CSCI 2040: Introduction to Python **2022-2023 Term 1**

Lab Assignment 5

Instructor: Yim Pan, CHUI Due: 23:59 on Wednesday, Dec. 14, 2022

Notes

1. You are allowed to form a group of two to do this lab assignment.
2. You are strongly recommended to bring your own laptop to the lab with Anaconda[1] and Pycharm[2] installed. You don't even have to attend the lab session if you know what you are required to do by reading this assignment.
3. Only **Python 3.x** is acceptable. You need to specify your python version as the first line in your script. For example, if your scripts are required to run in **Python 3.6**, the following line should appear **in the first line of your scripts**:

```
#python_version == '3.6'
```

4. For those of you using the Windows PC in SHB 924A (NOT recommended) with your CSDOMAIN account[3], please login and open "Computer" on the desktop to check if an "S:" drive is there. If not, then you need to click "Map network drive", use "S:" for the drive letter, fill in the path `\\ntsvr1\userapps` and click "Finish". Then open the "S:" drive, open the **Python3** folder, and click the "IDLE (Python 3.7 64-bit)" shortcut to start doing the lab exercises. You will also receive a paper document and if anything has changed, please be subject to the paper.
5. Your code should only contain specified functions. Please delete all the debug statements (e.g. print) before submission.

Exercise 1 **(50 marks)**

`Numpy` is an important package for scientific computation, e.g., in quantum computing and signal processing.[4] In this exercise, you are asked to write codes to illustrate the efficiency and convenience of `Numpy` over vanilla python lists.

Let's start from an example: the element-wise product. The element-wise product takes two vectors of the same size and produces another vector of the same size whose i^{th} element is the product of the original two vectors' i^{th} element. We will compare the time cost of calculating the element-wise product between using `Numpy` array and vanilla python list.

[1]An open data science platform powered by Python. `https://www.continuum.io/downloads`
[2]A powerful Python IDE. `https://www.jetbrains.com/pycharm/download/`
[3]A non-CSE student should ask the TA for a CSDOMAIN account.
[4]https://numpy.org

```
from time import time
import numpy as np
import random

def product_Plain(v1, v2):
    v3 = [];
    for i in range(len(v1)):
        v3.append(float(v1[i] * v2[i]))
    return v3

def product_Pythonic(v1, v2):
    # input v1 and v2 are lists.
    v3 = [float(x*y) for x, y in zip(v1, v2)]
    return v3

def product_Numpy(v1, v2):
    # input v1 and v2 are arrases.
    v3 = v1 * v2
    return v3

n = 5000000
vec_1 = [random.random() for _ in range(n)] # list
vec_2 = [random.random() for _ in range(n)] # list
vec_1_np = np.random.rand(5000000) # ndarray
vec_2_np = np.random.rand(5000000) # ndarray
# for example, vec_1 = [1,2,3] is a list and
# vec_2 = np.array([1,2,3]) is an array.

t = time()
vec_3 = product_Plain(vec_1, vec_2)
print("Plain", time() - t)

t = time()
vec_3 = product_Pythonic(vec_1, vec_2)
print("Pythonic", time() - t)

t = time()
vec_3 = product_Numpy(vec_1_np, vec_2_np)
print("Numpy", time() - t)
```

The output of this code is (the results vary across computers):

```
Plain 1.0459966659545898
Pythonic 0.8720011711120605
Numpy 0.07000112533569336
```

This first "Plain" case corresponds to the so-called "ugly" code in Lab 4's Exercise 4, while the second "Pythonic" is recommended by that Exercise. As we can see, the time cost of the "Pythonic" code is smaller than the "Plain" code case; yet another advantage of being "Pythonic". No worries! In this exercise, you are NOT required to obey the "Pythonic" rule—either the "ugly" or "Pythonic" codes are acceptable. Let's focus on the advantage of Numpy upon plain python lists: the Numpy code is 12 times faster than the "Pythonic" one!

Alright, fasten your seat belts, we're going into the cave.

1. **Matrix Multiplication (15 marks)**
   Suppose you are given two square matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{1000\times 1000}$. Please respectively use plain python list and Numpy array to calculate the product of these two matrices and print out the time cost of both cases as the above example shows.

   Recall that the output matrix $\boldsymbol{C}$ is also in $\mathbf{R}^{1000\times 1000}$ and its (i, j) entry c_{ij} can be calculated via

$$c_{ij} = \sum_{k=1}^{1000} a_{ik} b_{kj},$$

   where a_{ij} and b_{ij} are the (i, j) entry for matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively.

   Please implement a function in p1-1.py and follow the function names as below.

```
def MatrixMultiple_Plain(M1, M2):
    # input M1 and M2 are lists of lists.
    ...
    return M3


def MatrixMultiple_Numpy(M1, M2):
    # input M1 and M2 are 2d arrases.
    ...
    return M3

# n may vary
n = 1000
Matrix_1 = [[random.random() for _ in range(n)] for _ in range(n)]
Matrix_2 = [[random.random() for _ in range(n)] for _ in range(n)]
Matrix_1_np = np.random.rand(n, n)
Matrix_2_np = np.random.rand(n, n)
```

2. **Vandermonde Matrix (15 marks)**

   The Vandarmonde matrix has wide applications in linear algebra[5]. It is a matrix with the terms of a geometric progression in each row. An $m \times n$ matrix can be expressed as follows,

$$\boldsymbol{V} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \dots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^{n-1} \end{bmatrix}.$$

   Given a list of inputs $(x_1, x_2, \dots, x_m)$ and integer n, you are asked to write two functions (one based on plain python, the other via Numpy) to calculate the Vandarmonde matrix and compare their time costs as before.

   Please implement a function in `p1-2.py` and follow the function names as below.

```
def VMatrix_Plain(x, n):
    # input x is a list and input n is an integer.
    ...
    return VMatrix

def VMatrix_Numpy(x, n):
    # input x is an array and input n is an integer.
    ...
    return VMatrix

# n and m may vary
n = 1000
m = 500
x = [random.random() for _ in range(m)]
x_np = np.random.rand(m)
```

3. **Broadcasting Calculation (20 marks)**

   Broadcasting is a convenient feature of Numpy. It allows the basic calculations—e.g., element-wise product or addition— of two matrices (or one matrix and one vector) of different sizes via automatically filling "pseduo" dimensions. You can find a formal explanation of broadcasting in this link[6]. In this exercise, you are ask to write functions

[5]https://www.wikiwand.com/en/Vandermonde_matrix
[6]https://numpy.org/doc/stable/user/basics.broadcasting.html

(one based on plain python, the other on Numpy) for the following calculation:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ x_{31} & x_{32} & x_{33} & \dots & x_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix} \xrightarrow{f} \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ 2x_{21} & 2x_{22} & 2x_{23} & \dots & 2x_{2m} \\ 3x_{31} & 3x_{32} & 3x_{33} & \dots & 3x_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ nx_{n1} & nx_{n2} & nx_{n3} & \dots & nx_{nm} \end{bmatrix},$$

and compare both functions' time costs as the beginning example.

Please make sure to use at least one broadcasting operation in the Numpy case. Please implement a function in p1-3.py and follow the function names as below.

```
def MatrixTransfer_Plain(M):
    ...
    return MT

def MatrixTransfer_Numpy(M):
    ...
    return MT

# n and m may vary
n = 1000
m = 1000
Matrix_1 = [[random.random() for _ in range(m)] for _ in range(n)]
Matrix_1_np = np.random.rand(n, m)
```

Exercise 2 (50 marks)

Imagine that you are in your physics lab course to conduct experiments on Ohm's Law[7]. You have just measured several groups of currents and the corresponding voltages on a resistor. You recorded the data in the following table. (I denotes current in A while V denotes voltage in V)

Data group	1	2	3	4	5	6
I (A)	1.0	1.1	1.4	1.6	1.9	2.1
V (V)	4.8	5.4	7.2	7.9	9.7	10.6

Now, you will use Python to do data analysis and visualization tasks on the experimental data. Matplotlib and NumPy will come in handy in this problem. You can import them in the following way.

[7] https://en.wikipedia.org/wiki/Ohm%27s_law

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

1. **Bar chart: (10 marks)** You will visualize the current at each voltage using a bar chart. Please implement a function in `p2.py` to plot a bar chart for the currents of the resistor when the list of voltages are configured as the categories on the horizontal axis and the list of currents are configured as the heights of the bars. Use `bar()` as the plotting function.

   (a) Please save your figure as "`bar.png`" using `savefig()`. After you save the figure, you can use `clf()` to clear the pyplot figure, so that this bar chart won't intervene with the rest of visualization tasks.

   (b) Please use `xlabel()` and `ylabel()` to label the physical quantities together with their units on the axes of your figure. Put "V (V)" on the x-axis and "I (A)" on the y-axis.

   (c) Please note that the categories are in string type, not in numeric types. Otherwise, you will discover that the intervals between adjacent bars are not identical, but related to the distances between the adjacent voltage values instead.

2. **Poly fit with `Numpy`, scatter plot and line chart: (20 marks)** Now, in order to find the relationship between voltage and current based on the experiment, you will resort to `polyfit()`[8] in `NumPy` and fit the following linear equation from the data in the table.

$$V = a \cdot I + b \tag{1}$$

   In this equation, a and b are the parameters to be solved by using `polyfit()`. Make sure that you read the document on `polyfit()` thoroughly to fully understand the input parameters and outputs.

   Please implement a function in `p2.py` to

   (a) solve a and b from Equation 1 by applying `polyfit()` on the experimental data, and

   (b) print out the resulting coefficient a and b from `polyfit()` with only the first two digits after the decimal point, and

   (c) use `scatter()` in `Matplotlib` to create a scatter plot from the data points in the table, with currents on the x-axis and voltages on the y-axis and

   (d) apply `plot()` in `Matplotlib` to create a line chart from Equation 1, with the resulting a and b from `polyfit()`, and with currents on the x-axis and voltages on the y-axis.

[8] `https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html`

(e) Please note that it is required for you to plot the line chart and the scatter plot on the same figure. However, assign a different color to the line from the scatter points.

(f) Please use `xlabel()` and `ylabel()` to label the physical quantities together with their units on the axes of your figure. Put "I (A)" on the x-axis and "V (V)" on the y-axis.

(g) Please use `legend()` in `Matplotlib` to provide some information about the scatter plot and the line chart. Denote the scattered points as "data points" and denote the line as equation "$V = a \cdot I + b$". Please replace a and b here with what you have obtained from `polyfit()` while keeping only the first two digits after the decimal point.

(h) Finally, please save your figure as "`polyfit-np.png`" using `savefig()`. Once again, after saving the figure, you can use `clf()` to clear the pyplot figure, so that this figure won't intervene with the rest of visualization tasks.

3. **Self-implemented poly fit, scatter plot and line chart: (20 marks)** Notice that when you use `polyfit()` from `NumPy`, there is a non-zero intercept b. In reality, however, Ohm's law formula does not hold place for such a non-zero bias term b, as expressed in the following equation.

$$V = R \cdot I \tag{2}$$

In `polyfit()`, we are unable to cancel out the bias term. Therefore, in this section of this exercise, you are asked to implement a poly fit by yourself to fit the experimental data with Equation 2. You will solve the term R from the data points in the table. You are **NOT** allowed to use `SciPy` for fitting Equation 2 in this section of the exercise.

Here, we resort to least-squares method[9] with zero intercept. The least-squares method is the process of obtaining the best-fitting curve or line for the given dataset by reducing the sum of the squared residues (or fitting deviation) of the data points from the curve.

To solve R, we construct an error function, i.e. residue,

$$L(R) = \sum_{i=1}^{6} (V_i - R \cdot I_i)^2 \tag{3}$$

Minimizing the error function in Equation 3 can be solved by taking the first order derivative,

$$\frac{\mathrm{d}L(R)}{\mathrm{d}R} = -2\sum_{i=1}^{6} I_i V_i + 2R\sum_{i=1}^{6} I_i^2 \tag{4}$$

Then, we further assign zero to the first order derivative, and rearrange the terms to

[9] `https://en.wikipedia.org/wiki/Least_squares`

find the resulting R.

$$R = \frac{\sum_{i=1}^{6} I_i V_i}{\sum_{i=1}^{6} I_i^2} \tag{5}$$

Furthermore, it's evident that the second order derivative of $L(R)$ is always positive. Thus, the R in Equation 5 can minimize the error function $L(R)$. This suggests that the R in Equation 5 can best fit the data points.

Equation 3 and Equation 4 are just the intermediate steps to deduce the final Equation 5, and they won't be needed in your code. In other words, to solve R, you **ONLY** need Equation 5.

Please implement a function in `p2.py` and

(a) print out the resulting coefficient R from computing Equation 5 while keeping only the first two digits after the decimal point, and

(b) use `scatter()` in `Matplotlib` to create a scatter plot from the data points in the table, with currents on the x-axis and voltages on the y-axis and

(c) apply `plot()` in `Matplotlib` to create a line chart of Equation 2, with the R you have just calculated from Equation 5, and with currents on the x-axis and voltages on the y-axis.

(d) Please note that it is required for you to plot the line chart and the scatter plot on the same figure. However, assign a different color to the line from the scatter points.

(e) Please use `xlabel()` and `ylabel()` to label the physical quantities together with their units on the axes of your figure. Put "I (A)" on the x-axis and "V (V)" on the y-axis.

(f) Please use `legend()` in `Matplotlib` to provide some information about the scatter plot and the line chart. Denote the scattered points as "data points" and denote the line as equation "$V = R \cdot I$". Please replace R here with what you have obtained from Equation 5 while keeping only the first two digits after the decimal point.

(g) Finally, please save your figure as "`polyfit-self.png`" using `savefig()`.

Please contain all the functions in this exercise within `p2.py`. You only need to consider the data points in the table we have provided you with for this exercise, and please don't generalize your code to other data points. We don't expect you to parse any inputs in this exercise, so you can just hard-code the data points in your script.

We won't use automatic test scripts for the grading of this exercise. You don't have to submit the "`*.png`" image files along with your code. However, we do expect that when we run your `p2.py`, the required "`*.png`" image files will be generated automatically.

Submission rules

1. Please name the `functions` and script files with the **exact** names specified in this assignment. Any script that has any wrong name or syntax error will not be marked.
2. For each group, please pack all your script files as a single archive named as

   `<student-id1>_<student-id2>_lab5.zip`

   For example, `1155012345_1155054321_lab5.zip`, i.e., just replace `<student-id1>` and `<student-id2>` with your own student IDs. If you are doing the assignment alone, just leave `<student-id2>` empty, e.g, `1155012345_lab5.zip`.
3. Upload the zip file to your blackboard (`https://blackboard.cuhk.edu.hk`),
   - Only one member of each group needs to upload the archive file.
   - Subject of your file should be `<student-id1>_<student-id2>_lab5` if you are in a two-person group or `<student-id1>_lab5` if not.
   - No later than 23:59 on Wednesday, Dec. 14, 2022
4. Students in the same group would get the same marks. Marks will be deducted if you do not follow the submission rules. Anyone/Anygroup who is caught plagiarizing would get 0 score!