5. 과학계산용라이브러리

Yoonjoong Kim

Department of Computer Engineering, Hanbat National University

yjkim@hanbat.ac.kr

5. 과학계산용 라이브러리

- 5.1 Numpy 모듈
  - 5.1.1 개요
    1. numpy array의 정의 및 사용하기
       1. np.zeros, np.ones, np.arrange, broadcasting
    2. 원소단위연산
    3. Indexing slicing
    4. Array boolean indexing(mask)
    5. Numpy 함수
       1. 원소단위 연산
       2. 통계함수
       3. 기타함수
  - 5.1.2 리스트 3 종류
  - 5.1.3 행렬의곱
  - 5.1.4 UA,WA,F1의 계산
- 5.2 Scipy 라이브러리
  - 5.2.1 scipy.linalg.eig 고유값
  - 5.2.3 scipy.linalg.inv 역행렬

5.1 Numpy 모듈

5.1.1 numpy의 개요

- numpy는 C언어로 구현된 파이썬 라이브러리로써, 고성능의 수치계산을 위해 제작되었습니다.
- Numerical Python의 합성어이고 벡터 및 행렬 연산에 매우 편리한 기능을 제공합니다.
- 데이터분석을 할 때 사용되는 라이브러리인 pandas와 matplotlib의 기반으로 사용되기도 합니다.
- 숫자연산을 할 때 **리스트형**이나 내장 **array 클래스**의 객체를 사용하는 것보다 **Numpy 패키지**를 사용하면 더 효율적이고 편리하게 연산을 수행할 수 있다.
- 기본적으로 array라는 단위로 데이터를 관리하며 이에 대해 연산을 수행합니다. array는 말그대로 행렬이라는 개념으로 생각하시면 됩니다.

- numpy 자료형
  - 부호가 있는 정수 int(8, 16, 32, 64)
  - 부호가 없는 정수 uint(8 ,16, 32, 54)
  - 실수 float(16, 32, 64, 128)
  - 복소수 complex(64, 128, 256)
  - 불리언 bool
  - 문자열 string_
  - 파이썬 오프젝트 object
  - 유니코드 unicode_

1. numpy array의 정의 및 사용하기

```
>>> import numpy as np
>>> list1 = [1,2,3,  4,5]   #python List
>>> list2 = [1,2,3,3.5,4]  #python List
>>> arr1=np.array(list1)
>>> arr2=np.array(list2)
>>> arr1.shape
(5,)
>>> arr1.dtype
dtype('int32')
>>> arr2.dtype
dtype('float64')
>>> arr3=np.array([[1,2],[3,4],[5,6]])
>>> arr3
array([[1, 2],
       [3, 4],
       [5, 6]])
```

1.1 np.zeros, np.ones, np.arrange, broadcasting

```
>>> np.zeros((2,3))        #zeros
array([[0., 0., 0.],
       [0., 0., 0.]])
>>> np.ones(5)             #ones
array([1., 1., 1., 1., 1.])
>>> np.ones((2,3))
array([[1., 1., 1.],
       [1., 1., 1.]])
>>> np.arange(5)           #arange
array([0, 1, 2, 3, 4])
>>> np.arange(1,5)
array([1, 2, 3, 4])
>>> np.arange(1,5,2)
array([1, 3])

#broadcast
>>> arr1=np.array([[10,11,12],
...                [20,21,22]])
>>> arr2=np.array([11,12,13])
>>> arr1+arr2
array([[21, 23, 25],
       [31, 33, 35]])
#arr1+arr2
# = [[10,11,12], +[11,12,13]
#     [20,21,22]])
# = [[10,11,12], +[[11,12,13]
#    [20,21,22]])   [11,12,13]]
# = [[21, 23, 25],
#     [31, 33, 35]]
```

2. 원소단위연산

```
>>> arr1=np.array([[1,2,3],[4,5,6]])
>>> arr2=np.array([[11,12,13],[14,15,16]])
>>> arr1.shape
(2, 3)
>>> arr2.shape
(2, 3)
>>> arr1
array([[1, 2, 3],
       [4, 5, 6]])
>>> arr2
array([[11, 12, 13],
       [14, 15, 16]])
>>> arr1+arr2              #덧셈
array([[12, 14, 16],
       [18, 20, 22]])
>>> arr1-arr2              #뺄셈
array([[-10, -10, -10],
       [-10, -10, -10]])
>>> arr1*arr2              #곱셈
array([[11, 24, 39],
       [56, 75, 96]])
>>> arr1/arr2              #나눗셈
array([[0.09090909, 0.16666667, 0.23076923],
       [0.28571429, 0.33333333, 0.375     ]])
>>> arr3=np.array([11,12,13])
>>> arr1+arr3              # broad casting
Array([[12, 14, 16],
       [15, 17, 19]])
>>> arr1*10                # broad casting
array([[10, 20, 30],
       [40, 50, 60]])
>>> arr1**2                # broad casting
array([[ 1,  4,  9],
       [16, 25, 36]], dtype=int32)
```

3. Indexing slicing

```
>>> arr1=np.arange(5)
>>> arr1
array([0, 1, 2, 3, 4])
>>> arr1[0]    #0번째요소
0
>>> arr1[4]    #4번째요소
4
>>> arr1[0:2] #0~1번째요소
array([0, 1])
>>> arr1[:2]   #처음부터 2전까지
array([0, 1])
>>> arr1[2:-1] #2~마지막전까지
array([2, 3])
>>> arr1[2:]   #2~끝까지
array([2, 3, 4])
>>> arr2 = np.array([[1,2,3],
...                  [4,5,6],
...                  7,8,9]])
>>> arr2
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> arr2[1,:]       #1행
array([4, 5, 6])
>>> arr2[:,2]       #2열
array([3, 6, 9])
>>> arr2[:1,:1]     #
array([[1]])
>>> arr2[:2,:2]     #
array([[1, 2],
       [4, 5]])
```

4. Array boolean indexing(mask)

```
>>> names=np.array(['Kim','Lee',"Kim"])
>>> score=np.array([[70,71,72],
...                 [80,81,82],
...                 [90,91,92]])
>>> score[[True,False,True]]
array([[70, 71, 72],
       [90, 91, 92]])
>>> score[names=='Kim']
array([[70, 71, 72],
       [90, 91, 92]])
>>> score[(names=='Kim' ) | (names== 'Lee' )]
array([[70, 71, 72],
       [80, 81, 82],
       [90, 91, 92]])
>>> score[names=='Kim',:2]
array([[70, 71],
       [90, 91]])
```

5. Numpy 함수
5.1 원소단위 연산

```
>>> arr1=np.array([[1,2.4],
...       [11.6,-12]])
>>> np.abs(arr1)          #abs
array([[ 1. ,  2.4],
       [11.6, 12. ]])
>>> np.square(arr1)       #square
array([[  1.  ,   5.76],
       [134.56, 144.  ]])
>>> np.sqrt(arr1)         #sqrt
array([[1.        , 1.54919334],
       [3.40587727,        nan]])
>>> np.exp(arr1)          #exp
array([[2.71828183e+00, 1.10231764e+01],
       [1.09097799e+05, 6.14421235e-06]])
>>> np.log(arr1)          #log
array([[0.        , 0.87546874],
       [2.4510051 ,        nan]])
>>> np.log10(arr1)        #log10
array([[0.        , 0.38021124],
       [1.06445799,        nan]])
>>> np.log2(arr1)         #log2
array([[0.        , 1.26303441],
       [3.5360529 ,        nan]])
>>> np.sign(arr1)         #sign
array([[ 1.,  1.],
       [ 1., -1.]])
```

```
>>> np.ceil(arr1)         #ceil
array([[  1.,   3.],
       [ 12., -12.]])
>>> np.floor(arr1)        #floor
array([[  1.,   2.],
       [ 11., -12.]])
>>> np.isnan(arr1)        #isnan
array([[False, False],
       [False, False]])
>>> np.isnan(np.sqrt(arr1))
array([[False, False],
       [False,  True]])
>>> np.isinf(arr1)        #isinf
array([[False, False],
       [False, False]])
>>> np.cos(arr1)          #cos,cosh,sin,sinh,tan,tanh
array([[ 0.54030231, -0.73739372],
       [ 0.56828963,  0.84385396]])
>>>
```

```
>>> arr1=np.array([[0,1,2],
...                [3,4,5]])
>>> arr2=np.array([[1,0,2],
...                [3,5,6]])
>>> np.add(arr1,arr2)   #add substract,multiply,devide
array([[ 1,  1,  4],
       [ 6,  9, 11]])
>>> np.maximum(arr1,arr2) #maximum,minimum
array([[1, 1, 2],
       [3, 5, 6]])
```

5.2 통계함수
#sum, mean, std, min, max,argmax,argmin,cumsum,cumprod

```
arr1=np.array([[0,1,2],
...            [3,4,5]])
>>> np.sum(arr1)          #sum
15
>>> np.sum(arr1,axis=1)
array([ 3, 12])
>>> np.sum(arr1,axis=0)
array([3, 5, 7])
>>> np.mean(arr1)         #mean
2.5
>>> np.mean(arr1,axis=1)
array([1., 4.])
>>> np.mean(arr1,axis=0)
array([1.5, 2.5, 3.5])
>>> np.argmax(arr1)       #argmax
5
>>> np.argmax(arr1,axis=1)
array([2, 2], dtype=int64)
>>> np.argmax(arr1,axis=0)
array([1, 1, 1], dtype=int64)
>>> np.argmin(arr1)       #argmin
0
>>> np.argmin(arr1,axis=1)
array([0, 0], dtype=int64)
>>> np.argmin(arr1,axis=0)
array([0, 0, 0], dtype=int64)
>>> np.cumsum(arr1)       #cumsum
array([[ 0,  1,  3, 6, 10, 15] dtype=int32)
>>> np.cumsum(arr1,axis=1)
array([[ 0,  1,  3],
       [ 3,  7, 12]], dtype=int32)
>>> np.cumprod(arr1)      #cumprod
array([0, 0, 0, 0, 0, 0], dtype=int32)
>>> np.cumprod(arr1,axis=1)
array([[ 0,  0,  0],
       [ 3, 12, 60]], dtype=int32)
```

5.3 기타함수

```
>>> arr1=np.array([[20,1,4],
...                [9, 4,3]])
>>> np.sort(arr1)
array([[ 1,  4, 20],
       [ 3,  4,  9]])

>>> np.sort(arr1,axis=1)
array([[ 1,  4, 20],
       [ 3,  4,  9]])

>>> np.sort(arr1,axis=0)
array([[ 9,  1,  3],
       [20,  4,  4]])

>>> np.sort(arr1,axis=0)[::-1]
array([[20,  4,  4],
       [ 9,  1,  3]])
```

5.1.2 리스트 3 종류

5.1.2 리스트의 3 종류

- Python의 리스트
  - 선언과 연산
  - **다양한 데이터형 요소를 허용하지만**
- array 클래스 리스트
  - 내장 array 클래스를 이용
  - **균일한 숫자데이터형의 요소만 허용한다.**
- Numpy 리스트
  - 선언은 다양한 형의 요소 선언가능
  - 연산에서는 동일 형의 원소연산만 가능

1. Python list

```
>>> a=[1,2,3]
>>> b=[3,4,5]
>>> a*2
[1, 2, 3, 1, 2, 3]
>>> a+a
[1, 2, 3, 1, 2, 3]
>>> a+b
[1, 2, 3, 4, 5, 6]
```

2. Numpy array

```
>>> import numpy as np
>>> a=np.array([1,2,3])
>>> b=np.array([3,4,5])
>>> 2*a
array([2, 4, 6])
>>> a+a
array([2, 4, 6])
>>> c=np.array([3.0,4.0,5.0])
>>> a+c
array([4., 6., 8.])
>>> d=np.array([3.0,4,5.0])
>>> d
array([3., 4., 5.])
>>> d=np.array([3.0,4,'a'])
>>> d
array(['3.0', '4', 'a'], dtype='<U32')
>>> a+d
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: ufunc 'add' did not contain a loop with
signature matching types dtype('<U32') dtype('<U32')
dtype('<U32')
```

5.1.3 행렬의 곱

- 행렬의 곱

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 5 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} -1 & 3 & 5 \\ 1 & 4 & 2 \end{bmatrix}$$

$$\mathbf{AB}^T = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 5 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 3 & 4 \\ 5 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 1\cdot(-1)+2\cdot 3+3\cdot 5 & 1\cdot 1+2\cdot 4+3\cdot 2 \\ 3\cdot(-1)+2\cdot 3+5\cdot 5 & 3\cdot 1+2\cdot 4+5\cdot 2 \end{bmatrix}$$

$$= \begin{bmatrix} 20 & 15 \\ 28 & 21 \end{bmatrix},$$

$$\mathbf{A}^T\mathbf{B} = \begin{bmatrix} 1 & 3 \\ 2 & 2 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} -1 & 3 & 5 \\ 1 & 4 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 1\cdot(-1)+3\cdot 1 & 1\cdot 3+3\cdot 4 & 1\cdot 5+3\cdot 2 \\ 2\cdot(-1)+2\cdot 1 & 2\cdot 3+2\cdot 4 & 2\cdot 5+2\cdot 2 \\ 3\cdot(-1)+5\cdot 1 & 3\cdot 3+5\cdot 4 & 3\cdot 5+5\cdot 2 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 15 & 11 \\ 0 & 14 & 14 \\ 2 & 29 & 25 \end{bmatrix}.$$

```
>>> np.dot(a, b.transpose())
array([[20, 15],
       [28, 21]])
>>> np.dot(a.T, b)
array([[ 2, 15, 11],
       [ 0, 14, 14],
       [ 2, 29, 25]])
>>>
```

Transpose() 함수는 행렬의 전치행렬을 만드는 함수이고,
dot() 함수는 행렬의 내적을 계산하는 함수이다.
a.transpose() 대신에 a.T라고 쓸 수도 있다

5.1.4 UA,WA,F1 계산

•다음은 감정인식결과 혼돈행렬(confusion matrix)이다. UA(unweighted accuracy)와 WA(weighted accuracy)를 Numpy 모듈을 이용하여 계산하자.

	ang	neu	hap	sad	tot	recall		
ang	99	3	15	18	135	0.7333		
neu	2	3	0	4	9	0.3333		
hap	14	8	7	7	36	0.1944		
sad	3	1	3	43	50	0.8600		
tot	118	15	25	72	230			
precision	0.8390	0.2000	0.2800	0.5972			0.4791	AP
						0.5303	WA(AR)	
						0.6609	UA	
						0.5034	F1	

```
>>> cm=np.array(
...  [[99,  3, 15, 18],
...  [ 2,  3,  0,  4],
...  [14,  8,  7,  7],
...  [ 3,  1,  3, 43]])
>>> cmm=np.zeros((5,5))
>>> cmm[:-1,:-1]=cm
>>> cmm[:-1,-1]=cm.sum(1)
>>> cmm[-1,:-1]=cm.sum(0)

>>> AR=(np.diag(cm)/cm.sum(1)).mean()
>>> AP=(np.diag(cm)/cm.sum(0)).mean()
>>> F1=2*AP*AR/(AP+AR)
>>> UA=np.diag(cm).sum()/cm.sum()
>>> WA=AR
>>> UA,WA,F1
(0.66086, 0.53027, 0.50336)
```

```
 import numpy as np
>>> cm=np.array(
...  [[99,  3, 15, 18],
...  [ 2,  3,  0,  4],
...  [14,  8,  7,  7],
...  [ 3,  1,  3, 43]])
>>> cmm=np.zeros((5,5))
>>> cmm
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

```
>>> cmm[:4,:4]=cm
>>> cmm
array([[99.,  3., 15., 18.,  0.],
       [ 2.,  3.,  0.,  4.,  0.],
       [14.,  8.,  7.,  7.,  0.],
       [ 3.,  1.,  3., 43.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
>>> cmm[:4,4]= cm.sum(1)
>>> cmm
array([[ 99.,   3.,  15.,  18., 135.],
       [  2.,   3.,   0.,   4.,   9.],
       [ 14.,   8.,   7.,   7.,  36.],
       [  3.,   1.,   3.,  43.,  50.],
       [  0.,   0.,   0.,   0.,   0.]])
```

```
>>> cmm[-1,:]=cm.sum(0)
>>> cmm
array([[ 99.,   3.,  15.,  18., 135.],
       [  2.,   3.,   0.,   4.,   9.],
       [ 14.,   8.,   7.,   7.,  36.],
       [  3.,   1.,   3.,  43.,  50.],
       [118.,  15.,  25.,  72., 230.]])
```

5.2 Scipy라이브러리

- 수학, 과학, 공학분야에서 사용할 수 있는 다양한 패키지로 구성되어 있다.
  - 특수함수(scipy.special)
  - 신호처리(scipy.signal)
  - 영상처리(scipy.ndimage)
  - 푸리에변환(scipy.fftpack)
  - 최적화(scipy.optimize)
  - 수치적적분(scipy.integrate)
  - 선형대수(scipy.linalg)
  - 입출력(scipy.io)
  - 통계(scipy.stats)
  - 고속실행(scipy.weave)
  - 클러스터링알고리듬(scipy.cluster)
  - 희소행렬(sparse matrices) (scipy.sparse)
  - 보간(scipy.interpolate)
  - 기타(e.g. scipy.odr, scipy.maxentropy)

  - ScipyCookbook (https://scipy-cookbook.readthedocs.io/)

5.2 Scipy라이브러리(cont.)

5.2.1 scipy.linalg.eig(A) 다음행렬의 고유값과 고유벡터

- 다음행렬의 고유 값과 고유벡터
  - 정방행렬 A의 선형변환(A**v**) 결과가 자신의 상수 배(λ)가 되는 0이 아닌 벡터를 행렬A의 고유벡터(**v**)고한다.

$$A\mathbf{v}=\lambda\mathbf{v} \qquad \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}\begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}=\lambda\begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}$$

- 다음 행렬의 고유 값과 고유벡터를 구하라.

$$\mathbf{A}=\begin{bmatrix} 3 & 4 \\ 4 & 3 \end{bmatrix}$$

  - $\lambda=7,\ V=[\ \ 0.70710678\ 0.70710678].T$
  - $\lambda=-1, V=[-0.70710678\ 0.70710678].T$

```
import numpy as np
from scipy import linalg as LA
A = np.array([[3,4],[4,3]])
w, v = LA.eig(A)
print('eigen values are ', w)
print('The first eigen vector is ', v[:,0], \
  '\n and the correspoding eigen value is', w[0])
print('The second eigen vector is ', v[:,1], \
  '\n and the correspoding eigen value is', w[1])
```

```
eigen values are  [ 7. -1.]
The first eigen vector is  [ 0.70710678  0.70710678]
 and the correspoding eigen value is 7.0
The second eigen vector is  [-0.70710678  0.70710678]
 and the correspoding eigen value is -1.0
```

5.2 Scipy라이브러리(cont.)

5.2.2 scipy.linalg.inv() 역행렬

- 역행렬
  - 행렬A의 역행렬은 A와 곱해서 항등행렬 E가 나오는 행렬을 A의 역행렬 A^{-1} 이라 한다.

$$AB = BA = E, B = A^{-1}$$

- $A = \begin{bmatrix} 3 & 4 \\ 4 & 3 \end{bmatrix}$, $A^{-1} = ?$

```
>>> import scipy.linalg as LA
>>> A=np.array([[1,2],[3,4]])
>>> A
array([[1, 2],
       [3, 4]])
>>> B=LA.inv(A)
>>> B
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
>>> np.dot(A,B)
array([[1.0000000e+00, 0.0000000e+00],
       [8.8817842e-16, 1.0000000e+00]])
>>> np.dot(B,A)
array([[1.00000000e+00, 0.00000000e+00],
       [1.11022302e-16, 1.00000000e+00]])
>>>
```