ_from_LatLonAlt(lats, lons, alts)
    cy_xyz = XYZ_from_LatLonAlt_new(lats, lons, alts)

    lla = uvutils.LatLonAlt_from_XYZ(xyz, check_acceptability=False)
    cy_lla = np.asarray(LatLonAlt_from_XYZ_new(xyz, check_acceptability=False))

    print(f"Size {size}")
    print("\tResults equal LLA -> XYZ | XYZ -> LLA")
    print(f"\t\tcy:  {np.allclose(xyz, cy_xyz)} {np.allclose(lla, cy_lla)}")

    print("\tLLA -> XYZ")
    py_time = %timeit -q -o -r 100 -n 1000 uvutils.XYZ_from_LatLonAlt(lats,␣
 ↪lons, alts)
    cy_time = %timeit -q -o -r 100 -n 1000  XYZ_from_LatLonAlt_new(lats, lons,␣
 ↪alts)
    print(f"\t\tpython: {py_time}")
    print(f"\t\tnew:    {cy_time}")
    print("\tXYZ -> LLA")
    py_time = %timeit -q -o -r 100 -n 1000   uvutils.LatLonAlt_from_XYZ(xyz,␣
 ↪check_acceptability=False)
    cy_time = %timeit -q -o -r 100 -n 1000  LatLonAlt_from_XYZ_new(xyz,␣
 ↪check_acceptability=False)
    print(f"\t\tpython: {py_time}")
    print(f"\t\tnew:    {cy_time}")
```

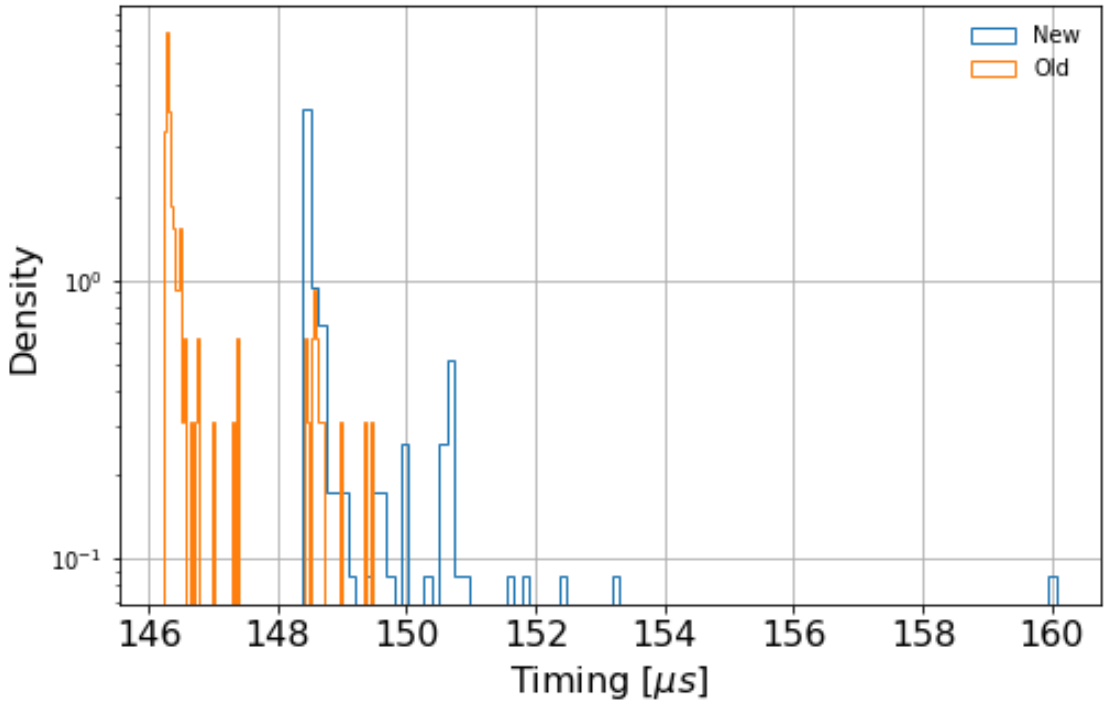
```
Size 1
        Results equal LLA -> XYZ | XYZ -> LLA
                cy:  True True
        LLA -> XYZ
                python: 4.5 µs ± 121 ns per loop (mean ± std. dev. of 100 runs,
```

```
1000 loops each)
                new:    4.66 µs ± 29.5 ns per loop (mean ± std. dev. of 100
runs, 1000 loops each)
        XYZ -> LLA
                python: 3.6 µs ± 713 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    3.74 µs ± 17.1 ns per loop (mean ± std. dev. of 100
runs, 1000 loops each)
Size 10
        Results equal LLA -> XYZ | XYZ -> LLA
                cy:  True True
        LLA -> XYZ
                python: 4.84 µs ± 68.8 ns per loop (mean ± std. dev. of 100
runs, 1000 loops each)
                new:    4.96 µs ± 19.9 ns per loop (mean ± std. dev. of 100
runs, 1000 loops each)
        XYZ -> LLA
                python: 5.93 µs ± 70.5 ns per loop (mean ± std. dev. of 100
runs, 1000 loops each)
                new:    6.23 µs ± 99.7 ns per loop (mean ± std. dev. of 100
runs, 1000 loops each)
Size 100
        Results equal LLA -> XYZ | XYZ -> LLA
                cy:  True True
        LLA -> XYZ
                python: 9.98 µs ± 100 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    10 µs ± 31.1 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
        XYZ -> LLA
                python: 32.2 µs ± 881 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    32.4 µs ± 59.3 ns per loop (mean ± std. dev. of 100
runs, 1000 loops each)
Size 500
        Results equal LLA -> XYZ | XYZ -> LLA
                cy:  True True
        LLA -> XYZ
                python: 33.7 µs ± 1.01 µs per loop (mean ± std. dev. of 100
runs, 1000 loops each)
                new:    32.9 µs ± 680 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
        XYZ -> LLA
                python: 146 µs ± 1.21 µs per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    149 µs ± 841 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
```

[38]:
```
py_time = %timeit -q -o -r 100  uvutils.LatLonAlt_from_XYZ(xyz,␣
 ↪check_acceptability=False)
new_time = %timeit -q -o -r 100 LatLonAlt_from_XYZ_new(xyz,␣
 ↪check_acceptability=False)
```

[39]:
```
fig, ax = plt.subplots(1, figsize=(8,5), facecolor="white")
_ ,_, _ =ax.hist((new_time.timings * units.s).to("microsecond"), bins=100,␣
 ↪log=True, density=True, histtype="step", label="New")
_ ,_, _ =ax.hist((py_time.timings * units.s).to("microsecond"), bins=100,␣
 ↪log=True, density=True, histtype="step", label="Old")

ax.grid()
ax.set_xlabel(r"Timing [$\mu s$]", fontsize=16)
ax.set_ylabel("Density", fontsize=16)
plt.setp(ax.get_xticklabels(), fontsize=16);
ax.legend(loc="upper right", frameon=False);
```



[54]:
```
for size in [1, 10, 100, 500]:# , 1000, 10000, 100000]:

    center_lat = -30.721
    center_lon = 21.428
    center_alt = 1052.5
    lats = center_lat  + np.random.normal(0, .0005, size=size)
```

```
    lons = center_lon+ np.random.normal(0, .0005, size=size)
    alts = np.random.uniform(1051, 1054, size=size)
    lats *= np.pi/180.
    lons *= np.pi/180.

    center_lat *= np.pi/180.
    center_lon *= np.pi/180.

    xyz = uvutils.XYZ_from_LatLonAlt(lats, lons, alts)

    enu = uvutils.ENU_from_ECEF(xyz, center_lat, center_lon, center_alt)
    ecef = uvutils.ECEF_from_ENU(enu, center_lat, center_lon, center_alt)

    cy_ecef = ECEF_from_ENU_new(enu, center_lat, center_lon, center_alt)
    cy_enu = ENU_from_ECEF_new(ecef, center_lat, center_lon, center_alt)

    print(f"Size: {size}")
    print("\tResults equal ECEF -> ENU | ENU -> ECEF")
    print(f"\t\tcy:  {np.allclose(enu, cy_enu)} {np.allclose(ecef, cy_ecef)}")


    print("\tECEF -> ENU")
    py_time = %timeit -q -o -r 100 -n 1000   uvutils.ENU_from_ECEF(ecef,␣
↪center_lat, center_lon, center_alt)
    cy_time = %timeit -q -o  -r 100 -n 1000  ENU_from_ECEF_new(ecef,␣
↪center_lat, center_lon, center_alt)
    print(f"\t\tpython: {py_time}")
    print(f"\t\tnew:    {cy_time}")
    print("\tENU -> ECEF")
    py_time = %timeit -q -o -r 100 -n 1000    uvutils.ECEF_from_ENU(enu,␣
↪center_lat, center_lon, center_alt)
    cy_time = %timeit -q -o  -r 100 -n 1000  ECEF_from_ENU_new(enu, center_lat,␣
↪center_lon, center_alt)
    print(f"\t\tpython: {py_time}")
    print(f"\t\tnew:    {cy_time}")
```

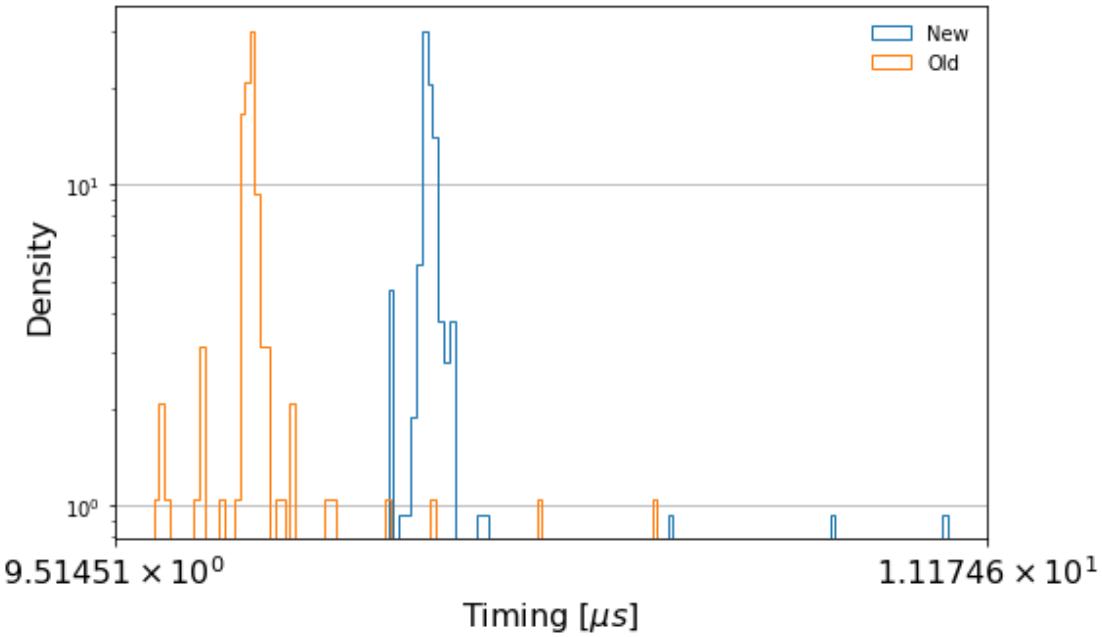
```
Size: 1
        Results equal ECEF -> ENU | ENU -> ECEF
                cy:  True True
        ECEF -> ENU
                python: 32.9 µs ± 1.16 µs per loop (mean ± std. dev. of 100
runs, 1000 loops each)
                new:    34.5 µs ± 161 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
        ENU -> ECEF
                python: 9.13 µs ± 178 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
```

```
                new:    9.72 µs ± 413 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
Size: 10
        Results equal ECEF -> ENU | ENU -> ECEF
                cy:  True True
        ECEF -> ENU
                python: 31.8 µs ± 265 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    34.3 µs ± 299 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
        ENU -> ECEF
                python: 7.86 µs ± 138 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    8.04 µs ± 58.2 ns per loop (mean ± std. dev. of 100
runs, 1000 loops each)
Size: 100
        Results equal ECEF -> ENU | ENU -> ECEF
                cy:  True True
        ECEF -> ENU
                python: 33.7 µs ± 367 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    36.7 µs ± 297 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
        ENU -> ECEF
                python: 8.4 µs ± 73.8 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    8.8 µs ± 187 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
Size: 500
        Results equal ECEF -> ENU | ENU -> ECEF
                cy:  True True
        ECEF -> ENU
                python: 41.1 µs ± 988 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    43.5 µs ± 678 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
        ENU -> ECEF
                python: 9.74 µs ± 105 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
                new:    10.3 µs ± 202 ns per loop (mean ± std. dev. of 100 runs,
1000 loops each)
```

[55]:
```
old_time = %timeit -q -o -r 100   uvutils.ECEF_from_ENU(enu, center_lat,␣
 ↪center_lon, center_alt)
new_time = %timeit -q -o  -r 100 ECEF_from_ENU_new(enu, center_lat, center_lon,␣
 ↪center_alt)
```

[56]:
```
fig, ax = plt.subplots(1, figsize=(8,5), facecolor="white")
_ ,_, _ =ax.hist((new_time.timings * units.s).to("microsecond"), bins=100,␣
 ↪log=True, density=True, histtype="step", label="New")
_ ,_, _ =ax.hist((old_time.timings * units.s).to("microsecond"), bins=100,␣
 ↪log=True, density=True, histtype="step", label="Old")

ax.grid()
ax.set_xlabel(r"Timing [$\mu s$]", fontsize=16)
ax.set_ylabel("Density", fontsize=16)
plt.setp(ax.get_xticklabels(), fontsize=16);
ax.legend(loc="upper right", frameon=False);
ax.set_xscale("symlog", linthresh=100)
```



[ ]: