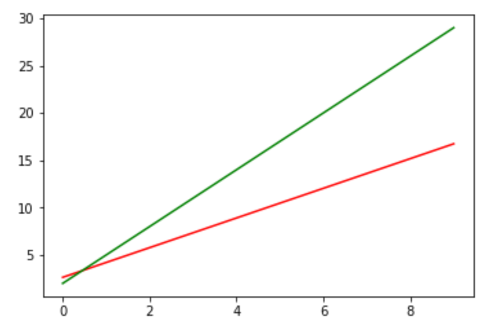
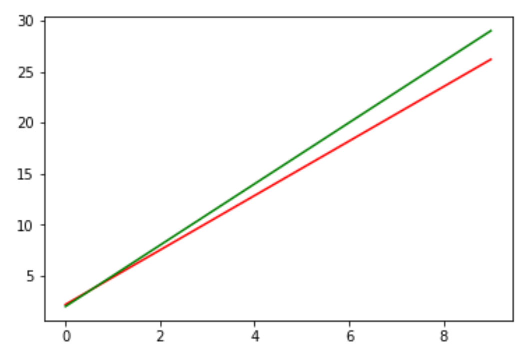
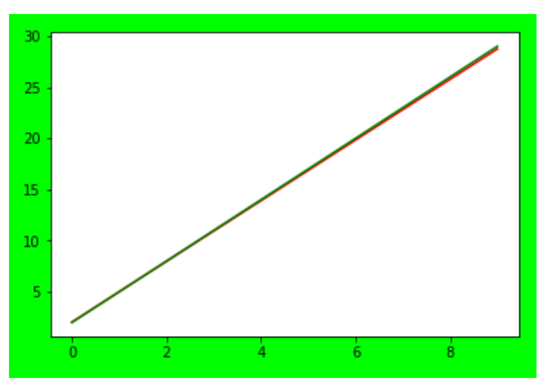
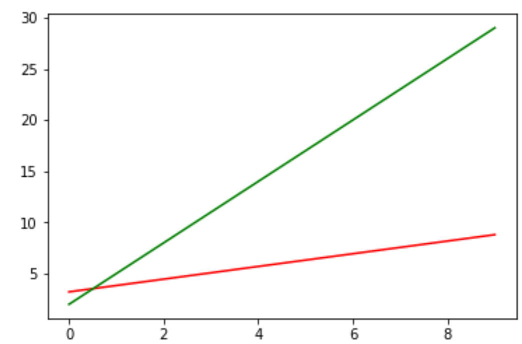
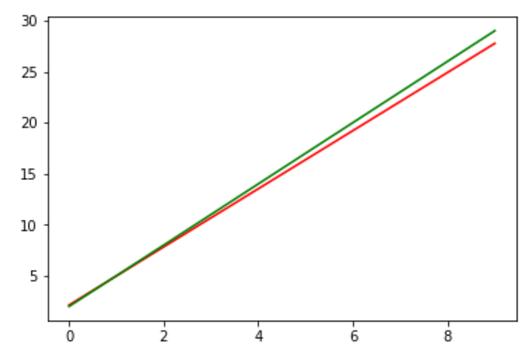
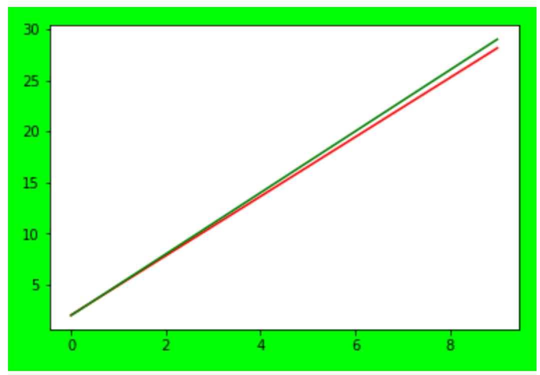
CMSC 422 PS4 Write Up

Problem 1:

In this problem, I have trained the algorithm using gradient descent and performed testing using the function ‘simplest_testing’ where the weights and bias which are received from the training algorithm is compared to the actual line equation (y=3x+2). I have computed the error percentage error from the actual and computed value of y and calculated the accuracy percentage of the algorithm using changes to parameters such as eta, iterations and step size.

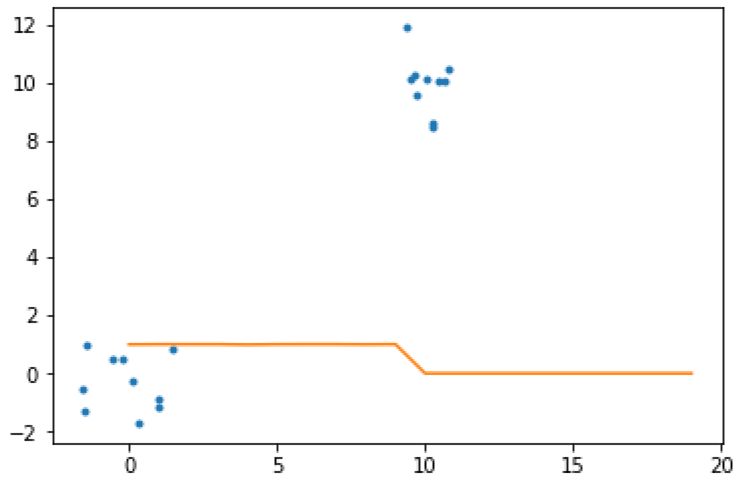
Results from the test cases were:

S.No.	‘n’ (data size)	‘k’ (iterations)	‘eta’	Accuracy Percentage	Graphs (y=3x+2 vs line from algorithm)
1	100	100	0.02	61.93%	
2	100	1000	0.02	98.21%	
3	100	10000	0.02	99.34%	

4	100	100	0.03	72.17%	
5	100	1000	0.03	98.69%	
6	100	10000	0.03	99.68%	

Problem 2:

For this problem, when we choose a linearly separable data set (Training set number 1), the algorithm correctly classified the points. The classification is shown in the test function using the probability of the point.

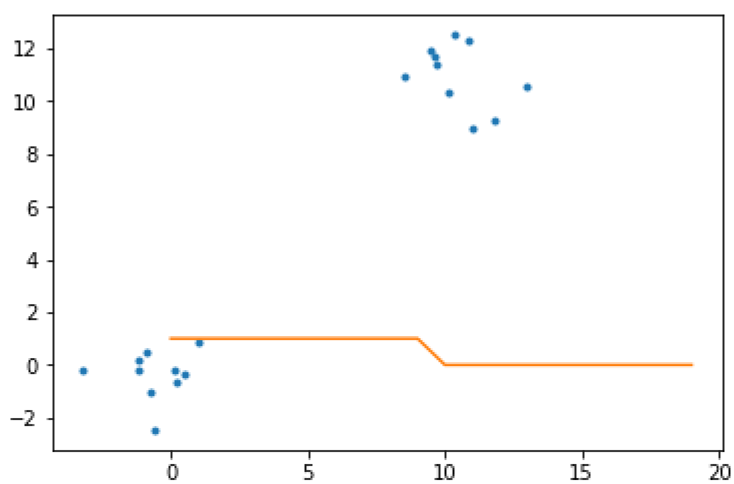


Graph 1

From the graph 1, we can clearly observe that the points near (10,10) have a probability of 0 while the points near (0,0) have a probability of 1. For this particular case, I have used the parameters as follows:

k=100000

eta=0.03



Graph 2

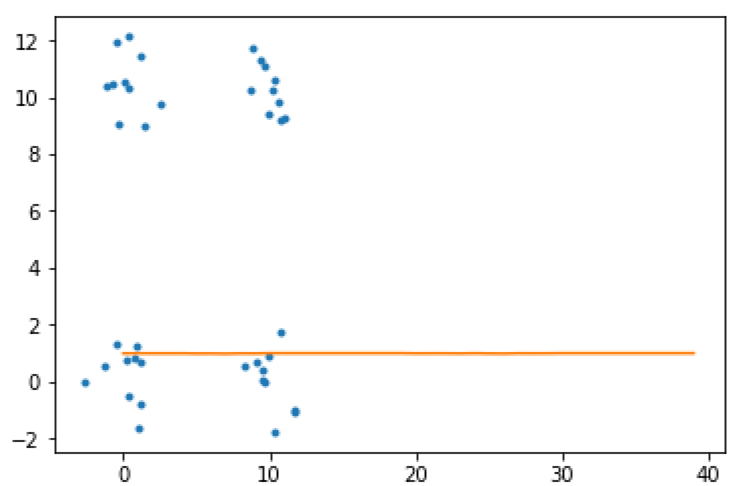
Also, from the graph 2, we can clearly observe that the points near (10,10) have a probability of 0 while the points near (0,0) have a probability of 1. For this particular case, I have used the parameters as follows:
k=1000000
eta=0.03

As both the graphs have identical results, we can safely consider that the algorithm has converged for the case 1 (k=100000, eta=0.03) itself. For testing the algorithm I observed the values returned by the testing function and checking if it satisfies the condition:
(Points near (10,10) have probability 0) AND (Points near (0,0) have probability 1)

Noting few samples from my testing cases:

1) *The point (0.9713701165690681 , -1.190080022008588) has probability 0.9940106047944518*
   So this a point near (0,0) and has a probability =1
2) *The point (9.714203679950991 , 9.594112657866141) has probability 4.875416073324236e-05*
   So this is a point new (10,10) and has probability =0

For the training set number 2, the data was not linearly separable and this caused the algorithm to misclassify the data points as the model of the problem is not capable of classifying the data into different classes.

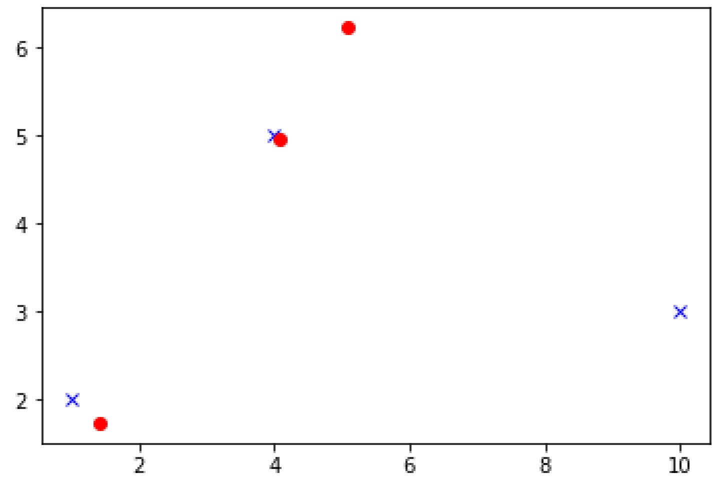


Graph 3

From the graph 3, we can clearly identify that the line is not able to classify the points into required classes as the points are non separable using a single line as the hyperplane.

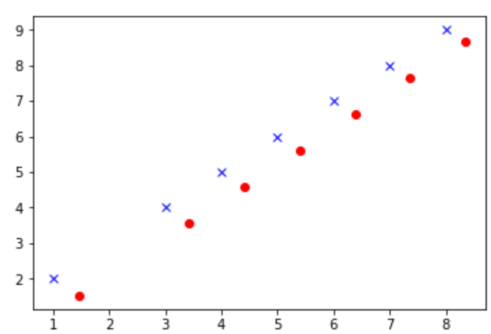
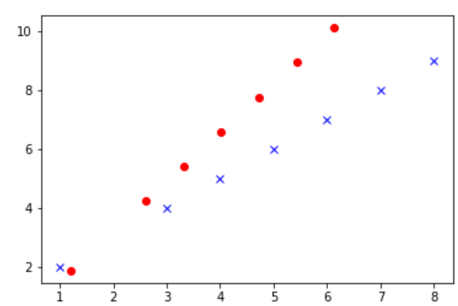
Problem 3:

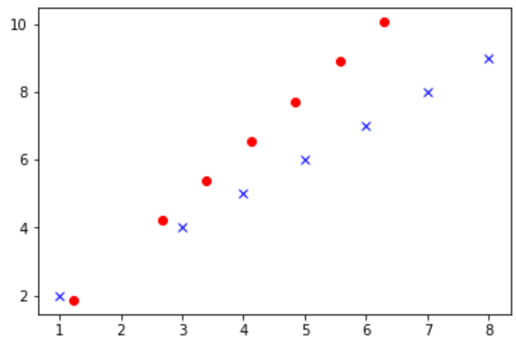
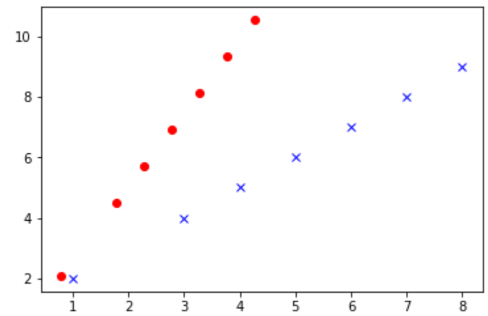
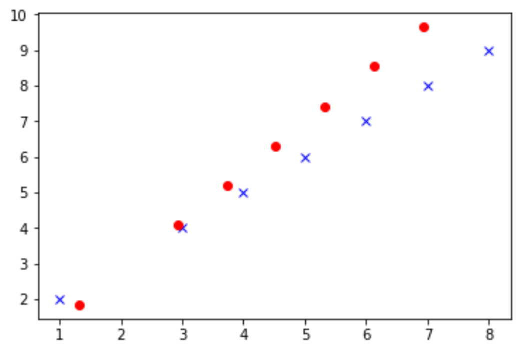
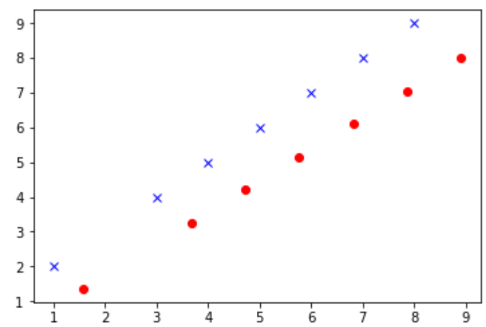
After training the data and calculating the weights and bias, I used the test case given in the question by substituting points (1,2) , (4,5) and (10,3) I got the result as (1.4, 1.90) , 4.07,4.97) and (5.10,6.2269) using the parameters as follows: k=650, eta= 0.01 and sigma=0



We can observe from the graph that the point (10,3) is a point quite far off from the line y=x+1 and we are training the algorithm to fit this line. That would mean the (10,3) is an outlier and would not be a good fit for the algorithm. That is the reason for such a huge difference between the algorithm output and the actual point. This is also the reason the first two points are having such good accuracy from the algorithm as they lie on the line y=x+1.

Next, testing the algorithm for sigma=0.1 and sigma=0, I get the following results:

Parameters	Graph (Sigma=0)	Graph (Sigma=0.1)
eta=0.01, k=1000		

eta=0.01, k=1500		
eta=0.01, k=1700		

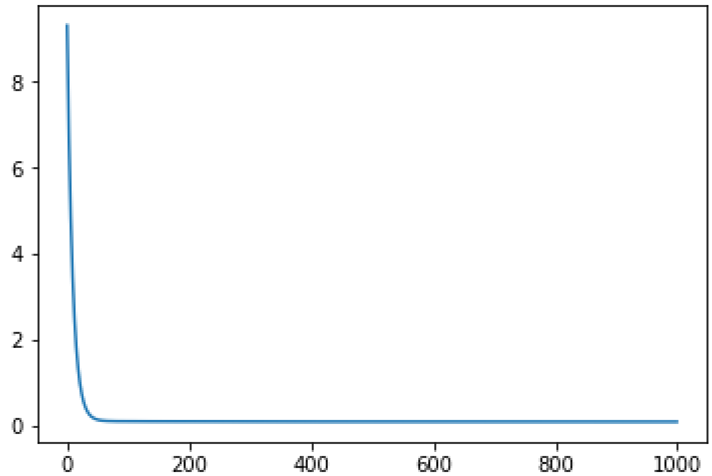
Observing the above graphs, we can observe the distinct difference between the two sigma values. The graphs with sigma=0 is fitting more accurately while for sigma=0.1, the fitting is less accurate. This is due to the variance which is being added to the training data as sigma is just adding noise to the existing data and then passing it to the algorithm.

The results vary drastically due to the change in sigma values using the same testing data for both the cases with input parameters also kept the same.

In [1]:
```
import numpy as np
import sys
import matplotlib.pyplot as plt

###Problem 1
###Provided function to create training data
def simplest_training_data(n):

    w = 3
    b = 2
    x = np.random.uniform(0,1,n)
    y = 3*x+b+0.3*np.random.normal(0,1,n)
    return (x,y)


def simplest_training(n, k, eta):
    # Generating random training data
    x,y=simplest_training_data(n)
    # initialising weight randomly from a Gaussian Distribution
    w=np.random.normal(0,1,1)
    # initialising bias to be 0
    b=0
    Loss_values=[]
    epochs=[]

    # Iterating over k epochs
    for i in range (k):
        dL_dw=0
        dL_db=0
        Loss_function=0
        # Iterating over training data points
        for j in range (n):
            a=w*x[j]+b
            Loss_function+=(a-y[j])**2
            dL_dw=dL_dw+2*x[j]*(w*x[j]+b-y[j])
            dL_db=dL_db+2*(w*x[j]+b-y[j])
        # Performing gradient descent
        Loss_function=Loss_function/n
        Loss_values.append(Loss_function)
        epochs.append(i)
        dL_dw=dL_dw/n
        dL_db=dL_db/n
        w=w-(eta)*(dL_dw)
        b=b-(eta)*(dL_db)
    theta=(w,b)
    plt.plot(epochs,Loss_values)
    return theta

def simplest_testing(theta, x):
    w=theta[0]
    b=theta[1]
    y=[]
    for l in range(np.size(x)):
        z=w*x[l]+b
        y.append(z)
    return y
```

In [2]:
```
theta=simplest_training(100,1000,0.02)
x=[0,1,2,3,4,5,6,7,8,9]
y=simplest_testing(theta,x)
print(y)
```

```
[array([1.96080508]), array([4.95471909]), array([7.94863309]), array([10.94254
71]), array([13.9364611]), array([16.93037511]), array([19.92428912]), array([2
2.91820312]), array([25.91211713]), array([28.90603113])]
```



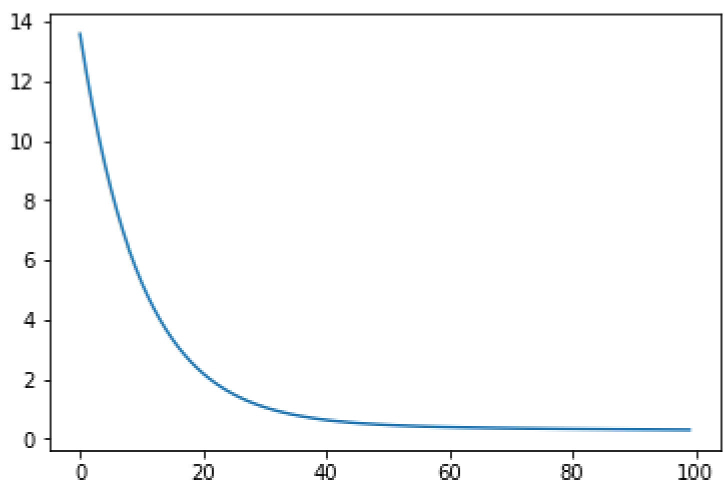
In [3]:
```
theta=(3,2)
x=[0,1,2,3,4,5,6,7,8,9]
y=simplest_testing(theta,x)
print(y)
```

```
[2, 5, 8, 11, 14, 17, 20, 23, 26, 29]
```

In [ ]:

In [4]:
```
## Parameters for this test condition are: n=100, k=100, eta=0.02
theta=simplest_training(100,100,0.02)
x=[0,1,2,3,4,5,6,7,8,9]
y1=simplest_testing(theta,x)
theta_default=(3,2)
x=[0,1,2,3,4,5,6,7,8,9]
y2=simplest_testing(theta_default,x)
percent=0
for i in range(np.size(y)):
    percent+=100-(abs(y1[i]-y2[i])/y2[i])*100
percent=percent/(np.size(y))
print("Percentage accuracy between the two set of values is: ",percent)
```

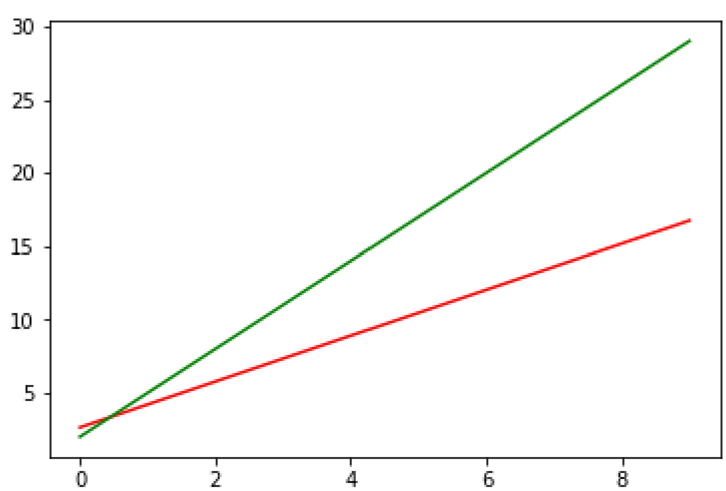
```
Percentage accuracy between the two set of values is:  [65.17342863]
```



In [5]:
```
print("Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)")
plt.plot(x,y1,'r-')
plt.plot(x,y2,'g-')
```

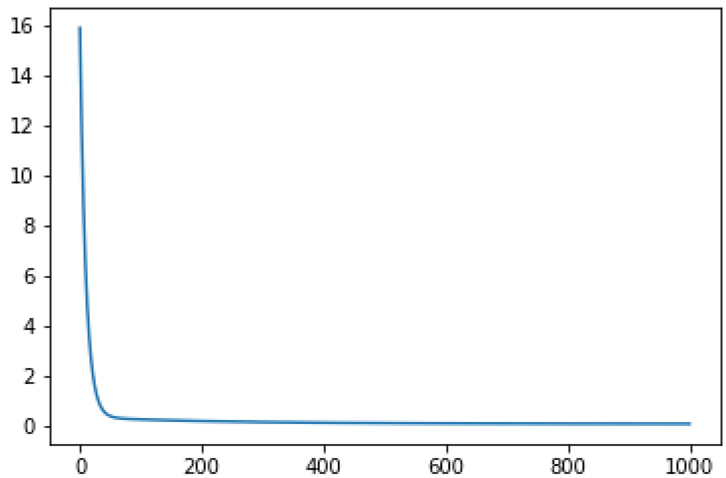
```
Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x13c51c76e48>]
```



In [6]:
```
## Parameters for this test condition are: n=100, k=1000, eta=0.02
theta=simplest_training(100,1000,0.02)
x=[0,1,2,3,4,5,6,7,8,9]
y1=simplest_testing(theta,x)
theta_default=(3,2)
x=[0,1,2,3,4,5,6,7,8,9]
y2=simplest_testing(theta_default,x)
percent=0
for i in range(np.size(y)):
    percent+=100-(abs(y1[i]-y2[i])/y2[i])*100
percent=percent/(np.size(y))
print("Percentage accuracy between the two set of values is: ",percent)
```

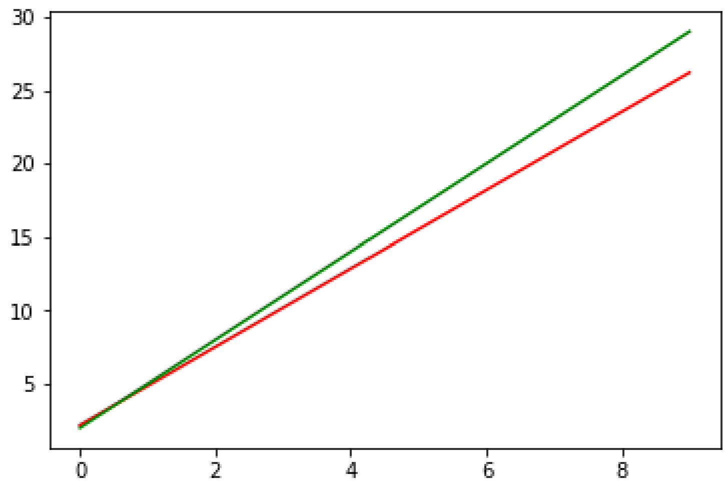
```
Percentage accuracy between the two set of values is:  [92.0043153]
```



In [7]:
```
print("Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)")
plt.plot(x,y1,'r-')
plt.plot(x,y2,'g-')
```

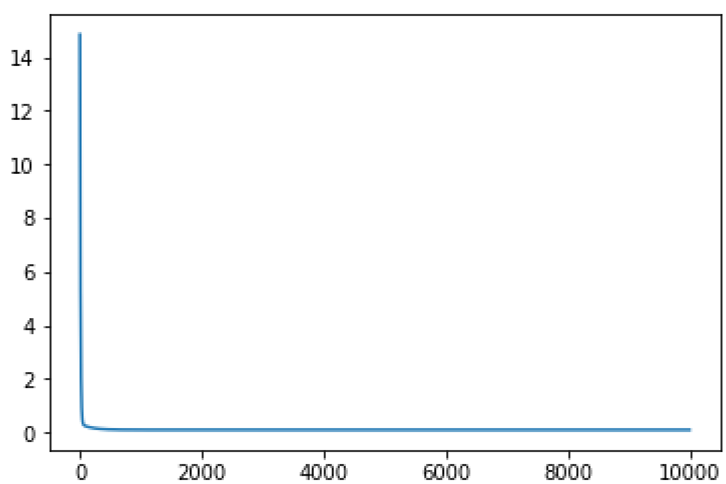
```
Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x13c51d443c8>]
```



In [8]:
```
## Parameters for this test condition are: n=100, k=10000, eta=0.02
theta=simplest_training(100,10000,0.02)
x=[0,1,2,3,4,5,6,7,8,9]
y1=simplest_testing(theta,x)
theta_default=(3,2)
x=[0,1,2,3,4,5,6,7,8,9]
y2=simplest_testing(theta_default,x)
percent=0
for i in range(np.size(y)):
    percent+=100-(abs(y1[i]-y2[i])/y2[i])*100
percent=percent/(np.size(y))
print("Percentage accuracy between the two set of values is: ",percent)
```

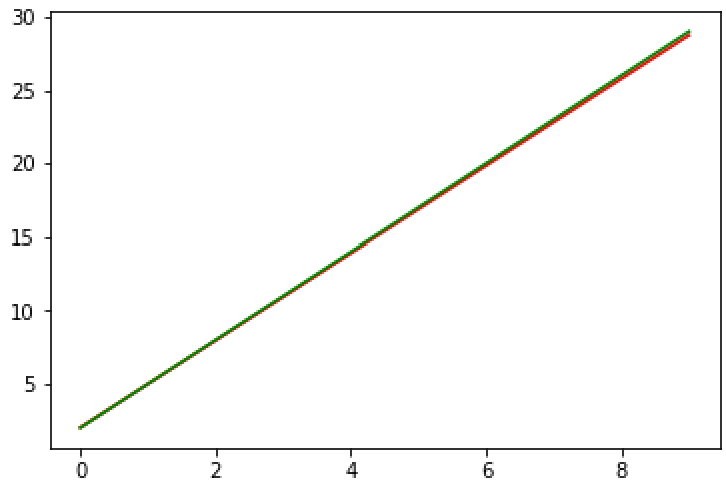
```
Percentage accuracy between the two set of values is:  [99.26290048]
```



In [9]:
```
print("Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)")
plt.plot(x,y1,'r-')
plt.plot(x,y2,'g-')
```

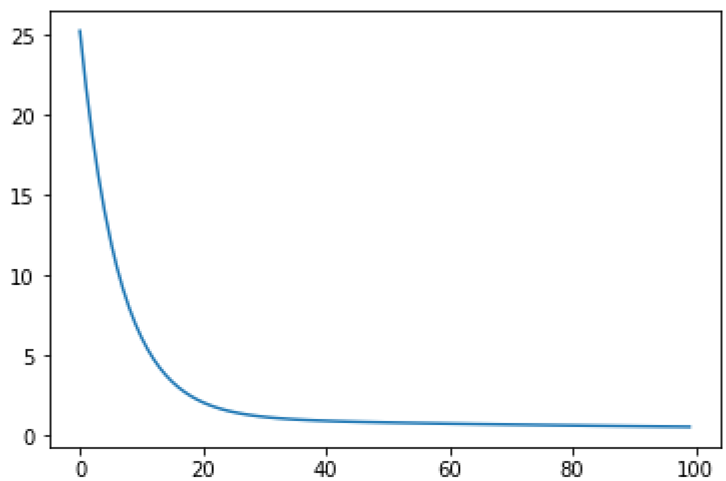
```
Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x13c51d44ef0>]
```



In [10]:
```
## Parameters for this test condition are: n=100, k=100, eta=0.03
theta=simplest_training(100,100,0.03)
x=[0,1,2,3,4,5,6,7,8,9]
y1=simplest_testing(theta,x)
theta_default=(3,2)
x=[0,1,2,3,4,5,6,7,8,9]
y2=simplest_testing(theta_default,x)
percent=0
for i in range(np.size(y)):
    percent+=100-(abs(y1[i]-y2[i])/y2[i])*100
percent=percent/(np.size(y))
print("Percentage accuracy between the two set of values is: ",percent)
```

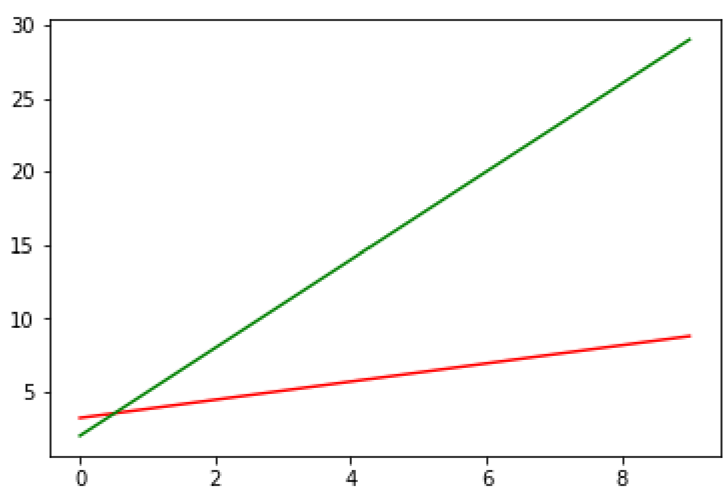
```
Percentage accuracy between the two set of values is:  [42.50037863]
```



In [11]:
```
print("Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)")
plt.plot(x,y1,'r-')
plt.plot(x,y2,'g-')
```

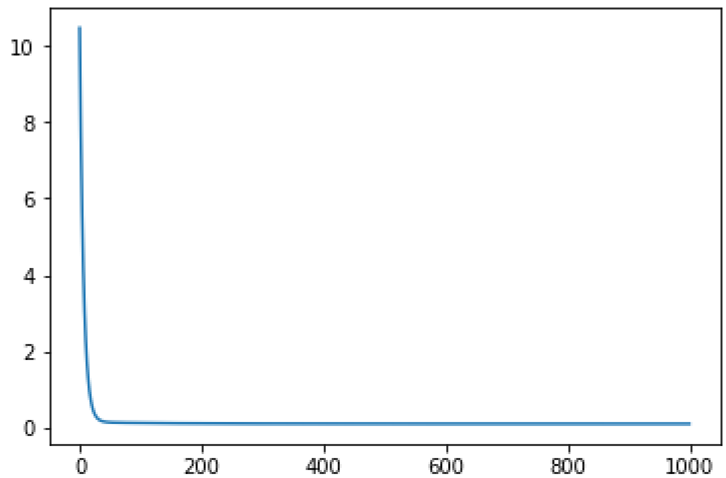
```
Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)
```

Out[11]: [<matplotlib.lines.Line2D at 0x13c51e852e8>]



In [12]:
```
## Parameters for this test condition are: n=100, k=1000, eta=0.03
theta=simplest_training(100,1000,0.03)
x=[0,1,2,3,4,5,6,7,8,9]
y1=simplest_testing(theta,x)
theta_default=(3,2)
x=[0,1,2,3,4,5,6,7,8,9]
y2=simplest_testing(theta_default,x)
percent=0
for i in range(np.size(y)):
    percent+=100-(abs(y1[i]-y2[i])/y2[i])*100
percent=percent/(np.size(y))
print("Percentage accuracy between the two set of values is: ",percent)
```

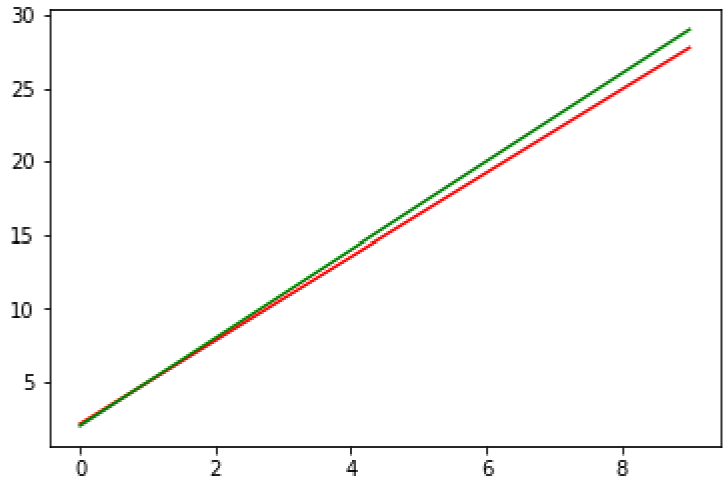
```
Percentage accuracy between the two set of values is:  [96.42856979]
```



In [13]:
```
print("Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)")
plt.plot(x,y1,'r-')
plt.plot(x,y2,'g-')
```

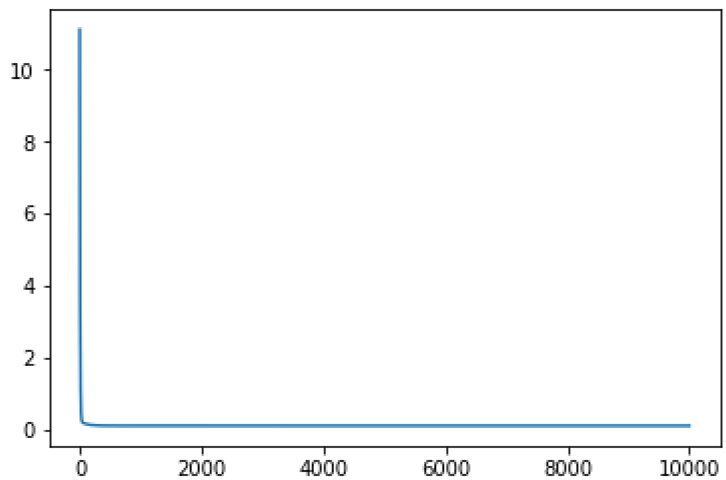
```
Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x13c51f80208>]
```



In [14]:
```
## Parameters for this test condition are: n=100, k=10000, eta=0.03
theta=simplest_training(100,10000,0.03)
x=[0,1,2,3,4,5,6,7,8,9]
y1=simplest_testing(theta,x)
theta_default=(3,2)
x=[0,1,2,3,4,5,6,7,8,9]
y2=simplest_testing(theta_default,x)
percent=0
for i in range(np.size(y)):
    percent+=100-(abs(y1[i]-y2[i])/y2[i])*100
percent=percent/(np.size(y))
print("Percentage accuracy between the two set of values is: ",percent)
```

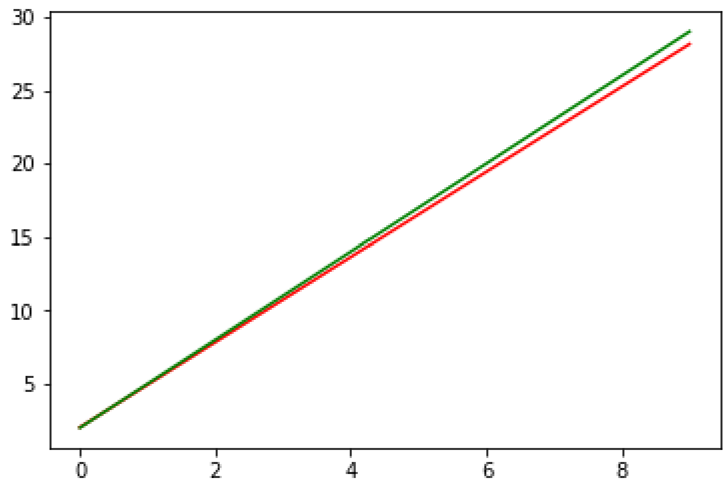
```
Percentage accuracy between the two set of values is:  [97.57492806]
```



In [15]:
```
print("Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)")
plt.plot(x,y1,'r-')
plt.plot(x,y2,'g-')
```

```
Plot between lines generated by algorithm(RED) and y=3x+2(GREEN)
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x13c51dff358>]
```

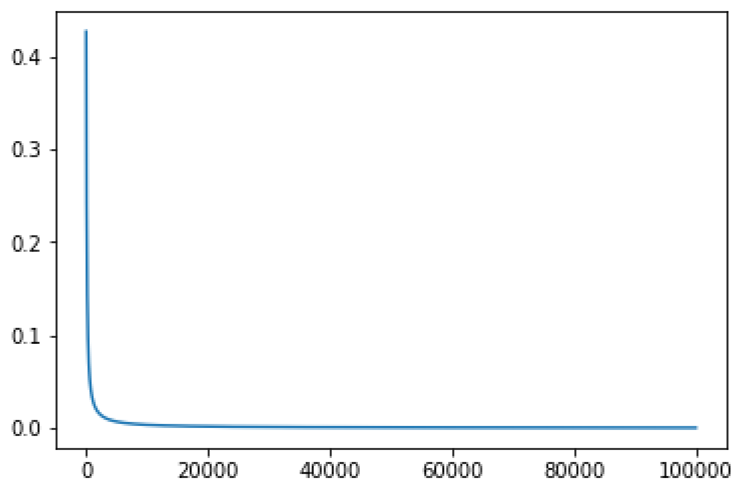


In [ ]:

In [67]:
```
import numpy as np
import sys
import matplotlib.pyplot as plt
import math
###Problem 2
###Provided function to create training data
def single_layer_training_data(trainset):
    n = 10
    if trainset == 1:
        # Linearly separable
        X = np.concatenate((np.random.normal((0,0),1,(n,2)), np.random.normal((1(
        y = np.concatenate((np.ones(n), np.zeros(n)),axis=0)

    elif trainset == 2:
        # Not Linearly Separable
        X = np.concatenate((np.random.normal((0,0),1,(n,2)), np.random.normal((1(
        y = np.concatenate((np.ones(2*n), np.zeros(2*n)), axis=0)

    else:
        print("function single_layer_training_data undefined for input", trainse
        sys.exit()

    return (X,y)

def single_layer_training(k, eta, trainset):
    w=np.random.normal(0,1,2)
    w1=w[0]
    w2=w[1]
    b=0
    X,y=single_layer_training_data(trainset)
    Loss_values=[]
    epochs=[]
    for i in range (k):
        dL_dw1=0
        dL_dw2=0
        dL_db=0
        Loss_function=0
        for j in range (20):
            sigmoid=1/(1+np.exp(-(w1*X[j][0]+w2*X[j][1]+b)))
            Loss_function+=-(y[j]*math.log(sigmoid)+(1-y[j])*(math.log(1-sigmoid
            dL_dw1+=X[j][0]*((sigmoid-(y[j])))
            dL_dw2+=X[j][1]*((sigmoid-(y[j])))
            dL_db=dL_db+(sigmoid-y[j])
        Loss_function=Loss_function/20
        Loss_values.append(Loss_function)
        epochs.append(i)
        dL_dw1=dL_dw1/20
        dL_dw2=dL_dw2/20
        dL_db=dL_db/20
        w1=w1-(eta)*(dL_dw1)
        w2=w2-(eta)*(dL_dw2)
        b=b-(eta)*(dL_db)
    theta=[w1,w2,b]
    plt.plot(epochs,Loss_values)
    return theta
```

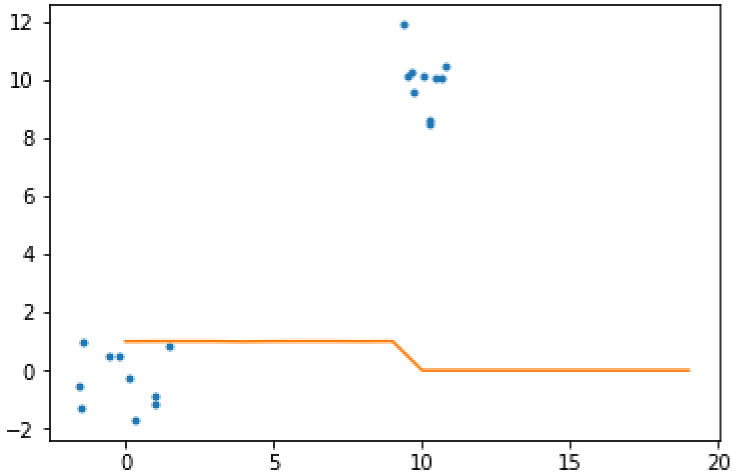
```
def single_layer_testing(theta, X):
    y=[]
    x1=[]
    x2=[]
    for i in range (np.size(X,0)):
        w1=theta[0]
        w2=theta[1]
        b=theta[2]
        t=sigmoid=1/(1+np.exp(-(w1*X[i][0]+w2*X[i][1]+b)))
        y.append(t)
        print("The point (",X[i][0],",",X[i][1],") has probability ",t)
        x1.append(X[i][0])
        x2.append(X[i][1])

    plt.plot(x1,x2,'.')
    plt.plot(y,'-')
    return y
```

In [68]:
```
theta=single_layer_training(100000,0.03,1)
```

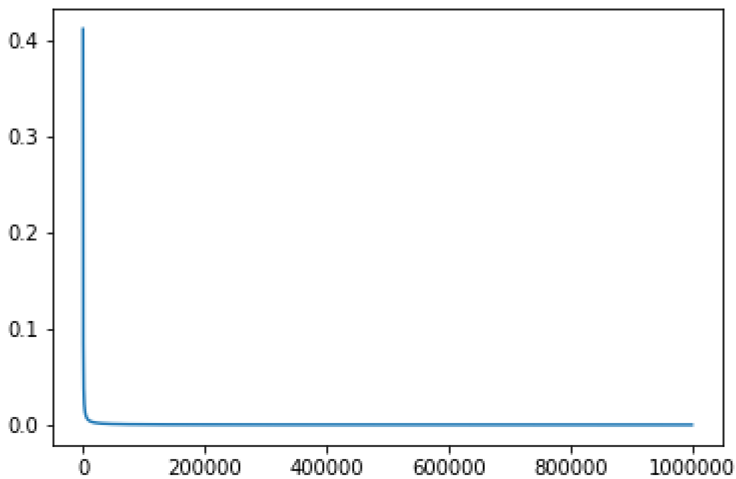


In [69]:
```
output=single_layer_testing(theta,single_layer_training_data(1)[0])
```

```
The point ( 0.9713701165690681 , -1.190080022008588 ) has probability  0.994010
6047944518
The point ( -0.550038559255467 , 0.49257604976448194 ) has probability  0.99961
89611457493
The point ( 0.14483431809977665 , -0.2422251435136072 ) has probability  0.9986
587281467791
The point ( -0.21597750148656672 , 0.49550273964529673 ) has probability  0.999
3120325997298
The point ( 1.4997579030339248 , 0.8529485039249879 ) has probability  0.986056
8541128296
The point ( 0.30851945095519406 , -1.7015502819429675 ) has probability  0.9981
007287022841
The point ( -1.559969963520815 , -0.5072393283830173 ) has probability  0.99993
35849433861
The point ( -1.4523409675215802 , 0.960171735448449 ) has probability  0.999924
2540081033
The point ( 0.9732190442917298 , -0.8918556992130664 ) has probability  0.99406
23155656307
The point ( -1.4727983259296797 , -1.2825402935281238 ) has probability  0.9999
200514319949
The point ( 9.714203679950991 , 9.594112657866141 ) has probability  4.87541607
3324236e-05
The point ( 9.371341982471995 , 11.88737358448934 ) has probability  9.80841931
107156e-05
The point ( 10.422539604550696 , 10.04759923443162 ) has probability  1.4173309
249766304e-05
The point ( 10.015993738951444 , 10.149377164519551 ) has probability  2.922420
8100360393e-05
The point ( 10.80408958085794 , 10.490257642642216 ) has probability  7.3437952
02233672e-06
The point ( 10.279073866766451 , 8.60836388291944 ) has probability  1.72427273
8425474e-05
The point ( 9.495489236419338 , 10.125083083768894 ) has probability  7.3349301
22489498e-05
The point ( 10.22978987892053 , 8.497288517108629 ) has probability  1.87304462
1930928e-05
The point ( 10.631197599884098 , 10.017607550225502 ) has probability  9.784978
324810073e-06
The point ( 9.616848375290278 , 10.23992297288972 ) has probability  5.94460340
0583938e-05
```

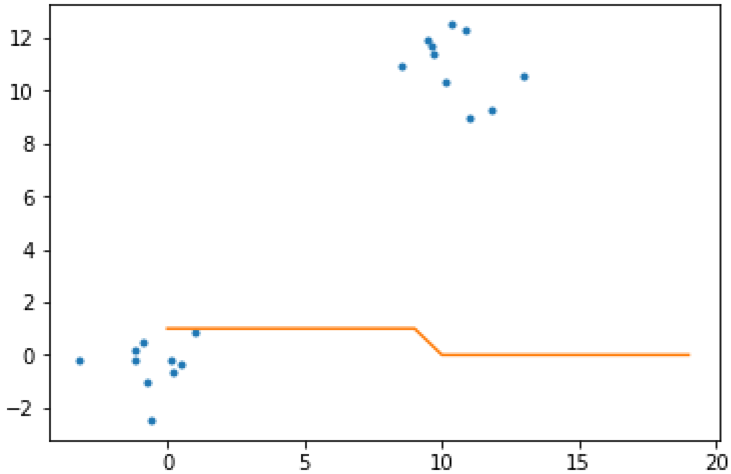


In [72]:
```
theta=single_layer_training(1000000,0.03,1)
```

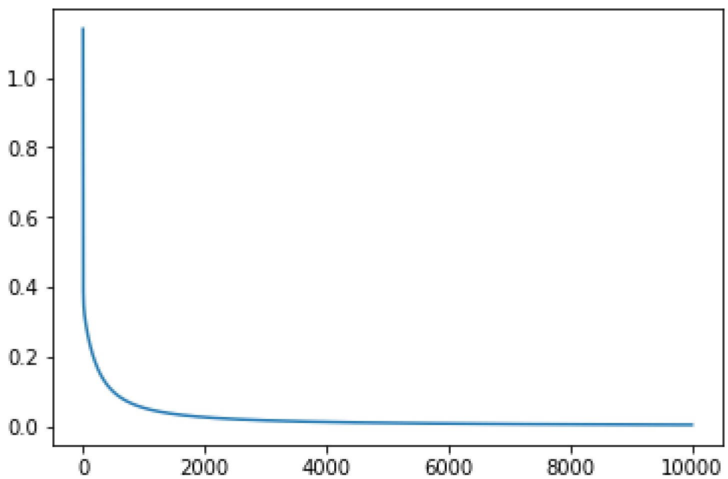


In [73]:
```
output=single_layer_testing(theta,single_layer_training_data(1)[0])
```

```
The point ( -0.7651093367329425 , -1.0567417856945172 ) has probability  0.9999
971721228633
The point ( -3.2165652485115452 , -0.1638171770482597 ) has probability  0.9999
997559934766
The point ( 0.9834954318855981 , 0.8610415829849435 ) has probability  0.999771
5929796185
The point ( -0.9171461678317312 , 0.46219701385917145 ) has probability  0.9999
889503414754
The point ( -1.180797133780758 , -0.2315849060456333 ) has probability  0.99999
62512652153
The point ( 0.11136273500023258 , -0.20565191410141284 ) has probability  0.999
9771984480058
The point ( 0.5066728877519454 , -0.36907021888654146 ) has probability  0.9999
668249595115
The point ( -0.5839767130225613 , -2.463089927484785 ) has probability  0.99999
91536051759
The point ( 0.21265452960732417 , -0.6824034602955346 ) has probability  0.9999
839965482177
The point ( -1.2066528804985197 , 0.20380398312537 ) has probability  0.9999943
227365555
The point ( 9.482490276565395 , 11.89947649874225 ) has probability  3.96973400
27566776e-07
The point ( 11.037754022930251 , 8.943957418446175 ) has probability  9.9445286
1922304e-07
The point ( 11.805867733144265 , 9.283477692751106 ) has probability  2.4310951
197526673e-07
The point ( 12.993861536096915 , 10.557222133118161 ) has probability  1.267885
692620274e-08
The point ( 10.154760983636432 , 10.360041497402626 ) has probability  7.742470
148112459e-07
The point ( 9.700059602750414 , 11.383133270252824 ) has probability  5.0211498
22156812e-07
The point ( 10.320696575175425 , 12.506948623854477 ) has probability  6.677778
77943406e-08
The point ( 8.52118049465189 , 10.914022049494362 ) has probability  4.13413785
9763449e-06
The point ( 9.653555321218505 , 11.715922168376594 ) has probability  3.7932462
428677484e-07
The point ( 10.83457177334059 , 12.32638648670944 ) has probability  3.96835626
1744105e-08
```



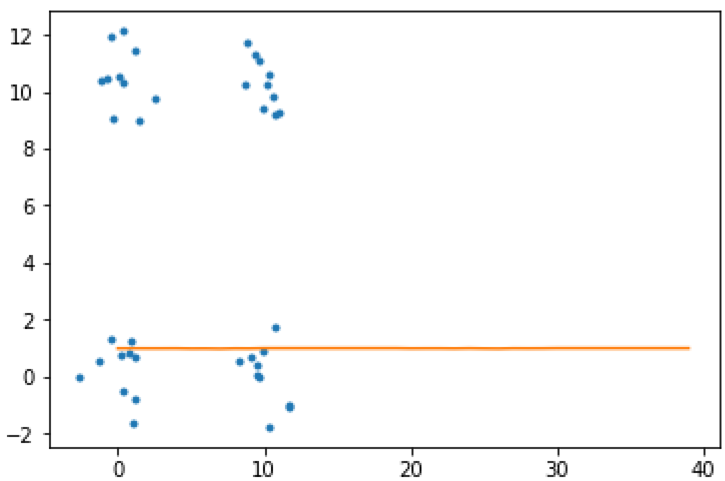
In [70]:
```
theta=single_layer_training(10000,0.02,2)
```



In [71]:
```
out=single_layer_testing(theta,single_layer_training_data(2)[0])
```

```
The point ( 0.7169427524819387 , 0.8479018896728545 ) has probability  0.993114
8446508865
The point ( -0.4537689929686338 , 1.3396006760373296 ) has probability  0.99426
2281765022
The point ( 0.19691065844210176 , 0.7502541710418565 ) has probability  0.99271
43018630233
The point ( 1.1354469667258351 , 0.6825009748467065 ) has probability  0.992684
6488257663
The point ( 0.891723652995416 , 1.2641081557754588 ) has probability  0.9942828
540932764
The point ( 1.146948602557443 , -0.8185411493741243 ) has probability  0.985981
4777498828
The point ( 0.36864980350430704 , -0.49521553589715456 ) has probability  0.987
5519494971964
The point ( 1.0013344526696404 , -1.6112646852377985 ) has probability  0.98019
96368194885
The point ( -1.3038865599854637 , 0.5492446675003941 ) has probability  0.99171
97591743357
The point ( -2.6359870346821928 , -0.01071102848893917 ) has probability  0.989
0562104176491
The point ( 9.591581700193046 , 11.082421982437285 ) has probability  0.9999384
984613565
The point ( 9.931198554881734 , 9.419887696795662 ) has probability  0.99987378
71197905
The point ( 10.96034468383559 , 9.258393275526702 ) has probability  0.99986828
41495851
The point ( 10.573533028727041 , 9.827995042535433 ) has probability  0.9998962
773605891
The point ( 10.196211308548875 , 10.270704461277166 ) has probability  0.999913
6760972713
The point ( 10.729414786354292 , 9.21597261176965 ) has probability  0.99986496
68326289
The point ( 9.42966704538955 , 11.27518104047817 ) has probability  0.999943227
7447359
The point ( 8.726067761162343 , 10.243705488828843 ) has probability  0.9999090
775909201
The point ( 10.390777596243552 , 10.640025867025496 ) has probability  0.999926
957332141
The point ( 8.853275942621753 , 11.716613128290554 ) has probability  0.9999524
675734034
The point ( 9.71911429015428 , -0.04627135960667561 ) has probability  0.992039
1086736039
The point ( 9.468826321689814 , 0.37432290352169734 ) has probability  0.993324
6040825435
The point ( 8.225270280360954 , 0.5760931658412668 ) has probability  0.9936763
068450124
The point ( 11.772752842325247 , -0.995985518608974 ) has probability  0.988628
5028078841
The point ( 10.73889178905942 , 1.7785644305858967 ) has probability  0.9965029
951757229
The point ( 11.757760232913252 , -1.047768662181464 ) has probability  0.988365
9587187354
The point ( 10.327321605217861 , -1.7653987223837502 ) has probability  0.98351
66218686502
The point ( 9.883518535940564 , 0.892445414955166 ) has probability  0.99473232
```

```
48297372
The point ( 9.448670546084621 , 0.08814172837850075 ) has probability  0.992435
7201725083
The point ( 9.09199502627472 , 0.723283794524721 ) has probability  0.994206506
5331445
The point ( 1.4387692124874176 , 9.006937760864508 ) has probability  0.9998093
75721506
The point ( 0.40271743869838683 , 12.135824862966203 ) has probability  0.99995
01927848274
The point ( -0.44149314810064866 , 11.961206721918838 ) has probability  0.9999
449822335575
The point ( 1.1954219944569853 , 11.452947816399599 ) has probability  0.999934
2637464932
The point ( -0.3310169357509503 , 9.031323528281975 ) has probability  0.999802
0800695967
The point ( 0.36965818834833286 , 10.296876003951937 ) has probability  0.99988
84465012436
The point ( -1.1437969935021874 , 10.384267332320112 ) has probability  0.99988
81136657027
The point ( 0.09414212883213804 , 10.542553037805854 ) has probability  0.99989
90714036166
The point ( -0.8118207207579594 , 10.444089832917452 ) has probability  0.99989
19889591712
The point ( 2.5468115751299294 , 9.73977687350097 ) has probability  0.99986582
40089563
```

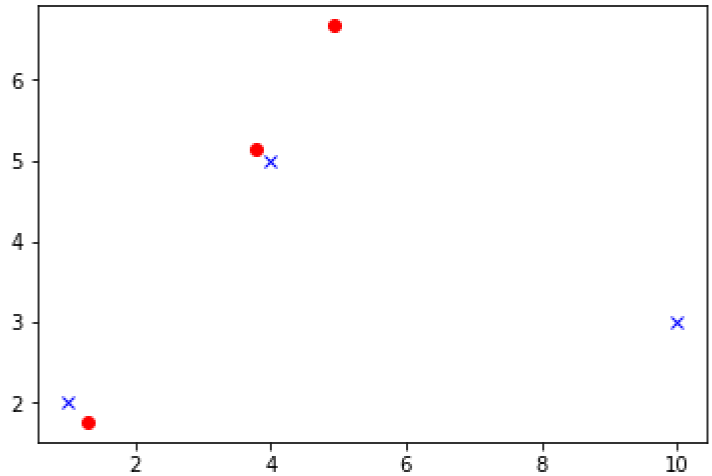


In [22]:
```
import numpy as np
import sys
import matplotlib.pyplot as plt
###Problem 3
###Provided function to create training data
def pca_training_data(n, sigma):
    m = 1
    b = 1
    x1 = np.random.uniform(0,10,n)
    x2 = m*x1+b
    X = np.array([x1,x2]).T
    X += np.random.normal(0,sigma,X.shape)
    return X

def pca_training(k, eta, n, sigma):
    x=pca_training_data(n,1)
    w=np.random.normal(0,1,4)
    w11=w[0]
    w12=w[1]
    w21=w[2]
    w22=w[3]
    b11=0
    b21=0
    b22=0
    theta=[]
    for i in range (k):
        theta_prev=theta
        dL_dw11=0
        dL_dw12=0
        dL_dw21=0
        dL_dw22=0
        dL_db11=0
        dL_db21=0
        dL_db22=0
        for j in range (n):
            h=w11*x[j][0]+w12*x[j][1]+b11
            z1=w21*h+b21
            z2=w22*h+b22
            dL_dw11+=2*(x[j][0]*w21)*(z1-x[j][0])+2*(x[j][0]*w22)*(z2-x[j][1])
            dL_dw12+=2*(x[j][1]*w21)*(z1-x[j][0])+2*(x[j][1]*w22)*(z2-x[j][1])
            dL_db11+=2*w21*(z1-x[j][0])+2*w22*(z2-x[j][1])
            dL_dw21=2*(z1-x[j][0])*h
            dL_dw22=2*(z2-x[j][1])*h
            dL_db21=2*(z1-x[j][0])
            dL_db22=2*(z2-x[j][1])
        dL_dw11=dL_dw11/n
        dL_dw12=dL_dw12/n
        dL_dw21=dL_dw21/n
        dL_dw22=dL_dw22/n
        dL_db11=dL_db11/n
        dL_db21=dL_db21/n
        dL_db22=dL_db22/n
        w11=w11-(eta)*(dL_dw11)
        w12=w12-(eta)*(dL_dw12)
        w21=w21-(eta)*(dL_dw21)
        w22=w22-(eta)*(dL_dw22)
```

```
        b11=b11-(eta)*(dL_db11)
        b21=b21-(eta)*(dL_db21)
        b22=b22-(eta)*(dL_db22)
        theta=[w11,w12,b11,w21,w22,b21,b22]
        if theta==theta_prev:
            print("Convergence at ",k," epoch.")
            break
    return theta

def pca_test(theta, X):
    l=[]
    w11=theta[0]
    w12=theta[1]
    b11=theta[2]
    w21=theta[3]
    w22=theta[4]
    b21=theta[5]
    b22=theta[6]
    Z1=[]
    Z2=[]
    XX1=[]
    XX2=[]
    percent1=0
    percent2=0
    for i in range(np.size(X,0)):
        h = w11*X[i][0] + w12*X[i][1] + b11
        z1 = w21*h + b21
        z2 = w22*h + b22
        Z1.append(z1)
        Z2.append(z2)
        XX1.append(X[i][0])
        XX2.append(X[i][1])
        per1=100-((abs(X[i][0]-z1)/X[i][0])*100)
        percent1+=100-((abs(X[i][0]-z1)/X[i][0])*100)
        per2=100-((abs(X[i][1]-z2)/X[i][1])*100)
        percent2+=100-((abs(X[i][1]-z2)/X[i][1])*100)
        print(" X1=",X[i][0]," z1= ",z1," accuracy=",per1," & X2=",X[i][1]," z2=
    percent1=percent1/(np.size(X,0))
    percent2=percent2/(np.size(X,0))
    print("The overall accuracy percentage of the algorithm for x1,z1=",percent1
    plt.plot(XX1,XX2,'bx')
    plt.plot(Z1,Z2,'ro')
    Z=[Z1,Z2]
    return Z
```

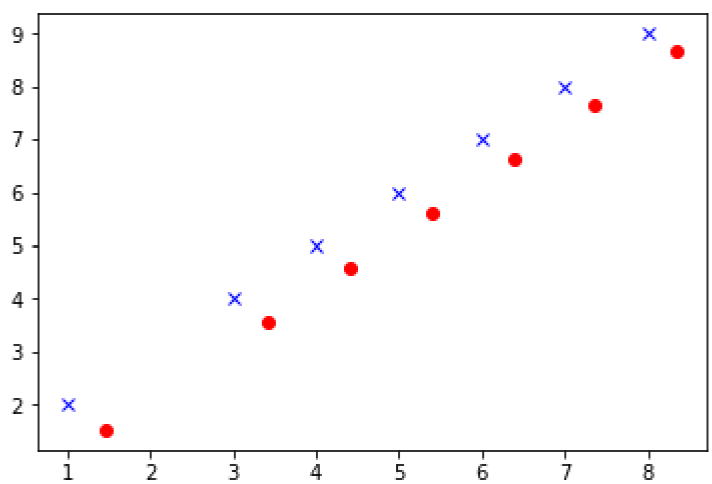
In [116]:
```
X=pca_training_data(10,0)
t=pca_training(300, 0.01, 100, 0)
a=pca_test(t, [[1,2], [4,5], [10, 3]])
```

```
 X1= 1  z1=  1.3067933520328963  accuracy= 69.32066479671037  & X2= 2  z2=  1.7
599288863636309  accuracy=  87.99644431818155 %
 X1= 4  z1=  3.8001378090115194  accuracy= 95.00344522528799  & X2= 5  z2=  5.1
41291187719788  accuracy=  97.17417624560424 %
 X1= 10  z1=  4.930248590614222  accuracy= 49.30248590614222  & X2= 3  z2=  6.6
73896914366198  accuracy=  -22.46323047887327 %
The overall accuracy percentage of the algorithm for x1,z1= 71.2088653093802  a
nd for x2,z2= 54.235796694970844
```



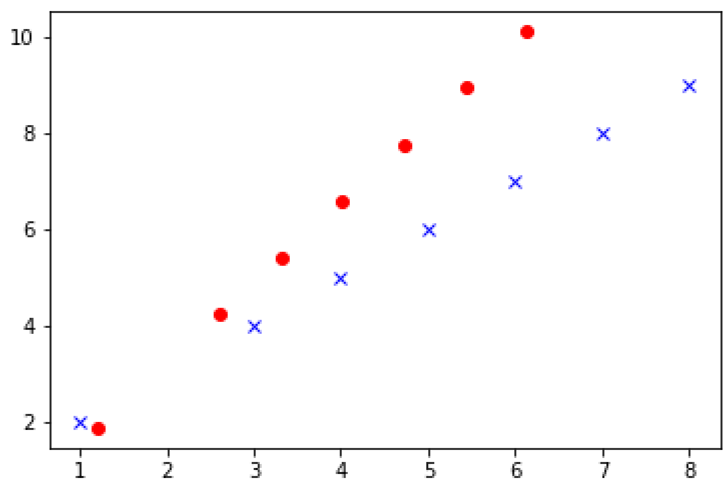
In [146]:
```
X=[[1,2],[3,4],[4,5],[5,6],[6,7],[7,8],[8,9]]
t=pca_training(1000, 0.01, 100, 0)
a=pca_test(t,X)
```

```
 X1= 1  z1=  1.4616559673238056  accuracy= 53.834403267619436  & X2= 2  z2=  1.5184494425129065  accuracy=  75.92247212564533 %
 X1= 3  z1=  3.4304840595210546  accuracy= 85.65053134929818  & X2= 4  z2=  3.561206971948219  accuracy=  89.03017429870548 %
 X1= 4  z1=  4.41489810561968  accuracy= 89.627547359508  & X2= 5  z2=  4.582585736665876  accuracy=  91.65171473331752 %
 X1= 5  z1=  5.399312151718304  accuracy= 92.01375696563392  & X2= 6  z2=  5.6039645013835315  accuracy=  93.39940835639219 %
 X1= 6  z1=  6.383726197816928  accuracy= 93.60456336971787  & X2= 7  z2=  6.625343266101187  accuracy=  94.64776094430268 %
 X1= 7  z1=  7.368140243915552  accuracy= 94.74085365834925  & X2= 8  z2=  7.646722030818843  accuracy=  95.58402538523553 %
 X1= 8  z1=  8.352554290014178  accuracy= 95.59307137482278  & X2= 9  z2=  8.6681007955365  accuracy=  96.31223106151667 %
The overall accuracy percentage of the algorithm for x1,z1= 86.43781819213564 and for x2,z2= 90.93539812930219
```



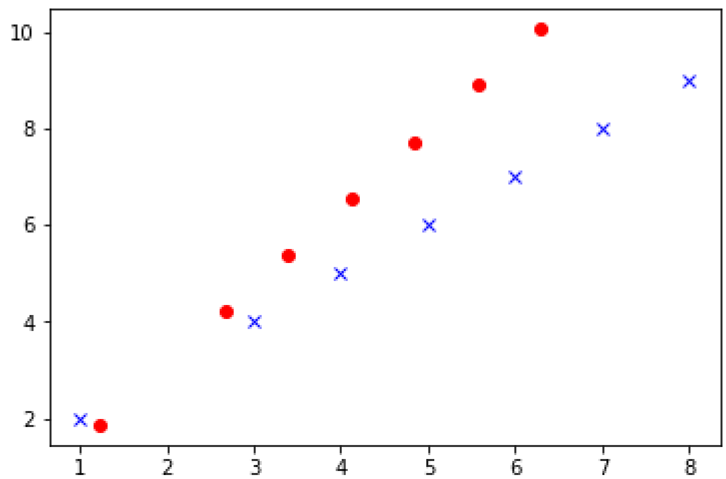
In [124]:
```
t=pca_training(1000, 0.01, 100, 0.1)
a=pca_test(t,X)
```

```
 X1= 1  z1=  1.2057679488880724  accuracy= 79.42320511119277  & X2= 2  z2=  1.8
849198026229919  accuracy=  94.2459901311496 %
 X1= 3  z1=  2.6133154181865534  accuracy= 87.11051393955178  & X2= 4  z2=  4.2
36209675710355  accuracy=  94.09475810724113 %
 X1= 4  z1=  3.3170891528357944  accuracy= 82.92722882089487  & X2= 5  z2=  5.4
11854612254038  accuracy=  91.76290775491924 %
 X1= 5  z1=  4.020862887485035  accuracy= 80.4172577497007  & X2= 6  z2=  6.587
499548797719  accuracy=  90.20834085337135 %
 X1= 6  z1=  4.724636622134275  accuracy= 78.74394370223791  & X2= 7  z2=  7.76
31444853414  accuracy=  89.0979359236943 %
 X1= 7  z1=  5.428410356783515  accuracy= 77.54871938262164  & X2= 8  z2=  8.93
8789421885081  accuracy=  88.26513222643648 %
 X1= 8  z1=  6.132184091432756  accuracy= 76.65230114290945  & X2= 9  z2=  10.1
14434358428763  accuracy=  87.61739601745819 %
The overall accuracy percentage of the algorithm for x1,z1= 80.40330997844417
and for x2,z2= 90.75606585918148
```



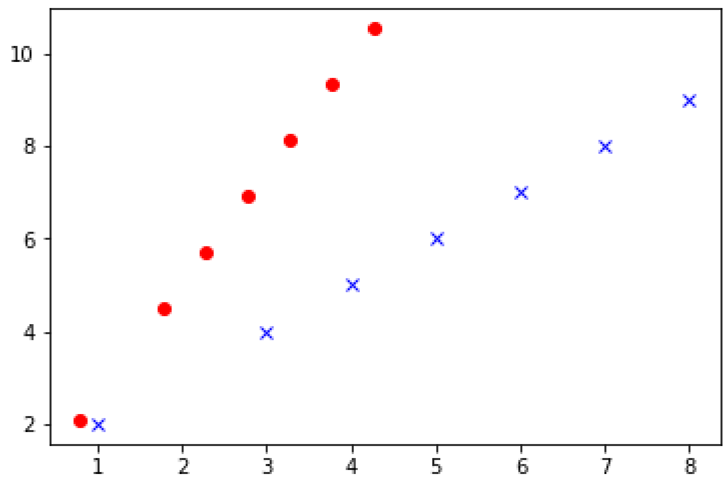
In [147]:
```
X=[[1,2],[3,4],[4,5],[5,6],[6,7],[7,8],[8,9]]
t=pca_training(1500, 0.01, 100, 0)
a=pca_test(t,X)
```

```
 X1= 1  z1=  1.2210120176261559  accuracy= 77.89879823738441  & X2= 2  z2=  1.8623644251443374  accuracy=  93.11822125721687 %
 X1= 3  z1=  2.6691206253205793  accuracy= 88.97068751068598  & X2= 4  z2=  4.204022497102738  accuracy=  94.89943757243155 %
 X1= 4  z1=  3.3931749291677913  accuracy= 84.82937322919479  & X2= 5  z2=  5.374851533081938  accuracy=  92.50296933836124 %
 X1= 5  z1=  4.117229233015004  accuracy= 82.34458466030007  & X2= 6  z2=  6.5456805690611395  accuracy=  90.90532384898101 %
 X1= 6  z1=  4.841283536862215  accuracy= 80.68805894770358  & X2= 7  z2=  7.71650960504034  accuracy=  89.76414849942371 %
 X1= 7  z1=  5.565337840709427  accuracy= 79.50482629584896  & X2= 8  z2=  8.88733864101954  accuracy=  88.90826698725576 %
 X1= 8  z1=  6.289392144556639  accuracy= 78.61740180695799  & X2= 9  z2=  10.05816767699874  accuracy=  88.24258136668065 %
The overall accuracy percentage of the algorithm for x1,z1= 81.83624724115369 and for x2,z2= 91.19156412433583
```



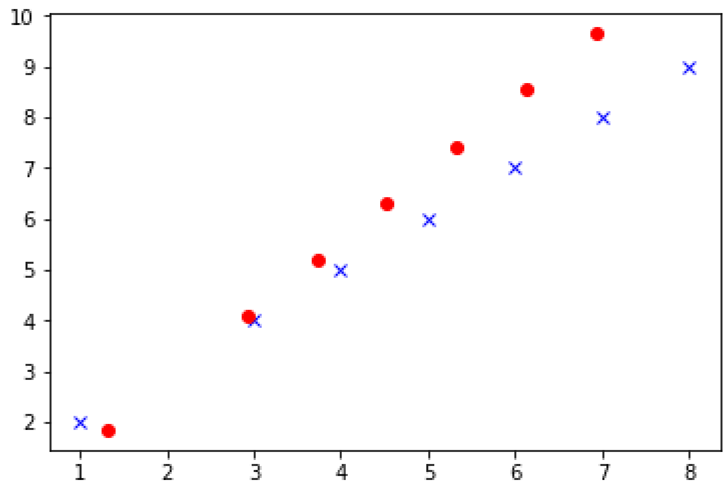
In [128]:
```
t=pca_training(1500, 0.01, 100, 0.1)
a=pca_test(t,X)
```

```
 X1= 1  z1=  0.7904523196245876  accuracy= 79.04523196245876  & X2= 2  z2=  2.0
988824426976462  accuracy=  95.0558778651177 %
 X1= 3  z1=  1.784557388866182  accuracy= 59.485246295539405  & X2= 4  z2=  4.5
09055633394691  accuracy=  87.27360916513271 %
 X1= 4  z1=  2.2816099234869798  accuracy= 57.0402480871745  & X2= 5  z2=  5.71
4142228743214  accuracy=  85.71715542513572 %
 X1= 5  z1=  2.778662458107777  accuracy= 55.573249162155534  & X2= 6  z2=  6.9
19228824091737  accuracy=  84.67951959847106 %
 X1= 6  z1=  3.275714992728574  accuracy= 54.59524987880957  & X2= 7  z2=  8.12
431541944026  accuracy=  83.93835115085344 %
 X1= 7  z1=  3.7727675273493713  accuracy= 53.896678962133876  & X2= 8  z2=  9.
329402014788784  accuracy=  83.3824748151402 %
 X1= 8  z1=  4.269820061970169  accuracy= 53.37275077462711  & X2= 9  z2=  10.5
34488610137306  accuracy=  82.95012655402994 %
The overall accuracy percentage of the algorithm for x1,z1= 59.001236446128395
and for x2,z2= 86.14244493912581
```



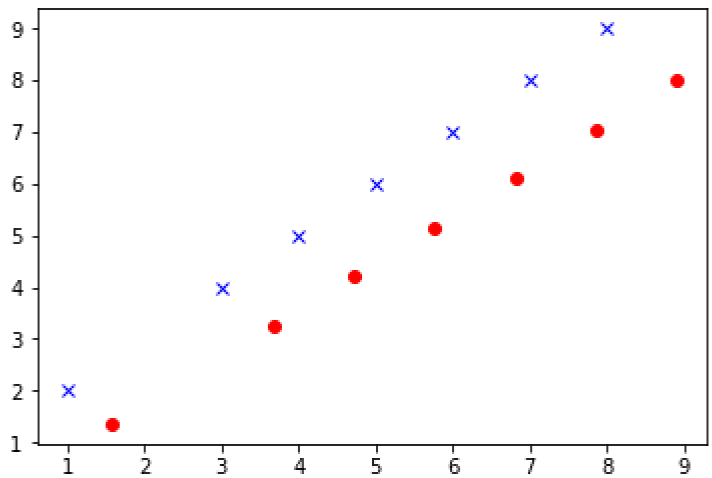
In [143]:
```
X=[[1,2],[3,4],[4,5],[5,6],[6,7],[7,8],[8,9]]
t=pca_training(1700, 0.01, 100, 0)
a=pca_test(t,X)
```

```
 X1= 1  z1=  1.3268107042177528  accuracy= 67.31892957822473  & X2= 2  z2=  1.8
397042337558847  accuracy=  91.98521168779423 %
 X1= 3  z1=  2.9280319504608516  accuracy= 97.60106501536173  & X2= 4  z2=  4.0
69923314371672  accuracy=  98.2519171407082 %
 X1= 4  z1=  3.7286425735824005  accuracy= 93.21606433956  & X2= 5  z2=  5.1850
32854679565  accuracy=  96.2993429064087 %
 X1= 5  z1=  4.5292531967039515  accuracy= 90.58506393407903  & X2= 6  z2=  6.3
0014239498746  accuracy=  94.99762675020901 %
 X1= 6  z1=  5.329863819825501  accuracy= 88.83106366375836  & X2= 7  z2=  7.41
52519352953545  accuracy=  94.06782949578064 %
 X1= 7  z1=  6.130474442947051  accuracy= 87.57820632781501  & X2= 8  z2=  8.53
0361475603248  accuracy=  93.3704815549594 %
 X1= 8  z1=  6.931085066068599  accuracy= 86.63856332585749  & X2= 9  z2=  9.64
5471015911141  accuracy=  92.82809982320954 %
The overall accuracy percentage of the algorithm for x1,z1= 87.39556516923663
and for x2,z2= 94.54292990843852
```



In [144]:
```
t=pca_training(1700, 0.01, 100, 0.1)
a=pca_test(t,X)
```

```
 X1= 1  z1=  1.5807278467538206  accuracy= 41.92721532461794  & X2= 2  z2=  1.3
6042205280957  accuracy=  68.0211026404785 %
 X1= 3  z1=  3.672795595753444  accuracy= 77.57348014155187  & X2= 4  z2=  3.25
88102829678878  accuracy=  81.47025707419719 %
 X1= 4  z1=  4.718829470253257  accuracy= 82.02926324366857  & X2= 5  z2=  4.20
8004398047047  accuracy=  84.16008796094093 %
 X1= 5  z1=  5.764863344753069  accuracy= 84.70273310493863  & X2= 6  z2=  5.15
7198513126207  accuracy=  85.95330855210344 %
 X1= 6  z1=  6.8108972192528805  accuracy= 86.48504634578532  & X2= 7  z2=  6.1
06392628205365  accuracy=  87.23418040293379 %
 X1= 7  z1=  7.856931093752694  accuracy= 87.75812723210437  & X2= 8  z2=  7.05
55867432845245  accuracy=  88.19483429105655 %
 X1= 8  z1=  8.902964968252506  accuracy= 88.71293789684367  & X2= 9  z2=  8.00
4780858363684  accuracy=  88.94200953737428 %
The overall accuracy percentage of the algorithm for x1,z1= 78.4555433270729  a
nd for x2,z2= 83.42511149415496
```



In [ ]: