SFWRTECH 3PR3:

Procedural and Objective Oriented Programming Concepts

(Assignment #4)

Student Name: Dojae Kim

Student Number: 400420323

Professor Name: Dr. Seshasai Srinivasan

SFWRTECH 3PR3 (Assignment 4)

Objective

The purpose of this Assignment 4:

1. To learn how to use list, tuples in Python programs.
2. To learn how to write loops and decision statements in Python
3. To learn how to build and package Python modules for reusability.
4. To learn how to read files in Python.
5. To learn how to print a graph in Python.

Equation

Part A:

1. $\hat{s}(t) = -\left(Asin\left(\frac{2\pi}{T}t\right) + \mu\right)e^{B^{(\frac{2\pi}{T}t)}}$ .....*(1)*

Part B:

1. $\hat{s}(t) = -\left(Asin\left(\frac{2\pi}{T}t\right) + \mu\right)e^{B^{(\frac{\frac{2\pi}{T}t}{C})}}$ .....*(3)*

Part C:

1. $\hat{s}(t) = -\left(Asin\left(\frac{2\pi}{T}t + \delta\right) + \mu\right)e^{B^{(\frac{\frac{2\pi}{T}t+\delta}{C})}}$ .....*(4)*

MSE

1. $MSE = \frac{1}{N}\sum_{t=1}^{N}(\hat{s}(t) - S(t))^2$ .....*(2)*

Part A Source Code

```
# Student name: Dojae Kim
# Student number: 400420323
# Student email: kim408@mcmaster.ca
# Lecture: SFWRTECH 3PR3
# Assignment 4 Part A

import math
from typing import List, Tuple
import matplotlib.pyplot as plt

DATA_FILENAME = "A04_sfwr_data_01.txt"


def getInputs(filename: str) -> List[float]:
    """
    Read data from file and return list of points
    :param: filename Name of file to read from
    :return: List of points from file
    """
    res: List[float] = []
    with open(filename) as file:
        for line in file:
            res.append(float(line))
    return res


def evaluateModel(t: float, A: float, B: float, mu: float) -> float:
    """
    Calculate an equation
    :param t: time
    :param A: A constant
    :param B: B constant
    :param mu: mu constant
    :return: Value of equation
    """
    value = (2 * math.pi * t) / 120
    return -1 * (A * math.sin(value) + mu) * (math.e ** (B * value))


def getFit(data: List[float], A: float, B: float, mu: float) -> float:
    """
    Calculate fit (MSE) for set of data and A, B and mu
    :param data: Data to check MSE on
    :param A: A for equation
    :param B: B for equation
    :param mu: mu for equation
    :return: MSE on set of data with A, B and mu
```

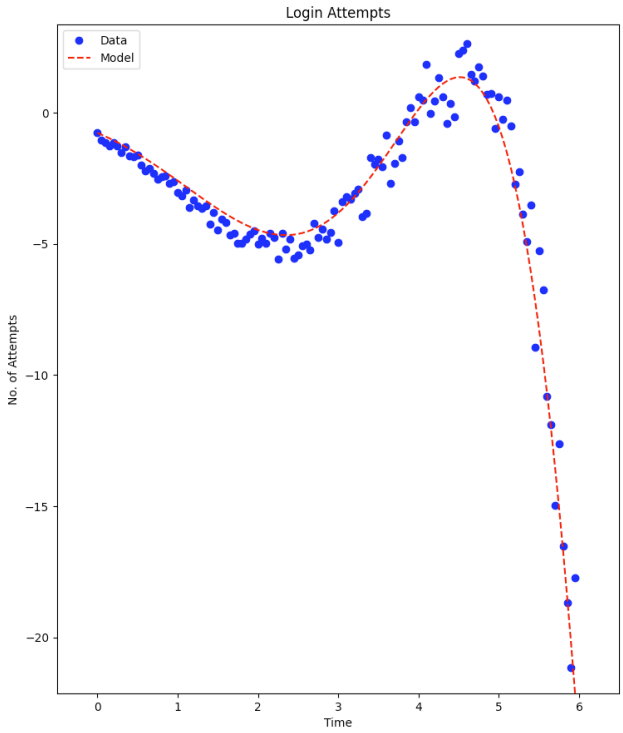
```
    """
    mse = 0
    for i in range(len(data)):
        mse += (evaluateModel(i + 1, A, B, mu) - data[i]) ** 2
    return mse / len(data)


def setParameter(data: List[float]) -> Tuple[List[float], float, float, float]:
    """
    Find A, B and mu that gives MSE less than 1
    :param data: Data to check MSE on
    :return: Tuple with List of MSE on the way to find MSE < 1 and A. B and mu
    """
    mse: List[float] = []
    step = 5
    A = 0
    B = 0
    mu = 0
    found = False

    for mu in range(0, 201, step):
        if found:
            break
        for A in range(0, 201, step):
            if found:
                break
            for B in range(0, 201, step):
                an_mse = getFit(data, A / 100, B / 100, mu / 100)
                if len(mse) == 0 or mse[-1] > an_mse:
                    mse.append(an_mse)
                if an_mse < 1:
                    found = True
                    break  # Found A, B and mu
    print("The value of A is:", A / 100)
    print("The value of B is:", B / 100)
    print("The value of mu is:", mu / 100)
    print("The MSE predicted by our model is:", round(mse[-1], 3))
    print("Number of iterations: ", len(mse))
    return mse, A / 100, B / 100, mu / 100


def plot_data(data: List[float],
              mse_data: Tuple[List[float], float, float, float]) -> None:
    """
    Plot data in 2 subplots
    :param data: Data for left subplot
    :param mse_data: Data for right subplot and line on left one
    :return: None
```

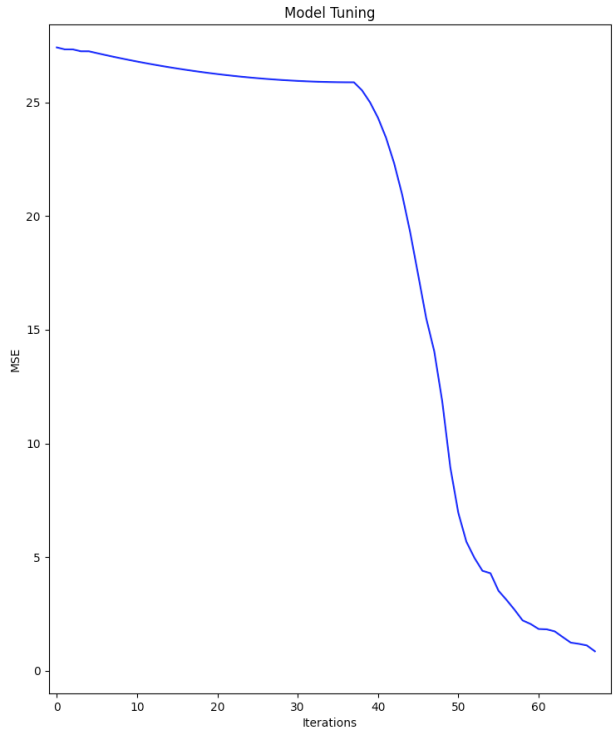
```
    """
    mse = mse_data[0]
    A = mse_data[1]
    B = mse_data[2]
    mu = mse_data[3]

    model = []
    for i in range(120):
        model.append(evaluateModel(i + 1, A, B, mu))

    times = []
    for i in range(120):
        times.append(i / 20)

    tries = []
    for i in range(len(mse)):
        tries.append(i)

    plt.subplot(1, 2, 1)  # the figure has 1 row, 2 columns, and this plot is the first plot.
    plt.title('Login Attempts')
    plt.xlabel('Time')
    plt.ylabel('No. of Attempts')
    plt.xlim([-0.5, 6.5])
    plt.ylim([min(min(data), min(data)-1), max(max(data), max(model) + 2)])

    plt.plot(times, data, 'bo', label='Data')
    plt.plot(times, model, '--r', label='Model')
    plt.legend(loc=2)

    plt.subplot(1, 2, 2)  # the figure has 1 row, 2 columns, and this plot is the second plot.
    plt.title('Model Tuning')
    plt.xlabel('Iterations')
    plt.ylabel('MSE')
    plt.xlim([-1, len(mse) + 1])
    plt.ylim([-1, max(mse) + 1])
    plt.plot(tries, mse, '-b')

    plt.show()


if __name__ == '__main__':
    data = getInputs(DATA_FILENAME)
    mse_data = setParameter(data)

    # Plot the data
    plot_data(data, mse_data)
```

Part A Sample Output

Sample 1:

```
The value of A is: 0.8
The value of B is: 0.55
The value of mu is: 0.7
The MSE predicted by our model is: 0.859
Number of iterations:  68
```





Part B Source Code

```
# Student name: Dojae Kim
# Student number: 400420323
# Student email: kim408@mcmaster.ca
# Lecture: SFWRTECH 3PR3
# Assignment 4 Part B

import math
from typing import List, Tuple
import matplotlib.pyplot as plt

DATA_FILENAME = "A04_sfwr_data_03.txt"


def getInputs(filename: str) -> List[float]:
    """
    Read data from file and return list of points
    :param: filename Name of file to read from
    :return: List of points from file
    """
    res: List[float] = []
    with open(filename) as file:
        for line in file:
            res.append(float(line))
    return res


def evaluateModel(t: float, A: float, B: float, C: float, mu: float) -> float:
    """
    Calculate an equation
    :param t: time
    :param A: A constant
    :param B: B constant
    :param C: C constant
    :param mu: mu constant
    :return: Value of equation
    """
    value = (2 * math.pi * t) / 120
    return -1 * (A * math.sin(value) + mu) * (math.e ** (B * value / C))


def getFit(data: List[float], A: float, B: float, C: float, mu: float) -> float:
    """
    Calculate fit (MSE) for set of data and A, B and mu
    :param data: Data to check MSE on
    :param A: A for equation
```

```
    :param B: B for equation
    :param C: C for equation
    :param mu: mu for equation
    :return: MSE on set of data with A, B, C and mu
    """
    mse = 0
    for i in range(len(data)):
        mse += (evaluateModel(i + 1, A, B, C, mu) - data[i]) ** 2
    return mse / len(data)


def setParameter(data: List[float]) -> \
        Tuple[List[float], float, float, float, float]:
    """
    Find A, B, C, and mu that gives MSE less than 0.5
    :param data: Data to check MSE on
    :return: Tuple with List of MSE on the way to find MSE < 0.5 and A. B, C and mu
    """
    mse: List[float] = []
    step = 5
    A = 0
    B = 0
    C = 10
    mu = 0
    found = False
    for A in range(0, 201, step):
        if found:
            break
        for B in range(0, 201, step):
            if found:
                break
            for C in range(10, 201, step):
                if found:
                    break
                for mu in range(0, 201, step):
                    an_mse = getFit(data, A / 100, B / 100, C / 100, mu / 100)
                    if len(mse) == 0 or mse[-1] > an_mse:
                        mse.append(an_mse)
                    if an_mse < 0.5:
                        found = True
                        break  # Found A, B and mu
    print("The value of A is:", A / 100)
    print("The value of B is:", B / 100)
    print("The value of C is:", C / 100)
    print("The value of mu is:", mu / 100)
    print("The MSE predicted by our model is:", round(mse[-1], 3))
    print("Number of iterations: ", len(mse))
    return mse, A / 100, B / 100, C / 100, mu / 100
```

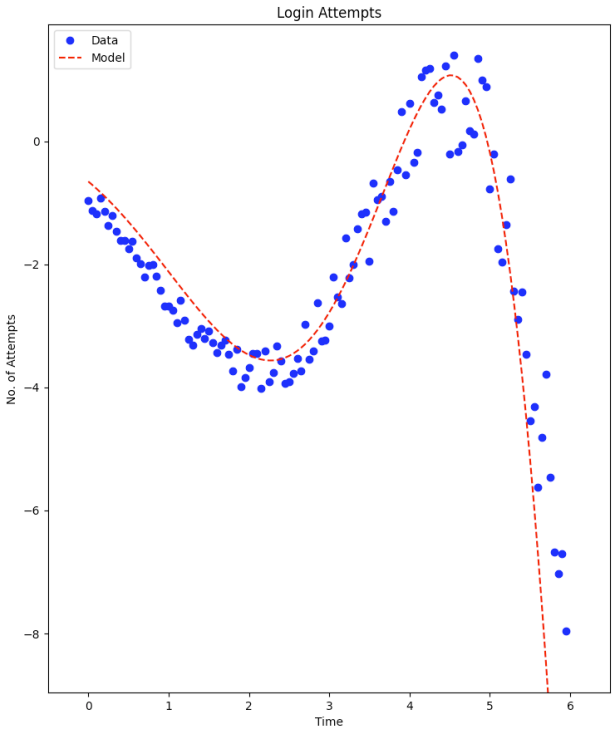
```
def plot_data(data: List[float],
              mse_data: Tuple[List[float], float, float, float, float]) -> None:
    """
    Plot data in 2 subplots
    :param data: Data for left subplot
    :param mse_data: Data for right subplot and line on left one
    :return: None
    """
    mse = mse_data[0]
    A = mse_data[1]
    B = mse_data[2]
    C = mse_data[3]
    mu = mse_data[4]

    model = []
    for i in range(120):
        model.append(evaluateModel(i + 1, A, B, C, mu))

    times = []
    for i in range(120):
        times.append(i / 20)

    tries = []
    for i in range(len(mse)):
        tries.append(i)

    plt.subplot(1, 2, 1)  # the figure has 1 row, 2 columns, and this plot is the first plot.
    plt.title('Login Attempts')
    plt.xlabel('Time')
    plt.ylabel('No. of Attempts')
    plt.xlim([-0.5, 6.5])
    plt.ylim([min(min(data), min(data) - 1), max(max(data), max(data) + 0.5)])

    plt.plot(times, data, 'bo', label='Data')
    plt.plot(times, model, '--r', label='Model')
    plt.legend(loc=2)

    plt.subplot(1, 2, 2)  # the figure has 1 row, 2 columns, and this plot is the second plot.
    plt.title('Model Tuning')
    plt.xlabel('Iterations')
    plt.ylabel('MSE')
    plt.xlim([-1, len(mse) + 1])
    plt.ylim([-1, max(mse) + 1])
    plt.plot(tries, mse, '-b')

    plt.show()
```

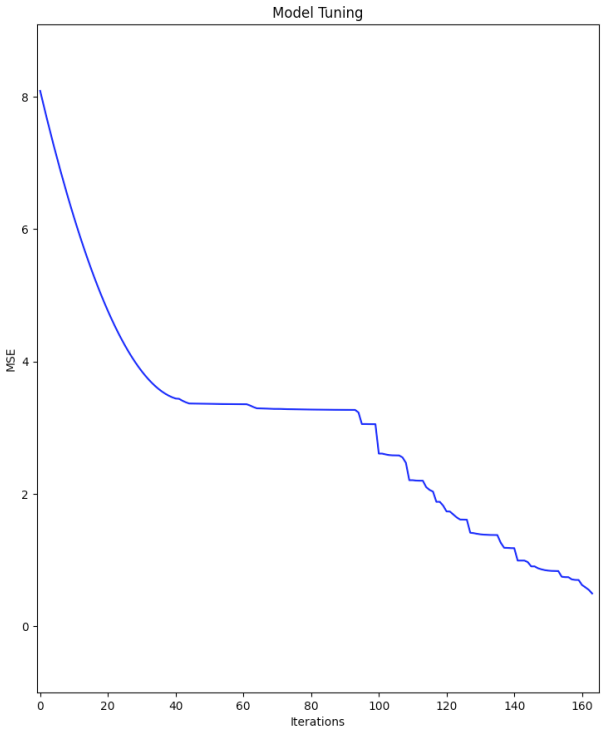
```
if __name__ == '__main__':
    data = getInputs(DATA_FILENAME)
    mse_data = setParameter(data)

    # Plot the data
    plot_data(data, mse_data)
```

Part B Sample Output

Sample 1:

```
The value of A is: 0.7
The value of B is: 0.2
The value of C is: 0.4
The value of mu is: 0.6
The MSE predicted by our model is: 0.497
Number of iterations:  164
```





Part C Source Code

```
# Student name: Dojae Kim
# Student number: 400420323
# Student email: kim408@mcmaster.ca
# Lecture: SFWRTECH 3PR3
# Assignment 4 Part C

import math
from typing import List, Tuple
import matplotlib.pyplot as plt

DATA_FILENAME = "A04_sfwr_data_05.txt"


def getInputs(filename: str) -> List[float]:
    """
    Read data from file and return list of points
    :param: filename Name of file to read from
    :return: List of points from file
    """
    res: List[float] = []
    with open(filename) as file:
        for line in file:
            res.append(float(line))
    return res


def evaluateModel(t: float, A: float, B: float, C: float, mu: float,
                  shift: float) -> float:
    """
    Calculate an equation
    :param t: time
    :param A: A constant
    :param B: B constant
    :param C: C constant
    :param mu: mu constant
    :param shift: shift constant
    :return: Value of equation
    """
    value = ((2 * math.pi * t) / 120) + shift
    return -1 * (A * math.sin(value) + mu) * (math.e ** (B * value / C))


def getFit(data: List[float], A: float, B: float, C: float, mu: float,
           shift: float) -> float:
    """
```

```
    Calculate fit (MSE) for set of data and A, B and mu
    :param data: Data to check MSE on
    :param A: A for equation
    :param B: B for equation
    :param C: C for equation
    :param mu: mu for equation
    :param shift: mu for equation
    :return: MSE on set of data with A, B, mu and shift
    """
    mse = 0
    for i in range(len(data)):
        mse += (evaluateModel(i + 1, A, B, C, mu, shift) - data[i]) ** 2
    return mse / len(data)


def setParameter(data: List[float]) -> Tuple[
    List[float], float, float, float, float, float]:
    """
    Find A, B and mu that gives MSE less than 0.1
    :param data: Data to check MSE on
    :return: Tuple with List of MSE on the way to find MSE < 0.1 and A. B, C,
    mu and shift
    """
    mse: List[float] = []
    step = 5
    A = 0
    B = 0
    C = 10
    mu = 0
    shift = 0
    found = False
    for mu in range(0, 201, step):
        if found:
            break
        for A in range(0, 201, step):
            if found:
                break
            for B in range(0, 201, step):
                if found:
                    break
                for C in range(10, 201, step):
                    if found:
                        break
                    for shift in range(0, 151, step):
                        an_mse = getFit(data, A / 100, B / 100, C / 100,
                                        mu / 100, shift / 100)
                        if len(mse) == 0 or mse[-1] > an_mse:
                            mse.append(an_mse)
```

```
                    if an_mse < 0.1:
                        found = True
                        break  # Found A, B and mu
    print("The value of A is:", A / 100)
    print("The value of B is:", B / 100)
    print("The value of C is:", C / 100)
    print("The value of mu is:", mu / 100)
    print("The value of shift is:", shift / 100)
    print("The MSE predicted by our model is:", round(mse[-1], 3))
    print("Number of iterations: ", len(mse))
    return mse, A / 100, B / 100, C / 100, mu / 100, shift / 100


def plot_data(data: List[float],
              mse_data: Tuple[
                  List[float], float, float, float, float, float]) -> None:
    """
    Plot data in 2 subplots
    :param data: Data for left subplot
    :param mse_data: Data for right subplot and line on left one
    :return: None
    """
    mse = mse_data[0]
    A = mse_data[1]
    B = mse_data[2]
    C = mse_data[3]
    mu = mse_data[4]
    shift = mse_data[5]

    model = []
    for i in range(120):
        model.append(evaluateModel(i + 1, A, B, C, mu, shift))

    times = []
    for i in range(120):
        times.append(i / 20)

    tries = []
    for i in range(len(mse)):
        tries.append(i)

    plt.subplot(1, 2, 1)  # the figure has 1 row, 2 columns, and this plot is the first plot.
    plt.title('Login Attempts')
    plt.xlabel('Time')
    plt.ylabel('No. of Attempts')
    plt.xlim([-0.5, 6.5])
    plt.ylim([min(min(data), min(data) - 0.5), max(max(data), max(data) + 0.5)])
```

```
    plt.plot(times, data, 'bo', label='Data')
    plt.plot(times, model, '--r', label='Model')
    plt.legend(loc=2)

    plt.subplot(1, 2, 2)  # the figure has 1 row, 2 columns, and this plot is the second plot.
    plt.title('Model Tuning')
    plt.xlabel('Iterations')
    plt.ylabel('MSE')
    plt.xlim([-1, len(mse) + 1])
    plt.ylim([-1, max(mse) + 1])
    plt.plot(tries, mse, '-b')

    plt.show()


if __name__ == '__main__':
    data = getInputs(DATA_FILENAME)
    mse_data = setParameter(data)

    # Plot the data
    plot_data(data, mse_data)
```

Part C Sample Output

Sample 1:

```
The value of A is: 0.75
The value of B is: 0.15
The value of C is: 0.6
The value of mu is: 0.65
The value of shift is: 0.7
The MSE predicted by our model is: 0.096
Number of iterations:  313
```

SFWRTECH 3PR3 (Assignment 4)