LECTURE 4

LINEAR ALGEBRA

COMPUTATION AND DATA MANIPULATION

DR. PRAPASSORN TANTIPHANWADI
INDUSTRIAL ENGINEERING, FACULTY OF ENGINEERING AT KHAMPAENGSAEN
DECEMBER 2565

CONTENT

- Matrix manipulation and Linear Algebra
- NumPy Arrays and Matrices
- Indexing and Slicing
- Panda
- Plotting and Visualsing: matplotlib

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- A python object that using arrays is **the list.**
- For example, we can create two lists as follows:
- Each type of object has a defined set of operations.

```
In [1]: a = [1, 2, 3, 4, 5]
        b = [20, 30, 40, 50, 60]

In [4]: a, b
Out[4]: ([1, 2, 3, 4, 5], [20, 30, 40, 50, 60])
```

- Add = concatenuate

```
In [5]: a+b
Out[5]: [1, 2, 3, 4, 5, 20, 30, 40, 50, 60]
```

NumPy Arrays and Matrices

```
In [6]: import numpy as np

In [7]: A = np.array([1, 2, 3, 4, 5])
        B = np.array([20, 30, 40, 50, 60])

In [8]: A, B
Out[8]: (array([1, 2, 3, 4, 5]), array([20, 30, 40, 50, 60]))

In [14]: C = A+B

In [15]: C
Out[15]: array([21, 32, 43, 54, 65])
```

- We can operate array as vector operation +, -, / , x

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

NumPy Arrays and Matrices

In [16]:
```
A = np.array(a)
B = np.array(b)
```

In [17]:
```
C = A+ B
D = A- B
E = A * B
F = A / B
```

In [25]:
```
C, D, E, F
```

Out[25]:
```
(array([21, 32, 43, 54, 65]),
 array([-19, -28, -37, -46, -55]),
 array([ 20,  60, 120, 200, 300]),
 array([0.05      , 0.06666667, 0.075     , 0.08      , 0.08333333]))
```

In [26]:
```
G
```

Out[26]:
```
700
```

In [21]:
```
H = np.cross(A,B)
```

```
------------------------------
ValueError: incompatible dimensions for cross product
(dimension must be 2 or 3)
```

In [22]:
```
import numpy as np
p = [[2, 2] , [3, 1]] #2-D
q = [[6, 7] , [5, 4]] # 2-D
```

In [23]:
```
import numpy as np
HH = np.cross(p,q)
```

In [27]:
```
HH
```

Out[27]:
```
array([2, 7])
```

In [24]:
```
GG = np.dot(p,q)
```

In [28]:
```
GG
```

Out[28]:
```
array([[22, 22],
       [23, 25]])
```

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

NumPy Arrays and Matrices

In [32]:
```
M1 = np.matrix([[2, 3], [-1, 5]])
M2 = np.matrix([[1, 2], [-10, 5.4]])
```

In [37]:
```
print(M1)
print(M2)
```

```
[[ 2  3]
 [-1  5]]
[[  1.    2. ]
 [-10.    5.4]]
```

In [38]:
```
MM = M1+M2
print(MM)
```

```
[[  3.    5. ]
 [-11.   10.4]]
```

In [40]:
```
M2tr = M2.transpose()
print(M2tr)
```

```
[[  1.  -10. ]
 [  2.    5.4]]
```

- **NumPy support matrices**
  - The SciPy package, we can use the linalg methods that will enable to do some typical linear algebra computations such as matrix inversion:

In [42]:
```
from numpy import array, dot
from scipy import linalg
```

In [44]:
```
x = array([[1, 1], [1, 2], [1, 3], [1, 4]])
y = array([[1], [2], [3], [4]])
```

In [45]:
```
print(x)
print(y)
```

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

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

NumPy Arrays and Matrices

In [50]:
```
n = linalg.inv(dot(x.T, x)) #invert a matrix with the .inv method
k = dot(x.T, y)
coef = dot(n, k)  #matrix multiplication with arrays can be done with the dot() funciton.
```

In [52]:
```
print(n)
```

```
[[ 1.5 -0.5]
 [-0.5  0.2]]
```

In [53]:
```
print(k)
```

```
[[10]
 [30]]
```

In [54]:
```
print(coef)
```

```
[[0.]
 [1.]]
```

$$n = (x^T x)^{-1}$$

$$k = x^T y$$

$$coef = nk = (x^T x)^{-1} x^T y$$

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

Indexing and Slicing

- In the case with lists, it is possible to access the contents of an N-dimensional array by indexing and / or slicing the array.
- We can do this using the usual notation **start:end:step** which will extract the appropriate elements starting at start in steps given by step and until end-1.

```
In [60]: a = np.arange(10)
         print(a[:]) ; print(a[2:6])

[0 1 2 3 4 5 6 7 8 9]
[2 3 4 5]
```

```
In [61]: print(a[1:9:3])

[1 4 7]
```

- The same notation can be used with arrays of more dimensions.

```
In [63]: b = np.array([np.arange(4), 2*np.arange(4)])
         print(b.shape)

(2, 4)
```

```
In [67]: print(b)

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

```
In [66]: print(b[:1, :])

[[0 1 2 3]]
```

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

Panda

- It allows us to manipulate indexed structured data with many variables, including work with time series, missing values and multiple datasets.
- In Pandas, a 1D array is called a series, whereas dataframes are collections of series.

```
import numpy as np

import pandas as pd
```

In [40]:
```
import numpy as np
import pandas as pd
```

In [41]:
```
array1 = np.array([14.1, 15.2, 16.3])
```

In [42]:
```
print(array1)
```

```
[14.1 15.2 16.3]
```

In [43]:
```
series1 = pd.Series(array1)
```

In [44]:
```
print(series1)
```

```
0    14.1
1    15.2
2    16.3
dtype: float64
```

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

Panda

Sample tabular data to be loaded into a Pandas dataframe.

Animal	Limbs	Herbivore
Python	0	No
Iberian Lynx	4	No
Giant Panda	4	Yes
Field Mouse	4	Yes
Octopus	8	No

- We can load this data into Python by creating lists with the appropriate information about the two features describing the animals in the table.
- We can load data into a Pandas dataframe with lists, dictionaries, arrays, tuples, etc.

```
In [45]: features = {'limbs': [0, 4, 4, 4, 8],\
                    'herbivore': ['No', 'No', 'Yes', 'Yes', 'No']}

In [46]: animals = ['Python', 'Iberian Lynx',\
                   'Giant Panda', 'Field Mouse', 'Octopus']

In [47]: df = pd.DataFrame(features, index = animals)

In [48]: print(df)

              limbs herbivore
Python            0        No
Iberian Lynx      4        No
Giant Panda       4       Yes
Field Mouse       4       Yes
Octopus           8        No
```

- limbs and herbivore as a **dictionary**
- where the keys will be the names of the columns in our Pandas dataframe, and the values correspond to the entries in the table.
- we are defining a **list** called animals that will be used as an index to identify each of the rows in the table.

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

Panda

- We can have a look at the first three entries in the dataframe df with the command head:

```
In [49]: df.head(3)
```

Out[49]:

	limbs	herbivore
Python	0	No
Iberian Lynx	4	No
Giant Panda	4	Yes

- we can refer to the column data by the name given to the column.

```
In [51]: df['limbs'][2:5]
```

```
Out[51]: Giant Panda    4
         Field Mouse    4
         Octopus        8
         Name: limbs, dtype: int64
```

- we have use slicing to select the data required. The information about a single row can be obtained by locating the correct index:

```
In [52]: df.loc['Python']
```

```
Out[52]: limbs        0
         herbivore    No
         Name: Python, dtype: object
```

- we can get a description of the various columns. If the data is numeric, the describe method will give us some basic descriptive statistics such as the count, mean, standard deviation, etc:

```
In [53]: df['limbs'].describe()
```

```
Out[53]: count    5.000000
         mean     4.000000
         std      2.828427
         min      0.000000
         25%      4.000000
         50%      4.000000
         75%      4.000000
         max      8.000000
         Name: limbs, dtype: float64
```

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

Panda

- if the data is categorical it provides a count, the number of unique entries, the top category, etc.

In [54]:
```
df['herbivore'].describe()
```

Out[54]:
```
count      5
unique     2
top       No
freq       3
Name: herbivore, dtype: object
```

- It is very easy to add new columns to a dataframe.

In [55]:
```
df['class'] = ['reptile', 'mammal', 'mammal',\
               'mammal', 'mollusc']
```

In [56]:
```
df
```

Out[56]:

	limbs	herbivore	class
Python	0	No	reptile
Iberian Lynx	4	No	mammal
Giant Panda	4	Yes	mammal
Field Mouse	4	Yes	mammal
Octopus	8	No	mollusc

- Pandas allows us to create groups and aggregations:

In [57]:
```
grouped = df['class'].groupby(df['herbivore'])
```

In [59]:
```
grouped.groups
```

Out[59]:
```
{'No': ['Python', 'Iberian Lynx', 'Octopus'], 'Yes': ['Giant Panda', 'Field Mouse']}
```

In [60]:
```
grouped.size()
```

Out[60]:
```
herbivore
No     3
Yes    2
Name: class, dtype: int64
```

- Pandas allows us to create aggregations:

In [62]:
```
from numpy import mean
```

In [63]:
```
limbs = df['limbs'].groupby(df['herbivore'])\
    .aggregate(mean)
```

In [64]:
```
print(limbs)
```

```
herbivore
No     4.0
Yes    4.0
Name: limbs, dtype: float64
```

MATRIX MANIPULATIONS AND LINEAR ALGEBRA

Panda

- Pandas is able to take data from a myriad of sources

Source	Command
Flat file	read_table read_csv read_fwf
Excel file	read_excel ExcelFile.parse
JSON	read_json json_normalize
SQL	read_sql_table read_sql_query read_sql
HTML	read_html

HOMEWORK

1) Creating a row vector, a column vector of values 1, 2, 3
2) Creating a matrix of 3 rows and 2 columns
3) Creating a sparse matrix from

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 3 & 0 \end{bmatrix}$$

4) Selecting element
   - Select third element of vector : vector[2]
   - Select second row, second column : matrix[1,1]

NumPy's arrays make that easy:

```
# Load library
import numpy as np

# Create row vector
vector = np.array([1, 2, 3, 4, 5, 6])

# Create matrix
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
```

5) Describe the shape, size, and dimensions of the matrix
   - Create matrix of value 1, 2. …, 12
   - View number of rows and columns
   - View number of elements
   - View number of dimensions

6) Perform some operation of the following matrix
   - Create function that adds 100 to something
   - Create vectorized function
   - Apply function to all elements in matrix
   - Return maximum element (in matrix)
   - Return minimum element (in matrix)
   - Return mean (in matrix)
   - Return variance (in matrix)
   - Return standard deviation (in matrix)
   - Find the mean value in each column (in matrix)

```
# Create row vector
vector = np.array([1, 2, 3, 4, 5, 6])

# Create matrix
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
```

HOMEWORK

7) Reshaping the following arrays
   - Create 4x3 matrix of value 1, 2, …, 12 and confirm size
   - Reshape matrix into 2x6 matrix and confirm size

8) Transpose a matrix
   - Create 3x3 matrix of value 1, 2, …, 9 and confirm size
   - Transpose the matrix

9) Transpose a vector
   - Create row vector of value 1, 2, …, 6 and confirm size
   - Transpose the row vector

10) Flattening a matrix
    - Create 3x3 matrix of value 1, 2, …, 9 and confirm size
    - Flatten the matrix
    - Reshape the 3x3 matrix to a row vector

11) Find the rank of the following matrix with explanation

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 10 \\ 1 & 1 & 15 \end{bmatrix}$$

12) Find with explanation
    - the determinant of the following matrix
    - The diagonal of the matrix
    - The trace of the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 8 & 9 \end{bmatrix}$$

13) Find Eigenvalues and Eigenvectors with explanation

$$\begin{bmatrix} 1 & -1 & 3 \\ 1 & 1 & 6 \\ 3 & 8 & 9 \end{bmatrix}$$

14) Calculate Dot products of the two matrics $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ and $\begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$

HOMEWORK

15) Matrix Operations of the two following matrices

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 3 & 1 \\ 1 & 3 & 1 \\ 1 & 3 & 8 \end{bmatrix}$$

- Adding
- Subtracting
- Multiplying two matrices
- Multiplying two matrices element-wise
- Inverting

16) Generating random values

- Generate 3 random floats between 0.0 and 1.0
- Generate 3 random integers between 1 and 10
- Draw 3 numbers from a normal distribution with mean = 0.0 and standard deviation of 1.0
- Draw 3 numbers from a logistic distribution with mean 0.0 and scale of 1.0
- Draw 3 numbers greater than or equal to 1.0 and less than 2.0

THE END