본 문건은 다음 링크의 내용을 기반으로 작성되었다.

* <https://docs.scipy.org/doc/numpy/user/quickstart.html>
* <https://docs.python.org/3/tutorial/>

NumPy에서 다루는 데이터의 형식은 동질(homogeneous)의 다차원 배열(array)이다. 그것은 양의 정수들의 튜플에 의해 인덱스 화 된 같은 타입의 요소들 (보통 숫자들)의 테이블이다. NumPy에서 차원은 축(axis)라고 불린다.

예를 들어, 3D 공간 [1, 2, 1]에있는 점의 좌표는 하나의 축(차원)을 만든다. 이 축에는 그 안에 3 개의 요소가 있으므로 길이가 3이라고 말한다(length = 3). 아래 배열에는 2 개의 축(하나는 x축 하나는 y축으로 생각해도 무방)이 있다. 첫 번째 축은 길이가 2이고 두 번째 축은 길이가 3이다.

[[ 1., 0., 0.],

[ 0., 1., 2.]]

NumPy의 배열 클래스는 ndarray라고 불린다. 단순히 array라고 할 수도 있다. Ndarray의 속성은 다음과 같은 것들이 있다.

ndarray.ndim # 축의 숫자

ndarray.shape # 차원에 관한 정보

ndarray.size # ndarray 원소의 총 수

Ndarray.dtype # 원소들의 유형(type)

만약 행이 n이고 열이 m 인 n x m 행렬인 경우 ndarray.shape 의 아웃풋은 (n, m) 이고 ndarray.size는 n\*m 이다. 아래 예들을 살펴보자.

>>> import numpy as np # numpy는 np 라는 이름으로 줄여서 사용하는 것이 일반적이다.

>>> a = np.arange(15).reshape(3, 5) # 원소의 수가 15이고 3\*5(행이 3, 열이 5)인 2차원 배열(즉 행렬)

그리고 각각 명령어와 아웃풋은 다음과 같다.

>>> print(a)

[[ 0 1 2 3 4]

[ 5 6 7 8 9]

[10 11 12 13 14]]

>>> a.shape

(3, 5)

>>> a.ndim

2

>>> a.dtype.name

'int64'

>>> a.itemsize

8

>>> a.size

15

>>> type(a)

<type 'numpy.ndarray'>

>>> b = np.array([6, 7, 8])

>>> b

array([6, 7, 8])

>>> type(b)

<type 'numpy.ndarray'>

배열(array)를 만드는 방법을 알아보자. 가장 쉬운 방법은 array 함수를 이용하는 것이다.

>>> a = np.array([2,3,4]) # 정수 array 만들기

>>> a.dtype # 정수로 이루어진 배열이므로 아웃풋은 dtype('int64')

>>> b = np.array([1.2, 3.5, 5.1]) # 실수 array 만들기

>>> b.dtype # 실수 array이므로 아웃풋은 dtype('float64')

흔히하는 실수는 [ ]를 빠뜨리는 것이다.

>>> a = np.array(1,2,3,4) # WRONG

>>> a = np.array([1,2,3,4]) # RIGHT

이차원 array는 다음과 같다. 고차원의 array도 비슷한 방식으로 만들 수 있다.

>>> np.array([(1.5,2,3), (4,5,6)]) # 방법 1

>>> np.array([[1.5,2,3], [4,5,6]]) # 방법 2

Array를 만들때 type을 명확하게 표현해줄 수 있다.

>>> np.array( [ [1,2], [3,4] ], dtype=complex ) # 복소수

특정한 숫자로 채워져 있는 array는 매우 쉽게 만들 수 있다.

>>> np.zeros( (3,4) ) # 0으로 채워진 3\*4 array

array([[ 0., 0., 0., 0.],

[ 0., 0., 0., 0.],

[ 0., 0., 0., 0.]])

>>> np.ones( (2,3,4), dtype=np.int16 ) # 정수 1로 채워진 2\*3\*4 array

array([[[ 1, 1, 1, 1],

[ 1, 1, 1, 1],

[ 1, 1, 1, 1]],

[[ 1, 1, 1, 1],

[ 1, 1, 1, 1],

[ 1, 1, 1, 1]]], dtype=int16)

수열(sequence)를 만드는 방법은 다음과 같다.

>>> np.arange( 10, 30, 5 ) # 10부터 시작하여 5간격으로 30미만(즉 25)까지

>>> np.arange( 0, 2, 0.3 ) # 0에서 시작하여 0.3 간격으로 2 미만까지

>>> np.linspace( 0, 2, 9 ) # 9 numbers from 0 to 2

다음은 0에서 2π까지 100개의 균등한 간격의 숫자를 뽑아내서 각 숫자에서 sin(x) 함수 값을 계산하고 이를 그래프로 그려보자.

>>> x = np.linspace( 0, 2\*np.pi, 100 ) # x is 100 numbers from 0 to 2π

>>> f = np.sin(x) # 각 x에서 sin 값을 평가

π값은 numpy에서 가져오므로 np.pi 가 되고, sin 함수도 numpy에서 가져와서 np.sin이 된다.

그래프를 그리기 위하여 matplotlib이라는 python 그래프 라이브러리(plotting library)를 부르고 그 중에서도 pyplot이라는 모듈(module)을 불러보자. 그리고 이를 이용하여 위 x 값들을 x-축으로 한 f 들의 그래프를 그릴 수 있다.

>>> import matplotlib.pyplot as plt

>>> plt.plot(x,f) # x-축은 x, 그래프는 f 값으로

>>> plt.show() # sin 함수를 보여준다.

그외에 수열을 만드는 방법은 다음과 같다.

>>> a = np.arange(6)

>>> print(a)

위와 같이 하면 결과값은 [0 1 2 3 4 5]이다.

>>> b = np.arange(12).reshape(4,3)

>>> print(b)

위의 결과값은 0에서 11까지 정수를 출력하여 4\*3 행렬로 만든다. 결과값은 다음과 같다.

[[ 0 1 2]

[ 3 4 5]

[ 6 7 8]

[ 9 10 11]]

비슷하게 3D array를 다음과 같이 만들 수 있다.

>>> c = np.arange(24).reshape(2,3,4) # 3d array

Array가 너무 크면 다음과 같이 중간값을 생략한다.

>>> print(np.arange(10000))

[ 0 1 2 ..., 9997 9998 9999]

>>> print(np.arange(10000).reshape(100,100))

[[ 0 1 2 ..., 97 98 99]

[ 100 101 102 ..., 197 198 199]

[ 200 201 202 ..., 297 298 299]

...,

[9700 9701 9702 ..., 9797 9798 9799]

[9800 9801 9802 ..., 9897 9898 9899]

[9900 9901 9902 ..., 9997 9998 9999]]

새로운 array를 만들고 산술연산을 할 수 있다.

>>> a = np.array( [20,30,40,50] )

>>> print(a)

[20 30 40 50]

>>> b = np.arange( 4 )

>>> b

array([0, 1, 2, 3])

>>> c = a-b

>>> print(c)

[20, 29, 38, 47]

>>> b\*\*2

array([0, 1, 4, 9])

>>> print(10\*np.sin(a)) # sine 함수다. Cosine, tangent 등도 연습해보자.

[ 9.12945251 -9.88031624 7.4511316 -2.62374854]

>>> a<35

array([ True, True, False, False])

행렬연산을 할 수 있다. \*은 원소끼리 곱하는 연산이다. 행렬곱은 @ 또는 dot 함수를 사용하자.

>>> A = np.array( [[1,1], [0,1]] )

>>> B = np.array( [[2,0], [3,4]] )

>>> A \* B # elementwise product

array([[2, 0],

[0, 4]])

>>> A @ B # matrix product

array([[5, 4],

[3, 4]])

>>> A.dot(B) # another matrix product

array([[5, 4],

[3, 4]])

>>> a = np.ones((2,3), dtype=int) # 정수 1로 채워진 2\*3 array다.

0과 1사이의 균등분포(uniform distribution)에서 추출된 무작위 숫자로 이루어진 2\*3 array는 다음과 같이 만들 수 있다.

>>> b = np.random.random((2,3))

>>> a \*= 3 # a = a\*3 을 간단히 표현했다.

>>> a

array([[3, 3, 3],

[3, 3, 3]])

>>> b += a # b = b+a를 간단히 표현했다.

>>> b

array([[ 3.417022 , 3.72032449, 3.00011437],

[ 3.30233257, 3.14675589, 3.09233859]])

>>> a += b #a는 정수로 정의되고, b는 실수를 사용하여 에러가 발생한다.

Traceback (most recent call last):

...

TypeError: Cannot cast ufunc add output from dtype('float64') to dtype('int64') with casting rule 'same\_kind'

복소수는 j를 사용한다. 예를 들어 실수부가 2이고 복소수부가 3인 복소수는 2.0+3.0j라고 표시한다.

>>> d = np.exp(2.0+3.0j)

>>> d

(-7.315110094901103+1.0427436562359045j)

>>> d.dtype.name

'complex128'

Array의 합계, 평균 등도 간단히 계산할 수 있다.

>>> a = np.random.random((2,3))

>>> a

array([[ 0.18626021, 0.34556073, 0.39676747],

[ 0.53881673, 0.41919451, 0.6852195 ]])

>>> a.sum()

2.5718191614547998

>>> a.min()

0.1862602113776709

>>> a.max()

0.6852195003967595

a.mean(), a.var(), a.std()는 짐작데로 평균, 분산, 표준편자(standard deviation)을 계산한다.

행과 열단위로도 계산할 수 있다.

>>> b = np.arange(12).reshape(3,4) # 0-12까지 정수를 3\*4행렬로 출력

>>> b

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

>>> b.sum(axis=0) # 열에 대한 합

array([12, 15, 18, 21])

>>> b.min(axis=1) # 행에 대한 합

array([0, 4, 8])

>>> b.cumsum(axis=1) # 행을 따라서 누적합

array([[ 0, 1, 3, 6],

[ 4, 9, 15, 22],

[ 8, 17, 27, 38]])

다음과 같은 원소별 연산도 가능하다.

>>> B = np.arange(3)

>>> B

array([0, 1, 2])

>>> np.exp(B)

array([ 1. , 2.71828183, 7.3890561 ])

>>> np.sqrt(B)

array([ 0. , 1. , 1.41421356])

>>> C = np.array([2., -1., 4.])

>>> np.add(B, C) # B+C와 같다.

array([ 2., 0., 6.])

1차원 array의 숫자를 추출하고 분할(slicing)하는 등의 작업을 해보자.

>>> a = np.arange(10)\*\*3 # \*\*는 제곱을 의미한다. \*\*3이므로 3제곱이다.

>>> a

array([ 0, 1, 8, 27, 64, 125, 216, 343, 512, 729])

>>> a[2]

8

>>> a[2:5]

array([ 8, 27, 64])

0번째부터 6번째 전 자리까지 2칸씩 뛰면서 -1000을 넣는다.

>>> a[:6:2] = -1000 # equivalent to a[0:6:2] = -1000; from start to position 6, exclusive, set every 2nd element to -1000

>>> a

array([-1000, 1, -1000, 27, -1000, 125, 216, 343, 512, 729])

Array를 뒤에서부터 보여준다.

>>> a[ : :-1] # reversed a

array([ 729, 512, 343, 216, 125, -1000, 27, -1000, 1, -1000])

>>> a = np.floor(10\*np.random.random((2,2))) # floor는 내림 함수다. (예) np.floor(3.7) = 3.0

>>> a

array([[ 8., 8.],

[ 0., 0.]])

>>> b = np.floor(10\*np.random.random((2,2)))

>>> b

array([[ 1., 8.],

[ 0., 4.]])

행으로 두 array를 쌓는다(stack).

>>> np.vstack((a,b))

array([[ 8., 8.],

[ 0., 0.],

[ 1., 8.],

[ 0., 4.]])

열 방향으로 두 array를 붙인다.

>>> np.hstack((a,b))

array([[ 8., 8., 1., 8.],

[ 0., 0., 0., 4.]])

Numpy를 이용해서 선형대수(Linear Algebra) 즉 행렬 연산을 할 수 있다.

>>> import numpy as np

>>> a = np.array([[1.0, 2.0], [3.0, 4.0]])

>>> print(a)

[[ 1. 2.]

[ 3. 4.]]

>>> a.transpose() # 전이행렬

array([[ 1., 3.],

[ 2., 4.]])

>>> np.linalg.inv(a) # 역행렬

array([[-2. , 1. ],

[ 1.5, -0.5]])

>>> u = np.eye(2) # 2x2 단위행렬(identify matrix). "eye" 는 "I"를 의미.

>>> u

array([[ 1., 0.],

[ 0., 1.]])

>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])

>>> j @ j # 행렬곱

array([[-1., 0.],

[ 0., -1.]])

>>> np.trace(u) # 트레이스(trace) 구하기

2.0

일차연립방정식도 풀 수 있다.

>>> y = np.array([[5.], [7.]])

>>> np.linalg.solve(a, y) # a\*x = y 에서 x를 구한다.

array([[-3.],

[ 4.]])

>>> np.linalg.eig(j) # 고유벡터(eigenvector) 구하기

(array([ 0.+1.j, 0.-1.j]), array([[ 0.70710678+0.j , 0.70710678-0.j ],

[ 0.00000000-0.70710678j, 0.00000000+0.70710678j]]))

히스토그램(histogram)을 그릴 수도 있다.

>>> import numpy as np

>>> import matplotlib.pyplot as plt # 그래프를 그리기 위한 라이브러리 부르기

평균이 2, 표준편차가 0.5인 정규분포의 난수(random number)를 10,000개 만들어서 히스토그램을 그려보자.

>>> mu, sigma = 2, 0.5 # mu = 2, sigma = 0.5 와 같은 표현

>>> v = np.random.normal(mu,sigma,10000) # 평균이 mu이고 표준편차가 sigma인 난수 10,000개 생성

10,000개의 데이터를 50개의 집단(bin)으로 묶어서 히스토그램을 만들어보자. density = 1 옵션을 주면 히스토그램의 면적이 1이 되어서 확률분포함수(probability density function)처럼 히스토그램을 만들어 준다.

>>> plt.hist(v, bins=50, density=1) # matplotlib version (plot)

>>> plt.show() # 히스토그램을 보여준다.

