**ENG TECH 3PR3**

Procedural and Object-Oriented Programming Concepts



Assignment 4

Submitted by: Qin Yang 400420584

Date: Feb 24th , 2022

**Part. A Modelling Login Attempts**

1. **Code**

import math

import matplotlib

matplotlib.use('TkAgg')

import matplotlib.pyplot as plt

import numpy as np

def get\_inputs(): # getting file and insert value to an array

file01 = open("/Users/qinyang/Desktop/A04\_sfwr\_data\_01.txt", "r")

s\_file = []

s\_file.extend(file01.readlines())

s\_file = map(float, s\_file)

return s\_file

def setparameters(): # set up parameters

for a in np.arange(0, 2.01):

a = format(a, ".2f")

for b in np.arange(0, 2.01):

b = format(b, ".2f")

for mu in np.arange(0, 2.01):

mu = format(mu, ".2f")

return mu, a, b

def getfit(): # conduct brute force search to minimize mse to less than 1.0

t\_total = 120

mse\_trend = []

mse\_better = 1000

s\_file = get\_inputs()

i = 0 # define how many iterations we compared mse

for a in np.arange(0, 2.01, 0.05):

for b in np.arange(0, 2.01, 0.05):

for mu in np.arange(0, 2.01, 0.05):

if mse\_better < 1:

break

else:

mse\_result = evaluatemode(a, b, mu, t\_total, s\_file)

if mse\_result < mse\_better:

i = i + 1

mse\_better = mse\_result

parameters\_mse\_min = [a, b, mu]

mse\_trend.append(mse\_better)

return mse\_better, parameters\_mse\_min, i, mse\_trend

def evaluatemode(a, b, mu, t\_total, s\_file): # calculate mse

n = t\_total

s\_calculated = []

for t in range(1, 121):

s = (-1) \* (a \* math.sin(2 \* math.pi \* t / t\_total) + mu) \* math.e \*\* (b \* 2 \* math.pi \* t / t\_total)

s\_calculated.append(s)

s\_subtract = np.subtract(s\_calculated, s\_file)

s\_square = np.square(s\_subtract)

s\_sum = np.sum(s\_square)

mse = s\_sum / n

return mse

s\_file = get\_inputs()

setparameters()

mse\_better, parameters\_mse\_min, i, mse\_trend = getfit()

s\_calculated = []

a = parameters\_mse\_min[0]

b = parameters\_mse\_min[1]

mu = parameters\_mse\_min[2]

T = 120

for t in range(1, 121):

s = (-1) \* (a \* math.sin(2 \* math.pi \* t / T) + mu) \* math.e \*\* (b \* 2 \* math.pi \* t / T)

print(s)

s\_calculated.append(s)

print('The Value of A is: ', parameters\_mse\_min[0])

print('The Value of B is: ', parameters\_mse\_min[1])

print('The Value of mu is: ', parameters\_mse\_min[2])

print('The MSE predicted by pur model is: ', format(mse\_better, ".3f"))

t\_x = np.arange(1, 121, 1)

plt.scatter(t\_x, s\_file)

plt.plot(t\_x, s\_calculated)

plt.xlabel('Time [s]')

plt.ylabel('No.of attempts')

plt.show()

i\_x = np.arange(0, i, 1)

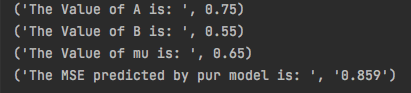
plt.plot(i\_x, mse\_trend)

plt.xlabel('iterations')

plt.ylabel('MSE')

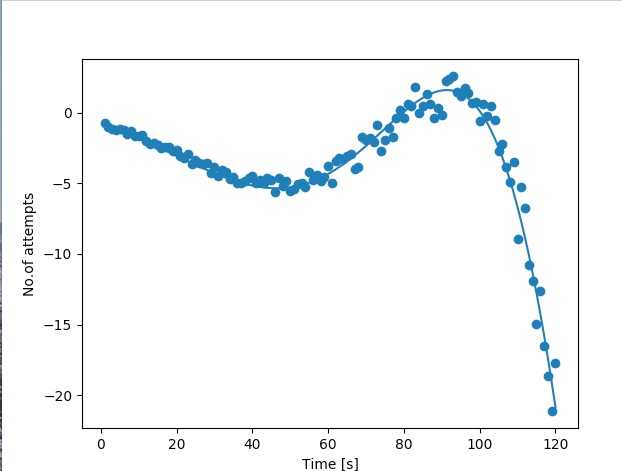
plt.show()

1. **Test Result**
2. Screenshot of print output

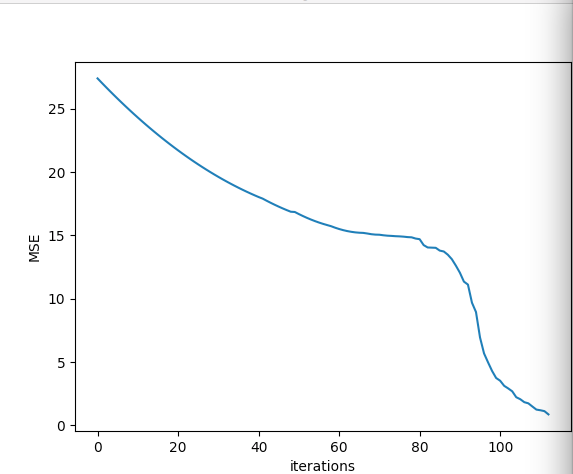


1. Screenshot of plot output

No. Attempts



MSE Trend:



**Part. B Some Variation is observed**

1. **Code**

import math

import matplotlib

matplotlib.use('TkAgg')

import matplotlib.pyplot as plt

import numpy as np

def get\_inputs(): # getting file and insert value to an array

file01 = open("/Users/qinyang/Desktop/A04\_sfwr\_data\_03.txt", "r")

s\_file = []

s\_file.extend(file01.readlines())

s\_file = map(float, s\_file)

return s\_file

def setparameters(): # set up parameters

for a in np.arange(0, 2.01):

a = format(a, ".2f")

for b in np.arange(0, 2.01):

b = format(b, ".2f")

for c in np.arange(0.01, 2.01):

c = format(c, ".2f")

for mu in np.arange(0, 2.01):

mu = format(mu, ".2f")

return mu, a, b, c

def getfit(): # conduct brute force search to minimize mse to less than 1.0

t\_total = 120

mse\_trend = []

mse\_better = 1000

s\_file = get\_inputs()

i = 0 # define how many iterations we compared mse

for a in np.arange(0, 2.01, 0.05):

for b in np.arange(0, 2.01, 0.05):

for mu in np.arange(0, 2.01, 0.05):

for c in np.arange(0.05, 2.01, 0.05):

if mse\_better < 0.5:

break

else:

mse\_result = evaluatemode(a, b, c, mu, t\_total, s\_file)

if mse\_result < mse\_better:

i = i + 1

mse\_better = mse\_result

parameters\_mse\_min = [a, b, c, mu]

mse\_trend.append(mse\_better)

return mse\_better, parameters\_mse\_min, i, mse\_trend

def evaluatemode(a, b, c, mu, t\_total, s\_file): # calculate mse

n = t\_total

s\_calculated = []

for t in range(1, 121):

s = (-1) \* (a \* math.sin(2 \* math.pi \* t / t\_total) + mu) \* math.e \*\* (b \* 2 \* math.pi \* t / (t\_total \* c))

s\_calculated.append(s)

s\_subtract = np.subtract(s\_calculated, s\_file)

s\_square = np.square(s\_subtract)

s\_sum = np.sum(s\_square)

mse = s\_sum / n

return mse

s\_file = get\_inputs()

setparameters()

mse\_better, parameters\_mse\_min, i, mse\_trend = getfit()

s\_calculated = []

a = parameters\_mse\_min[0]

b = parameters\_mse\_min[1]

c = parameters\_mse\_min[2]

mu = parameters\_mse\_min[3]

T = 120

for t in range(1, 121):

s = (-1) \* (a \* math.sin(2 \* math.pi \* t / T) + mu) \* math.e \*\* (b \* 2 \* math.pi \* t / T)

s\_calculated.append(s)

print('The Value of A is: ', format(parameters\_mse\_min[0], ".3f"))

print('The Value of B is: ', format(parameters\_mse\_min[1], ".3f"))

print('The Value of C is: ', format(parameters\_mse\_min[2], ".3f"))

print('The Value of mu is: ', format(parameters\_mse\_min[3], ".3f"))

print('The MSE predicted by pur model is: ', format(mse\_better, ".3f"))

t\_x = np.arange(1, 121, 1)

plt.scatter(t\_x, s\_file)

plt.plot(t\_x, s\_calculated)

plt.xlabel('Time [s]')

plt.ylabel('No.of attempts')

plt.show()

i\_x = np.arange(0, i, 1)

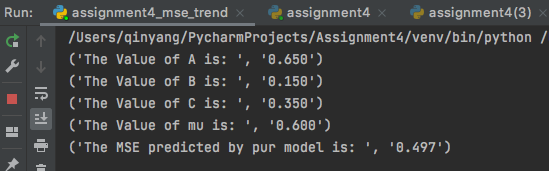
plt.plot(i\_x, mse\_trend)

plt.xlabel('iterations')

plt.ylabel('MSE')

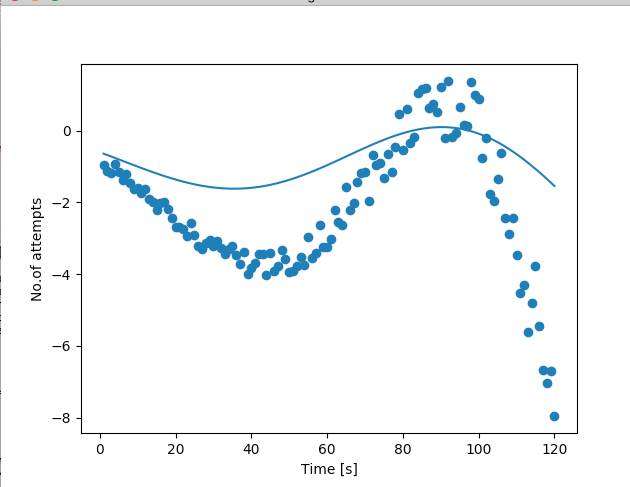
plt.show()

1. **Test Result**
2. Screenshot of print output

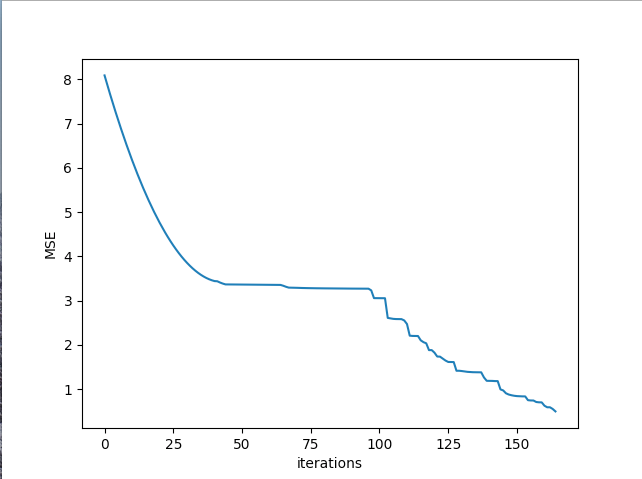


1. Screenshot of plot output

No. Attempts



MSE Trend:



**Part. C Towards a generic model**

1. **Code**

import math

import matplotlib

matplotlib.use('TkAgg')

import matplotlib.pyplot as plt

import numpy as np

def get\_inputs(): # getting file and insert value to an array

file01 = open("/Users/qinyang/Desktop/A04\_sfwr\_data\_05.txt", "r")

s\_file = []

s\_file.extend(file01.readlines())

s\_file = map(float, s\_file)

return s\_file

def setparameters(): # set up parameters

for a in np.arange(0, 2.01):

a = format(a, ".2f")

for b in np.arange(0, 2.01):

b = format(b, ".2f")

for c in np.arange(0.01, 2.01):

c = format(c, ".2f")

for mu in np.arange(0, 2.01):

mu = format(mu, ".2f")

for shift in np.arange(0, 1.51):

shift = format(shift, ".2f")

return mu, a, b, c, shift

def getfit(): # conduct brute force search to minimize mse to less than 1.0

t\_total = 120

mse\_trend = []

mse\_better = 1000

s\_file = get\_inputs()

i = 0 # define how many iterations we compared mse

for a in np.arange(0, 2.01, 0.05):

for b in np.arange(0, 2.01, 0.05):

for mu in np.arange(0, 2.01, 0.05):

for c in np.arange(0.05, 2.01, 0.05):

for shift in np.arange(0.05, 1.51, 0.05):

print(a, b, c, mu, mse\_better)

if mse\_better < 0.1:

break

else:

mse\_result = evaluatemode(a, b, c, mu, t\_total, s\_file, shift)

if mse\_result < mse\_better:

i = i + 1

mse\_better = mse\_result

parameters\_mse\_min = [a, b, c, mu, shift]

mse\_trend.append(mse\_better)

return mse\_better, parameters\_mse\_min, i, mse\_trend

def evaluatemode(a, b, c, mu, t\_total, s\_file, shift): # calculate mse

n = t\_total

s\_calculated = []

for t in range(1, 121):

s1 = (-1) \* ((a \* math.sin(2 \* math.pi \* t + shift) / t\_total) + mu)

s2 = math.e \*\* (b \* ((2 \* math.pi \* t / t\_total) + shift / c))

s = s1 \* s2

s\_calculated.append(s)

s\_subtract = np.subtract(s\_calculated, s\_file)

s\_square = np.square(s\_subtract)

s\_sum = np.sum(s\_square)

mse = s\_sum / n

return mse

s\_file = get\_inputs()

setparameters()

mse\_better, parameters\_mse\_min, i, mse\_trend = getfit()

s\_calculated = []

a = parameters\_mse\_min[0]

b = parameters\_mse\_min[1]

c = parameters\_mse\_min[2]

mu = parameters\_mse\_min[3]

shift = parameters\_mse\_min[4]

T = 120

for t in range(1, 121):

s1 = (-1) \* ((a \* math.sin(2 \* math.pi \* t + shift) / T ) + mu)

s2 = math.e \*\* (b \* ((2 \* math.pi \* t / T) + shift / c))

s = s1 \* s2

s\_calculated.append(s)

print('The Value of A is: ', format(parameters\_mse\_min[0], ".3f"))

print('The Value of B is: ', format(parameters\_mse\_min[1], ".3f"))

print('The Value of C is: ', format(parameters\_mse\_min[2], ".3f"))

print('The Value of mu is: ', format(parameters\_mse\_min[3], ".3f"))

print('The Value of shift is: ', parameters\_mse\_min[4])

print('The MSE predicted by pur model is: ', format(mse\_better, ".3f"))

t\_x = np.arange(1, 121, 1)

plt.scatter(t\_x, s\_file)

plt.plot(t\_x, s\_calculated)

plt.xlabel('Time [s]')

plt.ylabel('No.of attempts')

plt.show()

i\_x = np.arange(0, i, 1)

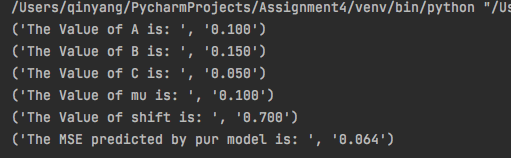
plt.plot(i\_x, mse\_trend)

plt.xlabel('iterations')

plt.ylabel('MSE')

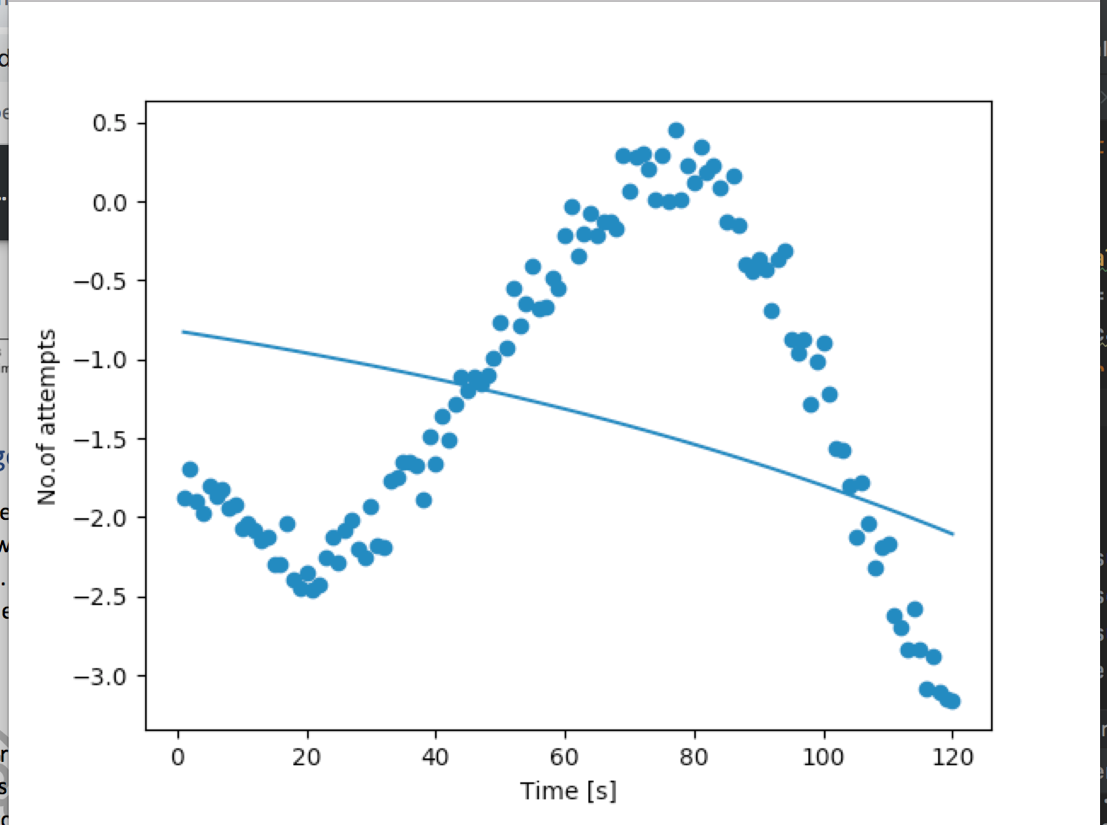
plt.show()

1. **Test Result**
2. Screenshot of print output



1. Screenshot of plot output

No Attempts:



MSE:

