East West University
Department of Computer Science and Engineering

CSE 303: Statistics for Data Science
LAB 07
Course Instructor: Dr. Mohammad Rezwanul Huq

Part I : Linear Algebra with NumPy

Lab Objective

Introducing Linear Algebra libraries in NumPy.

Lab Outcome

After completing this lab successfully, students will be able to:

1. **Understand** Linear algebra related functions.
2. **Apply** these functions to solve different problems in Linear Algebra.

Psychomotor Learning Levels

This lab involves activities that encompass the following learning levels in psychomotor domain.

Level	Category	Meaning	Keywords
P1	Imitation	Copy action of another; observe and replicate.	Relate, Repeat, Choose, Copy, Follow, Show, Identify, Isolate.
P2	Manipulation	Reproduce activity from instruction or memory	Copy, response, trace, Show, Start, Perform, Execute, Recreate.

Required Applications/Tools

- Anaconda Navigator (Anaconda3)
  - Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.
  - Popular Tools/IDEs: Spyder, Jupyter Notebook
- Google Colab: Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

Lab Activities

1. Dot Product, Outer Product and Norm of Vector and a Matrix

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

x = np.array([[1,2],[3,4]], dtype=np.int32)
y = np.array([[5,6],[7,8]], dtype=np.int32)
v = np.array([9,10])
w = np.array([11,12])
# Inner product of vectors
print('Inner Product: ')
```

```
print(v.dot(w))
print(np.dot(v, w))
print(np.inner(v, w))
# Outer product of vectors
print('Outer Product: ')
print(np.outer(v,w))
# Norm of a vector / matrix
print('Norm: ')
print(np.linalg.norm(v)) #Euclidean
print(np.linalg.norm(v,1)) #Manhattan
print(np.linalg.norm(v,np.inf)) #Infinite
print(np.linalg.norm(x))
print(np.linalg.norm(x, axis = 1)) #0 for column, 1 for row
```

2. **Vector-Matrix and Matrix-Matrix Multiplication**

```
# Matrix / vector product
print(x.dot(v))
print(np.dot(x, v))
# Matrix / matrix product (Matrix Multiplication)
print(x.dot(y))
print(np.dot(x, y))
print(np.matmul(x,y))
print(x @ y)
```

3. **Determinant, Trace and Rank of a Matrix**

Rank of a Matrix: https://stattrek.com/matrix-algebra/matrix-rank.aspx

```
m = np.array([[2,4,6],
              [1,5,9],
              [3,7,8]])
# determinant of a matrix
print('Determinant: %.2f'%np.linalg.det(m))
# trace of a matrix
print('Trace: %.2f'%np.trace(m))
# rank of a matrix
print('Rank: ', np.linalg.matrix_rank(m))
```

4. **Transpose and Inverse of a Matrix**

```
# transpose of a matrix
mt1 = m.T
print(mt1)
mt2 = np.transpose(m)
print(mt2)

# inverse of a matrix
inv_m = np.linalg.inv(m)
print(inv_m)
```

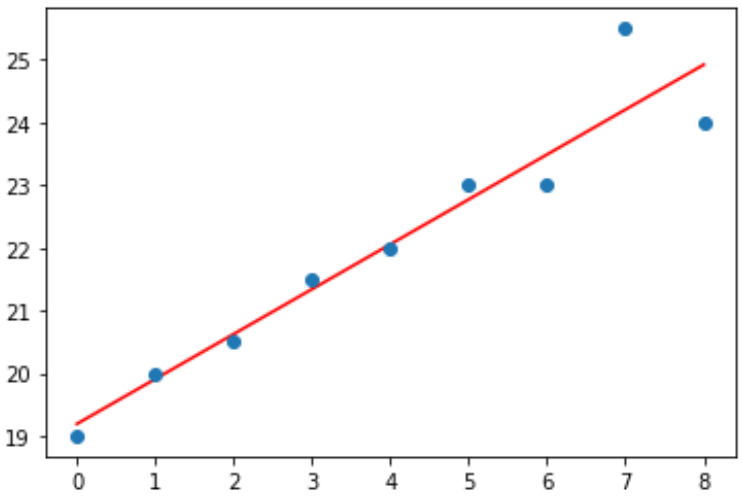
5. Solving a Linear System

```
# Solving a linear system
# Ax = Z => x = inv(A). Z
A = np.array([[1, 2], [3, 4]])
z = np.array([[5], [6]])

# using inverse of a matrix
print('Using Inverse: \n' , np.linalg.inv(A).dot(z))   # slow
computation

#using solve function
print('Using solve: \n', np.linalg.solve(A, z)) # fast computation
```

6. Least Square Solution to a Linear Matrix Equation

Computes the vector x that approximatively solves the equation A**x** = **b**. The equation may be under-, well-, or over-determined (i.e., the number of linearly independent rows of a can be less than, equal to, or greater than its number of linearly independent columns). If a is square and of full rank, then x (but for round-off error) is the "exact" solution of the equation. Else, x minimizes the Euclidean 2-norm || **b** - A**x** ||.

```
x = np.arange(0, 9)
A = np.array([x, np.ones(9)])
print (A)
# linearly generated sequence
y = [19, 20, 20.5, 21.5, 22, 23, 23, 25.5, 24]
# obtaining the parameters of regression line
w = np.linalg.lstsq(A.T, y, rcond=None)[0]
print(w)

# plotting the line
line = w[0]*x + w[1] # regression line
plt.plot(x, line, 'r-')
plt.plot(x, y, 'o')
plt.show()
```



7. Eigenvalues and Eigenvectors

Many problems present themselves in terms of an eigenvalue problem:

$$A \cdot \mathbf{v} = \lambda \cdot \mathbf{v}$$

In this equation A is an n-by-n matrix, **v** is a non-zero n-by-1 vector and λ is a scalar (which may be either real or complex). Any value of λ for which this equation has a solution is known as an eigenvalue of the matrix A. It is sometimes also called the characteristic value. The vector, v, which corresponds to this value is called an eigenvector. The eigenvalue problem can be rewritten as

$$\begin{aligned} A \cdot \mathbf{v} - \lambda \cdot \mathbf{v} &= 0 \\ A \cdot \mathbf{v} - \lambda \cdot I \cdot \mathbf{v} &= 0 \\ (A - \lambda \cdot I) \cdot \mathbf{v} &= 0 \end{aligned}$$

If v is non-zero, this equation will only have a solution if

$$|A - \lambda \cdot I| = 0$$

See more: https://lpsa.swarthmore.edu/MtrxVibe/EigMat/MatrixEigen.html

```
A = np.array([[0,1],[-2,-3]])
#function to calculate eigenvalues and vector
val, vect = np.linalg.eig(A)
print(val)
print(vect)
```

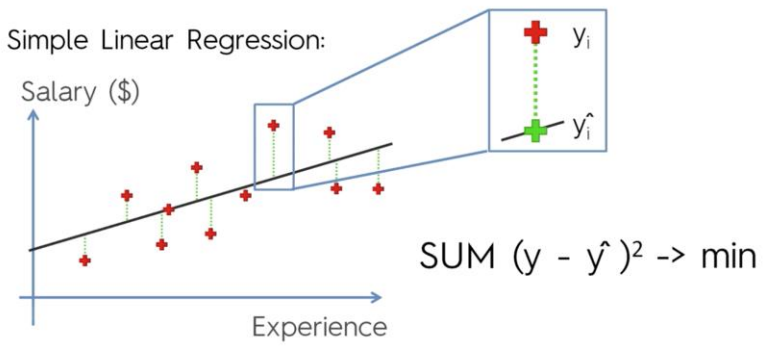
Part II: Simple Linear Regression in Python

Simple Linear Regression is a statistical method to find relationship between two continuous variables. Out of the two variables present, one is independent variable and the other is dependent variable.

This relationship is defined by the famous line equation which you would have learned in high school.

y = a + b*x

We have a dataset with two columns: YearsExperience (independent) and Salary (dependent). We want to predict the Salary based on the given YearsExperience.



1. **Importing the Dataset and Splitting into Train and Test Dataset**

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('SimpleLinearRegression.csv')
dataset.columns = ['YearsExperience', 'Salary']
X = dataset[['YearsExperience']]
y = dataset[['Salary']]
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 1/3, random_state = 0)
```

2. **Fitting the dataset into Simple Linear Regression Model**

```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

3. **Predicting the values of the Test Set**

```
y_pred = regressor.predict(X_test)
```

4. **Visualizing the Correlation**

```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

5. **Visualizing the Test Set Results**

   You have to do it!

6. **Model Evaluation for Test Dataset**

$$R^2 = 1 - \frac{\sum_{1=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{1=1}^{N} (y_i - \bar{y})^2}$$

```
from sklearn import metrics

# model evaluation for testing set

mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)

print("The model performance for testing set")
print("--------------------------------------")
print('MAE is %.2f'% mae)
print('MSE is %.2f'% mse)
print('R2 score is %.2f'% r2)
```

Lab Task (Lab 07: Linear Algebra and Linear Regression):

1. Write a Python script that can compute the inverse of a 3*3 square matrix.
You must not use any library functions.
This code should be submitted as soon as possible by tonight in the appropriate link of Google Classroom. The faster you submit the correct code, the more I will appreciate and remember your name, positively!

2. **Write a Python script that implements your Linear Regression Model for a simple linear regression problem (univariate).** You must not use any library functions to do that. You must show the plots as you have done in the example mentioned above.

3. **Write a Python script that implements your Linear Regression Model for a multivariate linear regression problem.** Use concepts of Linear Algebra to accomplish that. You must use your own built functions and modules and must not use any library functions for inverse or other purposes. You must show appropriate plotting.

Task 2 and Task 3 should be submitted in another Google Classroom link by 20th of May, 2021. The sooner, The Better, of course.