Exercise3 NumPy

April 14, 2018

1 Hochschule Bonn-Rhein-Sieg

2 Learning and Adaptivity, SS18

3 Assignment 01 (15-April-2018)

3.1 Sathiya Ramesh, Pradheep Krishna Muthukrishnan Padmanabhan, Naresh Kumar Gurulingan

4 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Library documentation: http://www.numpy.org/

```
In [1]: from numpy import *
```

5 Task 1: declare a vector using a list as the argument

```
In [2]: vector = array([1,2,3,4,5])
        vector

Out[2]: array([1, 2, 3, 4, 5])
```

6 Task 2: declare a matrix using a nested list as the argument

```
In [3]: matrix = array([[1,2,3], [4,5,6], [7,8,9]])
        matrix
```

```
Out[3]: array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
```

7 Task 3: initialize x or x and y using the following functions: arange, linspace, logspace, mgrid

```
In [4]: x, y = arange(0, 11, 1), arange(11, 22, 1)
        x, y
```

```
Out[4]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
         array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]))
```

```
In [5]: x, y = linspace(0, 11, num= 11), linspace(11, 22, num= 11)
        x, y
```

```
Out[5]: (array([ 0. ,  1.1,  2.2,  3.3,  4.4,  5.5,  6.6,  7.7,  8.8,  9.9, 11. ]),
         array([11. , 12.1, 13.2, 14.3, 15.4, 16.5, 17.6, 18.7, 19.8, 20.9, 22. ]))
```

```
In [6]: x, y = logspace(0, 11, num= 11), logspace(11, 22, num= 11)
        x, y
```

```
Out[6]: (array([1.00000000e+00, 1.25892541e+01, 1.58489319e+02, 1.99526231e+03,
                2.51188643e+04, 3.16227766e+05, 3.98107171e+06, 5.01187234e+07,
                6.30957344e+08, 7.94328235e+09, 1.00000000e+11]),
         array([1.00000000e+11, 1.25892541e+12, 1.58489319e+13, 1.99526231e+14,
                2.51188643e+15, 3.16227766e+16, 3.98107171e+17, 5.01187234e+18,
                6.30957344e+19, 7.94328235e+20, 1.00000000e+22]))
```

```
In [7]: x, y = mgrid, mgrid
        x, y
```

```
Out[7]: (<numpy.lib.index_tricks.nd_grid at 0x7f3b78182a90>,
         <numpy.lib.index_tricks.nd_grid at 0x7f3b78182a90>)
```

```
In [8]: from numpy import random
```

8 Task 4: what is difference between random.rand and random.randn

random.rand returns values sampled from a **uniform distribution** whereas **random.randn** returns values sampled from a **standard normal distribution**.

9 Task 5: what are the functions diag, itemsize, nbytes and ndim about?

```
In [9]: Z = random.randint(0, 20, (3,3))
        Z

Out[9]: array([[ 3,  4, 15],
               [ 2,  0, 12],
               [ 3, 10,  1]])
```

```
In [10]: # diag returns the diagonal elements..
         print('The diagonal elements of array Z are {}'.format(diag(Z)))

The diagonal elements of array Z are [3 0 1]
```

```
In [11]: # itemsize returns the size of a array element in bytes....
         print('The datatype of array Z is {}'.format(Z.dtype))
         print('Size of an element in Z is {} bytes'.format(Z.itemsize))

The datatype of array Z is int64
Size of an element in Z is 8 bytes
```

```
In [12]: # nbytes returns the total number of bytes used by an array...
         print('Number of elements in array Z is {}'.format(Z.size))
         print('Total space required for array Z is {} bytes'.format(Z.nbytes))

Number of elements in array Z is 9
Total space required for array Z is 72 bytes
```

```
In [13]: # ndim gives the number of dimensions in an array...
         print('Array Z has {} dimensions'.format(Z.ndim))

Array Z has 2 dimensions
```

```
In [14]: M = random.randint(0, 20, (3,3))
         M

Out[14]: array([[11, 13, 15],
                [11, 14,  3],
                [18,  4, 13]])
```

```
In [15]: # assign new value
         M[0,0] = 7
         M

Out[15]: array([[ 7, 13, 15],
                [11, 14,  3],
                [18,  4, 13]])
```

```
In [16]: M[0,:] = 0
         M

Out[16]: array([[ 0,  0,  0],
                [11, 14,  3],
                [18,  4, 13]])

In [17]: # slicing works just like with lists
         A = array([1,2,3,4,5])
         A[1:3]

Out[17]: array([2, 3])
```

10 Task 6: Using list comprehensions create the following matrix

array([[0, 1, 2, 3, 4], [10, 11, 12, 13, 14], [20, 21, 22, 23, 24], [30, 31, 32, 33, 34], [40, 41, 42, 43, 44]])

```
In [18]: A = array([[i for i in range(j, j+5)] for j in arange(0, 41, 10)])
         A

Out[18]: array([[ 0,  1,  2,  3,  4],
                [10, 11, 12, 13, 14],
                [20, 21, 22, 23, 24],
                [30, 31, 32, 33, 34],
                [40, 41, 42, 43, 44]])

In [19]: row_indices = [1, 2, 3]
         A[row_indices]

Out[19]: array([[10, 11, 12, 13, 14],
                [20, 21, 22, 23, 24],
                [30, 31, 32, 33, 34]])

In [20]: # index masking
         B = array([n for n in range(5)])
         row_mask = array([True, False, True, False, False])
         B[row_mask]

Out[20]: array([0, 2])
```

10.0.1 Linear Algebra

```
In [21]: v1 = arange(0, 5)
         v1

Out[21]: array([0, 1, 2, 3, 4])

In [22]: v1 + 2

Out[22]: array([2, 3, 4, 5, 6])
```

```
In [23]: v1 * 2

Out[23]: array([0, 2, 4, 6, 8])

In [24]: v1 * v1

Out[24]: array([ 0,  1,  4,  9, 16])

In [25]: dot(v1, v1)

Out[25]: 30

In [26]: dot(A, v1)

Out[26]: array([ 30, 130, 230, 330, 430])

In [27]: # cast changes behavior of + - * etc. to use matrix algebra
         M = asmatrix(A)
         M * M

Out[27]: matrix([[ 300,  310,  320,  330,  340],
                 [1300, 1360, 1420, 1480, 1540],
                 [2300, 2410, 2520, 2630, 2740],
                 [3300, 3460, 3620, 3780, 3940],
                 [4300, 4510, 4720, 4930, 5140]])

In [28]: # inner product
         v1.T * v1

Out[28]: array([ 0,  1,  4,  9, 16])

In [29]: C = asmatrix([[1j, 2j], [3j, 4j]])

In [30]: conjugate(C)

Out[30]: matrix([[0.-1.j, 0.-2.j],
                 [0.-3.j, 0.-4.j]])

In [31]: # inverse
         C.I

Out[31]: matrix([[0.+2. j, 0.-1. j],
                 [0.-1.5j, 0.+0.5j]])
```

10.0.2 Statistics

```
In [32]: mean(A[:,3])

Out[32]: 23.0

In [33]: std(A[:,3]), var(A[:,3])
```

```
Out[33]: (14.142135623730951, 200.0)

In [34]: A[:,3].min(), A[:,3].max()

Out[34]: (3, 43)

In [35]: d = arange(1, 10)
         sum(d), prod(d)

Out[35]: (45, 362880)

In [36]: cumsum(d)

Out[36]: array([ 1,  3,  6, 10, 15, 21, 28, 36, 45])

In [37]: cumprod(d)

Out[37]: array([     1,      2,      6,     24,    120,    720,   5040,  40320,
               362880])

In [38]: # sum of diagonal
         trace(A)

Out[38]: 110

In [39]: m = random.rand(3, 3)

In [40]: # use axis parameter to specify how function behaves
         m.max(), m.max(axis=0)

Out[40]: (0.8709077275056833, array([0.87090773, 0.46087396, 0.71438758]))

In [41]: # reshape without copying underlying data
         n, m = A.shape
         B = A.reshape((1,n*m))

In [42]: # modify the array
         B[0,0:5] = 5

In [43]: # also changed
         A

Out[43]: array([[ 5,  5,  5,  5,  5],
               [10, 11, 12, 13, 14],
               [20, 21, 22, 23, 24],
               [30, 31, 32, 33, 34],
               [40, 41, 42, 43, 44]])

In [44]: # creates a copy
         B = A.flatten()
```

```
In [45]: # can insert a dimension in an array
         v = array([1,2,3])
         v[:, newaxis], v[:,newaxis].shape, v[newaxis,:].shape

Out[45]: (array([[1],
                 [2],
                 [3]]), (3, 1), (1, 3))

In [46]: repeat(v, 3)

Out[46]: array([1, 1, 1, 2, 2, 2, 3, 3, 3])

In [47]: tile(v, 3)

Out[47]: array([1, 2, 3, 1, 2, 3, 1, 2, 3])

In [48]: w = array([5, 6])

In [49]: concatenate((v, w), axis=0)

Out[49]: array([1, 2, 3, 5, 6])

In [50]: # deep copy
         B = copy(A)
```