Applied Linear Algebra for IT

Anh H. Vo

1 Collections/Containers

List/Vector

Create a new vector/list as follows

```
import numpy as np

a1 = [0, 2, 4, 6, 8]
a2 = np.arange(0,10)
a3 = np.linspace(0,8,5)
a4 = np.random.randint(0,7, size
    = 5)
```

Add item

```
import numpy as np

lst = [0, 2, 4, 6, 8]
seq = [1, 4]
lst.append(-4) ## add item at
   end: [0, 2, 4, 6, 8, -4]
lst.insert(3, 7) ## insert a
   item at specific index: [0,
   2, 4, 7, 6, 8]
lst.extend(seq) ## add sequence
   of items at end: [0, 2, 4, 6,
    8, 1, 4]
```

Remove item(s)

```
import numpy as np

lst = [0, 2, 4, 6, 8, 4]
lst.remove(4) ## remove first
   item with value 4: [0, 2, 6,
   8, 4]
lst.pop(2) ## remove item at
   specific index: [0, 2, 6, 8,
   4]
del lst[1] ## remove item at
   specific index: [0, 4, 6, 8,
   4]
lst.clear() ## remove all items:
    []
```

To get value in a list: **lst[*index*]**

Set

- **Do not allow duplicate values**
- Set elements are **unchangable** (only remove/ insert new elements)
- Set elements are **unordered** (present in a different order every time when you run)
- **Can not access items in a set by specific index**

Add new item(s)

```
import numpy as np

s = {0, 2, 4, 6, 8}
s.add(5) ## add item: {0, 2, 4,
   5, 6, 8}
s.update({3,9})## add sequence
   of items: {0, 2, 3, 4, 5, 6,
   8, 9}

## unordered
ss = {'xa','ka'}
ss.add('nkk') ## {'ka','xa','nkk
   '}
```

Remove item

```
import numpy as np

s = {0, 2, 4, 6, 8}
s.remove(4) ## remove item: {0,
   2, 5, 6, 8}
s.discard(6) ## remove item: {0,
    2, 5, 8}
s.pop() ## remove first item:
   {2, 5, 8}
s.clear() ## remove all items:
   {}
```

Dictionary

A dictionary is used to store data as key-value pairs.

- A collection is **ordered**
- Set elements are **changable**
- **Do not allow duplicate values**

Add new item

```
import numpy as np

d = {1:'banana', 2:'monkey', 3:'
   apple', 4:'dog', 5:'cat'}
```

```
d.update({8:'tiger', 7: 'pig'})
   ## {1:'banana', 2:'monkey',
   3:'apple', 4:'dog', 5:'cat',
   7:'pig', 8:'tiger'}
d[6] = 'snake' ## {1:'banana',
   2:'monkey', 3:'apple', 4:'dog
   ', 5:'cat', 6:'snake', 7:'pig
   '}
```

Update item

```
import numpy as np

d = {1:'banana', 2:'monkey', 3:'
   apple', 4:'dog', 5:'cat'}
d.update({5:'tiger', 4: 'pig'})
   ## {1:'banana', 2:'monkey',
   3:'apple', 4:'pig', 5:'tiger
   '}
d[3] = 'orange' ## {1:'banana',
   2:'monkey', 3:'orange', 4:'
   pig', 5:'tiger'}
```

Remove item

```
import numpy as np

d = {1:'banana', 2:'monkey', 3:'
   apple', 4:'dog', 5:'cat', 6:'
   pig'}
d.pop(5) ## remove key = 5: {1:'
   banana', 2:'monkey', 3:'apple
   ', 4:'dog', 6:'pig'}
del d[1] ## remove key = 1:{2:'
   monkey', 3:'apple', 4:'dog',
   6:'pig'}
d.popitem() ## remove last item:
    {2:'monkey', 3:'apple', 4:'
   dog'}
d.clear() ## remove all items
```

To get value in a dictionary: **d.get(*key*), d[*key*]**

Tuple

This type data is utilized to store multiple items in a single variable. Tuple is **ordered and unchangeable**.

```
import numpy as np

## create a new tuple
t1 = tuple((1, "k", 7.4))
t2 = (1, "k", 7.4)
t3 = 1, "k", 7.4
```

2 Linear Algebra in python

Vector

Access elements in a vector

```
import numpy as np

lst = np.array([1, 3, 5, 7, 9])
x1 = lst[:-1] ## [1, 3, 5, 7]
   from 0 to len - 1
x_even = lst[::2] ## [1, 5, 9]
   get even positions
x_odd = lst[1::2] ## [3, 7] get
   odd positions
x2 = lst[-3:-1] ## [5, 7]
```

Reverse vector

```
import numpy as np

lst = np.array([1, 3, 5, 7, 9,
   10])
x1 = lst[::-1] ## [10, 9, 7, 5,
   3, 1]
lst.reverse() ## [10, 9, 7, 5,
   3, 1]
x2 = lst[::-2] ## [10, 7, 3]
```

Combine two vectors. It notes that axis = 0 means w.r.t columns and axis = 1 means w.r.t rows

```
import numpy as np

u = np.array([[1, 3, 5, 1, 2,
   3]])
v = np.array([[1, 3, 5, 7, 2,
   0]])

# horizontal stack:
# [1 3 5 7 2 0 1 3 5 1 2 3]
np.concatenate((v,u), axis=1)
np.hstack((v,u))
# vertical stack:
# [[1 3 5 7 2 0],
# [1 3 5 1 2 3]]
np.concatenate((v,u), axis=0)
np.vstack((v,u))
```

Find specific elements in vector with **np.where** or logic operations

```
import numpy as np

v = np.array([[1, 3, 5, 12, 2,
   30]])
v1 = v[np.where(((v%2==0) & (v
   >2)))] # [12, 30]
```

```
v2 = v[(v%2==0)&(v>2)] # [12, 30]
```

Find and replace

```
import numpy as np

v = np.array([[1, 3, 5, 12, 2, 30]])
# [[1  3  5  100  2  100]]
v1 = np.where(((v%2==0) & (v>2)), 100, v)
v[(v%2==0)&(v>2)] = 100
```

Vector operations: **+, -, **, /, *, dot product, inner product**

```
import numpy as np

u = np.array([[1, 3, 5, 1, 2]])
v = np.array([[1, 3, 5, 7, 2]])

# [[ 2  6 10  8  4  3]]
va = u + v
## [[ 0  0  0 -6  0  3]]
vs = u - v
# [[1, 9, 15, 1, 4]]
ve = u**2
# [[0.5 1.5 2.5 3.5 1.  0. ]]
vd = v/2
# [[ 1  9 25  7  4  0]]
vm = u*v
np.inner(u,v) # [[46]]
np.dot(u,v.T) # [[46]]
vdt = u@v.T # [[46]]
```

In addition, some functions are provided such as: **np.sum, np.max, np.min, np.mean, np.argmax, np.argmin, np.unique**

Matrix

Reshape vector/matrix

```
import numpy as np

lst = np.array([1, 3, 5, 7, 9, 10])
v1 = lst.reshape(2,3) ## [[1, 3, 5], [7, 9, 10]]
```

Create a matrix

```
import numpy as np
import np.random

# create a matrix manually
A = np.array([[1, 4, 2, 1],[0, 2, 3, 1],[1, 4, 3, 1]])
# create a random matrix
R = np.random.randint(-3, 5, size=(3,3))

# create special matrix
O = np.zeros((3, 3))
T = np.ones((3, 3))
I = np.eye((3, 3))
```

To add a row or column in matrix: **from numpy import ***

```
from numpy import *

M = np.array([[1, 1, 1],[2, 2, 2],[3, 3, 3]])
# Add a new row
M1 = append(M,[[5, 5, 5]],0)
# Add a new column
M2 = append(M,[[6], [6], [6]],1)
```

To delete row(s) or column(s) in matrix

```
from numpy import *

M = np.array([[1, 1, 1],[2, 2, 2],[3, 3, 3]])
# Delete row(s)
M1 = delete(m,[1,3],0)
# Delete column
M2 = delete(m,[0], 1)
```

Access elements in matrix

```
import numpy as np
import np.random

# create a matrix
A = np.array([[1, 4, 2, 1],[0, 2, 3, 1],[1, 4, 3, 1]])

#Get A[0, 0<=j<=3] - the first row in A
A1 = A[0,:]
#Get A[0<=j<=2, 1] - the second column in A
A2 = A[:,1]
#Get A[rows are odd, columns are even]
A3 = A[1::2,::2]
# The horizontal flip
A4 = np.flipud(A)
# The vertical flip
A4 = np.fliplr(A)
```

Some functions are also provided such as: **np.sum, np.max, np.min, np.mean, np.argmax, np.argmin, np.unique, np.sort**. If we compute matrix row-wise (axis = 1), otherwise column-wise will be axis = 0.

```python
import numpy as np
A = np.array([[1, 4, 2, 1],[0, 2, 3, 1]])
#Matrix is sorted w.r.t row
np.sort(A, axis=0)
#Matrix is sorted w.r.t column
np.sort(A, axis=1)
# Flatten and sort
np.sort(A, axis=None)
```

Find elements that satisfy condition:

```python
import numpy as np

A = np.array([[1, 4, 2, 1],[0, 2, 3, 1],[1, 4, 3, 1],[11, 4, -3, 1],[20, 54, -31, 10], [120, 504, -131, 100]])
# return a bool matrix
ind1 = A<10
# return a tuple with row and column
ind2 = np.where(A<10)
# return a bool value to verify that exists at least one true value
b1 = np.any(A<10)
# return a bool value to verify that all true value
b2 = np.all(A<10)
```

Find and replace elements in matrix

```python
import numpy as np

A = np.array([[1, 4, 2, 1],[0, 2, 3, 1]])
A1 = np.where(A==2, np.inf, A)
```

Matrix operations

```python
import numpy as np

A = np.array([[1, 4, 2, 1],[0, 2, 3, 1]])
B = np.array([[1, 1, -2, 1],[0, -2, 0, 1]])
# A+B
M_a = A + B
# A - B
M_s = A - B
# outer product
Mdt1 = A@B
Mdt2 = np.dot(A,B)
Mdt3 = np.matmul(A, B)
# element-wise product
Mel = A*B
```

Matrix analysis

```python
import numpy as np

v2 = np.array([[1 , 1],[ - 1, 0]])

# [[1, 1],[- 1, 0]]
t1 = np.transpose(v2)
# [[1 , 1],[-1, 0]]
t2 = v2.T
 # [[0, - 1], [1, 1]]
iv2 = np.linalg.inv(v2)
det_m = np.linalg.det(v2) # 1.0
Trace = np.trace(v2) # 1.0
diag = np.diag(v2) # [1, 0]
rank = np.linalg.matrix_rank(v2)
    # 2
```

Solving the system of equations $Ax = b$

```python
import numpy as np

A = np.array([[2, 1, -2],[3, 0, 1], [1, 1, -1]])
b = np.transpose([-3, 5, -2])
x = np.linalg.solve(A, b) # [1. -1. 2.]
```

Norm

- Norm-vector

```python
import numpy as np

k = [1, 0, 2, -1]
# Eclidean norm (l2)
l2 = np.linalg.norm(k)# 2.449
# l1
l1 = np.linalg.norm(k, 1) # 4
# l-inf (max)
l_inf_max = np.linalg.norm(k, np.inf)
# l-inf (min)
l_inf_min = np.linalg.norm(k, -np.inf)
```

- Norm-matrix

```python
import numpy as np

A = np.array([[1, 0],[-2, 1]])
# Euclidean-norm
e_n = np.linalg.norm(A)
# Frobenius
```

```
f_n = np.linalg.norm(A, 'fro')
# l-inf norm
l_inf = np.linalg.norm(A, np.inf) # 3.0
# l-1 norm
l1 = np.linalg.norm(A, 1) # 3.0
```

Matrix factorization:

- Eigen decomposition

```
import numpy as np

A = np.array([[2, 1, -2],[3, 0, 1], [1, 1, -1]])

# Find the eigen values of A
evalue = np.linalg.eigvals(A)
# Find the eigen values and eigen vectors
evalue, evector = np.linalg.eig(A)
```

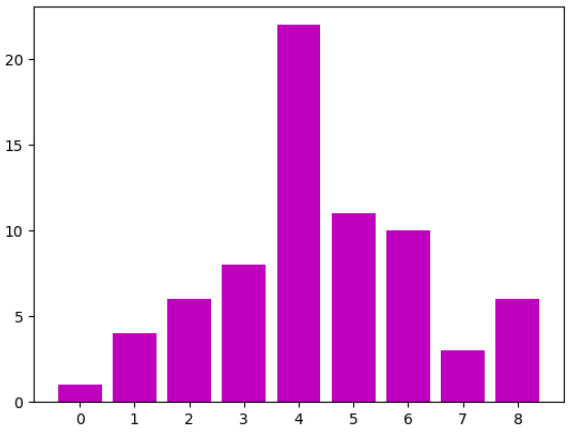
- Singular value decomposition

```
import numpy as np

A = np.array([[1,0], [-2, 1]])
U, S, V = np.linalg.svd(A)
```

3 Graph

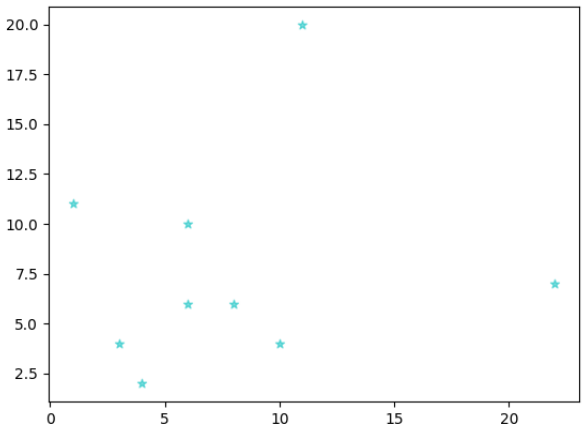
Draw a histogram of the following vector:

```
import matplotlib.pyplot as plt

l = [1, 4, 6, 8, 22, 11, 10, 3, 6]
x = [str(i) for i in np.arange(len(l))]
plt.bar(np.arange(len(l)),l, color='m')
plt.xticks(np.arange(len(l)),x)
plt.show()
```



Draw scatter plot:

```
import matplotlib.pyplot as plt
import numpy as np

x = [1, 4, 6, 8, 22, 11, 10, 3, 6]
y = [11, 2, 10, 6, 7, 20, 4, 4, 6]

plt.scatter(x, y, c = 'c', alpha =0.5, marker = '*')
plt.show()
```



4 Read and Write file

.mat file

Read .mat file

```
from numpy import *
import scipy.io

mat = scipy.io.loadmat('./data/lab08-09/house.mat')
data = mat['P']
```

Write .mat file

```
from numpy import *
import scipy.io

```

```
p = np.array([[1, 1],[2, 2],[3, 3]])
data = {"P":p}
scipy.io.savemat("data.mat", data)
```

.npy file

Read .npy file

```
from numpy import *

data = np.load('./data.npy')
```

Write .npy file

```
from numpy import *

p = np.array([[1, 1],[2, 2],[3, 3]])
np.save('data.npy', p)
```

.txt file

Read .txt file

```
from numpy import *

data = np.loadtxt('./data.txt', dtype=int)
```

Write .txt file

```
from numpy import *

p = np.array([[1, 1],[2, 2],[3, 3]])
np.savetxt('data.txt', p, fmt = '%d')
```