floating_point_assignment

February 8, 2025

1 Floating Point Assignment

To run this notebook, use the provided file `floating_point_assignment.ipynb` and place it into your working directory.

```
source venv/bin/activate
jupyter lab --ip 0.0.0.0
```

Then, follow the link from the standard out of the command. This contains a token to provide you access. To turn in your work, save this notebook and provide the file.

2 Definitions (25)

We will start with a definition of floating point and double using IEEE format definitions. They are

- 32-bit: 1 sign bit, 8 exp bits, 23 frac bits
- 64-bit: 1 sign bit, 11 exp bits, 52 frac bits
- 8-bit: 1 sign bit, 4 exp bits, 3 frac bits

1. Provide the representation of the value 3.14159265 in 32-bit IEEE format.
2. Provide the representation of the value 3.14159265 in 64-bit IEEE.
3. Provide the representation of the value 3.14159265 in the 8 bit format
4. Compare the precision loss of each value in comparison to 64-bit.
5. Choose a repeating number of your choosing (ie 1/3) and show the precision loss comparing the 8-bit format to 32-bit.

For numbers 1-3, provide your value in hex as that is easier. The above values should be completed by hand. You may use the techniques in part two to check your work.

3 Precision (25)

[1]:
```
import numpy as np
from pymap3d.ecef import geodetic2ecef
```

In this section, you will use latitude, longitude, altitude coordinates to perform some calculations. The first is a coordinate change into Earth Centered, Earth Fixed (ECEF). You will then use a method for calculating the euclidean distance. You will then make observations on the precision of your calculations using three different floating point definitions. I have chosen UCCS and UC Boulder as examples. Replace at least one of these locations with those that you choose.

3.1 Points (5 points each)

1. Replace location coordinates
2. Perform calculations with each of `np.float64`, `np.float32`, and `np.float16`
3. Brief discussion of observations

[2]:
```
help(geodetic2ecef)
```

```
Help on function geodetic2ecef in module pymap3d.ecef:

geodetic2ecef(lat, lon, alt, ell: 'Ellipsoid' = Ellipsoid(model='wgs84',
name='WGS-84 (1984)', semimajor_axis=6378137.0, semiminor_axis=6356752.31424518,
flattening=0.0033528106647473664, thirdflattening=0.0016792203863836474,
eccentricity=0.0818191908426201), deg: 'bool' = True) -> 'tuple'
    point transformation from Geodetic of specified ellipsoid (default WGS-84)
to ECEF

    Parameters
    ----------

    lat
           target geodetic latitude
    lon
           target geodetic longitude
    alt
         target altitude above geodetic ellipsoid (meters)
    ell : Ellipsoid, optional
          reference ellipsoid
    deg : bool, optional
          degrees input/output  (False: radians in/out)


    Returns
    -------

    ECEF (Earth centered, Earth fixed)  x,y,z

    x
        target x ECEF coordinate (meters)
    y
        target y ECEF coordinate (meters)
    z
        target z ECEF coordinate (meters)
```

[3]:
```
help(np.linalg.norm)
```

```
Help on _ArrayFunctionDispatcher in module numpy.linalg:
```

```
norm(x, ord=None, axis=None, keepdims=False)
    Matrix or vector norm.

    This function is able to return one of eight different matrix norms,
    or one of an infinite number of vector norms (described below), depending
    on the value of the ``ord`` parameter.

    Parameters
    ----------
    x : array_like
        Input array.  If `axis` is None, `x` must be 1-D or 2-D, unless `ord`
        is None. If both `axis` and `ord` are None, the 2-norm of
        ``x.ravel`` will be returned.
    ord : {non-zero int, inf, -inf, 'fro', 'nuc'}, optional
        Order of the norm (see table under ``Notes``). inf means numpy's
        `inf` object. The default is None.
    axis : {None, int, 2-tuple of ints}, optional.
        If `axis` is an integer, it specifies the axis of `x` along which to
        compute the vector norms.  If `axis` is a 2-tuple, it specifies the
        axes that hold 2-D matrices, and the matrix norms of these matrices
        are computed.  If `axis` is None then either a vector norm (when `x`
        is 1-D) or a matrix norm (when `x` is 2-D) is returned. The default
        is None.

        .. versionadded:: 1.8.0

    keepdims : bool, optional
        If this is set to True, the axes which are normed over are left in the
        result as dimensions with size one.  With this option the result will
        broadcast correctly against the original `x`.

        .. versionadded:: 1.10.0

    Returns
    -------
    n : float or ndarray
        Norm of the matrix or vector(s).

    See Also
    --------
    scipy.linalg.norm : Similar function in SciPy.

    Notes
    -----
    For values of ``ord < 1``, the result is, strictly speaking, not a
    mathematical 'norm', but it may still be useful for various numerical
    purposes.
```

```
The following norms can be calculated:

=====  ============================  ==========================
ord    norm for matrices             norm for vectors
=====  ============================  ==========================
None   Frobenius norm                2-norm
'fro'  Frobenius norm                --
'nuc'  nuclear norm                  --
inf    max(sum(abs(x), axis=1))      max(abs(x))
-inf   min(sum(abs(x), axis=1))      min(abs(x))
0      --                            sum(x != 0)
1      max(sum(abs(x), axis=0))      as below
-1     min(sum(abs(x), axis=0))      as below
2      2-norm (largest sing. value)  as below
-2     smallest singular value       as below
other  --                            sum(abs(x)**ord)**(1./ord)
=====  ============================  ==========================

The Frobenius norm is given by [1]_:

    :math:`||A||_F = [\sum_{i,j} abs(a_{i,j})^2]^{1/2}`

The nuclear norm is the sum of the singular values.

Both the Frobenius and nuclear norm orders are only defined for
matrices and raise a ValueError when ``x.ndim != 2``.

References
----------
.. [1] G. H. Golub and C. F. Van Loan, *Matrix Computations*,
       Baltimore, MD, Johns Hopkins University Press, 1985, pg. 15

Examples
--------
>>> from numpy import linalg as LA
>>> a = np.arange(9) - 4
>>> a
array([-4, -3, -2, …,  2,  3,  4])
>>> b = a.reshape((3, 3))
>>> b
array([[-4, -3, -2],
       [-1,  0,  1],
       [ 2,  3,  4]])

>>> LA.norm(a)
7.745966692414834
>>> LA.norm(b)
7.745966692414834
```

```
>>> LA.norm(b, 'fro')
7.745966692414834
>>> LA.norm(a, np.inf)
4.0
>>> LA.norm(b, np.inf)
9.0
>>> LA.norm(a, -np.inf)
0.0
>>> LA.norm(b, -np.inf)
2.0

>>> LA.norm(a, 1)
20.0
>>> LA.norm(b, 1)
7.0
>>> LA.norm(a, -1)
-4.6566128774142013e-010
>>> LA.norm(b, -1)
6.0
>>> LA.norm(a, 2)
7.745966692414834
>>> LA.norm(b, 2)
7.3484692283495345

>>> LA.norm(a, -2)
0.0
>>> LA.norm(b, -2)
1.8570331885190563e-016 # may vary
>>> LA.norm(a, 3)
5.8480354764257312 # may vary
>>> LA.norm(a, -3)
0.0

Using the `axis` argument to compute vector norms:

>>> c = np.array([[ 1, 2, 3],
…                [-1, 1, 4]])
>>> LA.norm(c, axis=0)
array([ 1.41421356,  2.23606798,  5.        ])
>>> LA.norm(c, axis=1)
array([ 3.74165739,  4.24264069])
>>> LA.norm(c, ord=1, axis=1)
array([ 6.,  6.])

Using the `axis` argument to compute matrix norms:

>>> m = np.arange(8).reshape(2,2,2)
>>> LA.norm(m, axis=(1,2))
```

```
array([  3.74165739,  11.22497216])
>>> LA.norm(m[0, :, :]), LA.norm(m[1, :, :])
(3.7416573867739413, 11.224972160321824)
```

3.1.1 64-bit

```
[4]: uccs_64 = np.array([38.8936117,-104.8005516, 1965.96])
     ucb_64 = np.array([40.0073943,-105.2662901, 1661.16])
```

```
[5]: uccs_64_ecef = np.array([*geodetic2ecef(*uccs_64)])
     uccs_64_ecef
```

```
[5]: array([-1270194.52877575, -4807305.93880946,  3984365.91566153])
```

```
[6]: ucb_64_ecef = np.array([*geodetic2ecef(*ucb_64)])
     ucb_64_ecef
```

```
[6]: array([-1288472.91829674, -4720774.50988504,  4079682.41622987])
```

```
[7]: np.linalg.norm(uccs_64_ecef - ucb_64_ecef)
```

```
[7]: 130027.00871657248
```

3.1.2 32-bit

```
[8]: uccs_32 = uccs_64.astype(np.float32)
     ucb_32 = ucb_64.astype(np.float32)
```

```
[9]: uccs_32_ecef = np.array([*geodetic2ecef(*uccs_32)])
     uccs_32_ecef
```

```
[9]: array([-1270194.43542948, -4807306.06965539,  3984365.91329425])
```

```
[10]: ucb_32_ecef = np.array([*geodetic2ecef(*ucb_32)])
      ucb_32_ecef
```

```
[10]: array([-1288472.84310863, -4720774.64402719,  4079682.24881102])
```

```
[11]: np.linalg.norm(uccs_32_ecef - ucb_32_ecef)
```

```
[11]: 130026.8880842706
```

3.1.3 16-bit

```
[12]: # Complete on your own
```

3.1.4 Discussion

Provide a brief discussion here

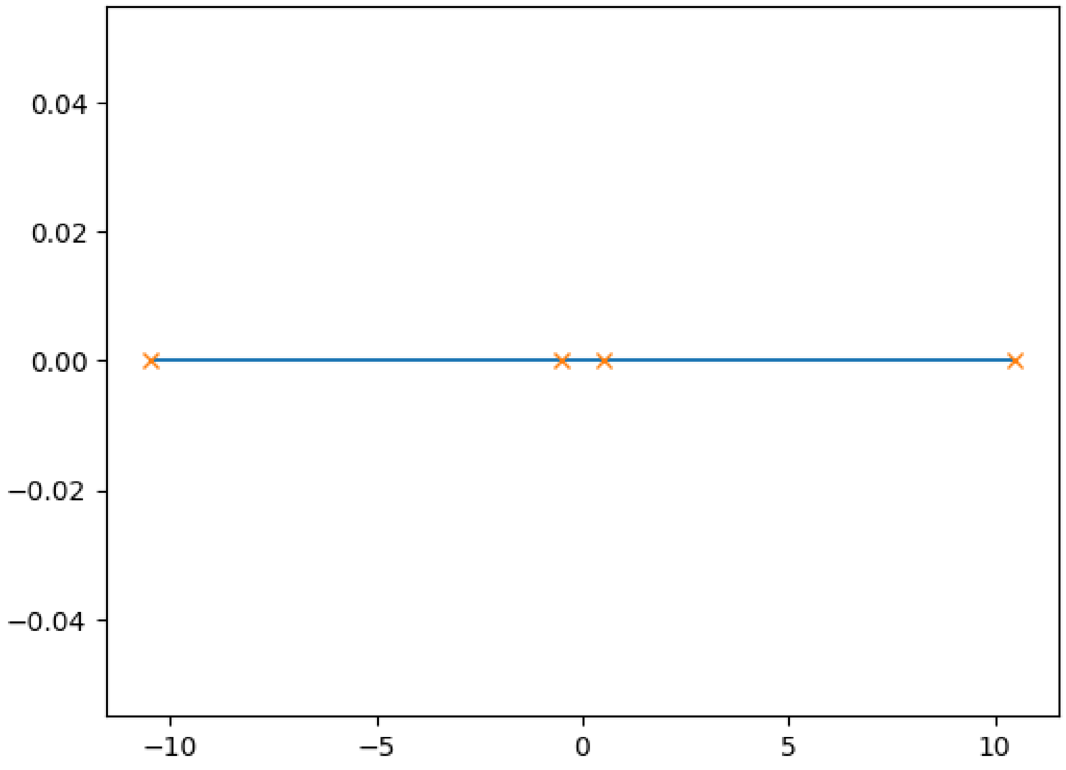
4 Number Line (10)

On a number line, place a sequence of at least 10 numbers (both +/-), along with the largest possible value (+/-) represented in the 8-bit tiny notation. You may choose the numbers yourself. You should observe the distance between the numbers, and make some comments. Here is an example of two numbers:

[13]:
```
import matplotlib.pyplot as plt
```

[14]:
```
x = np.array([-10.5, -0.5, 0.5, 10.5])
y = np.zeros_like(x)
```

[15]:
```
plt.plot(x, y);
plt.plot(x, y, 'x');
```



4.0.1 Discussion

Provide a brief discussion here