과학계산용 라이브러리

Numpy
Scipy
Matlpltlib

내용

1. Numpy 모듈
   - 리스트와 array()의 차이
   - 행렬, 다차원행렬
   - element-wise oepration :+,-,*,/,sqrt()
   - 행렬의 곱(내적)
   - UA,WA,F1의 계산
2. Scipy 라이브러리
   - scipy.linalg (선형대수)
     - 고유값,역행렬
3. Matplolib
   - 연속정현파신호 그리기
   - 이산정현파신호 그리기

4. Numpy 모듈

- 개요
  - Numpy는 과학연산을 위해서 설계된 다차원배열(multidimensional array) 처리를 위한 Python 확장 패키지이다.
  - 숫자연산을 할 때 리스트형이나 내장array 클래스의 객체를 사용하는 것보다 Numpy 패키지를 사용하면 더 효율적이고 편리하게 연산을 수행할 수 있다.
- 사용 방법
  - import numpy
    - 모듈의 함수나 변수를 사용할 때는 numpy.함수 또는 numpy.변수형식을 사용한다.
  - import numpy as newname
    - Numpy를 불러 들여 새로운 이름인 newname으로 사용한다. 즉, newname.함수 또는 newname.변수형식을 사용한다.
  - from numpy import *
    - Numpy모듈에 있는 모든 객체를 불러들여서 현재에 이름공간과 합치는 방법이므로 모듈 이름없이 함수와 변수를 바로 사용한다. Numpy모듈의 일부객체만 사용할 때는 *대신에 객체이름만 쓸 수도 있다.

1. Numpy 모듈

- 리스트 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

```
>>> 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]
```

```
>>> import array
>>> a=array.array('i',[1,2,3])
>>> a
array('i', [1, 2, 3])
>>> b=array.array('i',[3,4,5])
>>> a+b
array('i', [1, 2, 3, 3, 4, 5])
>>> a=array.array('i',[1,2,3.0])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: integer argument expected, got float
>>> c=array.array('f',[3,4,5])
>>> c
array('f', [3.0, 4.0, 5.0])
>>> a+c
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: bad argument type for built-in operation
```

```
>>> 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')
```

4. Numpy 모듈을 이용한 행렬연산

- 사용 예
  - 원소가 4 개인 1차원배열 a와 원소가 10 개인 2 차원배열 b를 만드는 예제
  - array( )는 리스트를 numpy의 배열로 변환하는 함수이다
  - arange( )는 주어진 구간에서 균일한 간격의 숫자를 만드는 함수이다. 앞예에서 numpy.arange(10)은 0부터 9까지 1 간격의 숫자를 만드는 함수이다.
  - Numpy의 주요객체는 동일한 데이터형을 갖는 원소들로 구성된 다차원배열이다.
  - 각 원소의 인덱스는 음이 아닌 정수의 튜플로 표시된다
  - Numpy에서 차원은 축(axis)이라고 부르고, 축의 개수를 rank라고 부른다.
  - 콘솔에서 help(arange)

```
>>> import numpy as np
>>> a=np.array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
>>> b=np.arange(10).reshape(2,5)   #2차원배열
>>> b
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> b.shape      #2차원배열 b의 구조
(2, 5)
>>> b.ndim       #2차원배열 b의 차원
2
>>> b.size       #2차원배열 b의 원소 수
10
>>> b[1,2]       #2차원배열 b의 원소 참조-튜플
7
>>> b[(1,2)]     #2차원배열 b의 원소 참조-튜플
7
>>> b[1][2]      #2차원배열 b의 원소 참조
7
```

4. Numpy 모듈을 이용한 행렬연산

- Numpy의 element-wise operation
  - Numpy의 배열에 대한 원소끼리의 연산
  - element-wise operator : +,-,*,/,sqrt
  - 예,예를 들어, 다음과 같은2×3 행렬

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

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} 1-1 & 2+3 & 3+5 \\ 3+1 & 2+4 & 5+2 \end{bmatrix} = \begin{bmatrix} 0 & 5 & 8 \\ 4 & 6 & 7 \end{bmatrix},$$

$$\mathbf{A} - \mathbf{B} = \begin{bmatrix} 1-(-1) & 2-3 & 3-5 \\ 3-1 & 2-4 & 5-2 \end{bmatrix} = \begin{bmatrix} 2 & -1 & -2 \\ 2 & -2 & 3 \end{bmatrix}.$$

$$A * B = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 5 \end{bmatrix} * \begin{bmatrix} -1 & 3 & 5 \\ 1 & 4 & 2 \end{bmatrix} = \begin{bmatrix} 1*(-1) & 2*3 & 3*5 \\ 3*1 & 2*4 & 5*2 \end{bmatrix} = \begin{bmatrix} -1 & 6 & 15 \\ 3 & 8 & 10 \end{bmatrix}$$

$$A * B = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 5 \end{bmatrix} * \begin{bmatrix} -1 & 3 & 5 \\ 1 & 4 & 2 \end{bmatrix} = \begin{bmatrix} 1*(-1) & 2*3 & 3*5 \\ 3*1 & 2*4 & 5*2 \end{bmatrix} = \begin{bmatrix} -1 & 6 & 15 \\ 3 & 8 & 10 \end{bmatrix}$$

$$A * 2 = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 5 \end{bmatrix} * 2 = \begin{bmatrix} 2 & 4 & 6 \\ 6 & 4 & 10 \end{bmatrix}$$

```
>>> a=np.array([1,2,3,3,2,5]).reshape(2,3)
>>> b=np.array([[-1,3,5],[1,4,2]])
>>> a
array([[1, 2, 3],
       [3, 2, 5]])
>>> b
array([[-1,  3,  5],
       [ 1,  4,  2]])
>>> a+b
array([[0, 5, 8],
       [4, 6, 7]])
>>> a-b
array([[ 2, -1, -2],
       [ 2, -2,  3]])
>>> a*b
array([[-1,  6, 15],
       [ 3,  8, 10]])
>>> a/b
array([[-1.        ,  0.66666667,  0.6       ],
       [ 3.        ,  0.5       ,  2.5       ]])
>>> a*2
array([[ 2,  4,  6],
       [ 6,  4, 10]])
>>> a**2
array([[ 1,  4,  9],
       [ 9,  4, 25]], dtype=int32)
>>> np.sqrt(a)
array([[1.        , 1.41421356, 1.73205081],
       [1.73205081, 1.41421356, 2.23606798]])
>>> np.sqrt(b)
__main__:1: RuntimeWarning: invalid value
encountered in sqrt
array([[       nan, 1.73205081, 2.23606798],
       [1.        , 2.        , 1.41421356]])
>>>
```

4. Numpy 모듈을 이용한 행렬 곱셈

- 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

4. Numpy 모듈을 이용한 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. 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. scipy.linalg.eig(A) 다음행렬의 고유값과 고유벡터

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

    Av=λ**v**

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

```
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])
```

- 다음 행렬의 고유 값과 고유벡터를 구하라.

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

  - λ=7, V=[0.70710678 0.70710678]^T
  - λ=−1,V=[-0.70710678 0.70710678]^T

```
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. 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]])
>>>
```

6. Matplotlib 모듈

- Matplotlib는 데이터 시각화에 이용되는 모듈이다.
  - plot() 함수 : 연속 데이터의 그래프 출력
  - stem() : 이산데이터의 그래프 출력

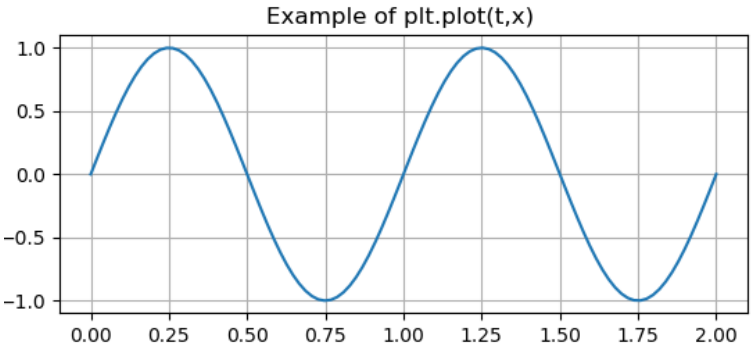
```
plt.plot(t,y)      #plot t and y using default line style
                    # and color
plt.plot(t,y,'bo')#plot t and y using blue circle marker
plt.plot(y)        #plot y using t as index array 0..N-1
plt.plot(y,'r+' ) #ditto, but with red pulses

plt.stem(n,y,linefmt='r-',markerfmt='bo',basefmt='k-')
```

- 연속 정현파 신호의 그래프 그리기 예
  - 시간 $t = [0,2]$의 폐구간에서 $y = x(t) = \sin(2\pi 1t)$ 의 그래프를 그리시오.

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

t=np.linspace(0,2,100) #2초간 100 samples
y=np.sin(2*np.pi*1*t)
plt.title('Example of plt.plot(t,x) ')
plt.plot(t,y)
plt.grid()
plt.show()
```



5. Matplotlib 모듈

- 이산 정현파 신호의 그래프 그리기
  - 시간 $t = [0,2]$의 폐구간의 연속정현파신호 $y = x(t) = \sin(2\pi 1t)$ 의 그래프를 샘플링주파수 fs 로 샘플링한 이산 신호 $y = x[nTs] = \sin(2\pi 1nTs)$ 를 그리시오.

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

def x(t) :
return np.sin(2*np.pi*1*t)

plt.subplot(211)
t=np.linspace(0,2,300) #2초간 300 samples그래프를 그리기 위하여
y=x(t)
plt.title('Plot (t,y) example of y=x(t)=sin(2 pi 1 t) ')
plt.plot(t,y)
plt.xlim([0,2])
plt.grid()

plt.subplot(212)
fs=10; Ts=1/fs      # fs=10 samples /1 sec
n=np.arange(0,2*fs) # max sample = 2초*fs, 2초간 20sample
y=x(n*Ts)
plt.xticks(n)
plt.stem(n,y,linefmt='r-',markerfmt='bo',basefmt='k-')
plt.xlim([0,20])
plt.title('Stem(n,Y) example of y[n]=x(nTs)=sin(2 pi 1 nTs) ')
plt.tight_layout()
plt.grid()
plt.show()
```