Exercise3 NumPy

April 15, 2018

1 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 [46]: from numpy import *
         import numpy as np
```

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

```
In [47]: a = np.array([1,2,3])
         type(a)

Out[47]: numpy.ndarray
```

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

```
In [48]: a = np.matrix([[1,2],[3,4]])
```

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

```
In [55]: from numpy import random

         x = np.arange(1,10,2)
```

```
print(x)

x = np.linspace(1,10,7)
print(x)

x = np.logspace(1,10,7)
print(x)

M_1 = np.mgrid[0:5,0:5]
print(M)

M = M_1[0]
```

```
[1 3 5 7 9]
[ 1.   2.5  4.   5.5  7.   8.5 10. ]
[1.00000000e+01 3.16227766e+02 1.00000000e+04 3.16227766e+05
 1.00000000e+07 3.16227766e+08 1.00000000e+10]
[[[7 7 7 7 7]
  [1 1 1 1 1]
  [2 2 2 2 2]
  [3 3 3 3 3]
  [4 4 4 4 4]]

 [[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]]]
```

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

np.random.rand() creates an array of the given shape and populates it with random samples from a uniform distribution i.e. each value is equally likely whereas np.random.randn() creates an array of the given shape and populates it with random samples from a normal distribution i.e. the values closer to the mean are more probable

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

```
In [73]: # assign new value
         M[0,0] = 7

         #M = np.array([[1,2,3],[3,4,5],[4,5,6]])

         a = np.diag(M)
         print(a)
         print(x.itemsize)
```

```
print(y.nbytes)
print(M.ndim)
```

```
[7]
8
400
2
```

```
In [57]: M[0,:] = 0
```

```
In [58]: # slicing works just like with lists
         A = array([1,2,3,4,5])
         A[1:3]
```

```
Out[58]: array([2, 3])
```

7 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 [59]: row_indices = [1, 2, 3]
         A[row_indices]
```

```
Out[59]: array([2, 3, 4])
```

```
In [60]: # index masking
         B = array([n for n in range(5)])
         row_mask = array([True, False, True, False, False])
         B[row_mask]
```

```
Out[60]: array([0, 2])
```

7.0.1 Linear Algebra

```
In [61]: v1 = arange(0, 5)
```

```
In [62]: v1 + 2
```

```
Out[62]: array([2, 3, 4, 5, 6])
```

```
In [63]: v1 * 2
```

```
Out[63]: array([0, 2, 4, 6, 8])
```

```
In [64]: v1 * v1
```

```
Out[64]: array([ 0,  1,  4,  9, 16])
```

```
In [65]: dot(v1, v1)
```

```
Out[65]: 30
```

```
In [66]: dot(A, v1)
```

```
Out[66]: 40
```

```
In [ ]: # inner product
        v.T * v
```

```
In [ ]: # inner product
        v.T * v
```

```
In [14]: C = matrix([[1j, 2j], [3j, 4j]])
```

```
In [15]: conjugate(C)
```

```
Out[15]: matrix([[0.-1.j, 0.-2.j],
                 [0.-3.j, 0.-4.j]])
```

```
In [16]: # inverse
         C.I
```

```
Out[16]: matrix([[0.+2. j, 0.-1. j],
                 [0.-1.5j, 0.+0.5j]])
```

7.0.2 Statistics

```
In [ ]: mean(A[1,:])
```

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

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

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

```
In [ ]: cumsum(d)
```

```
In [ ]: cumprod(d)
```

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

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

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

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

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

```
In [ ]: # also changed
        A
```

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

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

```
In [ ]: repeat(v, 3)
```

```
In [ ]: tile(v, 3)
```

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

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

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