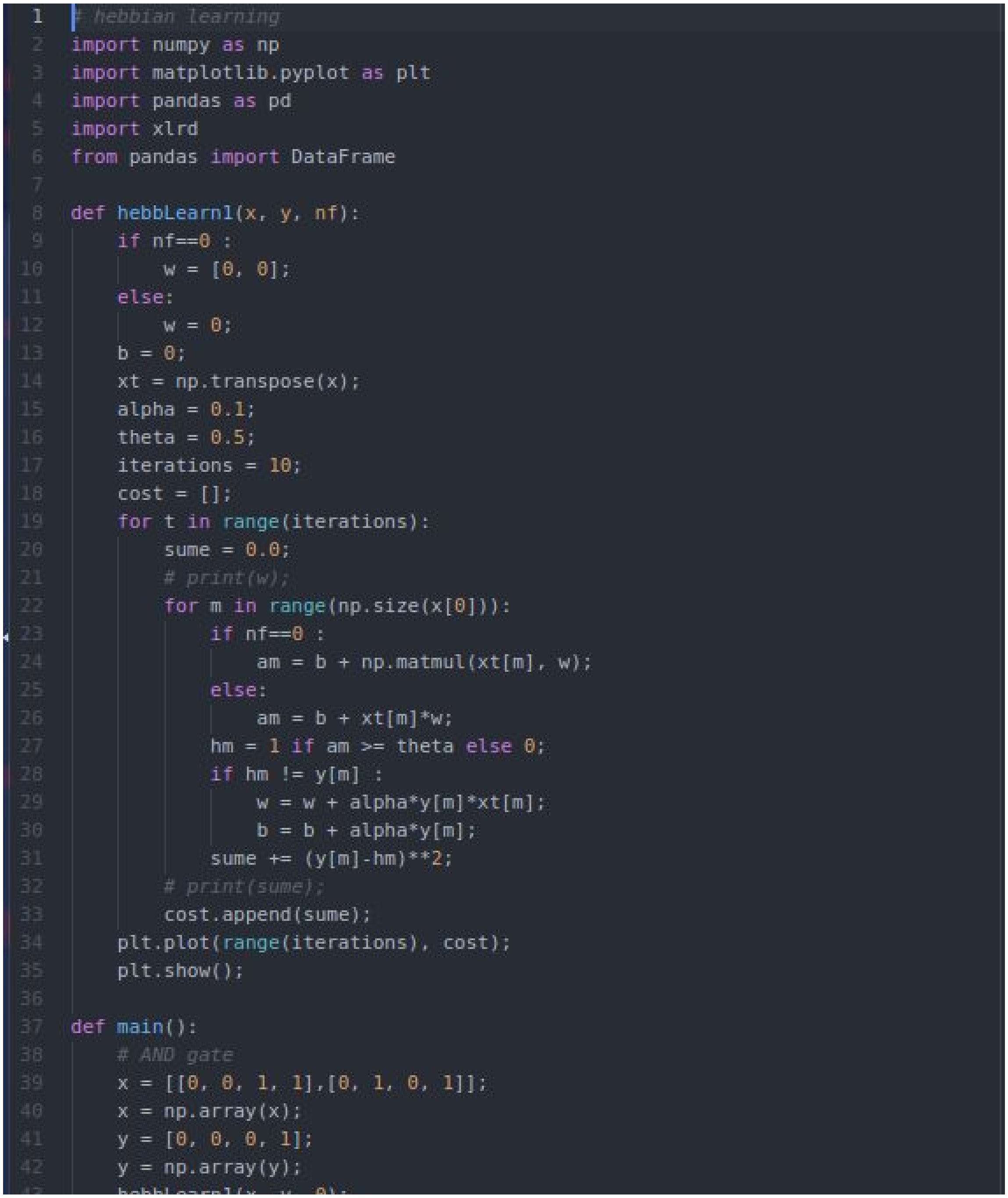
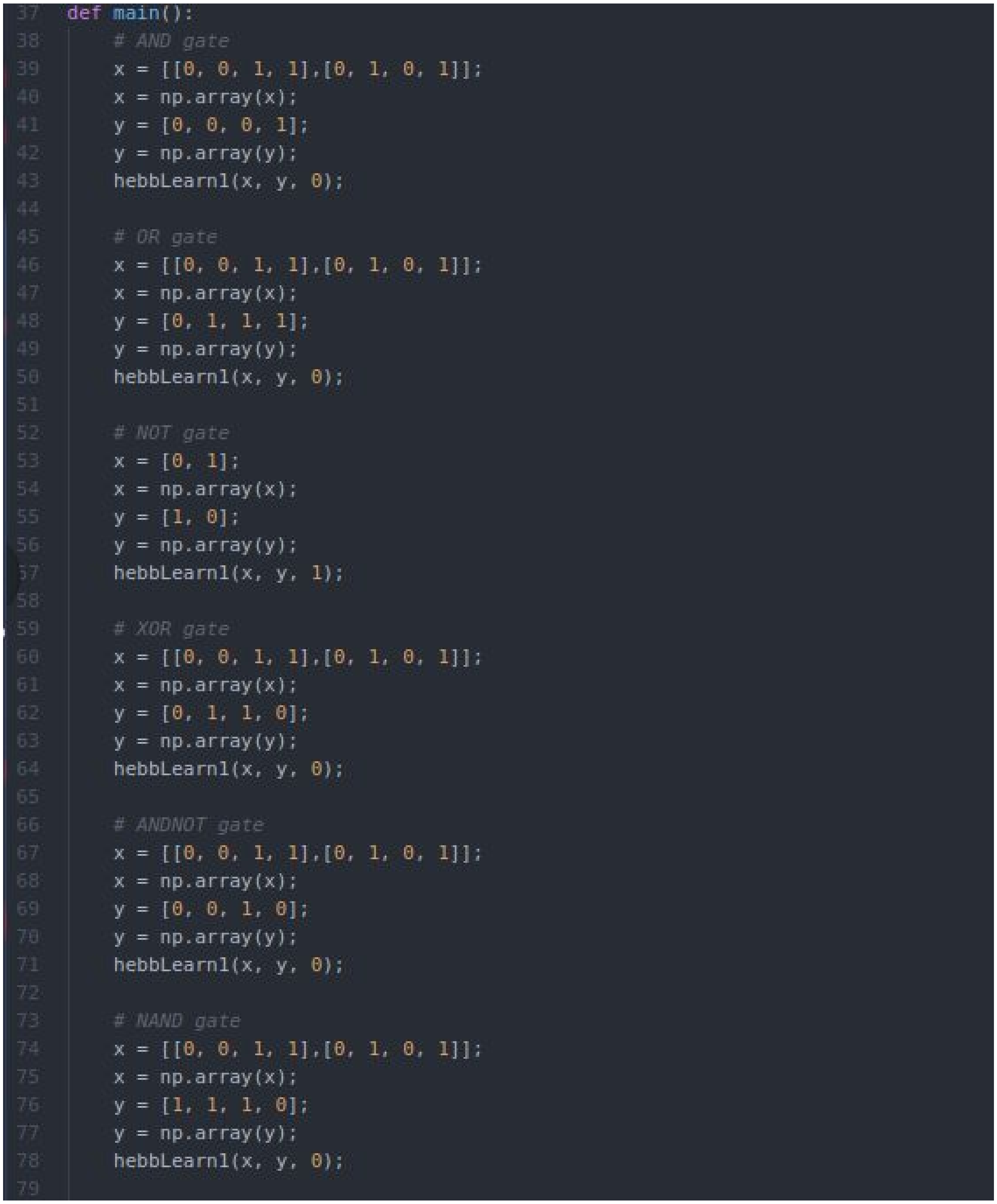
NEURAL NETWORK AND FUZZY LOGIC

ASSIGNMENT-2

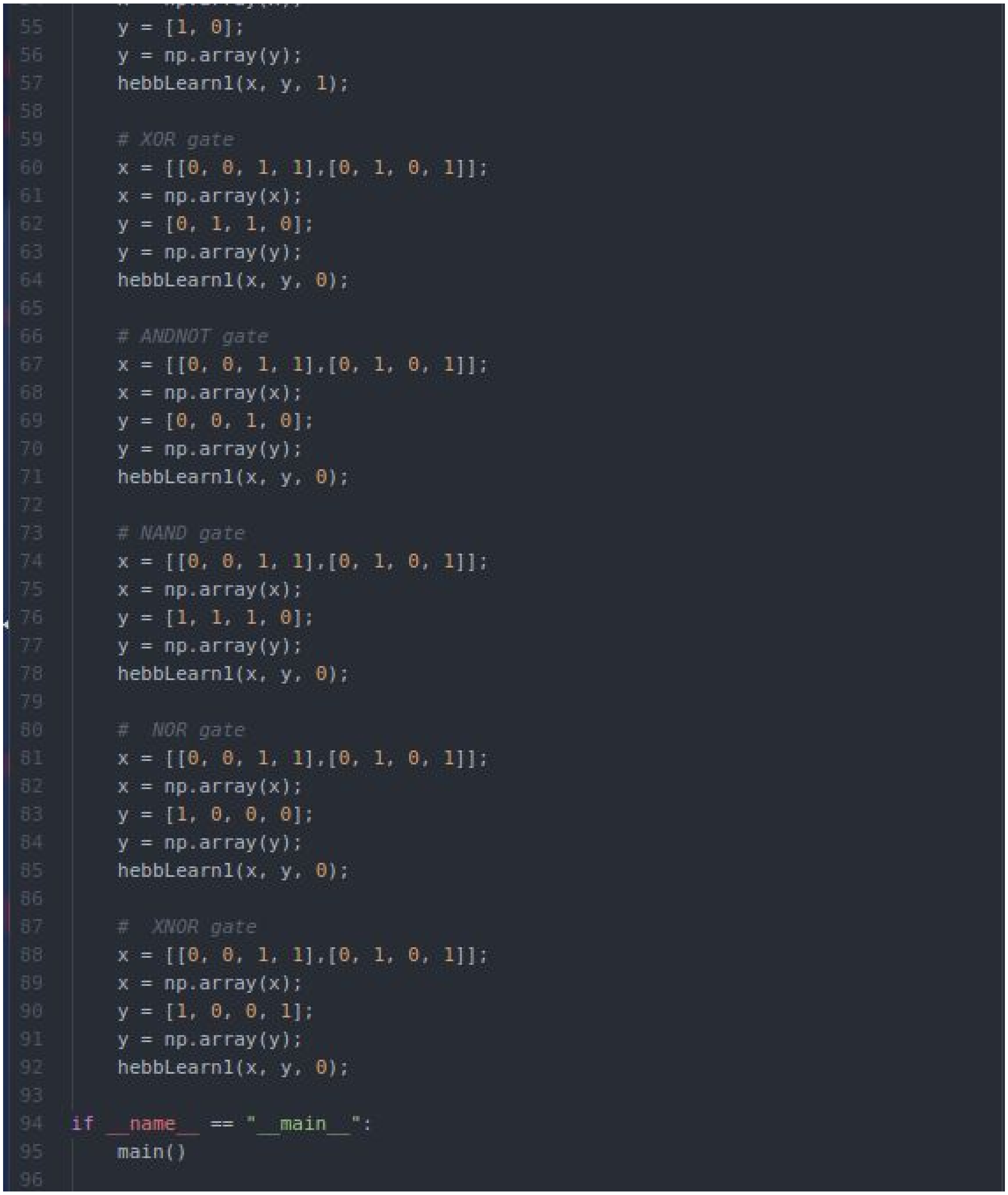
PROBLEM 1



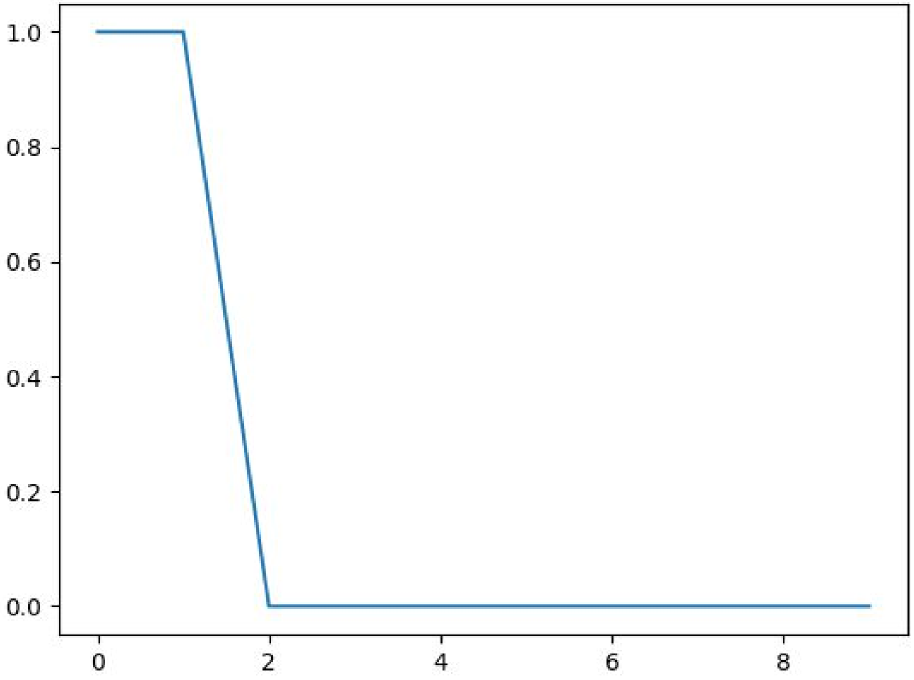
```
# hebbian learning
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame

def hebbLearn1(x, y, nf):
    if nf==0 :
        w = [0, 0];
    else:
        w = 0;
    b = 0;
    xt = np.transpose(x);
    alpha = 0.1;
    theta = 0.5;
    iterations = 10;
    cost = [];
    for t in range(iterations):
        sume = 0.0;
        # print(w);
        for m in range(np.size(x[0])):
            if nf==0 :
                am = b + np.matmul(xt[m], w);
            else:
                am = b + xt[m]*w;
            hm = 1 if am >= theta else 0;
            if hm != y[m] :
                w = w + alpha*y[m]*xt[m];
                b = b + alpha*y[m];
            sume += (y[m]-hm)**2;
        # print(sume);
        cost.append(sume);
    plt.plot(range(iterations), cost);
    plt.show();

def main():
    # AND gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [0, 0, 0, 1];
    y = np.array(y);
    hebbLearn1(x, y, 0);
```

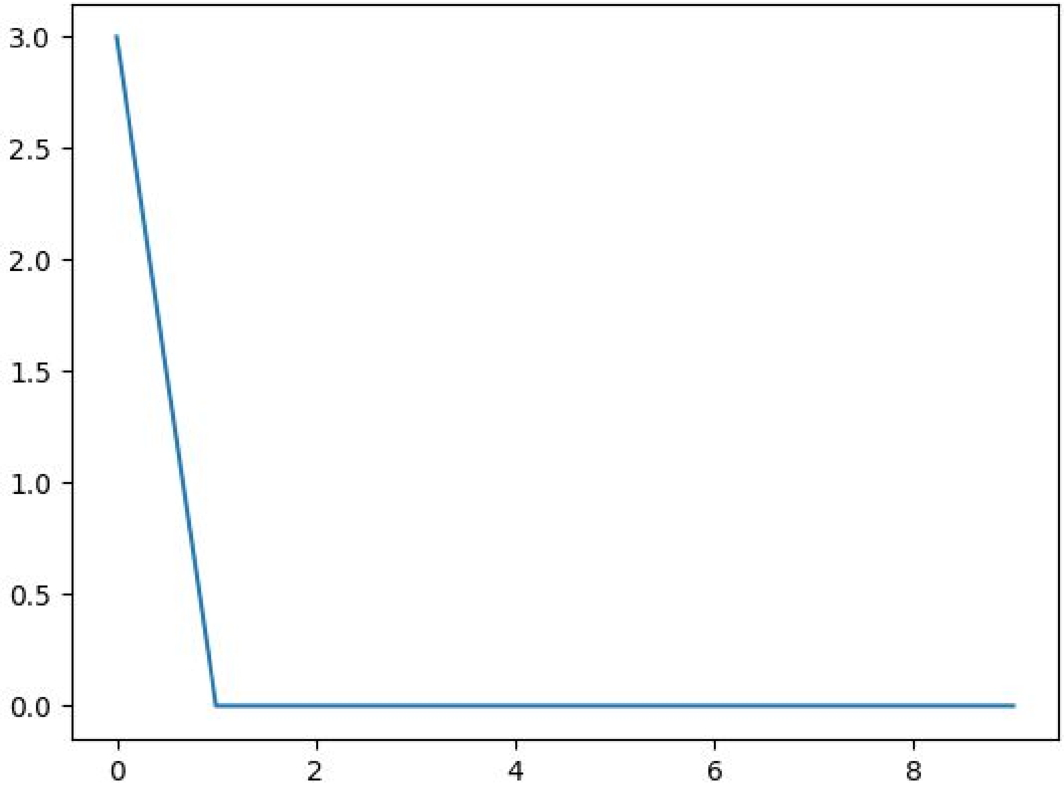


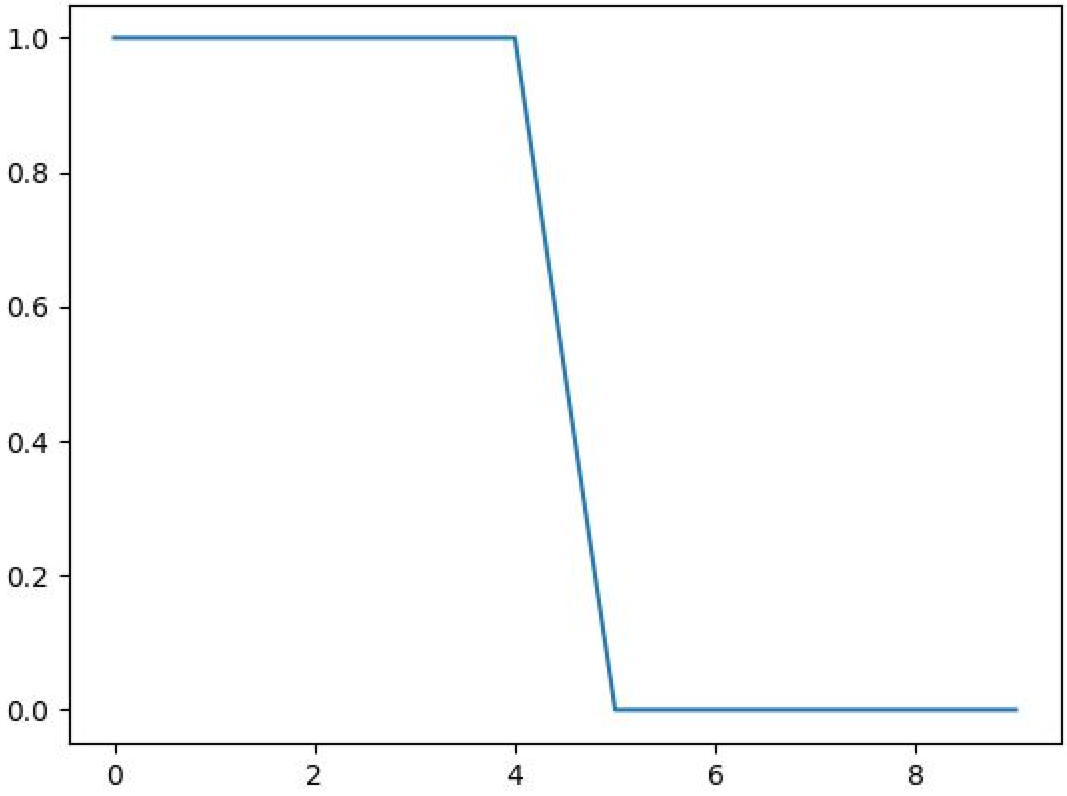
```
def main():
    # AND gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [0, 0, 0, 1];
    y = np.array(y);
    hebbLearn1(x, y, 0);

    # OR gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [0, 1, 1, 1];
    y = np.array(y);
    hebbLearn1(x, y, 0);

    # NOT gate
    x = [0, 1];
    x = np.array(x);
    y = [1, 0];
    y = np.array(y);
    hebbLearn1(x, y, 1);

    # XOR gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [0, 1, 1, 0];
    y = np.array(y);
    hebbLearn1(x, y, 0);

    # ANDNOT gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [0, 0, 1, 0];
    y = np.array(y);
    hebbLearn1(x, y, 0);

    # NAND gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [1, 1, 1, 0];
    y = np.array(y);
    hebbLearn1(x, y, 0);

```

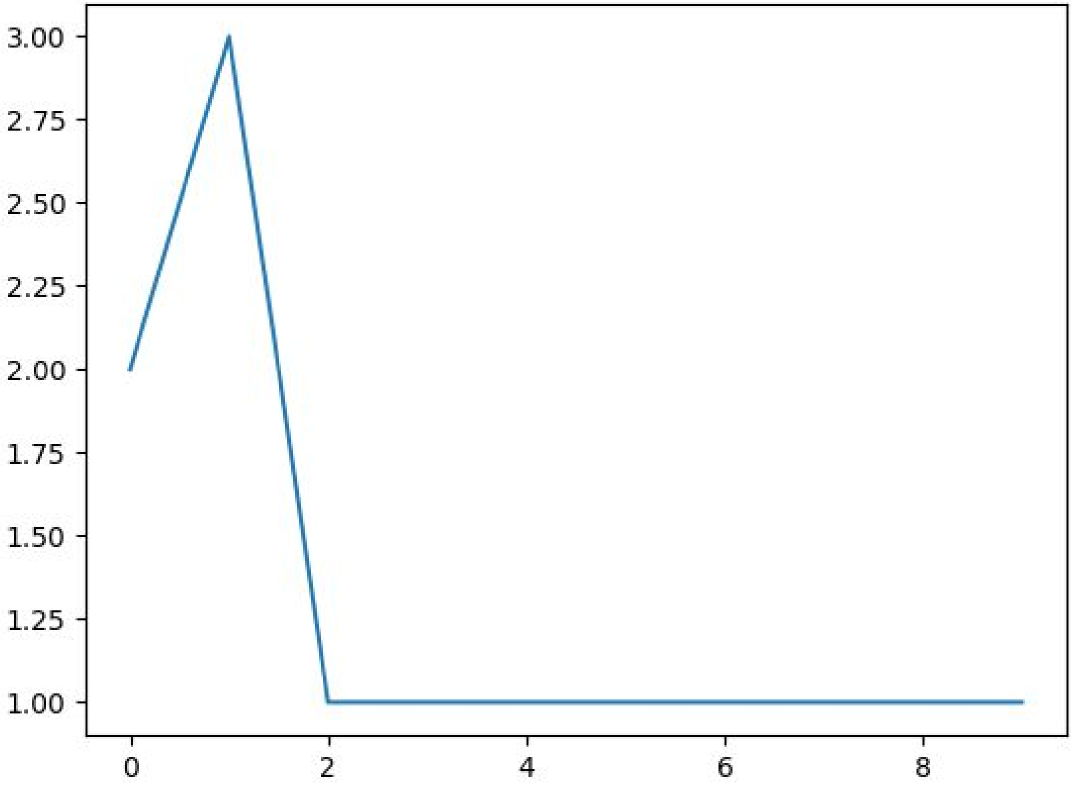


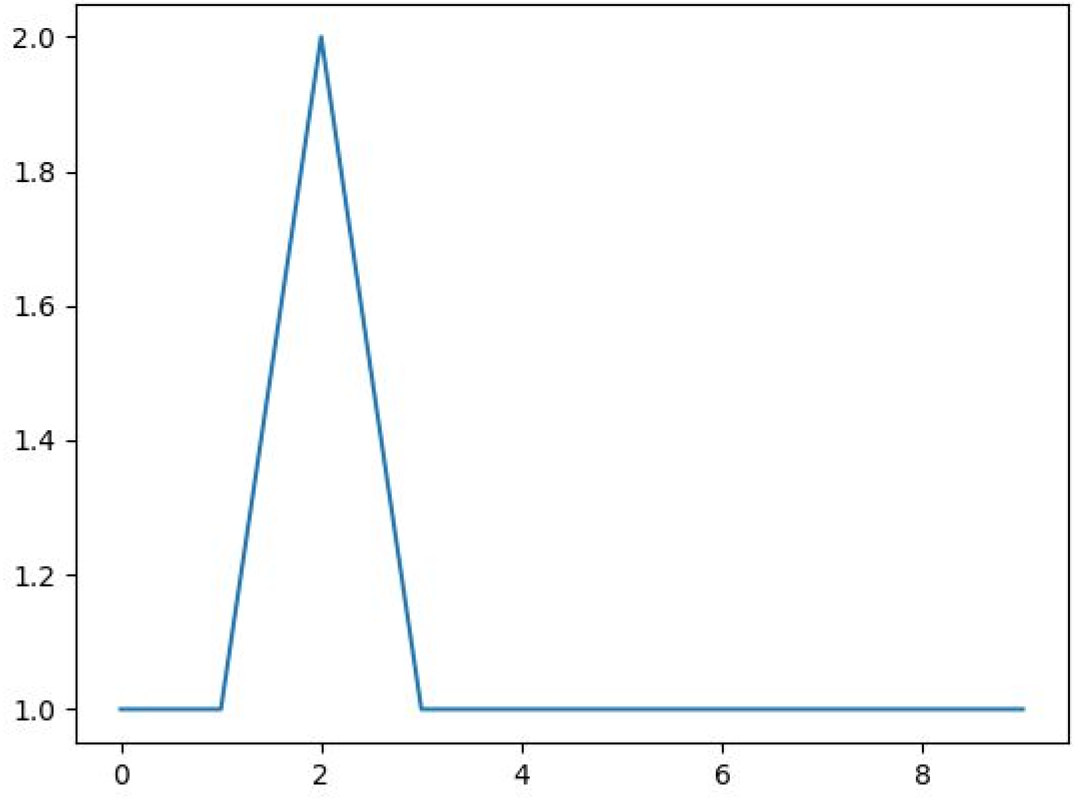
```
    x = np.array(x);
    y = [1, 0];
    y = np.array(y);
    hebbLearn1(x, y, 1);

    # XOR gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [0, 1, 1, 0];
    y = np.array(y);
    hebbLearn1(x, y, 0);

    # ANDNOT gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [0, 0, 1, 0];
    y = np.array(y);
    hebbLearn1(x, y, 0);

    # NAND gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [1, 1, 1, 0];
    y = np.array(y);
    hebbLearn1(x, y, 0);

    #  NOR gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [1, 0, 0, 0];
    y = np.array(y);
    hebbLearn1(x, y, 0);

    #  XNOR gate
    x = [[0, 0, 1, 1],[0, 1, 0, 1]];
    x = np.array(x);
    y = [1, 0, 0, 1];
    y = np.array(y);
    hebbLearn1(x, y, 0);

if __name__ == "__main__":
    main()

```

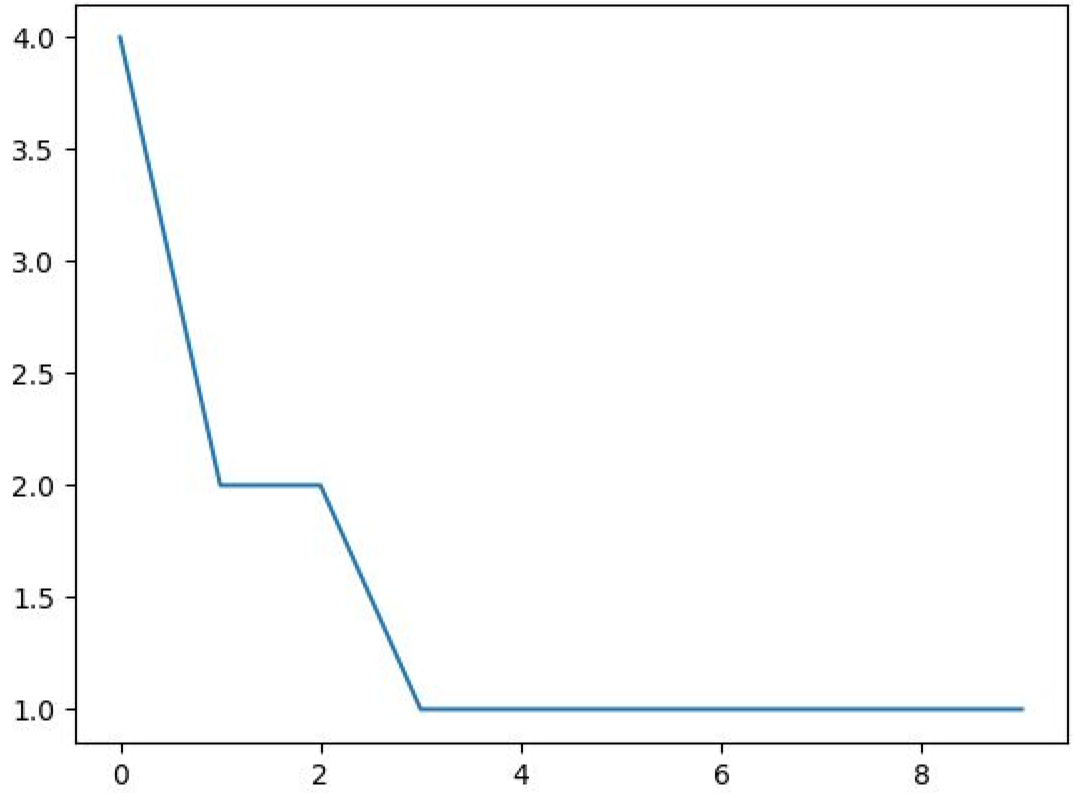

1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8

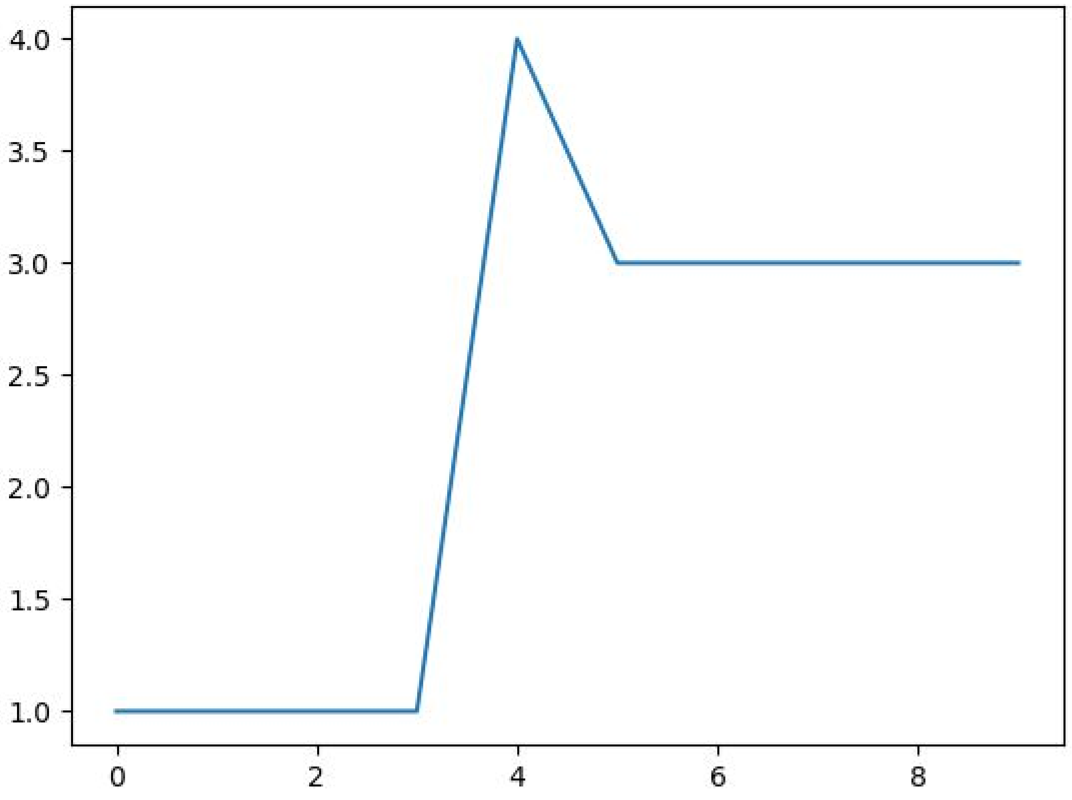

3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8

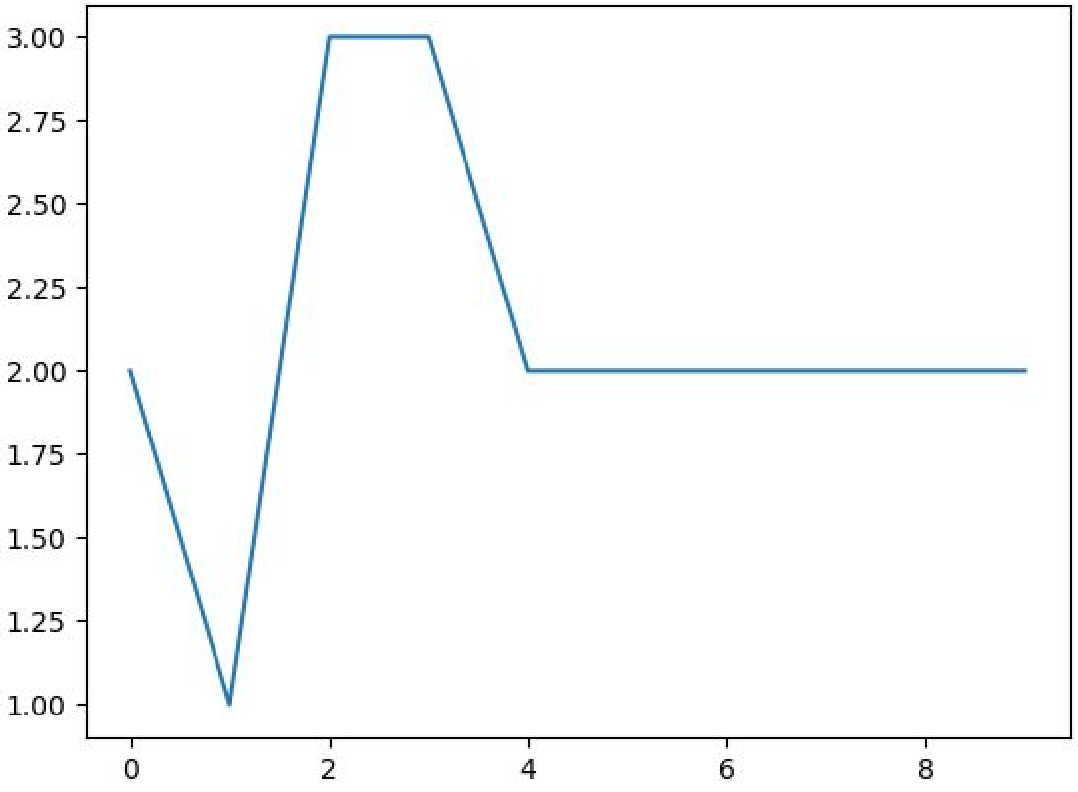

1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8


3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
2
4
6
8

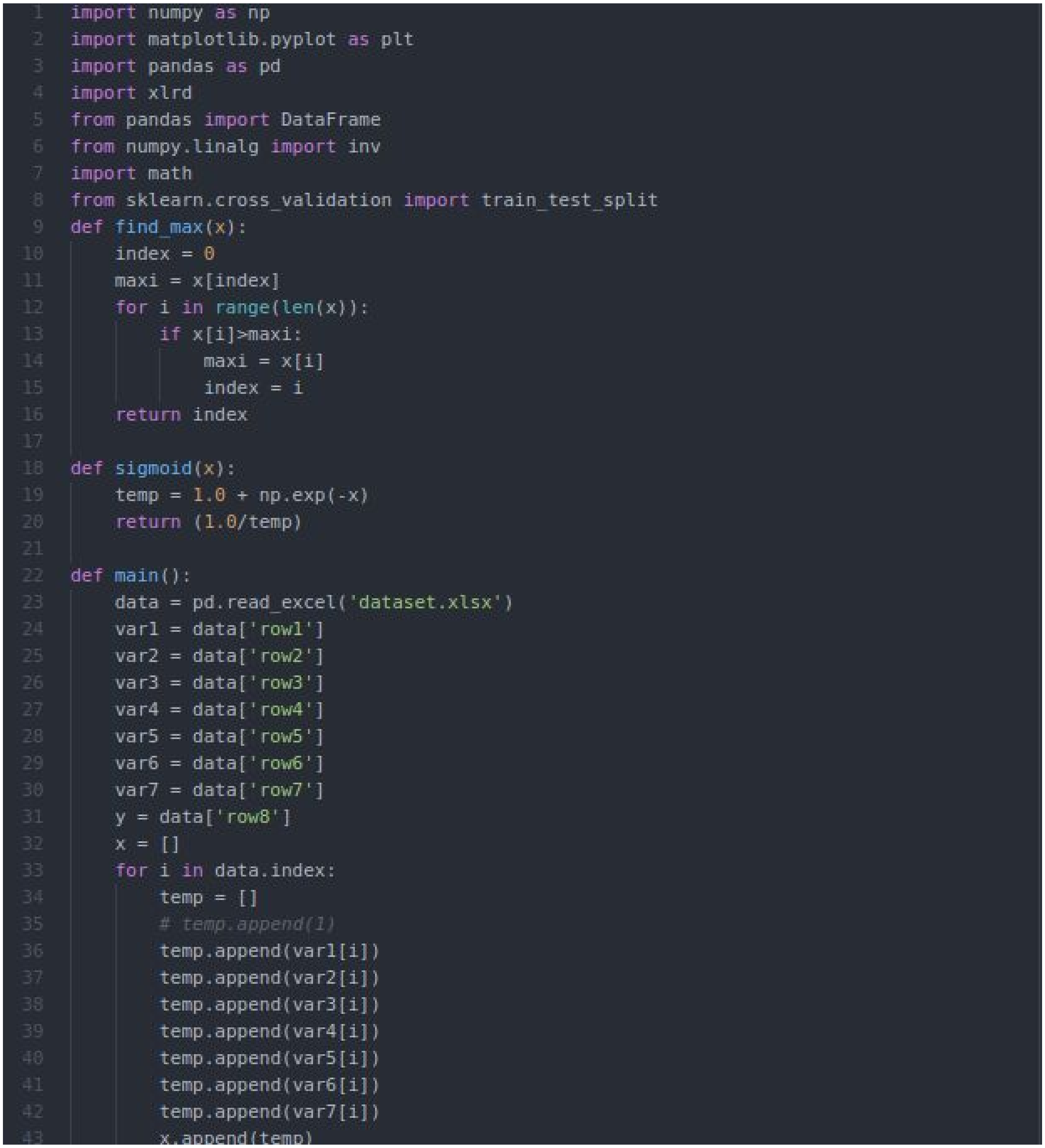

2.0
1.8
1.6
1.4
1.2
1.0
0
2
4
6
8


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
2
4
6
8

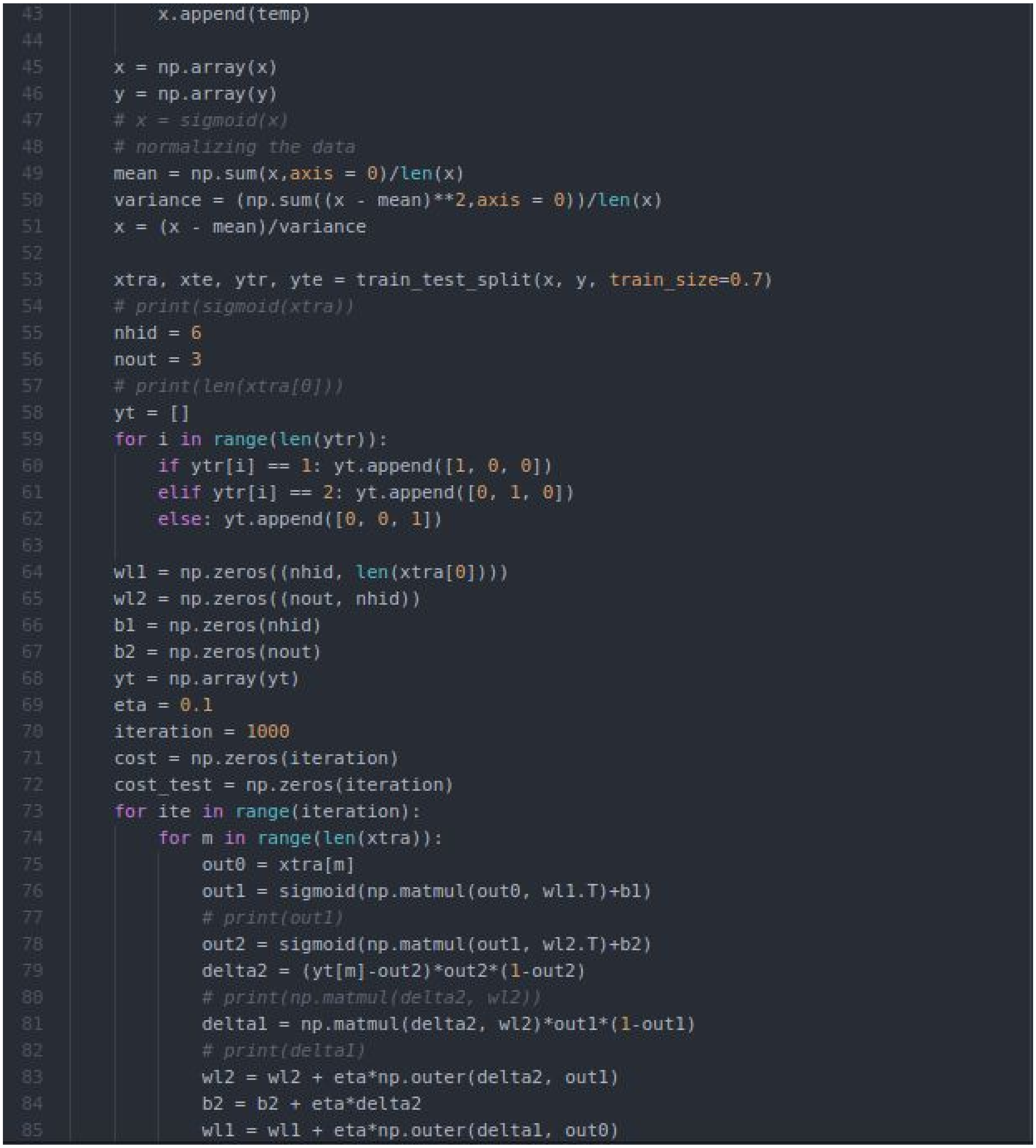

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
2
4
6
8


3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
2
4
6
8

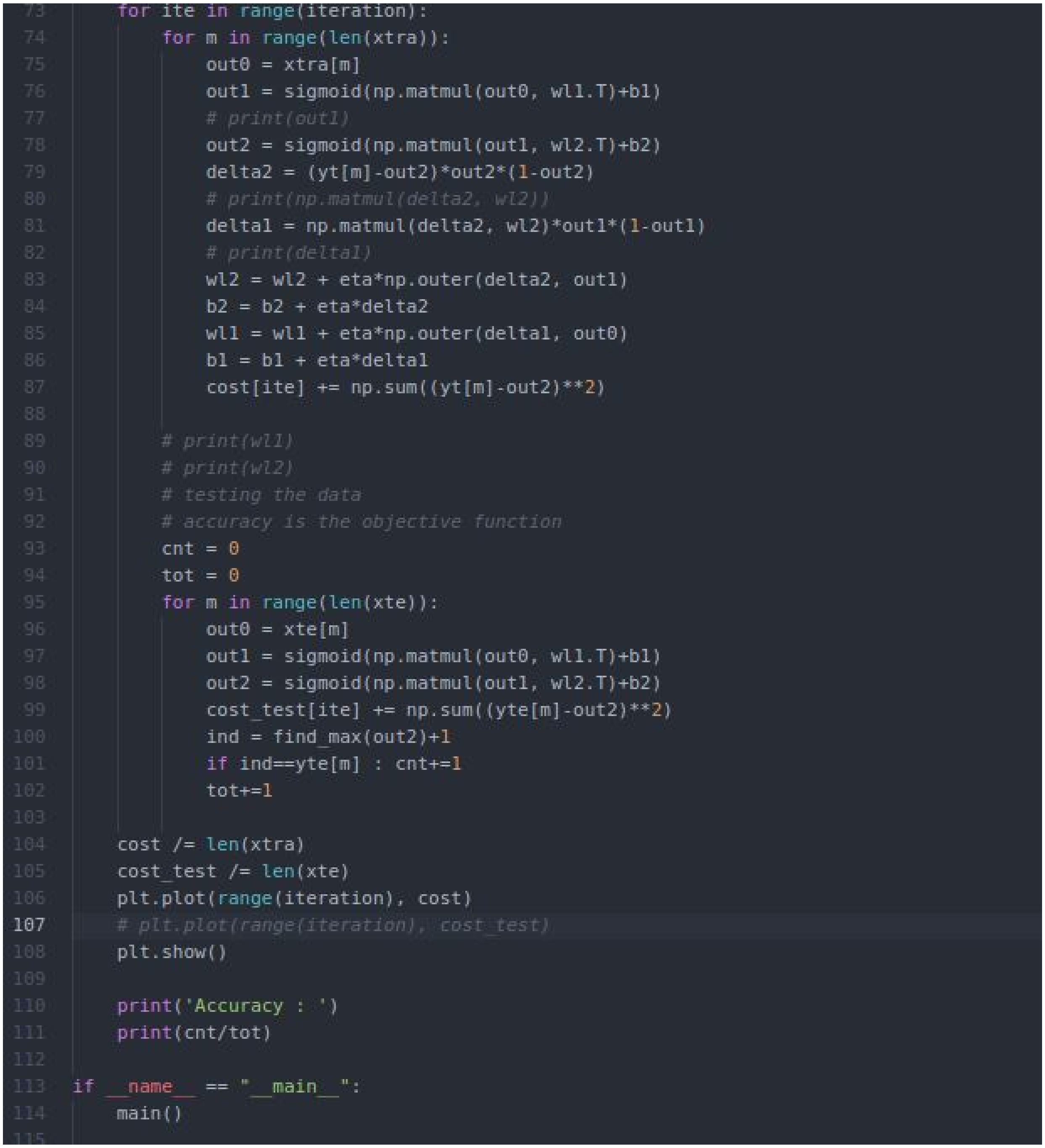
PROBLEM 2



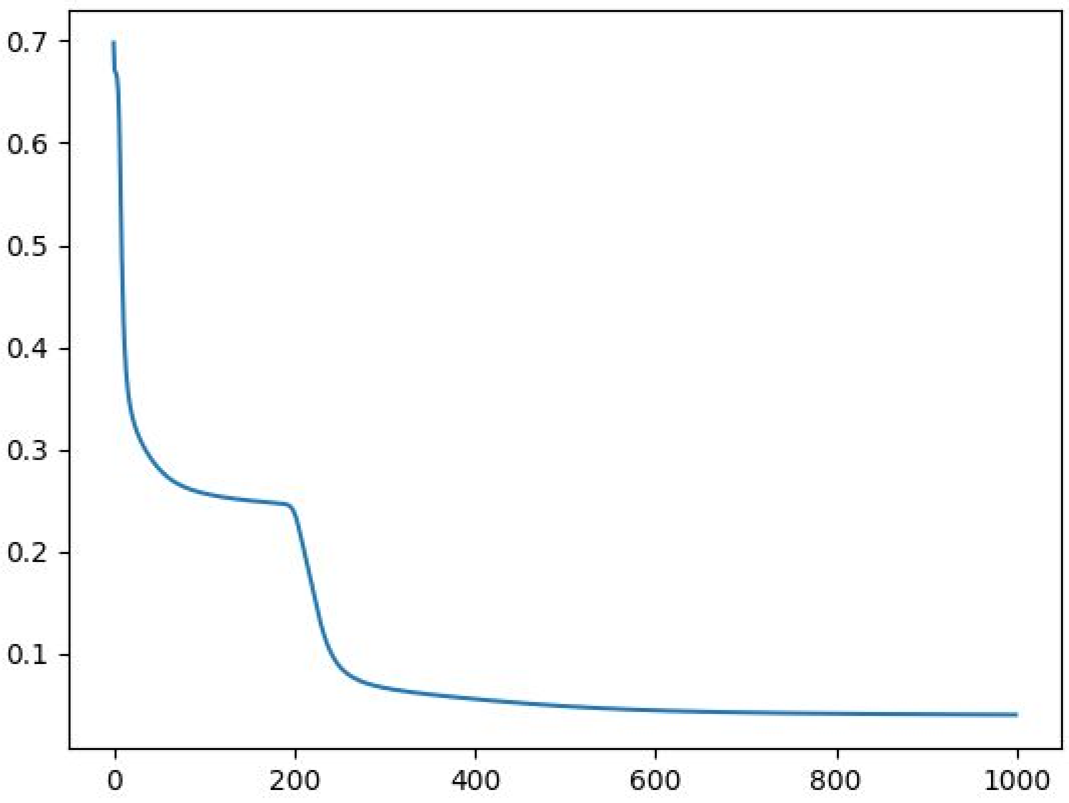
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
from pandas import DataFrame
from numpy.linalg import inv
import math
from sklearn.cross_validation import train_test_split
def find_max(x):
    index = 0
    maxi = x[index]
    for i in range(len(x)):
        if x[i]>maxi:
            maxi = x[i]
            index = i
    return index

def sigmoid(x):
    temp = 1.0 + np.exp(-x)
    return (1.0/temp)

def main():
    data = pd.read_excel('dataset.xlsx')
    var1 = data['row1']
    var2 = data['row2']
    var3 = data['row3']
    var4 = data['row4']
    var5 = data['row5']
    var6 = data['row6']
    var7 = data['row7']
    y = data['row8']
    x = []
    for i in data.index:
        temp = []
        # temp.append(1)
        temp.append(var1[i])
        temp.append(var2[i])
        temp.append(var3[i])
        temp.append(var4[i])
        temp.append(var5[i])
        temp.append(var6[i])
        temp.append(var7[i])
        x.append(temp)
```



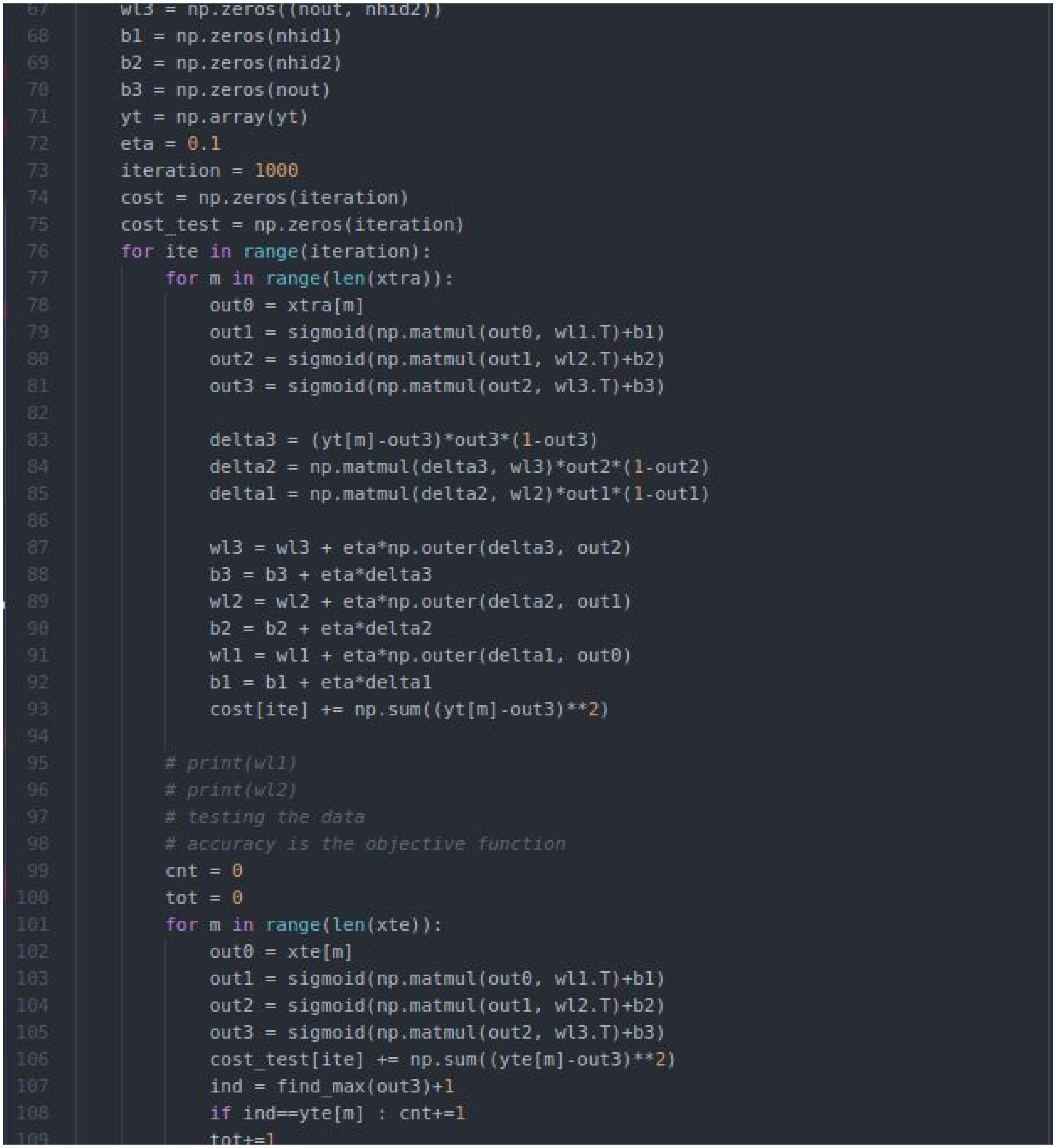
```python
        x.append(temp)

    x = np.array(x)
    y = np.array(y)
    # x = sigmoid(x)
    # normalizing the data
    mean = np.sum(x,axis = 0)/len(x)
    variance = (np.sum((x - mean)**2,axis = 0))/len(x)
    x = (x - mean)/variance

    xtra, xte, ytr, yte = train_test_split(x, y, train_size=0.7)
    # print(sigmoid(xtra))
    nhid = 6
    nout = 3
    # print(len(xtra[0]))
    yt = []
    for i in range(len(ytr)):
        if ytr[i] == 1: yt.append([1, 0, 0])
        elif ytr[i] == 2: yt.append([0, 1, 0])
        else: yt.append([0, 0, 1])

    wl1 = np.zeros((nhid, len(xtra[0])))
    wl2 = np.zeros((nout, nhid))
    b1 = np.zeros(nhid)
    b2 = np.zeros(nout)
    yt = np.array(yt)
    eta = 0.1
    iteration = 1000
    cost = np.zeros(iteration)
    cost_test = np.zeros(iteration)
    for ite in range(iteration):
        for m in range(len(xtra)):
            out0 = xtra[m]
            out1 = sigmoid(np.matmul(out0, wl1.T)+b1)
            # print(out1)
            out2 = sigmoid(np.matmul(out1, wl2.T)+b2)
            delta2 = (yt[m]-out2)*out2*(1-out2)
            # print(np.matmul(delta2, wl2))
            delta1 = np.matmul(delta2, wl2)*out1*(1-out1)
            # print(delta1)
            wl2 = wl2 + eta*np.outer(delta2, out1)
            b2 = b2 + eta*delta2
            wl1 = wl1 + eta*np.outer(delta1, out0)
```



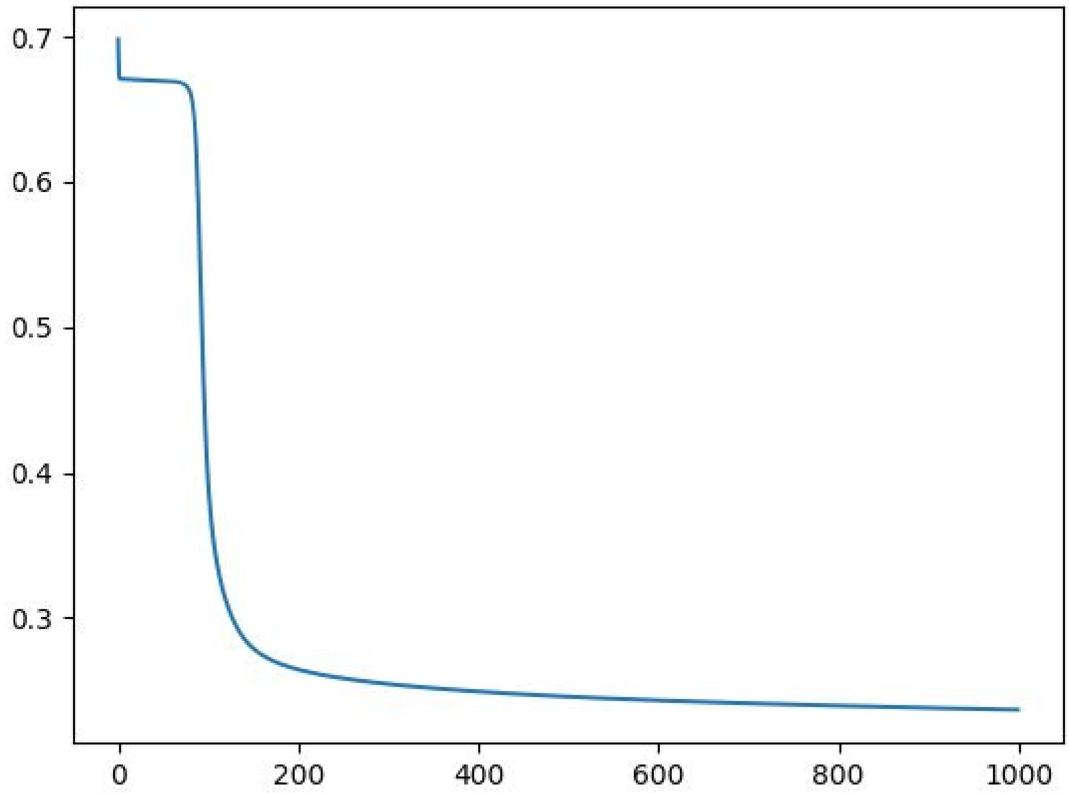
```python
    for ite in range(iteration):
        for m in range(len(xtra)):
            out0 = xtra[m]
            out1 = sigmoid(np.matmul(out0, wl1.T)+b1)
            # print(out1)
            out2 = sigmoid(np.matmul(out1, wl2.T)+b2)
            delta2 = (yt[m]-out2)*out2*(1-out2)
            # print(np.matmul(delta2, wl2))
            delta1 = np.matmul(delta2, wl2)*out1*(1-out1)
            # print(delta1)
            wl2 = wl2 + eta*np.outer(delta2, out1)
            b2 = b2 + eta*delta2
            wl1 = wl1 + eta*np.outer(delta1, out0)
            b1 = b1 + eta*delta1
            cost[ite] += np.sum((yt[m]-out2)**2)

        # print(wl1)
        # print(wl2)
        # testing the data
        # accuracy is the objective function
        cnt = 0
        tot = 0
        for m in range(len(xte)):
            out0 = xte[m]
            out1 = sigmoid(np.matmul(out0, wl1.T)+b1)
            out2 = sigmoid(np.matmul(out1, wl2.T)+b2)
            cost_test[ite] += np.sum((yte[m]-out2)**2)
            ind = find_max(out2)+1
            if ind==yte[m] : cnt+=1
            tot+=1

    cost /= len(xtra)
    cost_test /= len(xte)
    plt.plot(range(iteration), cost)
    # plt.plot(range(iteration), cost_test)
    plt.show()

    print('Accuracy : ')
    print(cnt/tot)

if __name__ == "__main__":
    main()
```



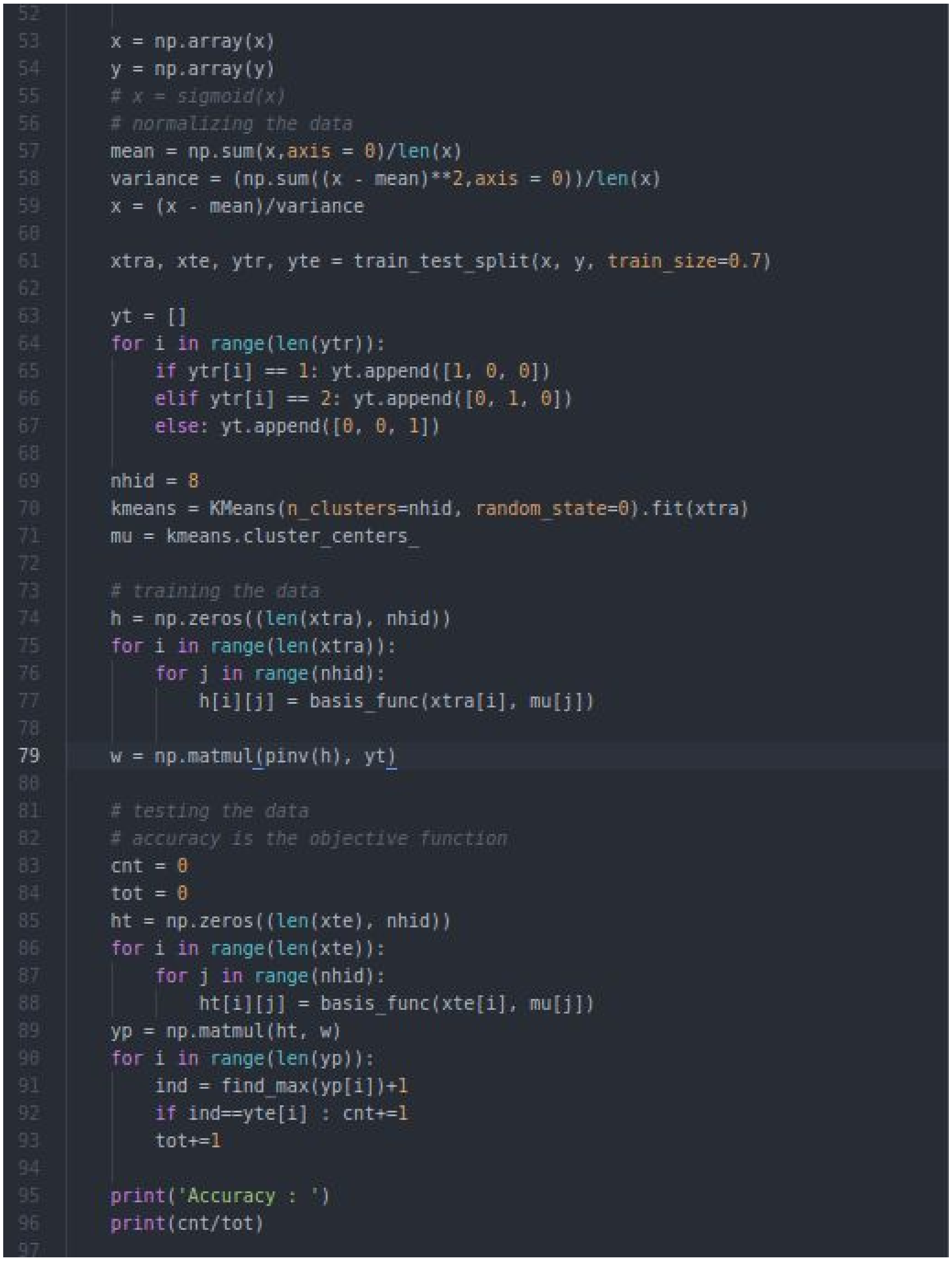
PROBLEM 3



```
    wl3 = np.zeros((nout, nhid2))
    b1 = np.zeros(nhid1)
    b2 = np.zeros(nhid2)
    b3 = np.zeros(nout)
    yt = np.array(yt)
    eta = 0.1
    iteration = 1000
    cost = np.zeros(iteration)
    cost_test = np.zeros(iteration)
    for ite in range(iteration):
        for m in range(len(xtra)):
            out0 = xtra[m]
            out1 = sigmoid(np.matmul(out0, wl1.T)+b1)
            out2 = sigmoid(np.matmul(out1, wl2.T)+b2)
            out3 = sigmoid(np.matmul(out2, wl3.T)+b3)

            delta3 = (yt[m]-out3)*out3*(1-out3)
            delta2 = np.matmul(delta3, wl3)*out2*(1-out2)
            delta1 = np.matmul(delta2, wl2)*out1*(1-out1)

            wl3 = wl3 + eta*np.outer(delta3, out2)
            b3 = b3 + eta*delta3
            wl2 = wl2 + eta*np.outer(delta2, out1)
            b2 = b2 + eta*delta2
            wl1 = wl1 + eta*np.outer(delta1, out0)
            b1 = b1 + eta*delta1
            cost[ite] += np.sum((yt[m]-out3)**2)

        # print(wl1)
        # print(wl2)
        # testing the data
        # accuracy is the objective function
        cnt = 0
        tot = 0
        for m in range(len(xte)):
            out0 = xte[m]
            out1 = sigmoid(np.matmul(out0, wl1.T)+b1)
            out2 = sigmoid(np.matmul(out1, wl2.T)+b2)
            out3 = sigmoid(np.matmul(out2, wl3.T)+b3)
            cost_test[ite] += np.sum((yte[m]-out3)**2)
            ind = find_max(out3)+1
            if ind==yte[m] : cnt+=1
            tot+=1
```

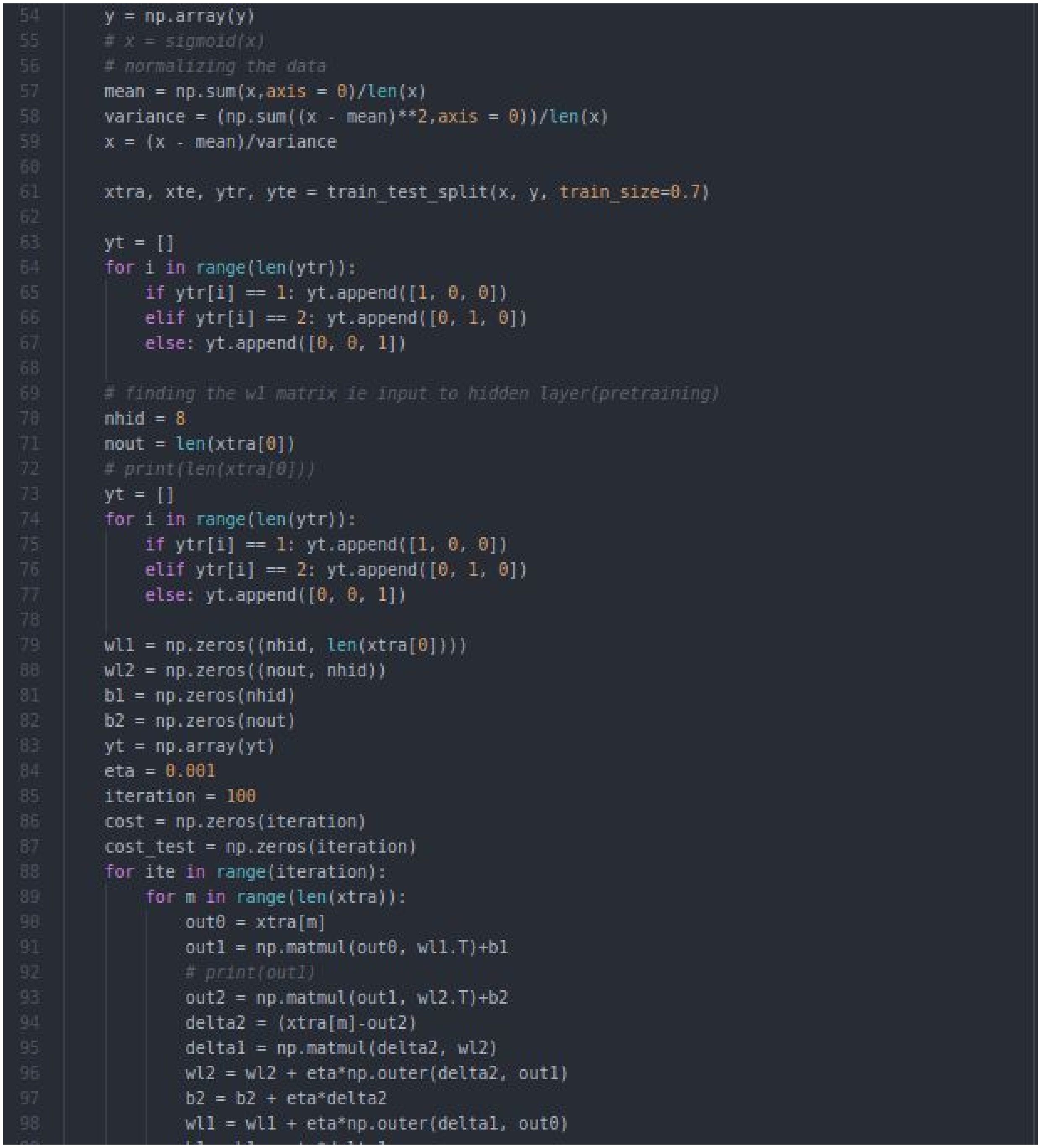


PROBLEM 4

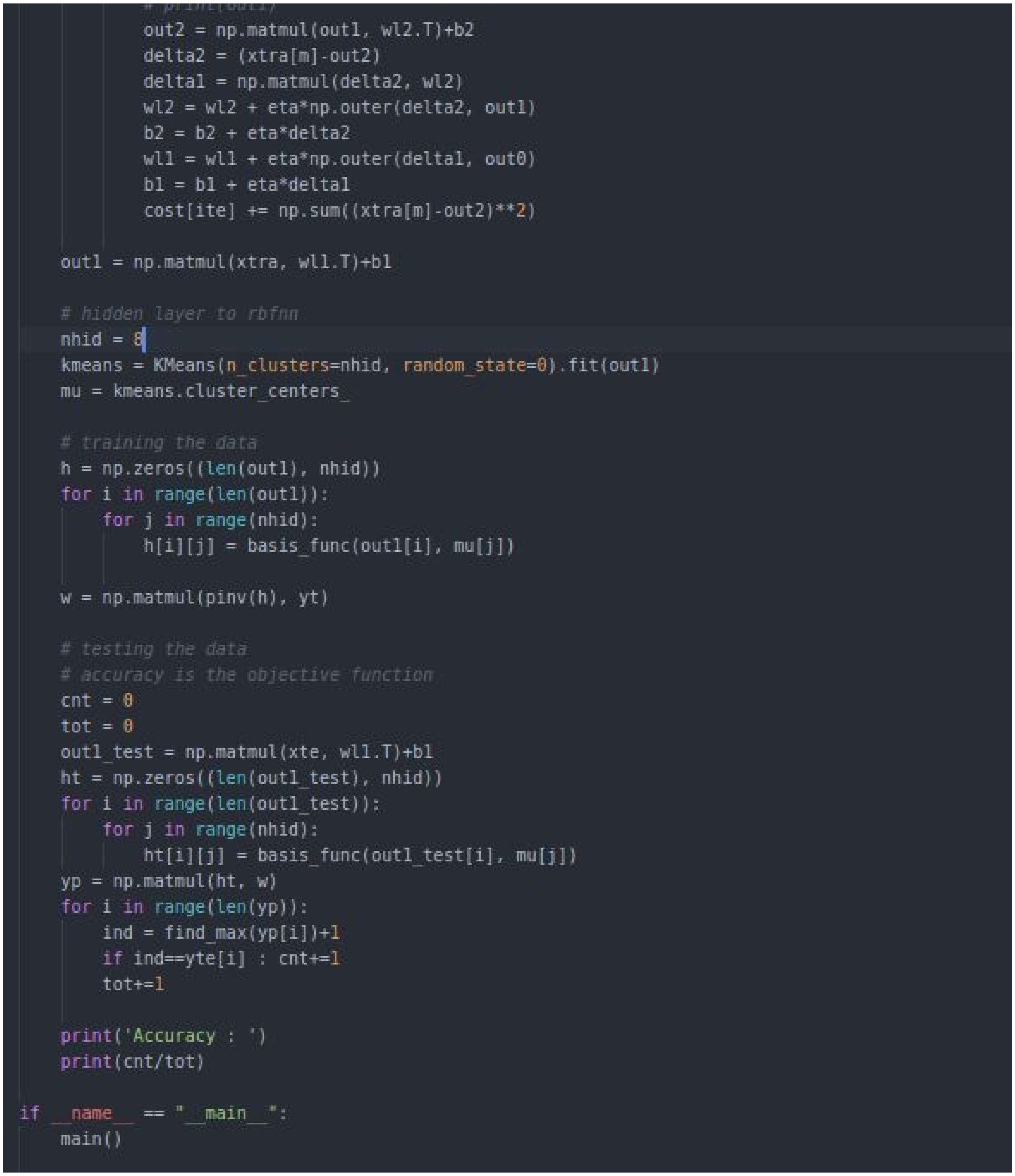


```
    
    x = np.array(x)
    y = np.array(y)
    # x = sigmoid(x)
    # normalizing the data
    mean = np.sum(x,axis = 0)/len(x)
    variance = (np.sum((x - mean)**2,axis = 0))/len(x)
    x = (x - mean)/variance

    xtra, xte, ytr, yte = train_test_split(x, y, train_size=0.7)

    yt = []
    for i in range(len(ytr)):
        if ytr[i] == 1: yt.append([1, 0, 0])
        elif ytr[i] == 2: yt.append([0, 1, 0])
        else: yt.append([0, 0, 1])

    nhid = 8
    kmeans = KMeans(n_clusters=nhid, random_state=0).fit(xtra)
    mu = kmeans.cluster_centers_

    # training the data
    h = np.zeros((len(xtra), nhid))
    for i in range(len(xtra)):
        for j in range(nhid):
            h[i][j] = basis_func(xtra[i], mu[j])

    w = np.matmul(pinv(h), yt)

    # testing the data
    # accuracy is the objective function
    cnt = 0
    tot = 0
    ht = np.zeros((len(xte), nhid))
    for i in range(len(xte)):
        for j in range(nhid):
            ht[i][j] = basis_func(xte[i], mu[j])
    yp = np.matmul(ht, w)
    for i in range(len(yp)):
        ind = find_max(yp[i])+1
        if ind==yte[i] : cnt+=1
        tot+=1

    print('Accuracy : ')
    print(cnt/tot)
```

PROBLEM 5



```
    y = np.array(y)
    # x = sigmoid(x)
    # normalizing the data
    mean = np.sum(x,axis = 0)/len(x)
    variance = (np.sum((x - mean)**2,axis = 0))/len(x)
    x = (x - mean)/variance

    xtra, xte, ytr, yte = train_test_split(x, y, train_size=0.7)

    yt = []
    for i in range(len(ytr)):
        if ytr[i] == 1: yt.append([1, 0, 0])
        elif ytr[i] == 2: yt.append([0, 1, 0])
        else: yt.append([0, 0, 1])

    # finding the w1 matrix ie input to hidden layer(pretraining)
    nhid = 8
    nout = len(xtra[0])
    # print(len(xtra[0]))
    yt = []
    for i in range(len(ytr)):
        if ytr[i] == 1: yt.append([1, 0, 0])
        elif ytr[i] == 2: yt.append([0, 1, 0])
        else: yt.append([0, 0, 1])

    wl1 = np.zeros((nhid, len(xtra[0])))
    wl2 = np.zeros((nout, nhid))
    b1 = np.zeros(nhid)
    b2 = np.zeros(nout)
    yt = np.array(yt)
    eta = 0.001
    iteration = 100
    cost = np.zeros(iteration)
    cost_test = np.zeros(iteration)
    for ite in range(iteration):
        for m in range(len(xtra)):
            out0 = xtra[m]
            out1 = np.matmul(out0, wl1.T)+b1
            # print(out1)
            out2 = np.matmul(out1, wl2.T)+b2
            delta2 = (xtra[m]-out2)
            delta1 = np.matmul(delta2, wl2)
            wl2 = wl2 + eta*np.outer(delta2, out1)
            b2 = b2 + eta*delta2
            wl1 = wl1 + eta*np.outer(delta1, out0)
```



```
            # print(out1)
            out2 = np.matmul(out1, wl2.T)+b2
            delta2 = (xtra[m]-out2)
            delta1 = np.matmul(delta2, wl2)
            wl2 = wl2 + eta*np.outer(delta2, out1)
            b2 = b2 + eta*delta2
            wl1 = wl1 + eta*np.outer(delta1, out0)
            b1 = b1 + eta*delta1
            cost[ite] += np.sum((xtra[m]-out2)**2)

    out1 = np.matmul(xtra, wl1.T)+b1

    # hidden layer to rbfnn
    nhid = 8
    kmeans = KMeans(n_clusters=nhid, random_state=0).fit(out1)
    mu = kmeans.cluster_centers_

    # training the data
    h = np.zeros((len(out1), nhid))
    for i in range(len(out1)):
        for j in range(nhid):
            h[i][j] = basis_func(out1[i], mu[j])

    w = np.matmul(pinv(h), yt)

    # testing the data
    # accuracy is the objective function
    cnt = 0
    tot = 0
    out1_test = np.matmul(xte, wl1.T)+b1
    ht = np.zeros((len(out1_test), nhid))
    for i in range(len(out1_test)):
        for j in range(nhid):
            ht[i][j] = basis_func(out1_test[i], mu[j])
    yp = np.matmul(ht, w)
    for i in range(len(yp)):
        ind = find_max(yp[i])+1
        if ind==yte[i] : cnt+=1
        tot+=1

    print('Accuracy : ')
    print(cnt/tot)

if __name__ == "__main__":
    main()
```