East West University
Department of Computer Science and Engineering

CSE 303: Statistics for Data Science
LAB 05
Course Instructor: Dr. Mohammad Rezwanul Huq

Introducing Numpy and Matplotlib Libraries

Lab Objective

Introducing Numpy and Matplotlib libraries for n-dimensional arrays manipulations and appropriate plotting.

Lab Outcome

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

1. **Understand** different Numpy functions and techniques to manipulate arrays.
2. **Apply** Numpy functions to perform different operations and **Produce** plotting accordingly using Matplotlib.

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. **What is Numpy?**
   NumPy is a module for Python. The name is an acronym for "Numeric Python" or "Numerical Python". It is an extension module for Python, mostly written in C. This makes sure that the precompiled mathematical and numerical functions and functionalities of Numpy guarantee great execution speed.

2. **Comparison between Core Python and Numpy**
   The advantages of Core Python:
   - high-level number objects: integers, floating point
   - containers: lists with cheap insertion and append methods, dictionaries with fast lookup

Advantages of using Numpy with Python:

- array oriented computing
- efficiently implemented multi-dimensional arrays
- designed for scientific computation

3. First look into Numpy

```
import numpy as np
# create a Python list of temperature in degree celcius
cvalues = [20.1, 20.8, 21.9, 22.5, 22.7, 22.3, 21.8, 21.2, 20.9,
20.1]
# converting this list into one-dimensional Numpy array
C = np.array(cvalues)
print(cvalues)
print(type(cvalues))
print(C)
print(type(C))
```

4. Element-wise Operations in Numpy (Scalar Operations)

Convert the values in cvalues list in Fahrenheit scale.
Write the code using list comprehension.

In Numpy, it is much easier because Numpy supports scalar operations.

```
F = C * 9/5 + 32
print(F)

# A few other examples of scalar operations
A = np.array([[1,2,3],[4,5,6]])
print(A)
print(A.shape)
B = np.array([[7,8,9],[10,11,12]])
print(B)
print(B.shape)
C = A + B
print(C)
print(C.shape)
```

5. Array Indexing

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
b = a[:,0:2]
print(b)
print(a[0,0])
print(a)
```

6. Boolean Array Indexing (for Filtering)

```
a = np.array([[1,2], [3, 4], [5, 6]])
bool_idx = (a > 2)
print(bool_idx)
print(a[bool_idx])
# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

7. Numpy Simple Math

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
# Elementwise sum
print(x + y)
print(np.add(x, y))
# Elementwise difference
print(x - y)
print(np.subtract(x, y))
# Elementwise product
print(x * y)
print(np.multiply(x, y))
# Elementwise division
print(x / y)
print(np.divide(x, y))
# Elementwise square root
print(np.sqrt(x))
```

8. Numpy Dot product and Vector and Matrix Multiplication

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
v = np.array([9,10])
w = np.array([11, 12])
# Inner product of vectors
print(v.dot(w))
print(np.dot(v, w))
# Matrix / vector product
print(x.dot(v))
print(np.dot(x, v))
# Matrix / matrix product
print(x.dot(y))
print(np.dot(x, y))
```

9. Numpy Mathematical Functions

```
x = np.array([[1,2],[3,4]])
print(np.sum(x))  # Compute sum of all elements
print(np.sum(x, axis=0))  # Compute sum of each column
print(np.sum(x, axis=1))  # Compute sum of each row
```

Full List of Mathematical Functions is given in this link. You should have a look in these functions.
https://numpy.org/doc/stable/reference/routines.math.html

10. Numpy Statistical Functions

```
data1 = np.arange(1.5)
print(np.average(data1))
data2 = np.arange(6).reshape(3,2)
print(data2)
print(np.average(data2, axis = 0))
print(np.average(data2, axis = 1))
```

Full List of Statistical Functions is given in this link. You must have a look in these functions.
https://numpy.org/doc/stable/reference/routines.statistics.html

11. Broadcasting

Broadcasting is a powerful mechanism that allows Numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

Adding a constant vector to each row of a matrix

```
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)

# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v
```

Better Solution

```
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))

y = x + vv
print(y)
```

Using Broadcasting

```
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)
```

12. Some special Numpy Arrays

```
np.zeros(5)
np.zeros((2,3))
np.random.rand(2,3)
np.full((2,2),7)
np.eye(3)
np.arange(2,10,2)
np.linspace(0,1,5)

a = np.array([3,6,9,12])
np.reshape(a,(2,2))

a = np.ones((2,2))
b = a.flatten()

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

13. Basic Plotting

```
import matplotlib.pyplot as plt

# x axis values
x = np.array([1,2,3])
# corresponding y axis values
y = np.array([2,4,1])

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('My first graph!')

# function to show the plot
plt.show()
```

```
import matplotlib.pyplot as plt

# x-coordinates of left sides of bars
left = [1, 2, 3, 4, 5]

# heights of bars
height = [10, 24, 36, 40, 5]

# labels for bars
tick_label = ['one', 'two', 'three', 'four', 'five']

# plotting a bar chart
plt.bar(left, height, tick_label = tick_label, width = 0.8, color =
['red', 'green'])

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')

# function to show the plot
plt.show()
```