Numpy and Matplotlib

[Background Knowledge for Data Analysis]

algebra

- relational algebra(SQL)
- linear algebra (Numpy)

Visualization

Crawling

Transformation

Basic Deep learning

Numpy

- Linear algebra library
- Fundamental package for working with N-dimensional array objects (vector, matrix, tensor, …)
- Numpy arrays are a fundamental data type for some other packages to use
- Numpy has many specialized modules and functions:

numpy.linalg (Linear algebra)	**numpy.random (Random sampling)**
numpy.fft (Discrete Fourier transform)	**sorting/searching/counting**
math functions	**numpy.testing (unit test support)**

Practice Together!

Jun 2019 Data Programming - Numpy & Matplotlib

Practice: Numpy Library

Basic numpy methods covered in class. All required codes are already written so that you can practice easily.
You might want these as a reference of solving later image classification problem.

---

1. Import Numpy Library

```
import numpy as np
import numpy.random as npr
```

```
# Practice here!
import numpy as np
import numpy.random as npr
```

Numpy array

- Simple array creation

```
import numpy as np

a = np.array([1,2,3,4])
b = np.array([2,3,4,5])
print(a)
print(b)
```

```
[1 2 3 4]
[2 3 4 5]
```

- Each Numpy array has some attributes:
  - shape(a tuple of the size in each dimension), dtype(data type of entries), size(total # of entries), ndim(# of dimensions), T(transpose)

```
print("shape = ", a.shape, ", dtype = ", a.dtype, ", size = ", a.size, ", ndim = ", a.ndim)
```

```
shape =  (4,) , dtype =  int64 , size =  4 , ndim =  1
```

Vectors

초기화할 때 넘겨주는 것은 원소 개수가 아니라 shape을 넘겨주는 것!

- Vectors are 1d arrays (or 1st order tensors)

```
import numpy as np
import numpy.random as npr
```

```
np.zeros(4) # Return a new array of given shape and type, filled with zeros.
```

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

```
np.ones(5) # Return a new array of given shape and type, filled with ones.
```

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

```
npr.randn(3) # Return samples from the “standard normal” distribution.
```

정규분포에서 샘플링

```
array([ 1.03548977, -0.10369842, -1.6403447 ])
```

```
np.linspace(0, 2, 5)  # 5 uniform values in [0, 2]
```

```
array([ 0. ,  0.5,  1. ,  1.5,  2. ])
```

```
np.arange(8) # create array from 0 to 7
```

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

Matrices

- Matrices are 2d arrays (or 2^{nd} order tensors)

```
np.zeros((2,5))
```

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

```
npr.randn(3, 3)
```

```
array([[ 0.84257344,  0.17978292, -0.62112465],
       [ 1.16650643,  0.87555025,  0.05225127],
       [ 1.19749645, -1.38333744,  0.2157709 ]])
```

Array shape

- Shape returns a tuple listing the length of the array along each dimension

```
a = np.array([1,2,3,4])
b = npr.randn(3,3)
c = npr.randn(3,2,4)
print("a = ", a, "\nb = \n", b, "\nc = \n", c)
print("a.shape = ", a.shape, "b.shape = ", b.shape, "c.shape = ", c.shape)
```

```
a =  [1 2 3 4]
b =
 [[-0.6129468  -1.53607928  1.10003604]
 [-0.29755292  0.54630051 -1.8317307 ]
 [ 0.02114839  0.02257444 -0.22226038]]
c =
 [[[-1.5262782  -0.26953168  1.10735491  0.14370897]
  [-0.07652912 -0.85846648  0.89918118 -0.44788444]]

 [[-1.29420815  0.25608476 -0.39793983  0.63340769]
  [-0.32379019  1.46029366 -0.07129954 -0.34766979]]

 [[-0.34066753 -0.56823573  1.02060155 -0.15245486]
  [-2.26962498 -1.45004611 -1.13198899  0.58360711]]]
a.shape =  (4,) b.shape =  (3, 3) c.shape =  (3, 2, 4)
```

Reshaping an array

- Reshape return a new array with a different shape, but it cannot change the number of elements in an array

```
A = np.arange(8)
print(A)
```

```
[0 1 2 3 4 5 6 7]
```

```
A.reshape(2,4)
```

```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

```
A.reshape(3,3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-57-63445bd75ed1> in <module>()
----> 1 A.reshape(3,3)

ValueError: cannot reshape array of size 8 into shape (3,3)
```

Array indexing

- Array Slicing

  >>> A[0, 3:5]
  array([3, 4])
  >>> A[4:, 4:]
  array([[28, 29], [34, 35]])
  >>> A[:, 2]
  array([2, 8, 14, 20, 26, 32])

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

- Slices are references to memory in the original array

슬라이싱은 참조이기 때문에 값이 바뀌면 바뀐 값을 그대로 가져옴.

```
a = np.array((0,1,2,3,4))
b = a[2:4]
print(b)
```

```
[2 3]
```

```
b[0] = 10
print(a)
```

Changing values in a slice also changes the original array!

```
[ 0  1 10  3  4]
```

Array indexing

- Indexing by position

```
a = np.arange(0, 80, 10)
print("a = ", a)
indices = [1, 2, -3]
y = a[indices]
print("y = ", y)
```

```
a =  [ 0 10 20 30 40 50 60 70]
y =  [10 20 50]
```

- Indexing with Booleans

```
mask = np.array([0, 1, 1, 0, 0, 1, 0, 0], dtype=bool)
y = a[mask]
print(y)
y = a[a > 20]
print(y)
```

broadcasting으로 자동으로 type casting을 해주기 때문에 가능.

```
[10 20 50]
[30 40 50 60 70]
```

Indexing with newaxis

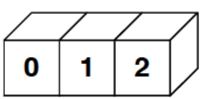
축을 추가해주는 기능.

- newaxis is a special index that inserts a new axis in the array at the specified location
- Each newaxis increases the array's dimensionality by 1
- Each newaxis expands the dimensions by adding one unit-length dimension

Use np.newaxis or write "from numpy import newaxis"

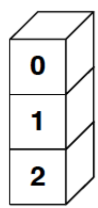
1 X 3

```
>>> shape(a)
(3,)
>>> y = a[newaxis,:]
>>> shape(y)
(1, 3)
```



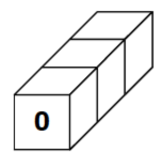
3 X 1

```
>>> y = a[:,newaxis]
>>> shape(y)
(3, 1)
```



1 X 1 X 3

```
> y = a[newaxis, newaxis, :]
> shape(y)
(1, 1, 3)
```



Datatype

- Every numpy array is a grid of elements of the same type
- Numpy tries to guess a datatype when you create an array, but you can also explicitly specify the datatype

```
a = np.array([1,2])
print(a.dtype)
```

```
int64
```

```
a = np.array([1.0, 2.0])
print(a.dtype)
```

```
float64
```

```
a = np.array([1,2], dtype=np.int64)
print(a.dtype)
```

```
int64
```

Basic Type	Available NumPy types	Code	Comments
Boolean	`bool`	`b`	Elements are 1 byte in size.
Integer	`int8, int16, int32, int64, int128, int`	`i`	int defaults to the size of long in C for the platform.
Unsigned Integer	`uint8, uint16, uint32, uint64, uint128, uint`	`u`	uint defaults to the size of unsigned long in C for the platform.
Float	`float16, float32, float64, float,longfloat`	`f`	float is always a double precision floating point value (64 bits). longfloat represents large precision floats. Its size is platform dependent.
Complex	`complex64, complex128, complex, longcomplex`	`c`	The real and imaginary elements of a complex64 are each represented by a single precision (32 bit) value for a total size of 64 bits.
Strings	`str, unicode`	`S` or `a`, `U`	For example, dtype='S4' would be used for an array of 4-character strings.
DateTime	`datetime64, timedelta64`	See section	Allow operations between dates and/or times. New in 1.7.
Object	`object`	`O`	Represent items in array as Python objects.
Records	`void`	`V`	Used for arbitrary data structures.

Mathematical operations

- Basic mathematical functions operate element-wise on arrays

```
a = np.array([[1,2],[3,4]], dtype=np.float64)
b = np.array([[5,6],[7,8]], dtype=np.float64)
print("a = \n", a, "\nb = \n", b)
```

```
a =
 [[ 1.  2.]
 [ 3.  4.]]
b =
 [[ 5.  6.]
 [ 7.  8.]]
```

```
a + b
```

```
array([[  6.,   8.],
       [ 10.,  12.]])
```

```
a * b
```

```
array([[  5.,  12.],
       [ 21.,  32.]])
```

```
a - b
```

```
array([[-4., -4.],
       [-4., -4.]])
```

```
a / b
```

```
array([[ 0.2       ,  0.33333333],
       [ 0.42857143,  0.5       ]])
```

```
np.sqrt(a)
```

```
array([[ 1.        ,  1.41421356],
       [ 1.73205081,  2.        ]])
```

```
a ** b
```

```
array([[  1.00000000e+00,   6.40000000e+01],
       [  2.18700000e+03,   6.55360000e+04]])
```

sum()

- sum(a, axis=j) defaults to adding up all the values in an array along the jth axis
  - if a is of $d_0 \times d_1 \times \cdots \times d_i$, the dimension of sum(a, axis=j) is $d_0 \times \cdots \times d_{j-1} \times d_{j+1} \times \cdots \times d_i$

```
a = np.array([[1,2,3],
[4,5,6]])
```

```
np.sum(a)
```

```
21
```

```
np.sum(a, axis = 0)
```

```
array([5, 7, 9])
```

```
np.sum(a, axis = 1)
```

```
array([ 6, 15])
```

prod()

- prod(a, axis=j) defaults to multiply all the values in an array along the jth axis
  - if a is of $d_0 \times d_1 \times \cdots \times d_i$, the dimension of prod(a, axis=j) is $d_0 \times \cdots \times d_{j-1} \times d_{j+1} \times \cdots \times d_i$

```
a = np.array([[1,2,3],[4,5,6]])
a.prod(axis = 0)
```

```
array([ 4, 10, 18])
```

```
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
a.prod(axis = 1)
```

```
array([  6, 120, 504])
```

Dot product

매트릭스 단위로 쓰려면 dot 으로!

- dot() implements the dot product on vectors

```
np.array([1,2,3]).dot(np.array([4,5,6]))
```

```
32
```

1*4 + 2*5 + 3*6

- For 2D arrays, it is the matrix product

```
a = np.array([[1,1], [3,2]])
b = np.array([[4,1], [2,2]])
print(a.dot(b))
```

```
[[ 6  3]
 [16  7]]
```

$$\begin{bmatrix} 1 & 1 \\ 3 & 2 \end{bmatrix} * \begin{bmatrix} 4 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 1*4+2*1 & 1*1+1*2 \\ 4*3+2*2 & 1*3+2*2 \end{bmatrix}$$

Min/Max

- min() and max() return minimum value and maximum value, respectively
- argmin() and argmax() return index of minimum value and index of maximum value, respectively

```
a = np.array([2.,3.,0.,1.])
```

```
a.min(axis = 0)
```

```
0.0
```

```
a.argmin(axis = 0)
```

```
2
```

```
a.max(axis = 0)
```

```
3.0
```

```
a.argmax(axis = 0)
```

```
1
```

Mean/Std/Var

- mean(), std(), and var() return mean value, standard deviation, and variance along the specified axis

```
a = np.array([[1,2,3],
[4,5,6]])
```

```
a.mean(axis = 0)
```

```
array([ 2.5,  3.5,  4.5])
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

axis=0

```
a.std(axis = 0)
```

```
array([ 1.5,  1.5,  1.5])
```

```
a.var(axis = 0)
```

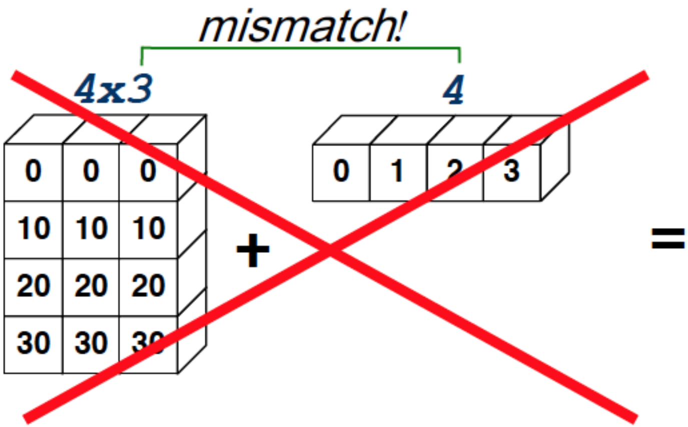
```
array([ 2.25,  2.25,  2.25])
```

Broadcasting

- NumPy arrays of different dimensionality can be combined in the same expression
- Arrays with *smaller* dimension are *broadcasted to* match the *larger* arrays, without copying data, so that they have equal size

Broadcasting Rules

- NumPy *compares* their shapes element-wise in a reverse order
- Two dimensions to *compare* are compatible when they are equal, or one of them is 1
  - ValueError: shape mismatch: objects cannot be broadcast to a single shape" exception

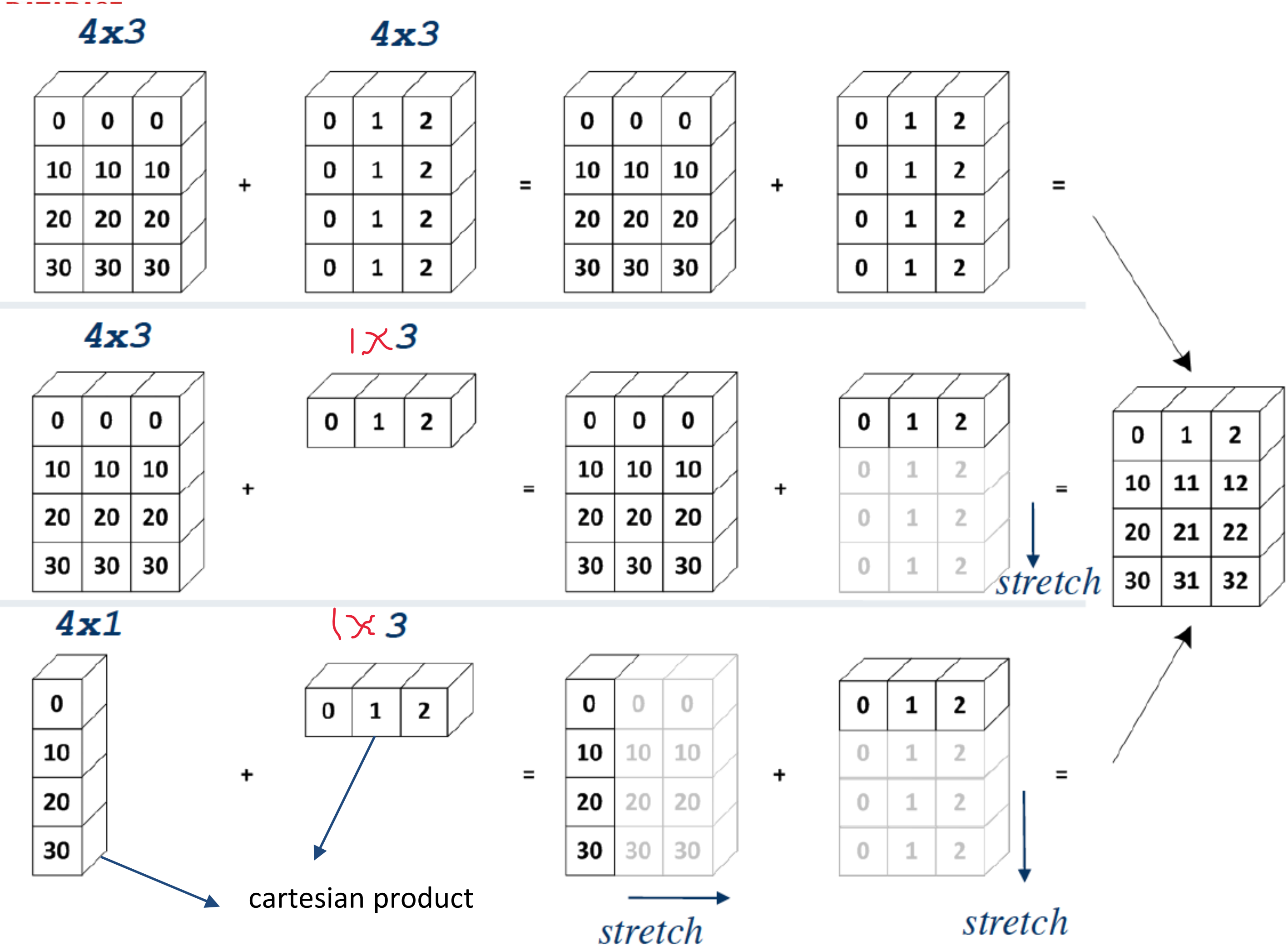


Simulating a Cartesian product using broadcasting rather than nested for-loops

```
For each a in A
    For each b in B
      y[a.pos][b.pos] = a op b
```

```
Y = A op B
```

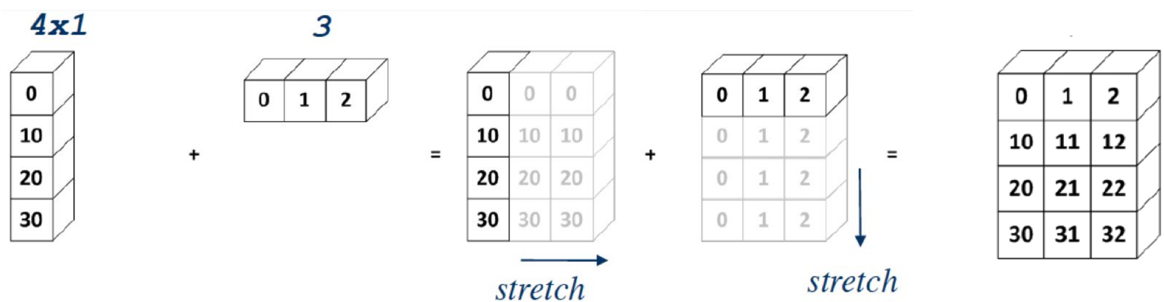
(This is much faster!)


4x3
4x3
4x3
1x3
4x1
1x3
stretch
cartesian product
stretch
stretch

Broadcasting example

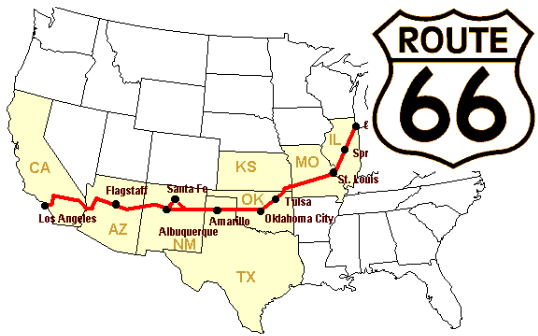
```
a = np.array((0,10,20,30))
b = np.array((0,1,2))
y = a[:, np.newaxis] + b
print(a[:, np.newaxis], "+", b, "=\n\n", y)
```

```
[[ 0]                  [[ 0  1  2]
 [10]                   [10 11 12]
 [20]                   [20 21 22]
 [30]] + [0 1 2] =      [30 31 32]]
```



Application: Distances between cities of Route 66

- Given a 1-D array of distances to all cities in Route 66 from Chicago



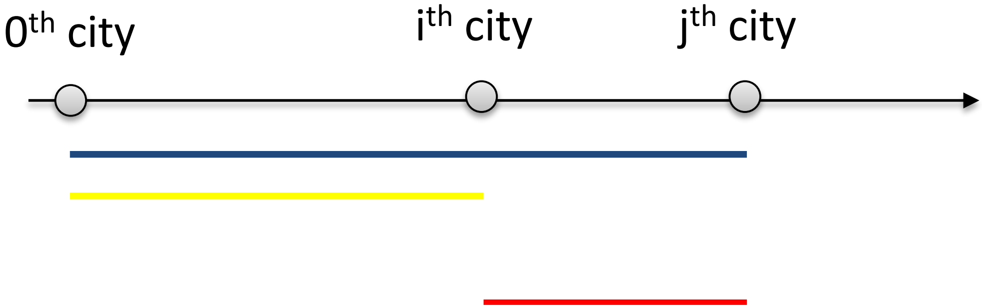
[0, 198, 303, 736, 871, 1175, 1475, 1544, 1913, 2448]

Chicago

Los Angeles

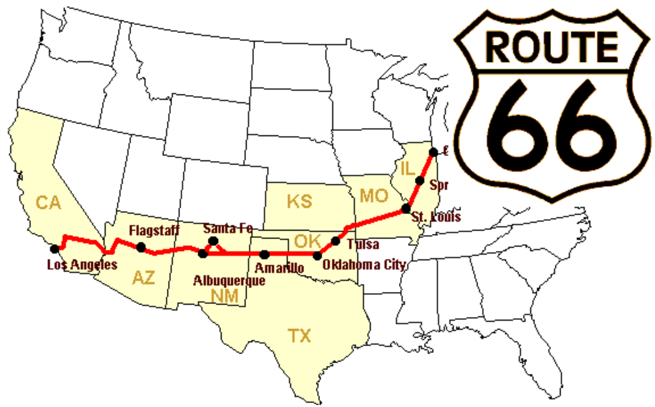
- Construct a 2D array of distances between pairs of cities

$$dist'(i, j) = |dist(0, j) - dist(0, i)|$$



```python
mileposts = np.array([0, 198, 303, 736, 871, 1175, 1475, 1544, 1913, 2448])
distance_array = np.abs(mileposts - mileposts[:, np.newaxis])
print(distance_array)
```

```
[[   0  198  303  736  871 1175 1475 1544 1913 2448]
 [ 198    0  105  538  673  977 1277 1346 1715 2250]
 [ 303  105    0  433  568  872 1172 1241 1610 2145]
 [ 736  538  433    0  135  439  739  808 1177 1712]
 [ 871  673  568  135    0  304  604  673 1042 1577]
 [1175  977  872  439  304    0  300  369  738 1273]
 [1475 1277 1172  739  604  300    0   69  438  973]
 [1544 1346 1241  808  673  369   69    0  369  904]
 [1913 1715 1610 1177 1042  738  438  369    0  535]
 [2448 2250 2145 1712 1577 1273  973  904  535    0]]
```



Grid-based or network-based problems can also use broadcasting

Matplotlib

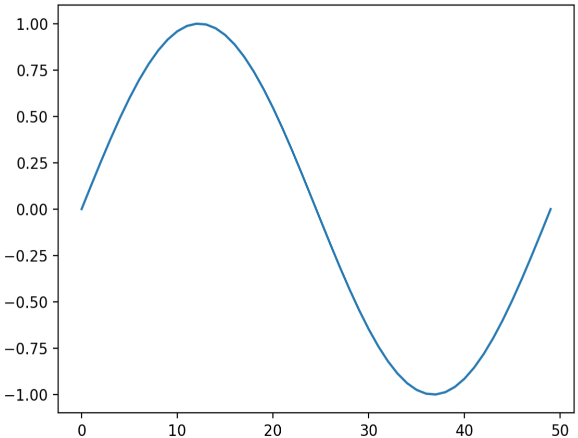
- 2D Python plotting library (matplotlib.pyplot mostly used)
- matplotlib.pyplot can do many types of visualizations including:
  - Line plots (using plot)
  - Scatter plots (using scatter)
  - Histograms, bar charts (using hist)
  - Error bars on plots, box plots (using boxplot, errorbar)
  - Images (matrix to image) (using imshow)
  - Pie charts, Polar charts (using pie, polar)
  - Contour maps (using contour or tricontour)
  - Stream plots which show derivatives at many locations (streamplot)

Line plot

- A line is created connecting each data point together

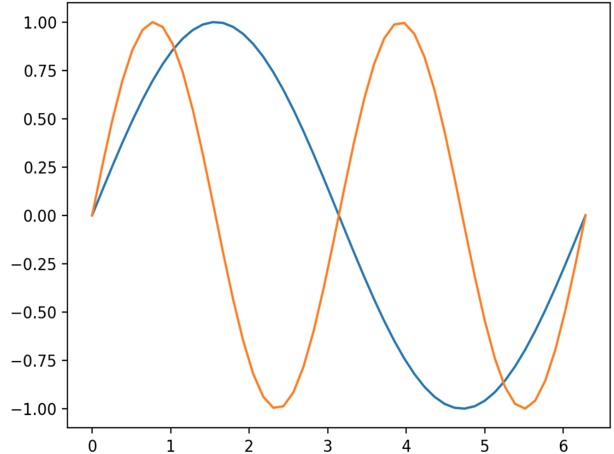
- Plot against indices

```
%matplotlib notebook
import matplotlib.pyplot as plt
from numpy import *
x = linspace(0, 2*pi, 50)
plt.plot(sin(x))
```



- Multiple datasets

```
plt.plot(x, sin(x), x, sin(2*x))
```

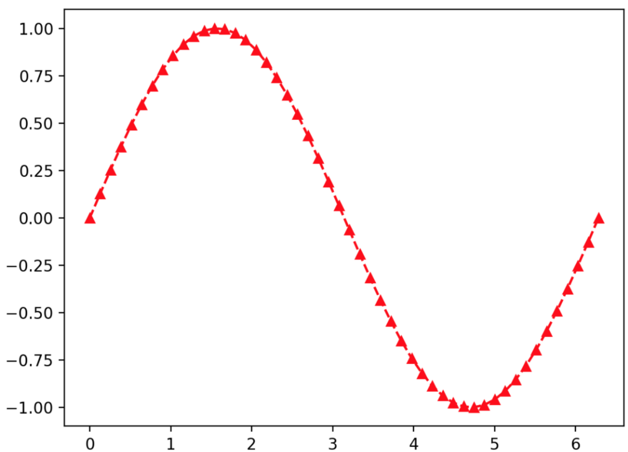


Line formatting

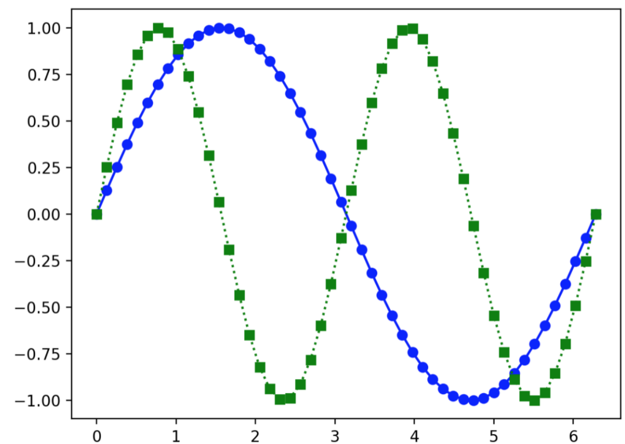
option을 자세하게 주어야 함!!!

```
plt.plot(x, sin(x), 'b-o') # blue, solid line, circle points
plt.plot(x, sin(x), 'r--^') # red, dashed line, triangle_up points
plt.plot(x, sin(x), 'g:s') # green, dotted line, square points
```

```
plt.plot(x, sin(x), 'r--^')
```



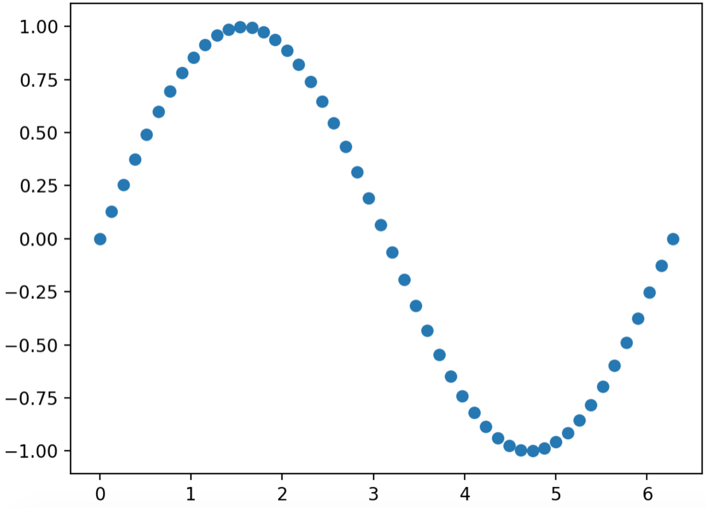

```
plt.plot(x, sin(x), 'b-o',
         x, sin(2*x), 'g:s')
```



Scatter plot

- display data as a collection of points

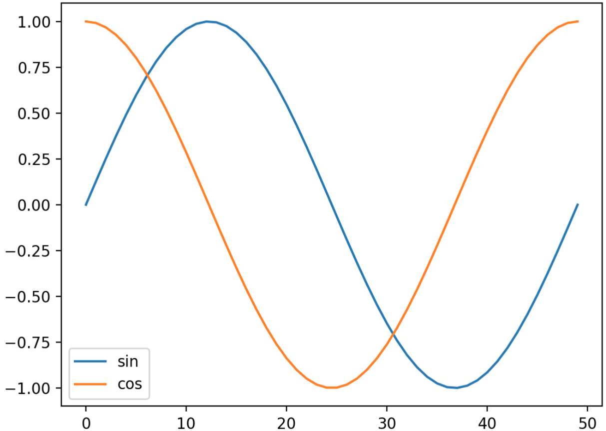
```
x = linspace(0, 2*pi, 50)
y = sin(x)
plt.scatter(x, y)
```



Legend

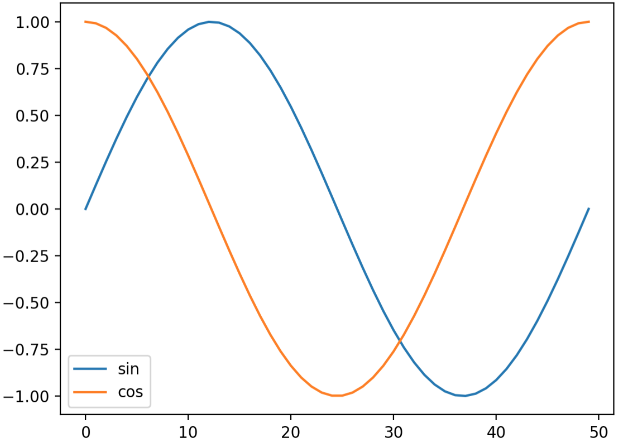
- Add labels in plot command

```
plt.plot(sin(x), label='sin')
plt.plot(cos(x), label='cos')
plt.legend()
```



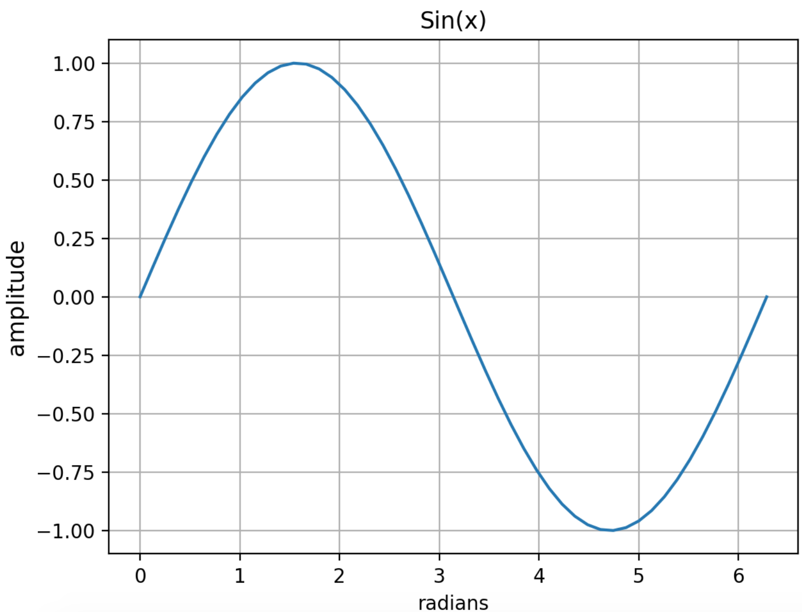
- Or as a list in legend()

```
plt.plot(sin(x))
plt.plot(cos(x))
plt.legend(['sin', 'cos'])
```



Titles and Grid

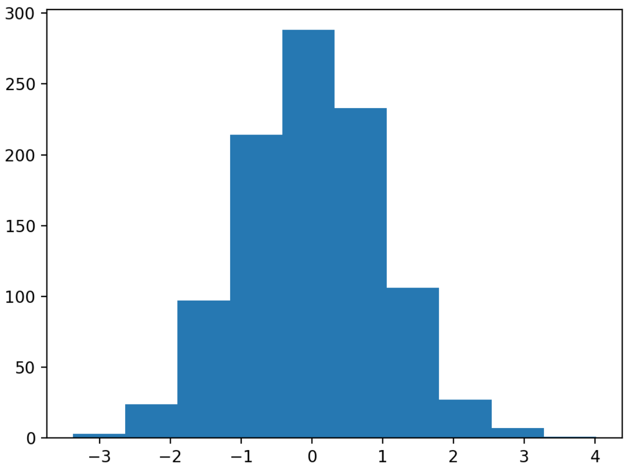
```
plt.plot(x, sin(x))
plt.xlabel('radians')
plt.ylabel('amplitude', fontsize='large')
plt.title('Sin(x)')
plt.grid()
```



Histograms

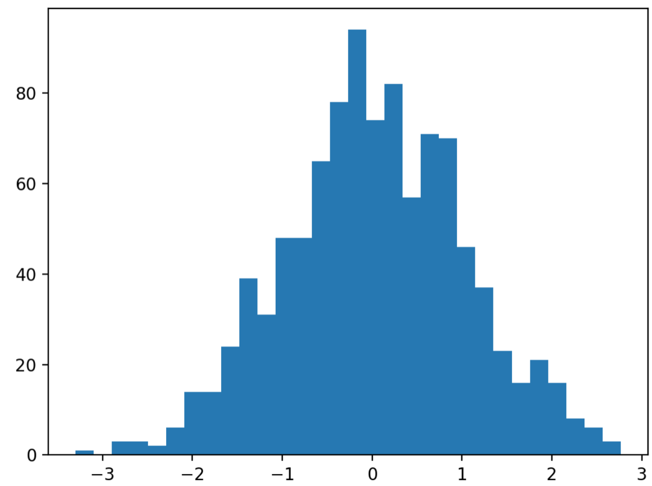
- Plot histogram, defaults to 10 bins

```
plt.hist(npr.randn(1000))
```



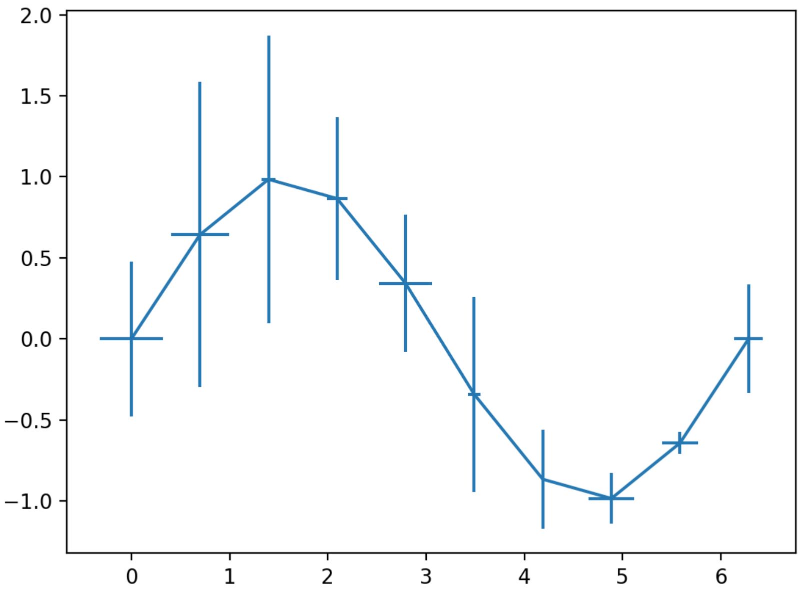
- Change the number of bins

```
plt.hist(npr.randn(1000), 30)
```



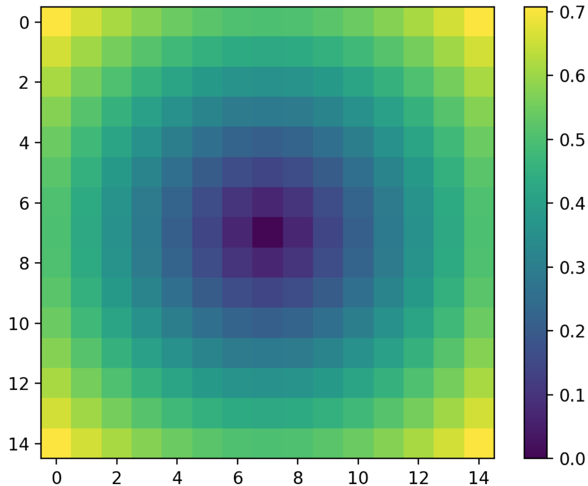
Plot with error bars

```
x = linspace(0,2*pi,10)
y = sin(x)
yerr = npr.rand(10)
xerr = npr.rand(10)/3
plt.errorbar(x, y, yerr, xerr)
```



Color Bar

```
a = linspace(0, 1, 15) - 0.5 # a.shape = (1, 15)
b = a[:, newaxis] # b.shape = (15, 1)
dist2 = a**2 + b**2 # broadcasting sum
dist = sqrt(dist2)
plt.imshow(dist); plt.colorbar()
```



Try plotting
your distance matrix