Data Processing and Analysis in Python
Lecture 16
Linear Algebra

UNIVERSITY OF
MARYLAND

ROBERT H. SMITH
SCHOOL OF BUSINESS

DR. ADAM LEE

NumPy Matrix Versus Array

NumPy Matrix (matlib)	NumPy Array (ndarray)
Strictly 2-dimensional	N-dimensional
Subclass of ndarray, so they inherit all the attributes and methods of ndarray	Attributes and methods
intended to facilitate linear algebra computations specifically	Intended to be general purpose for many kinds of numerical computing
The main advantage is to provide a convenient notation for matrix manipulations: • e.g. If A and B are matrices, then (A * B) is their matrix product	

NumPy Matrix

```
>>> import numpy as np
>>> dir(np.matrix)
>>> help(np.matrix)
```

```
>>> import numpy.matlib
```

https://numpy.org/doc/stable/reference/routines.matlib.html

```
>>> dir(numpy.matlib)
>>> help(numpy.matlib)
```

NumPy Matrix

- Functions:
  - **empty(shape, dtype=None)** fill with random data
  - **eye(shape, k=0, dtype=None)** fill with 1's on diagonal and 0's elsewhere
  - **identity(shape, dtype=None)** fill with 1's on diagonal and 0's elsewhere
  - **ones(shape, dtype=None)** fill with 1's
  - **zeros(shape, dtype=None)** fill with 0's
- Attributes:
  - **A** − return self as an ndarray object
  - **H** − return the (complex) conjugate transpose of self
  - **I** − return the (multiplicative) inverse of invertible self
  - **T** − return the transpose of the matrix
  - **shape** − tuple of array dimensions
  - **size** − number of elements in the array

NumPy Matrix

```
>>> np.matlib.empty((2, 2))
matrix([[  6.76425276e-320,   9.79033856e-307],
        [  7.39337286e-309,   3.22135945e-309]])
>>> np.matlib.zeros((2, 3))
matrix([[0.,  0.,  0.],
        [0.,  0.,  0.]])
>>> np.matlib.ones((2, 3))
matrix([[1.,  1.,  1.],
        [1.,  1.,  1.]])
>>> np.matlib.identity(3, dtype=int)
matrix([[1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]])
>>> np.matlib.eye(3, k=1, dtype=float)
matrix([[0.,  1.,  0.],
        [0.,  0.,  1.],
        [0.,  0.,  0.]])
```

NumPy Matrix

```
>>> A = np.matrix("1.0 2.0; 1.0 -1.0")
>>> A
matrix([[ 1.,  2.],
        [ 1., -1.]])
>>> type(A)
<class 'numpy.matrix'>
>>> print(A.H) # transpose
[[ 1.  1.],
 [ 2. -1.]]
>>> print(A.I) # inverse
[[ 0.33333333  0.66666667]
 [ 0.33333333 -0.33333333]]
>>> Y = np.matrix("7.0; 1.0")
>>> print(A.I * Y) # multiplication
[[3.]
 [2.]])
>>> numpy.linalg.solve(A, Y) # solve linear equations
matrix([[3.],
        [2.]])
```

$AX - Y = 0$

$AX = Y$

$A^{-1}AX = A^{-1}Y$

$X = A^{-1}Y$

$x_1 + 2x_2 = 7$

$x_1 - x_2 = 1$

$x_1 = 3$

$x_2 = 2$

NumPy Linear Algebra

- All linear algebra routines expect an object that can be converted into a 2-dimensional array
- The output is also a two-dimensional array
  - **dot(a, b[, out])** dot product of two arrays
  - **trace(a[, offset, axis1, axis2, dtype, out])** returns the sum along diagonals of the array
  - **inv(a)** computes the inverse of a matrix
  - **eig(a)** eigenvalues and right eigenvectors of a square array
  - **solve(a, b)** solves a linear matrix equation, or system of linear scalar equations

NumPy LinAlg

```
>>> from numpy import *
>>> from numpy.linalg import *
>>> A = array([[1.0,2.0],[3.0,4.0]])
>>> print(A)
[[1. 2.]
 [3. 4.]]
>>> A.transpose()
array([[ 1., 3.],
       [ 2., 4.]])
>>> inv(A)
array([[-2. , 1. ],
       [ 1.5, -0.5]])
```

NumPy LinAlg

```
>>> U = eye(2) # unit 2x2 matrix; "eye" ~ "I"
>>> U
array([[ 1., 0.],
       [ 0., 1.]])
>>> trace(U)
2.0
>>> A = array([[0.0,-1.0],[1.0,0.0]])
>>> dot(A, A) # matrix product
array([[-1.,  0.],
       [ 0., -1.]])
>>> eig(A) # eigenvalue & eigenvectors
(array([0.+1.j, 0.-1.j]),
 array([[0.70710+0.j      , 0.70710+0.j      ],
        [0.00000-0.70710j, 0.00000+0.70710j]]))
```

Supply and Demand

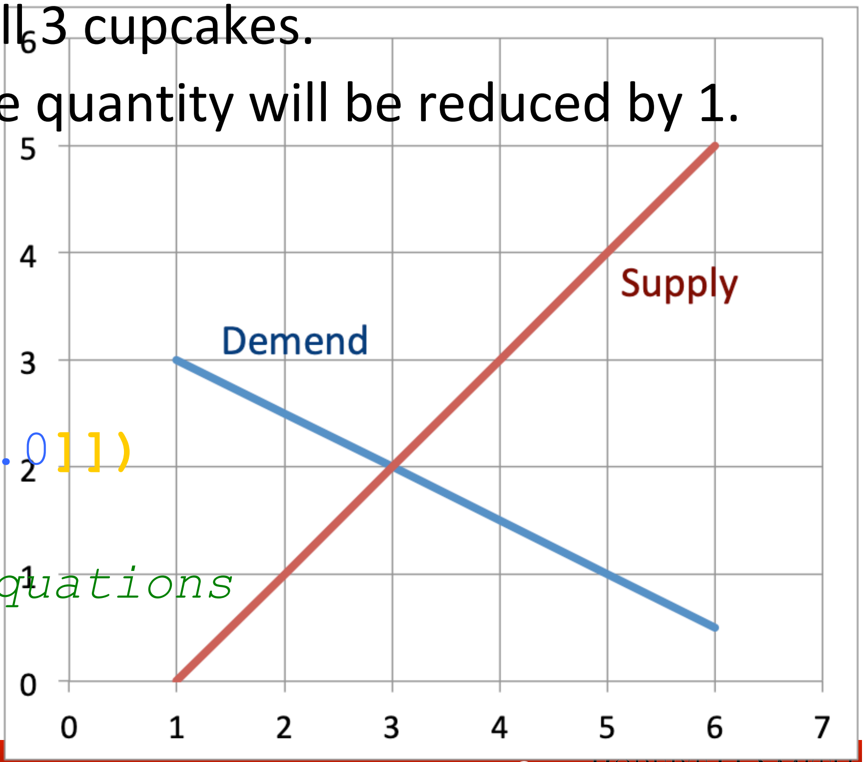
- Supply:
  - If we are willing to spend $1 on a cupcake, nobody sell any to us.
  - Every $1 increment on unit price, we can buy 1 more cup cake.
- Demand:
  - If we resell a cupcake for $1, we can sell 3 cupcakes.
  - Every $2 increment on sale price, resale quantity will be reduced by 1.

p: price $\quad p - q = 1$

q: quantity $\quad p + 2q = 7$

$$A X = Y$$

```
>>> A = array([[1.0,-1.0],[1.0,2.0]])
>>> Y = array([[1.0],[7.0]])
>>> solve(A, Y) # solve linear equations
array([[ 3.],
       [ 2.]])
```



Linear Algebra - Supply and Demand

```
from numpy import array
from numpy.linalg import solve
# generate model: p - q = 1, p + 2q = 7
A = array([[1.0,-1.0],[1.0,2.0]])
Y = array([[1.0],     [7.0]])

# solve linear equations
X = solve(A, Y)
print("Solution: price = %d & quantity = %d"%(X[0],X[1]))

# plot the model and solution
import matplotlib.pyplot as plt
plt.plot([1,6],[0,  5],  'r',label="Supply")
plt.plot([0,6],[3.5,0.5],'b',label="Demend")
plt.legend()
plt.show()
```

Product Mix

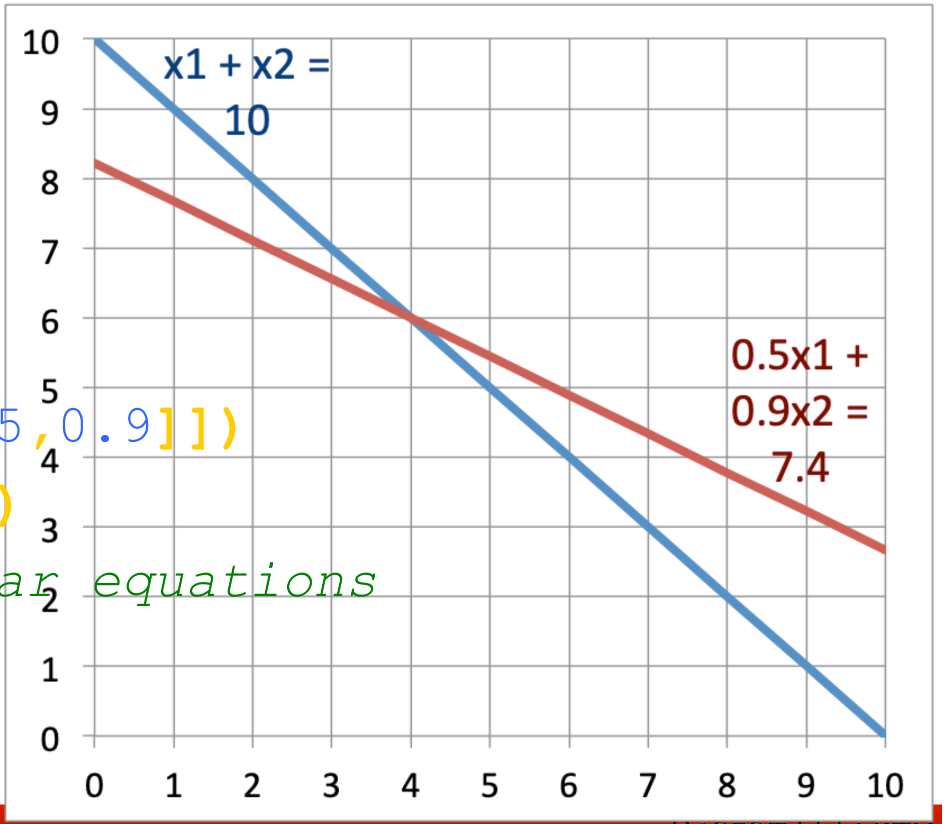
- There are two alcohol solutions: 50% & 90%
- How many gallons of each solution to be mixed to get 10 gallons of 74% alcohol solution?

$x_1 + x_2 = 10$

$0.5x_1 + 0.9x_2 = 0.74 * 10 = 7.4$

A X = Y

```
>>> A = array([[1.0,1.0],[0.5,0.9]])
>>> Y = array([[10.0],[7.4]])
>>> solve(A, Y) # solve linear equations
array([[ 4.],
       [ 6.]])
```



ROBERT H. SMITH SCHOOL OF BUSINESS

Tailwind and Headwind

- A drone flies with the wind could cover 60 miles in 2 hours.
- The return trip against the same wind took 2.5 hours.
- How fast was the drone? d
- What was the air speed? w

Trip	Rate	Time	Distance
Tailwind	$d + w$	2	60
Headwind	$d - w$	2.5	60

$(d + w) * 2 = 2d + 2w = 60$

$(d - w) * 2.5 = 2.5d - 2.5w = 60$

```
>>> A = array([[2.0,2.0],[2.5,-2.5]])
>>> Y = array([[60.0],[60.0]])
>>> solve(A, Y) # solve linear equations
array([[ 27.],
       [  3.]])
```

UNIVERSITY OF MARYLAND

ROBERT H. SMITH SCHOOL OF BUSINESS