# Week3_LinearAlgebraInPythonByNumpy

October 9, 2020

# 1 Week3: Numpy For Linear Algebra

- Edited by LUXP
- @Copyright: Macau University of Science and Technology

## 1.1 Vector in Numpy

[4]:
```
import numpy as np
vec = np.array([1,2,3,4])
print(vec)

print(np.shape(vec))
print(vec.shape)

print('Broadcasting: ', vec + 1)
print('Sum: ', vec.mean())
print('Vectorization: ', vec**2, vec + vec)
```

```
[1 2 3 4]
(4,)
(4,)
Broadcasting:  [2 3 4 5]
Sum:  2.5
Vectorization:  [ 1  4  9 16] [2 4 6 8]
```

## 1.2 Matrix in Numpy

[5]:
```
## Matrix Generation
myZero = np.zeros([3,5]) # generate a 3*5 zero matrix
print(myZero)
myOnes = np.ones([3,5])  # generate a 3*5 ones matrix
print(myOnes)
myRand = np.random.rand(3,5) # generate a 3*5 matrix with the number in (0~1)
print(myRand)
```

```python
myEyes = np.eye(3,5)    # generate a 3*5 identity matrix, np.eye(3) will get a␣
 ↪square matrix
print(myEyes)
print('Matrix Stack: Horizontal\n', np.hstack((myZero, myOnes)))
print('Matrix Stack: Verticle\n', np.vstack((myZero, myOnes)))
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
[[0.27581062 0.18132378 0.63936813 0.57180763 0.85697761]
 [0.97500135 0.99926431 0.7337     0.95007361 0.53443222]
 [0.65082094 0.50176237 0.48572451 0.76043032 0.45044425]]
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]]
Matrix Stack: Horizontal
 [[0. 0. 0. 0. 0. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 1. 1. 1. 1. 1.]]
Matrix Stack: Verticle
 [[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

[6]:

```python
np.identity(3)
```

[6]:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[7]:

```python
## Matrix Operations
myOnes = np.ones([3,5])
myEyes = np.eye(3,5)
print('myOnes:\n', myOnes)
print('myEyes:\n', myEyes)
## Visit elements:
print(myEyes[1,1], myEyes[1][1])
###### Matrix Sum
print('Sum of Matrix:myOnes+myEyes\n', myOnes + myEyes)
###### sum of all elements of Matrix
```

```
print('Sum of elements: np.sum(myOnes)\n', np.sum(myOnes))  ## NOTICE: np.sum␣
↪is to sum all elements; while sum is to sum each column
print('Sum of each columns: sum(myOnes)\n', sum(myOnes))

## Matrix Multiplication
NewMat = np.mat([[1,2,3], [2, 3, 1],[1,0,0]])
print('Matrix:\n',NewMat)
print('Scalar Matrix Multiplication:\n', 10*NewMat)
print('Matrix Multiplication:\n', NewMat*NewMat)
print('Matrix Elements Multiplication: np.multiply\n', np.multiply(NewMat,␣
↪NewMat))
print('Matrix Elements Power: np.power\n', np.power(NewMat, 2))
```

```
myOnes:
 [[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
myEyes:
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]]
1.0 1.0
Sum of Matrix:myOnes+myEyes
 [[2. 1. 1. 1. 1.]
 [1. 2. 1. 1. 1.]
 [1. 1. 2. 1. 1.]]
Sum of elements: np.sum(myOnes)
 15.0
Sum of each columns: sum(myOnes)
 [3. 3. 3. 3. 3.]
Matrix:
 [[1 2 3]
 [2 3 1]
 [1 0 0]]
Scalar Matrix Multiplication:
 [[10 20 30]
 [20 30 10]
 [10  0  0]]
Matrix Multiplication:
 [[ 8  8  5]
 [ 9 13  9]
 [ 1  2  3]]
Matrix Elements Multiplication: np.multiply
 [[1 4 9]
 [4 9 1]
 [1 0 0]]
Matrix Elements Power: np.power
```

```
 [[1 4 9]
 [4 9 1]
 [1 0 0]]
```

[10]:
```
NewMat
```

[10]:
```
matrix([[1, 2, 3],
        [2, 3, 1],
        [1, 0, 0]])
```

[15]:
```
NewMat[0,0] = 5
```

[16]:
```
NewMat
```

[16]:
```
matrix([[5, 2, 3],
        [2, 3, 1],
        [1, 0, 0]])
```

[7]:
```
## Matrix Transpose
NewMat = np.mat([[1,2,3],[4,5,6],[7,8,9]])
print('Matrix: \n',NewMat)
print('Matrix Transpose:\n',NewMat.T)
print('Matrix after .T: \n', NewMat)
print('Matrix Transpose:\n', NewMat.transpose()) ## same as the .T
print('Matrix after Transpose():\n', NewMat)
```

```
Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix Transpose:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
Matrix after .T:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix Transpose:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
Matrix after Transpose():
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[11]:
```python
## Matrix Shape, Copy, slice, split...
NewMat = np.mat([[1,2,3],[4, 5, 6],[7, 8, 9 ],[10,1,1]])
[m,n] = np.shape(NewMat)
print('Matrix:\n',NewMat)
print('Row and Columns:', m,n)
print('First Row:', NewMat[0]) # same as: NewMat[0,](Bad) and NewMat[0,:]
print('First Column:', NewMat.T[0])
print('Matrix Elements: ', NewMat[2,1]) ## Notice: first number is 0!!!!
print('Matrix Rows:\n', NewMat[[0,1],:]) ## first two rows
print('Matrix Colmuns:\n', NewMat[:,[1,2]])
# copy a matrix to other variable
NewMat2 = NewMat.copy()
# compare two matrix by elements
print(NewMat<NewMat2)
print('New Shape: \n', NewMat.reshape((2,6)))
```

```
Matrix:
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10  1  1]]
Row and Columns: 4 3
First Row: [[1 2 3]]
First Column: [[ 1  4  7 10]]
Matrix Elements:  8
Matrix Rows:
 [[1 2 3]
 [4 5 6]]
Matrix Colmuns:
 [[2 3]
 [5 6]
 [8 9]
 [1 1]]
[[False False False]
 [False False False]
 [False False False]
 [False False False]]
New Shape:
 [[ 1  2  3  4  5  6]
 [ 7  8  9 10  1  1]]
```

### 1.3 Linear Algebra

- Determinant
- Matrix Inverse
- Matrix Rank
- Solving Linear System

- Eigen Values

[9]:
```python
## Linear Algebra
NewMat = np.mat([[1,2,3],[4,5,6],[1,0,0]])
print("matrix:\n",NewMat)
print('Determinant:', np.linalg.det(NewMat))
print('Inverse Matrix:\n', np.linalg.inv(NewMat))
print('check: inv(A)*A\n', np.linalg.inv(NewMat)*NewMat)
print('Rank:\n', np.linalg.matrix_rank(NewMat))
b = [1,2,1]
print('Solve Linear System:\n', np.linalg.solve(NewMat, b))
b2 = [1,2,1]
b2T = np.mat(b2).T
print('Check by: inv(A)*b\n', np.linalg.inv(NewMat)*b2T)

# Eigen values
Evals, Evecs = np.linalg.eig(NewMat)
print('Eigen values: \n', Evals)
print('Eigen Vectors:\n', Evecs) #NOTICE: eigenvectors are stored in columns
print('AX , lambda*X:\n', NewMat*Evecs[:,0], Evals[0]*Evecs[:,0])

# use similarity matrix
sigma = Evals * np.eye(3)
print('sigma,\n ',sigma)
print('V*sigma*V^-1 = A:\n', Evecs*sigma*np.linalg.inv(Evecs))
```

```
matrix:
 [[1 2 3]
 [4 5 6]
 [1 0 0]]
Determinant: -2.9999999999999982
Inverse Matrix:
 [[ 4.44089210e-16 -2.22044605e-16  1.00000000e+00]
 [-2.00000000e+00  1.00000000e+00 -2.00000000e+00]
 [ 1.66666667e+00 -6.66666667e-01  1.00000000e+00]]
check: inv(A)*A
 [[ 1.00000000e+00 -2.22044605e-16  0.00000000e+00]
 [ 4.44089210e-16  1.00000000e+00  4.44089210e-16]
 [ 0.00000000e+00 -1.11022302e-16  1.00000000e+00]]
Rank:
 3
Solve Linear System:
 [ 1.         -2.          1.33333333]
Check by: inv(A)*b
 [[ 1.        ]
 [-2.        ]
 [ 1.33333333]]
Eigen values:
```

```
 [ 6.81573612 -1.1866583   0.37092218]
Eigen Vectors:
 [[-0.3482453  -0.75825177  0.19151266]
 [-0.93600993 -0.12945231 -0.8347106 ]
 [-0.05109431  0.63898072  0.51631494]]
AX , lambda*X:
 [[-2.37354807]
 [-6.37959666]
 [-0.3482453 ]] [[-2.37354807]
 [-6.37959666]
 [-0.3482453 ]]
sigma,
  [[ 6.81573612 -0.          0.        ]
 [ 0.         -1.1866583   0.        ]
 [ 0.         -0.          0.37092218]]
V*sigma*V^-1 = A:
 [[ 1.00000000e+00  2.00000000e+00  3.00000000e+00]
 [ 4.00000000e+00  5.00000000e+00  6.00000000e+00]
 [ 1.00000000e+00 -2.10500438e-16 -8.67941471e-17]]
```

### 1.4 Generate Random Number

Ref: Random

[25]:
```
## Generate a random float number
import numpy as np
np.random.seed(123) ## if set seed(123), the random number will be fixed!!!
print(np.random.rand())

print(np.random.randint(1,7))
```

```
0.6964691855978616
3
```

[ ]:
```
```

[ ]:
```
```