LAB04-02_NumPy (ASSIGN04)

October 31, 2021

Practice with NumPy

If you have never studied linear algebra you may find this notebook to be overly difficult. Let me know if that´s the case while we are in the lab. Linear algebra is at the basis of data science and machine learning, which makes extensive use of matrix formalism. However, Python is solving all linear algebra problems for you.

Fill in the missing code in the exercises and submit this notebook as part of your Assignment 3. Go as far down the list as you can.

This is a collection of exercises that have been gathered online and in the NumPy documentation. These are meant to be solved on your own during the lab or at home.

Each exercise is tagged with stars, showing the level of challenge. ( ) easy ( ) medium difficulty ( ) advanced

If you encouter difficulties, check my notebooks. If you still encounter difficulties, check the online documentation.

0.0.1 N.B.: the following exercises are also meant for you to familiarise with functions and operations that have not been introduced in class

Please refer to the NumPy official documentation to find out how to solve as many cases as you can. https://docs.scipy.org/doc/numpy/

1. Import the numpy package under the name `np` ( )

```
[1]: import numpy as np
```

2. Print the numpy version and the configuration ( )

```
[2]: print(np)
```

```
<module 'numpy' from '/opt/conda/lib/python3.9/site-packages/numpy/__init__.py'>
```

3. Create a null vector of size 10 ( )

```
[3]: import numpy as np
     x = np.zeros(10)
     print(x)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

4. How to find the memory size of any array ( )

```
[4]: np.size
```

```
[4]: <function numpy.size(a, axis=None)>
```

6. Create a null vector of size 10 but the fifth value which is 1 ( )

```
[5]: x =np.zeros(10)
     x[4] = 1
     print(x)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

7. Create a vector with values ranging from 10 to 49 ( )

```
[6]: x = np.arange(10, 50)
     print(x)
```

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
```

8. Reverse a vector (first element becomes last) ( )

```
[7]: x = x[::-1]
     print(x)
```

```
[49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26
 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10]
```

9. Create a 3x3 matrix with values ranging from 0 to 8 ( )

```
[8]: import numpy as np
     x = np.arange(0,9).reshape(3, 3)
     print(x)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

10. Find indices of non-zero elements from [1,2,0,0,4,0] ( )

```
[9]: x = [1, 2, 0, 0, 4, 0]
     np.nonzero(x)
```

```
[9]: (array([0, 1, 4]),)
```

11. Create a 3x3 identity matrix ( )

```
[10]: import numpy as np
      x= np.identity(3)
```

```
print(x)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

12. Create a 3x3x3 array with random values ( )

[11]:
```
x = np.random.random((3,3,3))
print(x)
```

```
[[[0.50678297 0.62582934 0.74922861]
  [0.17520123 0.40763461 0.16797375]
  [0.84999126 0.00118447 0.71640433]]

 [[0.46655646 0.18516468 0.68689182]
  [0.63876579 0.16999872 0.06894434]
  [0.79501794 0.63493477 0.53127617]]

 [[0.53379473 0.02753648 0.23600209]
  [0.4011877  0.4471707  0.14733219]
  [0.58282113 0.26922874 0.0406232 ]]]
```

13. Create a 10x10 array with random values and find the minimum and maximum values ( )

[12]:
```
x = np.random.random((10,10))
print(x)
minElement = np.amin(x)
print("Minimum value is ", minElement)
```

```
[[0.35035501 0.24773057 0.39634046 0.91570685 0.14980706 0.50762157
  0.62151196 0.56697437 0.78909124 0.76743458]
 [0.4591669  0.12163533 0.02028146 0.52420243 0.48496696 0.40107108
  0.18453862 0.61839059 0.95497079 0.73697851]
 [0.36806573 0.65224547 0.99606131 0.33116469 0.26578667 0.36252263
  0.38484488 0.45717245 0.96921236 0.78711142]
 [0.23673475 0.07216545 0.19777208 0.9473789  0.20547716 0.36746459
  0.43907206 0.6444392  0.40619087 0.80245534]
 [0.69895891 0.5735303  0.18446311 0.64564193 0.39085388 0.79960566
  0.67302114 0.88740639 0.5778195  0.72155251]
 [0.01132694 0.11564321 0.83319573 0.01093676 0.47588611 0.32265263
  0.87553601 0.84706861 0.11333095 0.19763802]
 [0.70699494 0.67079292 0.78763624 0.28340793 0.03237107 0.89414921
  0.48904822 0.636936   0.44222547 0.60804442]
 [0.13601622 0.80738766 0.77516272 0.69829999 0.69896692 0.22683675
  0.44906862 0.87740678 0.65978052 0.34707535]
 [0.24233911 0.46898104 0.64432358 0.92857911 0.75024988 0.59363658
  0.68780529 0.93971965 0.69038707 0.34192225]
 [0.15979474 0.91718856 0.90549972 0.16684173 0.40891926 0.01901552
  0.7241735  0.18407135 0.45098387 0.72747618]]
Minimum value is  0.010936757774734462
```

14. Create a random vector of size 30 and find the mean value ( )

[13]:
```
x = np.random.random(30)
print(x)
mean = np.mean(x)
print("Avarage is:",mean)
```

```
[0.20721079 0.68330409 0.35965734 0.91484678 0.6533464  0.61486396
 0.69252352 0.99892393 0.48378295 0.63433409 0.11998296 0.90651762
 0.14573205 0.75121093 0.97829487 0.16003907 0.73697977 0.26637362
 0.07409586 0.81442131 0.52039743 0.0032002  0.68658725 0.58315185
 0.55100985 0.81128904 0.1658922  0.33110667 0.24889959 0.53909145]
Avarage is: 0.5212355802251281
```

15. Create a 2d array with 1 on the border and 0 inside ( )

[14]:
```
x = np.ones((8,8))
x[1:-1,1:-1] = 0
print(x)
```

```
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]
```

16. How to add a border (filled with 0's) around an existing array? ( )

[15]:
```
#By using np.pad function we can allocate an instant value to the around of the␣
 ↪existing array
#np.pad(array, pad_width=1, mode='constant', constant_values=0)
```

18. Create a 5x5 matrix with values 1,2,3,4 just below the diagonal ( )

[16]:
```
newArray = np.diag(1+np.arange(4), k=-1)
print(newArray)
```

```
[[0 0 0 0 0]
 [1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]]
```

19. Create a 8x8 matrix and fill it with a checkerboard pattern ( )

[17]:
```
array = np.zeros((8,8))
array[1::2, ::2] =1
array[::2, 1::2] = 1
print(array)
```

```
[[0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]]
```

24. Multiply a 5x3 matrix by a 3x2 matrix (real matrix product) ( )

[18]:
```
array1 = np.random.random((5,3))
array2 = np.random.random((3,2))
dot= np.dot(array1, array2)
print(dot)
```

```
[[0.57564739 0.65950572]
 [0.8569689  0.74150172]
 [0.4693859  0.56780322]
 [0.33936001 0.41931713]
 [0.90785271 0.60413686]]
```

25. Given a 1D array, negate all elements which are between 3 and 8, in place. ( )

[19]:
```
arr = np.arange(15)
arr[3:9]= np.multiply(arr[3:9],-1)
print(arr)
```

```
[ 0  1  2 -3 -4 -5 -6 -7 -8  9 10 11 12 13 14]
```

34. How to get all the dates corresponding to the month of July 2016? ( )

[20]:
```
print("July, 2016")
print(np.arange('2017-07', '2017-08', dtype='datetime64[D]'))
```

```
July, 2016
['2017-07-01' '2017-07-02' '2017-07-03' '2017-07-04' '2017-07-05'
 '2017-07-06' '2017-07-07' '2017-07-08' '2017-07-09' '2017-07-10'
 '2017-07-11' '2017-07-12' '2017-07-13' '2017-07-14' '2017-07-15'
 '2017-07-16' '2017-07-17' '2017-07-18' '2017-07-19' '2017-07-20'
 '2017-07-21' '2017-07-22' '2017-07-23' '2017-07-24' '2017-07-25'
 '2017-07-26' '2017-07-27' '2017-07-28' '2017-07-29' '2017-07-30'
 '2017-07-31']
```

35. How to compute ((A+B)*(-A/2)) ? ( )

[21]:
```
A= np.ones(3)*4
B = np.ones(3)*6
print(A)
print(B)
np.add(A,B,out=B)
np.divide(A,2,out=A)
np.negative(A,out=A)
np.multiply(A,B,out=A)
```

```
[4. 4. 4.]
[6. 6. 6.]
```

[21]:
```
array([-20., -20., -20.])
```

36. Extract the integer part of a random array ( )

[22]:
```
randArray = np.random.uniform(0,20,10)
print(randArray)
print (randArray - randArray%1)
print (np.floor(randArray))
print (np.ceil(randArray)-1)
print (randArray.astype(int))
print (np.trunc(randArray))
```

```
[ 4.92041885 18.01108225 17.79564648  8.72524919 17.78457967 11.82326419
 17.51400918  3.19434042 15.40172529  1.9704843 ]
[ 4. 18. 17.  8. 17. 11. 17.  3. 15.  1.]
[ 4. 18. 17.  8. 17. 11. 17.  3. 15.  1.]
[ 4. 18. 17.  8. 17. 11. 17.  3. 15.  1.]
[ 4 18 17  8 17 11 17  3 15  1]
[ 4. 18. 17.  8. 17. 11. 17.  3. 15.  1.]
```

37. Create a 5x5 matrix with row values ranging from 0 to 4 ( )

[23]:
```
matrix1 = np.zeros((5,5))
matrix1 += np.arange(5)
print(matrix1)
```

```
[[0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]]
```

39. Create a vector of size 10 with values ranging from 0 to 1, both excluded ( )

[24]:
```
vect1 = np.linspace(0,1,12,endpoint=True)[1:-1]
print(vect1)
```

```
[0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455
 0.63636364 0.72727273 0.81818182 0.90909091]
```

40. Create a random vector of size 10 and sort it ( )

[25]:
```
import numpy as np
randVector = np.random.random(10)
print(randVector)
randVector.sort()
print("sorted vector is:", randVector)
```

```
[0.02913775 0.71219139 0.93555986 0.08233815 0.14714781 0.41588502
 0.60803623 0.98229723 0.47036144 0.67523606]
sorted vector is: [0.02913775 0.08233815 0.14714781 0.41588502 0.47036144
0.60803623
 0.67523606 0.71219139 0.93555986 0.98229723]
```

42. Consider two random arrays A and B, check if they are equal ( )

[26]:
```
array1 = np.random.randint(0,2,5)
array2 = np.random.randint(0,2,5)
equal = np.allclose(array1,array2)
print(array1)
print(array2)
print("result is :", equal)
```

```
[1 1 1 0 1]
[0 0 0 1 1]
result is : False
```

45. Create random vector of size 10 and replace the maximum value by 0 ( )

[27]:
```
rand_vec = np.random.random(10)
print(rand_vec)
rand_vec[rand_vec.argmax()] = 0
print(rand_vec)
```

```
[0.59041843 0.00452111 0.8621934  0.50725031 0.525351   0.152866
 0.82694739 0.69050381 0.3130946  0.66557683]
[0.59041843 0.00452111 0.         0.50725031 0.525351   0.152866
 0.82694739 0.69050381 0.3130946  0.66557683]
```

46. Create a structured array with x and y coordinates covering the [0,1]x[0,1] area ( )

[28]:
```
array= np.zeros((5,5), [('x',float),('y', float)])
array['x'], array['y'] = np.meshgrid(np.linspace(0,1,5),
np.linspace(0,1,5))
print(array)
```

```
[[(0.  , 0.  ) (0.25, 0.  ) (0.5 , 0.  ) (0.75, 0.  ) (1.  , 0.  )]
 [(0.  , 0.25) (0.25, 0.25) (0.5 , 0.25) (0.75, 0.25) (1.  , 0.25)]
 [(0.  , 0.5 ) (0.25, 0.5 ) (0.5 , 0.5 ) (0.75, 0.5 ) (1.  , 0.5 )]
 [(0.  , 0.75) (0.25, 0.75) (0.5 , 0.75) (0.75, 0.75) (1.  , 0.75)]
 [(0.  , 1.  ) (0.25, 1.  ) (0.5 , 1.  ) (0.75, 1.  ) (1.  , 1.  )]]
```

50. How to find the closest value (to a given scalar) in a vector? ( )

[29]:
```
vect1 = np.arange(10)
print(vect1)
value = np.random.uniform(0,10)
index = (np.abs(vect1-value)).argmin()
print(vect1[index])
```

```
[0 1 2 3 4 5 6 7 8 9]
6
```

55. What is the equivalent of enumerate for numpy arrays? ( )

[30]:
```
a = np.array([[4, 30], [5, 3]])

for index, x in np.ndenumerate(a):

    print(index, x)
```

```
(0, 0) 4
(0, 1) 30
(1, 0) 5
(1, 1) 3
```

57. How to randomly place p elements in a 2D array? ( )

[31]:
```
n = 4
p = 1
array = np.zeros((4,4))
np.put(array, np.random.choice(range(n*n), p, replace=False),5)
print(array)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 5. 0.]
 [0. 0. 0. 0.]]
```

58. Subtract the mean of each row of a matrix ( )

[32]:
```
randomArray = np.random.rand(4, 4)
print("matrix is:\n", randomArray)
Y = randomArray - randomArray.mean(axis=1, keepdims=True)
print("after subtracting :\n", Y)
```

```
matrix is:
 [[0.3776536  0.48163042 0.11251871 0.54838427]
 [0.06074421 0.02036398 0.07453862 0.34920264]
 [0.63867852 0.53026336 0.6307018  0.8688009 ]
 [0.56042905 0.84086842 0.70217296 0.07193944]]
after subtracting :
 [[-0.00239315  0.10158367 -0.26752804  0.16833752]
 [-0.06546815 -0.10584838 -0.05167374  0.22299028]
 [-0.02843263 -0.13684779 -0.03640934  0.20168975]
 [ 0.01657658  0.29701595  0.15832049 -0.47191303]]
```

60. How to tell if a given 2D array has null columns? ( )

[33]:
```python
array = np.random.randint(0,2,(4,4))
print(array)

print((~array.any(axis=0)).any())
```

```
[[0 1 1 1]
 [1 1 0 1]
 [1 1 0 0]
 [0 0 0 0]]
False
```

69. How to get the diagonal of a dot product? ( )

[34]:
```python
array1 = np.random.uniform(0,1,(2,2))
print("Array 1 is\n", array1)
print("Array 2 is\n", array2)
array2 = np.random.uniform(0,1,(2,2))
print("diagonal array1*array2 is :\n", np.diag(np.dot(array1, array2)))
```

```
Array 1 is
 [[0.73152437 0.01679797]
 [0.93646023 0.83850754]]
Array 2 is
 [0 0 0 1 1]
diagonal array1*array2 is :
 [0.35266548 0.75804054]
```

70. Consider the vector [1, 2, 3, 4, 5], how to build a new vector with 3 consecutive zeros interleaved between each value? ( )

[35]:
```python
array1 = np.array([1,2,3,4,5])
ziros = 3
znum = np.zeros(len(array1) + (len(array1)-1)*(ziros))
znum[::ziros+1] = array1
print(znum)
```

```
[1. 0. 0. 0. 2. 0. 0. 0. 3. 0. 0. 0. 4. 0. 0. 0. 5.]
```

89. How to get the n largest values of an array ( )

[36]:
```
numberOfLarg = int(input("insert the number of largest value :"))
array = np.arange(20)
print("Original array is :\n", array)
np.random.shuffle(array)
print ("largest values are :", array[np.argsort(array)[-numberOfLarg:]])
```

```
insert the number of largest value : 4

Original array is :
 [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
largest values are : [16 17 18 19]
```

94. Considering a 10x3 matrix, extract rows with unequal values (e.g. [2,2,3]) ( )

[37]:
```
array = np.random.randint(0,3,(10,3))

x = np.logical_and.reduce(array[:,1:] == array[:,:-1], axis=1)

unequal = array[~x]

print("original array is : \n", array)

print("new arrays with unequal values is :\n", unequal)
```

```
original array is :
 [[1 0 1]
 [0 0 1]
 [0 1 1]
 [0 2 2]
 [2 1 2]
 [0 2 2]
 [2 2 1]
 [0 1 2]
 [0 1 0]
 [1 1 0]]
new arrays with unequal values is :
 [[1 0 1]
 [0 0 1]
 [0 1 1]
 [0 2 2]
 [2 1 2]
 [0 2 2]
 [2 2 1]
 [0 1 2]
 [0 1 0]
 [1 1 0]]
```

```
[ ]:
```