HW6-RyanSponzillipdf

October 11, 2024

1 ASTR 310 HW 6

1.0.1 1. Working with NumPy

Avoid explicit iterations over arrays. In other words, do all your computations without using for- or while-loops, list comprehensions or other techniques of that nature. You may use them in printing the output of your code, if you want to format it nicely.

a) The function `numpy.random.random_sample()` allows you to create an array with random values in the range [0.0, 1.0). You can multiply the output by a scaling factor and add an offset to it to get it to lie in a different range. Create a 10 x 10 array initialized with random numbers that lie between 3 and 10, then compute the average of the array. [5 pts]

[2]:
```
import numpy as np

sample = (np.random.random_sample(100).reshape((10,10)) * 7) + 3
print(sample)
print(sample.mean())
```

```
[[3.34340957 8.47609813 6.97934897 3.26704454 8.57493448 6.52821741
  5.03385851 8.20270673 8.49291355 3.51261185]
 [4.52551492 6.42231223 4.61468879 7.44790129 4.66538997 5.33945399
  8.94561209 8.74483127 6.25188523 7.08350782]
 [8.98834529 9.98926405 6.13428888 8.9058151  4.60941899 8.91117867
  3.85733379 7.43546198 9.62888526 3.0566145 ]
 [4.28795574 3.19051283 7.05601689 4.26085392 4.63685347 3.72520596
  6.17567324 5.28220011 8.68198804 5.08286435]
 [4.573754   9.82224187 9.12866064 8.52364591 5.94397284 7.71777694
  5.49030202 9.95875185 4.74165914 3.98326033]
 [3.09039225 4.13366622 6.1359299  8.92680114 8.94807998 5.98912825
  9.14393164 9.66015735 9.99188966 3.83133995]
 [3.03975931 5.01293432 3.4990766  7.76225316 9.32149671 8.99327039
  8.42057921 6.70292789 7.62130684 4.58443607]
 [4.21550363 8.80964565 3.54469173 8.35388297 3.56722597 6.14460381
  8.09886327 8.62735881 7.85264633 4.96222122]
 [3.47500821 4.55216113 4.98640317 9.24831589 5.98782506 3.9404525
  6.70515001 5.15571709 3.65836042 7.26324646]
 [7.23231665 7.27304719 4.9718048  4.67797344 9.11185466 7.98153091
  5.97457409 6.85787267 8.90513606 7.2987916 ]]
```

```
6.444765401642202
```

b) Create a 1D array containing angles in degrees: 0, 15, 30, … 90 (i.e., every 15 degrees up to 90). Now create 3 new arrays with the sine, cosine, and tangent of the elements of the first array. Finally, calculate the inverse sine, inverse cosine, and inverse tangent of the arrays above and compare to the original angles. Compute the fractional error where possible. [5 pts]

[3]:
```
deg = np.linspace(0,90,7)
rads = np.pi/180

deg_sin = np.sin(deg*rads)
deg_cos = np.cos(deg*rads)
deg_tan = np.tan(deg*rads)

deg_sin_asin = np.arcsin(deg_sin)
deg_cos_acos = np.arccos(deg_cos)
deg_tan_atan = np.arctan(deg_tan)

error = deg - deg_sin_asin / rads

print(deg)
print(deg_sin)
print(deg_cos)
print(deg_tan)
print(error)
```

```
[ 0. 15. 30. 45. 60. 75. 90.]
[0.         0.25881905 0.5        0.70710678 0.8660254  0.96592583
 1.        ]
[1.00000000e+00 9.65925826e-01 8.66025404e-01 7.07106781e-01
 5.00000000e-01 2.58819045e-01 6.12323400e-17]
[0.00000000e+00 2.67949192e-01 5.77350269e-01 1.00000000e+00
 1.73205081e+00 3.73205081e+00 1.63312394e+16]
[0.00000000e+00 1.77635684e-15 3.55271368e-15 7.10542736e-15
 7.10542736e-15 0.00000000e+00 0.00000000e+00]
```

c) NumPy has a standard deviation function, numpy.std(), but here we'll write our own that works on a 1D array. The standard deviation is a measure of the "width" of the distribution of numbers in the array. Given an array a and its average $\bar{a}$, the standard deviation is given by

$$\sigma = \left[\frac{1}{N} \sum_{i=0}^{N-1} (a_i - \bar{a})^2 \right]^{1/2} .$$

Card-carrying statistics aficionados will remind you that if you have to compute $\bar{a}$ from the data themselves you should normalize by $1/(N-1)$ rather than $1/N$, but we won't worry about that for now. Write a function to calculate the standard deviation for an input array a. Test your function on a random array with 100 elements, and compare to the built-in numpy.std(). [5 pts]

[4]:
```
def my_std(a: np.ndarray):
    ave = np.average(a)
    a = a - ave
    a = a**2
    s = np.sum(a)
    return (s / a.size)**0.5
```

[5]:
```
test = np.random.random_sample(100)
print(my_std(test))
print(np.std(test))
```

```
0.2935595145551184
0.2935595145551184
```

d) Given a scalar value x and a NumPy array v, write code to find the value in v that is closest to x. Do not assume v is sorted. Test your code with an array containing 100 random numbers between 5 and 50 and $x = 25$. [5 pts]

[6]:
```
def closest(x, v: np.ndarray):
    temp = np.abs(v - x)
    return v[np.argmin(temp)]
```

[7]:
```
test2 = np.random.random_sample(100)*45 + 5

print(closest(25, test2))
```

```
24.944819976571548
```

e) Create a one-dimensional array x containing 100 randomly selected values between 0 and 2π. Sort the array. Create another 1D array y in which the jth element contains

$$y_j = \sin^2 x_j \cos x_j \ .$$

Next use the trapezoidal rule to integrate y on this interval. Recall that the trapezoidal rule is given by

$$\int_a^b y(x)\,dx \approx \sum_{j=0}^{N-2} \frac{1}{2}(y_j + y_{j+1})(x_{j+1} - x_j) \ .$$

Finally, produce two arrays containing, respectively, those x- and y-points from the grid for which $y > 0.25$. [5 pts]

[28]:
```
x = np.random.random_sample(100)*2*np.pi
x.sort()
y = np.sin(x)*np.sin(x)*np.cos(x)
print(x)
print(y)
```

```
y_integral = np.sum(0.5 * (y[1:] + y[:-1]) * np.diff(x))
print(y_integral)

xx, yy = np.meshgrid(x,y)
mask = y > 0.25
print(xx[mask])
print(yy[mask])
```

```
[4.14981188e-03 4.61232142e-03 1.59439711e-02 7.58159783e-02
 1.62828787e-01 2.78465216e-01 3.83550712e-01 4.62570704e-01
 4.84303053e-01 5.15010228e-01 5.61478549e-01 6.55904200e-01
 7.22507366e-01 8.94693816e-01 1.03878336e+00 1.06253469e+00
 1.07013976e+00 1.12109563e+00 1.24629684e+00 1.26866157e+00
 1.27605944e+00 1.30377022e+00 1.31201492e+00 1.32536800e+00
 1.36517577e+00 1.46010027e+00 1.47830579e+00 1.49538370e+00
 1.51976398e+00 1.55512885e+00 1.60680865e+00 1.65059526e+00
 1.72702160e+00 1.77507732e+00 1.81155877e+00 1.84758329e+00
 1.86247470e+00 1.88173660e+00 1.98303724e+00 2.07353176e+00
 2.19052561e+00 2.19105546e+00 2.34705614e+00 2.36651969e+00
 2.42389004e+00 2.49280063e+00 2.49816818e+00 2.61627501e+00
 2.70294874e+00 2.74017024e+00 2.82043708e+00 2.88034665e+00
 2.96872749e+00 3.00397988e+00 3.06811204e+00 3.19676440e+00
 3.21360980e+00 3.35797263e+00 3.48556592e+00 3.54676458e+00
 3.60604214e+00 3.62829399e+00 3.66477876e+00 3.70951351e+00
 3.74452722e+00 3.75622283e+00 3.76135490e+00 3.84149390e+00
 3.86097677e+00 3.99954283e+00 4.01034582e+00 4.10877936e+00
 4.18815123e+00 4.32649430e+00 4.34103093e+00 4.35809072e+00
 4.50432824e+00 4.90226017e+00 4.96747986e+00 4.99390822e+00
 5.22859128e+00 5.34160249e+00 5.34630818e+00 5.38301155e+00
 5.44556049e+00 5.45991860e+00 5.50591581e+00 5.68547541e+00
 5.69481615e+00 5.72382186e+00 5.78854867e+00 5.79166885e+00
 5.83290730e+00 5.91047501e+00 5.96746008e+00 6.01968598e+00
 6.05995586e+00 6.07692761e+00 6.16614618e+00 6.22830092e+00]
[ 1.72206915e-05  2.12731318e-05  2.54156366e-04  5.72057701e-03
  2.59321130e-02  7.26485287e-02  1.29862411e-01  1.78211294e-01
  1.91846338e-01  2.11131223e-01  2.39965923e-01  2.94768249e-01
  3.28021791e-01  3.80728073e-01  3.76737636e-01  3.71400256e-01
  3.69408503e-01  3.52555581e-01  2.86423193e-01  2.71212679e-01
  2.65975739e-01  2.45492763e-01  2.39144624e-01  2.28627923e-01
  1.95663191e-01  1.09121985e-01  9.15708880e-02  7.49135088e-02
  5.08774659e-02  1.56629879e-02 -3.59578673e-02 -7.92077328e-02
 -1.51823960e-01 -1.94514637e-01 -2.24886432e-01 -2.52860297e-01
 -2.63781534e-01 -2.77314271e-01 -3.36344498e-01 -3.69966547e-01
 -3.84879348e-01 -3.84873833e-01 -3.56709894e-01 -3.49809500e-01
 -3.25818645e-01 -2.90906548e-01 -2.87957694e-01 -2.17580317e-01
 -1.63303516e-01 -1.40532228e-01 -9.45486601e-02 -6.44472975e-02
 -2.91449470e-02 -1.86401367e-02 -5.37514625e-03 -3.03620714e-03
 -5.16408801e-03 -4.50192592e-02 -1.07062534e-01 -1.42795308e-01
```

```
 -1.79387550e-01 -1.93353519e-01 -2.16248417e-01 -2.43898979e-01
 -2.64860431e-01 -2.71674758e-01 -2.74630478e-01 -3.17374038e-01
 -3.26593446e-01 -3.74276697e-01 -3.76469440e-01 -3.84737044e-01
 -3.75137857e-01 -3.23065994e-01 -3.15096027e-01 -3.05174822e-01
 -1.97749190e-01  1.82009771e-01  2.36266750e-01  2.56373222e-01
  3.73333667e-01  3.84683117e-01  3.84509365e-01  3.81442163e-01
  3.69502903e-01  3.65634100e-01  3.50621486e-01  2.61783175e-01
  2.56234403e-01  2.38670338e-01  1.98341029e-01  1.96379931e-01
  1.70533326e-01  1.23494939e-01  9.16483343e-02  6.54982406e-02
  4.77931097e-02  4.10533644e-02  1.35424403e-02  3.00474105e-03]
-0.002388277461297956
[[4.14981188e-03 4.61232142e-03 1.59439711e-02 … 6.07692761e+00
  6.16614618e+00 6.22830092e+00]
 [4.14981188e-03 4.61232142e-03 1.59439711e-02 … 6.07692761e+00
  6.16614618e+00 6.22830092e+00]
 [4.14981188e-03 4.61232142e-03 1.59439711e-02 … 6.07692761e+00
  6.16614618e+00 6.22830092e+00]
 …
 [4.14981188e-03 4.61232142e-03 1.59439711e-02 … 6.07692761e+00
  6.16614618e+00 6.22830092e+00]
 [4.14981188e-03 4.61232142e-03 1.59439711e-02 … 6.07692761e+00
  6.16614618e+00 6.22830092e+00]
 [4.14981188e-03 4.61232142e-03 1.59439711e-02 … 6.07692761e+00
  6.16614618e+00 6.22830092e+00]]
[[0.29476825 0.29476825 0.29476825 … 0.29476825 0.29476825 0.29476825]
 [0.32802179 0.32802179 0.32802179 … 0.32802179 0.32802179 0.32802179]
 [0.38072807 0.38072807 0.38072807 … 0.38072807 0.38072807 0.38072807]
 …
 [0.35062149 0.35062149 0.35062149 … 0.35062149 0.35062149 0.35062149]
 [0.26178317 0.26178317 0.26178317 … 0.26178317 0.26178317 0.26178317]
 [0.2562344  0.2562344  0.2562344  … 0.2562344  0.2562344  0.2562344 ]]
```

1.1 2. Rise and set times of Betelgeuse with Astropy

Using Astropy we can get the altitude and azimuth of an object at a given location and time, eg. using

```
from astropy.coordinates import EarthLocation, SkyCoord, AltAz
from astropy.time import Time


obsloc   = EarthLocation.of_address("1002 W. Green St., Urbana, IL")
target   = SkyCoord.from_name("Betelgeuse")
obstime  = Time(Time.now(), location=obsloc)
altaz    = target.transform_to(AltAz(obstime=obstime, location=obsloc))
```

Here `obstime` by default uses the UTC time scale, and the `altaz` object has altitude (alt) and azimuth (az) attributes that are Angle objects. Notice that here the observation time is taken to be "now," i.e. whenever the code is run. The Time constructor can also accept different time specifications; use `help(Time)` or the Astropy documentation to find out how to create a Time

object for a given date and time.

With this information, write a function to use binary search to find the times on a given date when Betelgeuse has an altitude of zero to within one second, and determine which is the rise time and which is the set time. We suggest that you specify the date using a (month, day, year) tuple. You can lift your binary search code from HW 3. Hints:

a) Take your initial search intervals to be (obstime-12*u.hour, obstime) and (obstime, obstime+12*u.hour), where obstime is taken to be noon CST on the specified date.

b) If you difference two Time objects you get a TimeDelta object that can be directly compared to a float representing a number of seconds, or added to another Time object. So in implementing a binary search you can iterate until the difference between your upper search limit and your lower search limit is less than one second. You can also add one-half the difference to the lower search limit to get the time corresponding to the middle of the range.

c) To determine whether the next iteration of the binary search should search in the range (low, middle) or (middle, high), use the altitude of Betelgeuse at each of the three times. For example, if the altitude is negative for the lower time and positive for the middle time, then the time when the altitude is zero must lie in the (low, middle) interval, and you would set the higher time equal to the middle time for the next iteration.

Evaluate your function for October 14, 2024 and print the rise and set times in a user-friendly format. Express times in our local time zone (ie. CST).

Don't use the `riseset.py` routine that we worked with in Lecture 12, though you can check your results that way if you like.

[25 pts]

[114]:
```
from astropy.coordinates import EarthLocation, SkyCoord, AltAz
from astropy.time import Time, TimezoneInfo
from astropy import units as u
from datetime import datetime

def find_betelgeuse_rise_set(month, day, year):
    obsloc   = EarthLocation.of_address("1002 W. Green St., Urbana, IL")
    target   = SkyCoord.from_name("Betelgeuse")
    # get observing time at noon on the specified date
    date = datetime(year, month, day, tzinfo=TimezoneInfo(utc_offset=-5*u.hour))
    obstime = Time(date) + 12 * u.hour

    # Find Rise
    low = obstime - (12*u.hour)
    high = obstime + (12*u.hour)
    middle = obstime
    while ((high-low).to(u.second) > 1*u.second):
        alt_low = target.transform_to(AltAz(obstime=low, location=obsloc)).alt
        alt_middle = target.transform_to(AltAz(obstime=middle,␣
↪location=obsloc)).alt
        alt_high = target.transform_to(AltAz(obstime=high, location=obsloc)).alt
```

```
        if (alt_low < 0) & (alt_middle > 0):
            high = middle
            middle = low + (high - low) / 2

        elif (alt_middle < 0) & (alt_high > 0):
            low = middle
            middle = low + (high - low) / 2

    rise_time = low

    # Find Set
    low = obstime - (12*u.hour)
    high = obstime + (12*u.hour)
    middle = obstime
    while ((high-low).to(u.second) > 1*u.second):
        alt_low = target.transform_to(AltAz(obstime=low, location=obsloc)).alt
        alt_middle = target.transform_to(AltAz(obstime=middle,␣
↪location=obsloc)).alt
        alt_high = target.transform_to(AltAz(obstime=high, location=obsloc)).alt

        if (alt_low > 0) & (alt_middle < 0):
            high = middle
            middle = low + (high - low) / 2

        elif (alt_middle > 0) & (alt_high < 0):
            low = middle
            middle = low + (high - low) / 2

    set_time = low

    return (target.transform_to(AltAz(obstime=rise_time, location=obsloc)),␣
↪target.transform_to(AltAz(obstime=set_time, location=obsloc)))
```

[115]:
```
from datetime import datetime
from astropy.time import Time, TimezoneInfo

rise, set = find_betelgeuse_rise_set(10, 14, 2024)
cdt = TimezoneInfo(utc_offset=-5*u.hour)
print(f"Betelgeuse rises at {rise.obstime.to_datetime(timezone=cdt).
↪strftime("%H:%M:%S on %m/%d/%y")} and sets at {set.obstime.
↪to_datetime(timezone=cdt).strftime("%H:%M:%S on %m/%d/%y")}")
```

```
Betelgeuse rises at 22:47:32 on 10/14/24 and sets at 11:39:41 on 10/14/24
```

[ ]:
```

```