Learning and Memory Assignment-2

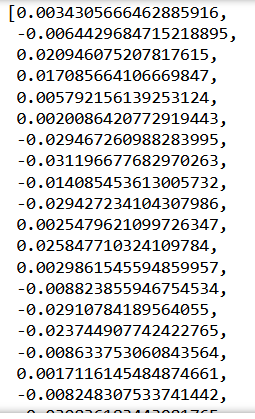
Average data per day 1 is 0.129938212695416 sec

Average data per day 2 is 0.1518654402902913 sec

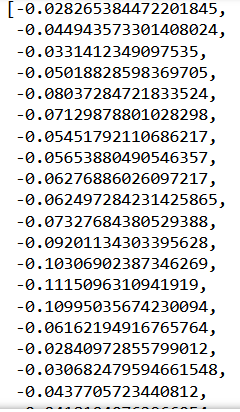
Average data per day 3 is 0.1704535584009131 sec

Average data per day 4 is 0.226222335245639 sec

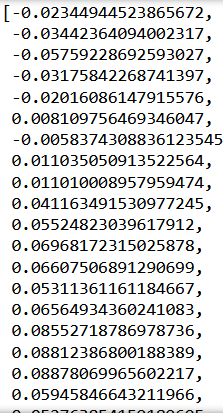
For Day1



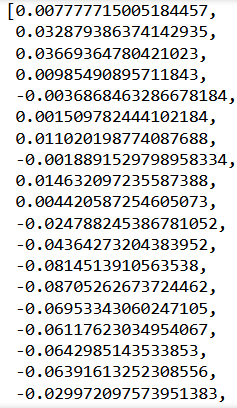
For Day 2



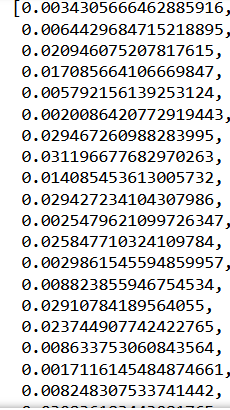
Day 3



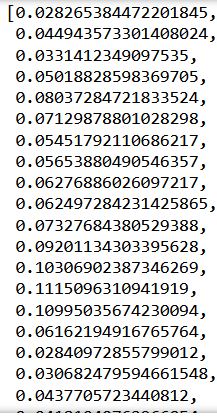
Day 4



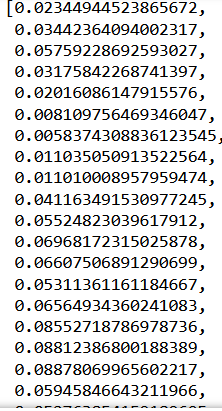
1. For day 1



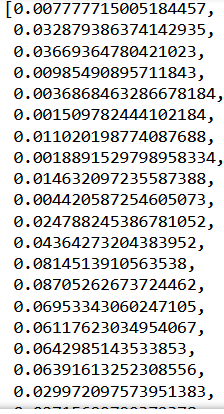
Day 2



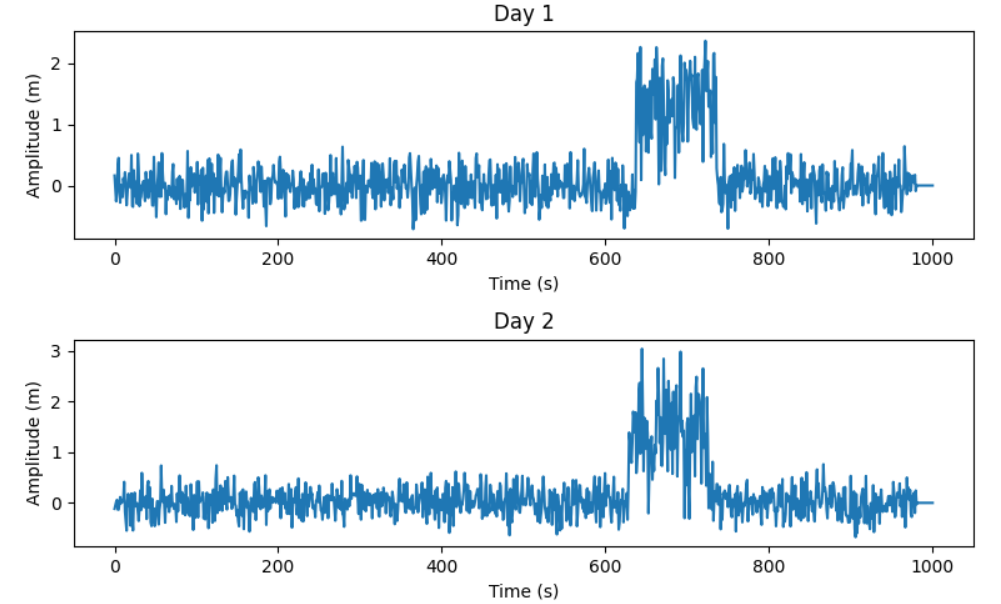
Day 3

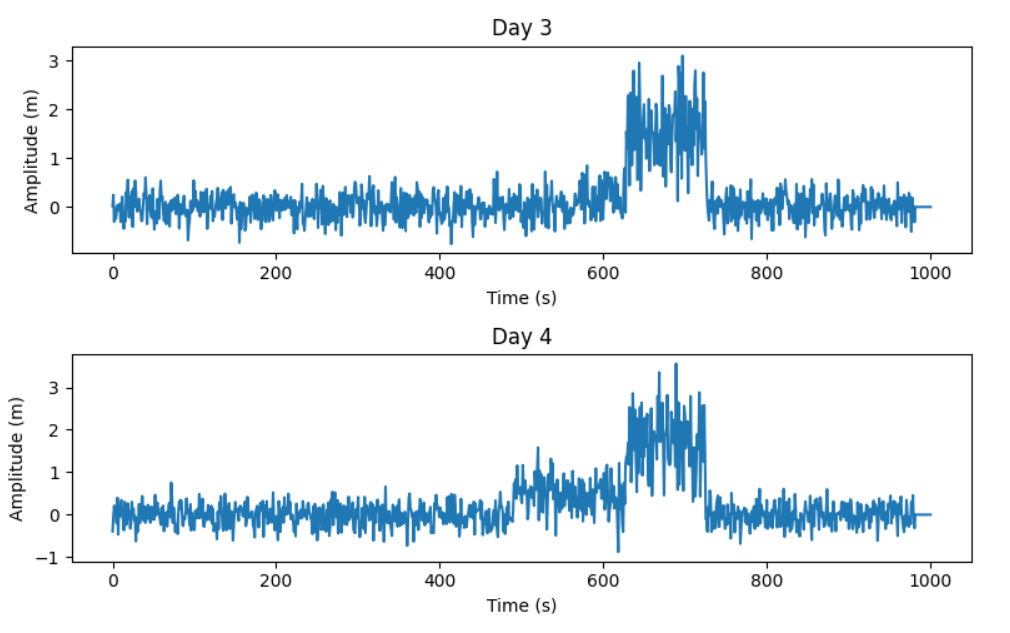


Day 4

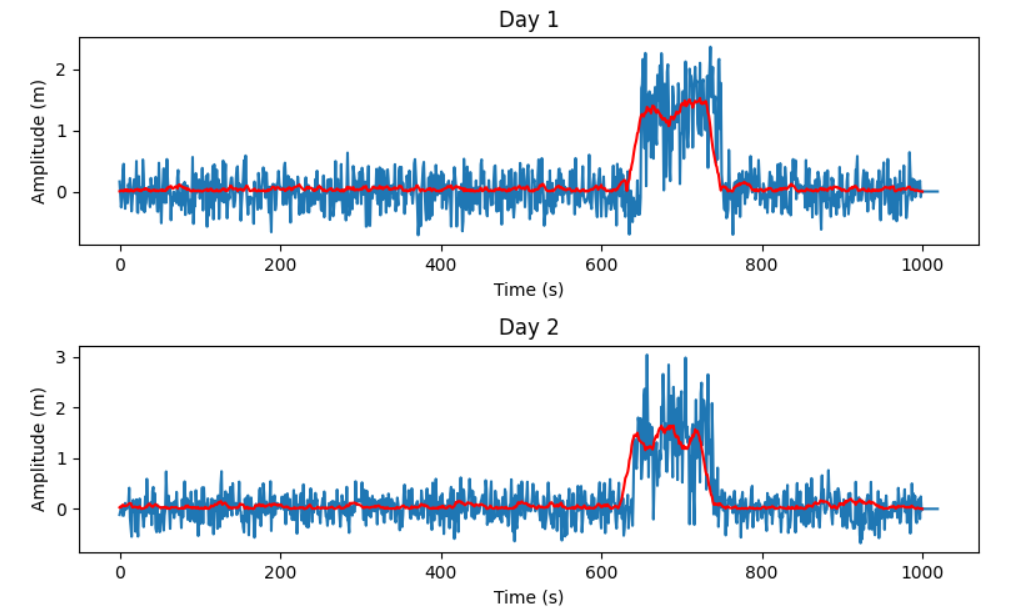


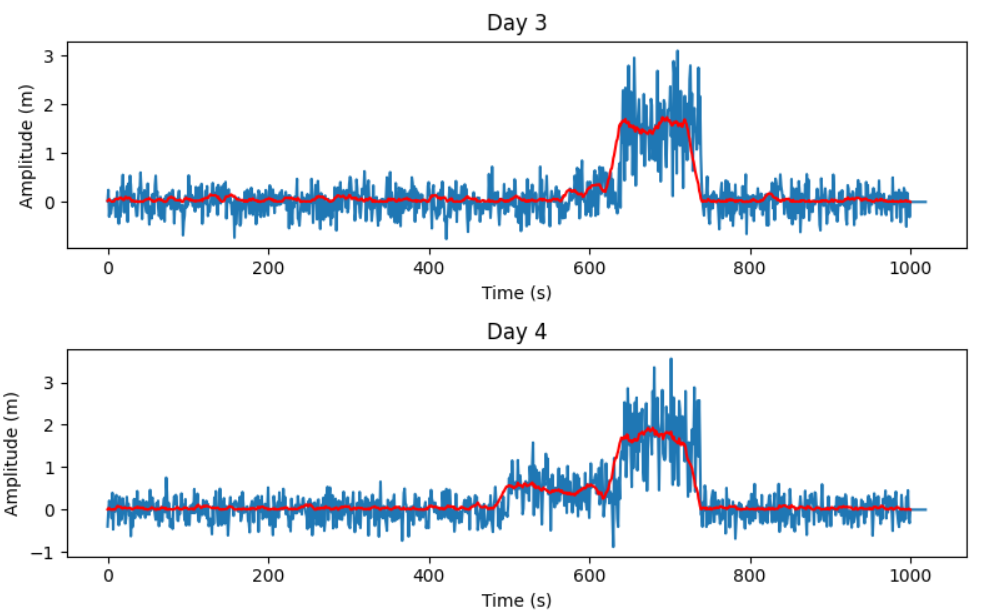












Code:

The learning mechanism involved is Conditioned Stimulus which makes a response with an UnConditioned Stimulus.

The necessary another thing will be involved will be:

1. Fast Recovery
2. Elimination
3. Make that these stimuli don’t correspond to other predicted stimuli.

Peaks:

Day 1:

736

Day 2

657

Day 3

710

Day 4

702

Code:

#!/usr/bin/env python

# coding: utf-8

# In[14]:

import pandas as pd

import numpy as np

from scipy.signal import find\_peaks

# In[15]:

read = pd.read\_excel('Data-Assignment2A.xlsx')

# In[16]:

read

# In[17]:

col1 = read.loc[0:,"Unnamed: 1"]

col1

# In[18]:

s = 'Unnamed: '

w = str(1)

s + w

new= s+ w

read.loc[0:,new]

# In[19]:

newArrayDay1.clear()

# In[20]:

newArrayDay1 = []

for i in range(1,1001):

    new = 'Unnamed: ' + str(i)

    newArrayDay1.append(read.loc[0:,new].mean())

newArrayDay1

m1 = sum(newArrayDay1)/len(newArrayDay1)

print("Thus average data per day 1 is {} sec".format(m1))

# In[23]:

maxIndex = newArrayDay1.index(max(newArrayDay1))

maxIndex

# In[10]:

# In[24]:

read2 = pd.read\_excel('Data-Assignment2A.xlsx',sheet\_name=1)

read2

# In[25]:

newArrayDay2 = []

for i in range(1,1001):

    new = 'Unnamed: ' + str(i)

    newArrayDay2.append(read2.loc[0:,new].mean())

newArrayDay2

m2 = sum(newArrayDay2)/len(newArrayDay2)

m2

print("Thus average data per day 2 is {} sec".format(m2))

# In[26]:

maxIndex = newArrayDay2.index(max(newArrayDay2))

maxIndex

# In[27]:

read3 = pd.read\_excel('Data-Assignment2A.xlsx',sheet\_name=2)

newArrayDay3 = []

for i in range(1,1001):

    new = 'Unnamed: ' + str(i)

    newArrayDay3.append(read3.loc[0:,new].mean())

newArrayDay3

m3 = sum(newArrayDay3)/len(newArrayDay3)

m3

print("Thus average data per day 3 is {} sec".format(m3))

# In[28]:

maxIndex = newArrayDay3.index(max(newArrayDay3))

maxIndex

# In[29]:

read4 = pd.read\_excel('Data-Assignment2A.xlsx',sheet\_name=3)

newArrayDay4 = []

for i in range(1,1001):

    new = 'Unnamed: ' + str(i)

    newArrayDay4.append(read4.loc[0:,new].mean())

newArrayDay4

m4 = sum(newArrayDay4)/len(newArrayDay4)

print("Thus average data per day 4 is {} sec".format(m4))

# In[30]:

maxIndex = newArrayDay4.index(max(newArrayDay4))

maxIndex

# In[13]:

# In[14]:

# Example usage

data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

window\_size = 3

moving\_average\_data = moving\_average(newArrayDay1, 20)

print(len(moving\_average\_data))

# In[15]:

len(newArrayDay1)

# Moving average filter&#39; across the averaged signal with a window width of 20 ms to get a

# filtered signal. Ensure that the raw and filtered signal are of the same length.

# In[16]:

for i in range(20):

    newArrayDay1.append(0)

# In[17]:

AverageFilterDay1 = []

last = 20

for i in range(0,1001):

    m1 = sum(newArrayDay1[i:last])/len(newArrayDay1[i:last])

    AverageFilterDay1.append(m1)

    last+=1

AverageFilterDay1

# In[ ]:

# In[19]:

for i in range(20):

    newArrayDay2.append(0)

AverageFilterDay2 = []

last = 20

for i in range(1001):

    m1 = sum(newArrayDay2[i:last])/len(newArrayDay2[i:last])

    AverageFilterDay2.append(m1)

    last+=1

AverageFilterDay2

# In[20]:

for i in range(20):

    newArrayDay3.append(0)

AverageFilterDay3 = []

last = 20

for i in range(1001):

    m1 = sum(newArrayDay3[i:last])/len(newArrayDay3[i:last])

    AverageFilterDay3.append(m1)

    last+=1

AverageFilterDay3

# In[21]:

for i in range(20):

    newArrayDay4.append(0)

AverageFilterDay4 = []

last = 20

for i in range(1001):

    m1 = sum(newArrayDay4[i:last])/len(newArrayDay4[i:last])

    AverageFilterDay4.append(m1)

    last+=1

AverageFilterDay4

# In[21]:

AverageFilterDay1

# Do a full wave rectification of the above moving average filtered signal.

# In[22]:

rectifiedDay1.clear()

# In[22]:

rectifiedDay1=[]

for i in range(len(AverageFilterDay1)):

    rectifiedDay1.append(np.abs(AverageFilterDay1[i]))

rectifiedDay1

# In[24]:

rectifiedDay2.clear()

# In[23]:

rectifiedDay2=[]

for i in range(len(AverageFilterDay2)):

    rectifiedDay2.append(np.abs(AverageFilterDay2[i]))

rectifiedDay2

# In[24]:

rectifiedDay3=[]

for i in range(len(AverageFilterDay3)):

    rectifiedDay3.append(np.abs(AverageFilterDay3[i]))

rectifiedDay3

# In[27]:

rectifiedDay4.clear()

# In[25]:

rectifiedDay4=[]

for i in range(len(AverageFilterDay4)):

    rectifiedDay4.append(np.abs(AverageFilterDay4[i]))

rectifiedDay4

# In[29]:

amplitude average ka hai

# In[26]:

import matplotlib.pyplot as plt

# In[32]:

time1 = np.linspace(0,1000,1020)

fig, axs = plt.subplots(nrows=4, ncols=1, figsize=(8, 10))

axs[0].plot(time1,newArrayDay1)

time2 = np.linspace(0,1000,1020)

axs[1].plot(time2,newArrayDay2)

time3 = np.linspace(0,1000,1020)

axs[2].plot(time3,newArrayDay3)

axs[3].plot(time3,newArrayDay4)

axs[0].set\_xlabel("Time (s)")

axs[1].set\_xlabel("Time (s)")

axs[2].set\_xlabel("Time (s)")

axs[3].set\_xlabel("Time (s)")

axs[0].set\_ylabel("Amplitude (m)")

axs[1].set\_ylabel("Amplitude (m)")

axs[2].set\_ylabel("Amplitude (m)")

axs[3].set\_ylabel("Amplitude (m)")

axs[0].set\_title("Day 1")

axs[1].set\_title("Day 2")

axs[2].set\_title("Day 3")

axs[3].set\_title("Day 4")

fig.tight\_layout()

# Display the plot

plt.show()

# Plot the amplitude vs time of the filtered and rectified signal (as red curve) – one signal for each

# day on top of the raw signal in the same subplots.

# In[ ]:

newArrayDay1.append(0)

# In[31]:

#filered Signal

rectifiedDay1

time1 = np.linspace(0,1000,1001)

fig, axs = plt.subplots(nrows=4, ncols=1, figsize=(8, 10))

axs[0].plot(newArrayDay1)

axs[0].plot(rectifiedDay1,color='red')

axs[1].plot(newArrayDay2)

axs[1].plot(rectifiedDay2,color='red')

axs[2].plot(newArrayDay3)

axs[2].plot(rectifiedDay3,color='red')

axs[3].plot(newArrayDay4)

axs[3].plot(rectifiedDay4,color='red')

axs[0].set\_xlabel("Time (s)")

axs[1].set\_xlabel("Time (s)")

axs[2].set\_xlabel("Time (s)")

axs[3].set\_xlabel("Time (s)")

axs[0].set\_ylabel("Amplitude (m)")

axs[1].set\_ylabel("Amplitude (m)")

axs[2].set\_ylabel("Amplitude (m)")

axs[3].set\_ylabel("Amplitude (m)")

axs[0].set\_title("Day 1")

axs[1].set\_title("Day 2")

axs[2].set\_title("Day 3")

axs[3].set\_title("Day 4")

fig.tight\_layout()

# Display the plot

plt.show()

# In[ ]:

# In[ ]:

# In[ ]:

2.

1. Exp1 :

Asymptote learning for experiment 1 is 0.7913065284884426

Learning rate for experiment 1 is 0.1399442931489643

R sqared of experiment 1 is 0.9955534656756603

Exp2:

Asymptote learning for experiment 2 is 0.9493844910080161

Asymptote for experiment 2 is 0.6504204783621171

R sqared of experiment 2 is 0.9998285356242271

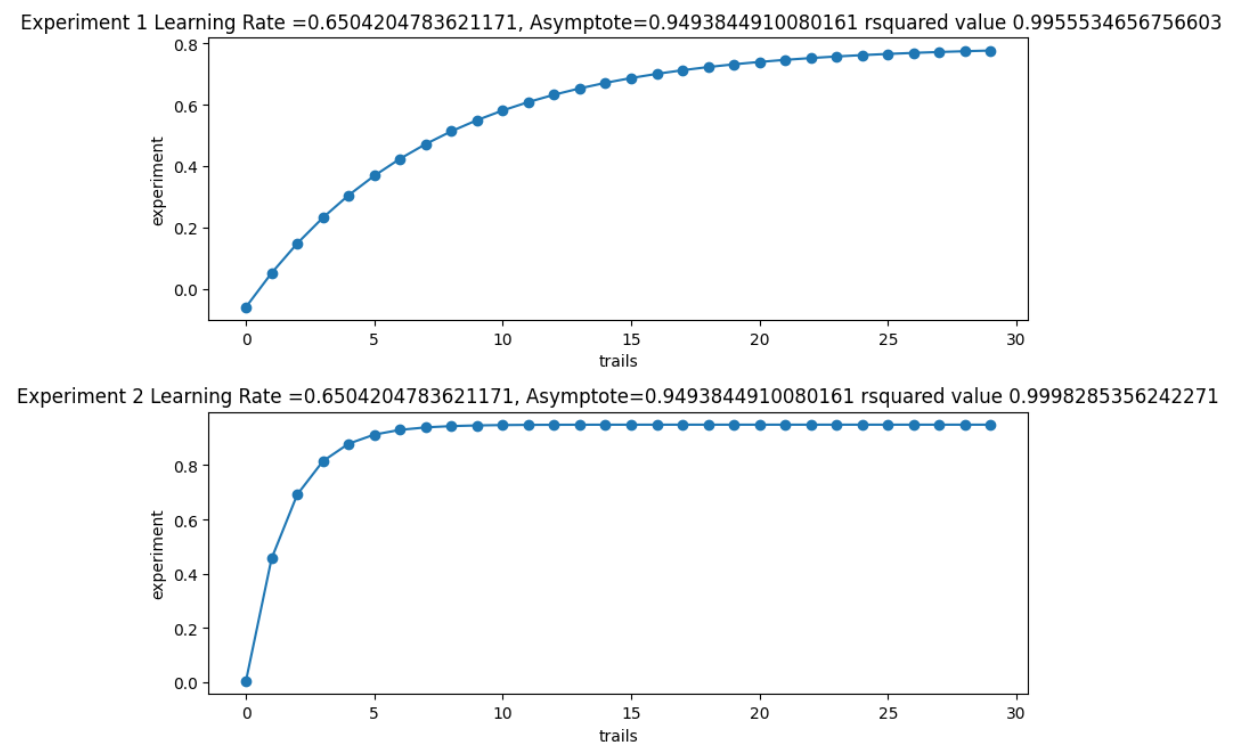
Exp3:

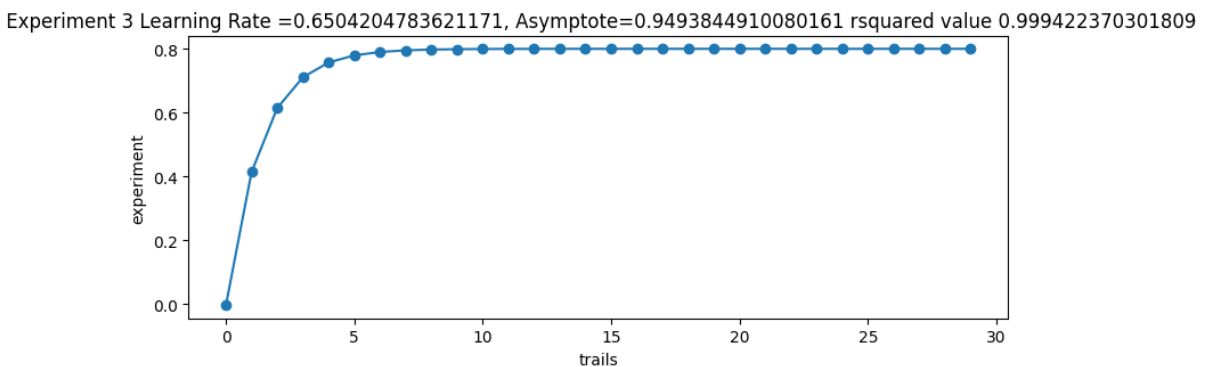
Asymptote learning for experiment 3 is 0.8008236290427434

Learning rate for experiment 3 is 0.7334091931092159

R sqared of experiment 3 is 0.999422370301809







Code:

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import pandas as pd

import numpy as np

from lmfit import Model as md

from scipy.optimize import curve\_fit as cf

import matplotlib.pyplot as plt

from sklearn.model\_selection import learning\_curve

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

# In[2]:

read = pd.read\_excel('Data-Assignment2B.xlsx',header=None)

# In[3]:

read

# In[14]:

# In[4]:

row1 = read.iloc[0]

row2 = read.iloc[1]

row3 = read.iloc[2]

# In[5]:

row1

# In[11]:

def sigmoid(xVAli, aVAli, bVaLi, cVali):

    return aVAli \* np.exp(-bVaLi \* xVAli) + cVali

# In[63]:

x\_val.clear()

# In[64]:

x\_val = []

for i in range(30):

    x\_val.append(i)

x\_val

# In[24]:

# In[71]:

pt1, pv = cf(sigmoid, x\_val, row1[1:])

AsymtoteExp1 = pt[2]

learingrateExp1 = pt[1]

print("Asymptote learning for experiment 1 is {}".format(pt1[2]))

print("Learning rate for experiment 1 is {}".format(pt1[1]))

# In[72]:

pt2, pv = cf(sigmoid, x\_val, row2[1:])

AsymtoteExp2 = pt[2]

learingrateExp2 = pt[1]

print("Asymptote learning for experiment 2 is {}".format(pt2[2]))

print("Asymptote for experiment 2 is {}".format(pt2[1]))

# In[73]:

pt3, pv = cf(sigmoid, x\_val, row3[1:])

AsymtoteExp3 = pt[2]

learingrateExp3 = pt[1]

print("Asymptote learning for experiment 3 is {}".format(pt3[2]))

print("Learning rate for experiment 3 is {}".format(pt3[1]))

# In[51]:

val1 = [sigmoid(x\_val,\*pt1) for x\_val in x\_val]

val2 = [sigmoid(x\_val,\*pt2) for x\_val in x\_val]

val3 = [sigmoid(x\_val,\*pt3) for x\_val in x\_val]

# In[57]:

from sklearn.metrics import r2\_score

sqq = sigmoid(1000,\*pt1)

r1 = r2\_score(row1[1:],val1)

r1

print("R sqared of experiment 1 is {}".format(r1))

# In[58]:

sqq = sigmoid(1000,\*pt2)

r2 = r2\_score(row2[1:],val2)

r2

print("R sqared of experiment 2 is {}".format(r2))

# In[60]:

sqq = sigmoid(1000,\*pt3)

r3 = r2\_score(row3[1:],val3)

r3

print("R sqared of experiment 3 is {}".format(r3))

# In[53]:

# In[54]:

# In[55]:

# Create three subplots for three experiments as part of one larger plot to graph the individual data

# points (as open circle markers; black colour) and overlay of the learning curve (blue colour) on each

# subplot. Indicate the Learning rate and Learning asymptote on top of each subplot (as title).

# In[59]:

# In[98]:

fig, axe = plt.subplots(nrows=3, ncols=1, figsize=(8, 10))

axe[0].scatter(x\_val,val1)

axe[0].plot(x\_val,val1)

axe[0].set\_title(" Experiment 1 Learning Rate ={}, Asymptote={} rsquared value {}".format(learingrateExp1,AsymtoteExp1,r1))

axe[0].set\_xlabel("trails")

axe[0].set\_ylabel("experiment")

axe[1].scatter(x\_val,val2)

axe[1].plot(x\_val,val2)

axe[1].set\_title("Experiment 2 Learning Rate ={}, Asymptote={} rsquared value {}".format(learingrateExp2,AsymtoteExp2,r2))

axe[1].set\_xlabel("trails")

axe[1].set\_ylabel("experiment")

axe[2].scatter(x\_val,val3)

axe[2].plot(x\_val,val3)

axe[2].set\_title(" Experiment 3 Learning Rate ={}, Asymptote={} rsquared value {}".format(learingrateExp3,AsymtoteExp3,r3))

axe[2].set\_xlabel("trails")

axe[2].set\_ylabel("experiment")

plt.tight\_layout()

plt.show()

# In[ ]:

# In[22]:

# In[ ]:

# In[ ]:

For one metric goodness of fit, I can r squared value. Higher the value better is the results

b. That will be strength in which an Unconditioned stimuli provides a good response with the conditioned stimuli. Here at start it generates a strong response then after it becomes constant as in Pavlov's dog experiment. It knows the sound already so it don’t generated strong response.