In [1]:
```
import numpy as np

import matplotlib.pyplot as plt
```

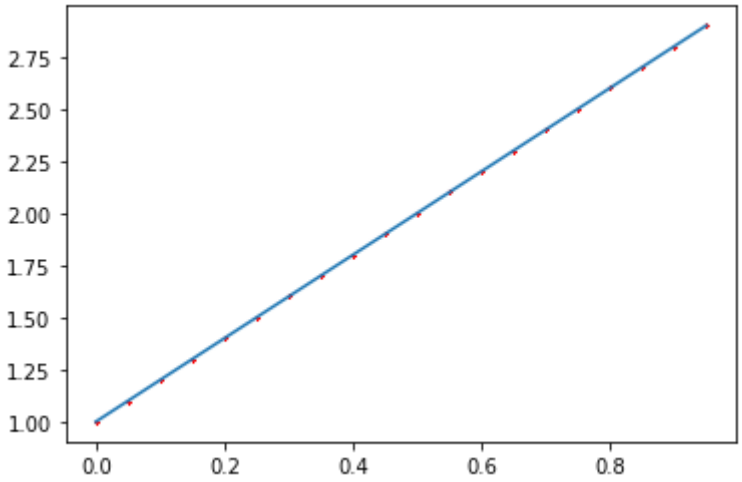
This tiny project Aims to practice with expectation–maximization algorithm, using iterative way to find max likelihood of random generated-data

In [2]:
```
# calculate perpendicular distance of 2d Point
def R(points_List):
#     theta = np.random.normal(0,0.1,3)
    n = len(points_List)
    xsum = 0
    ysum = 0
    x2 = 0
    xy = 0
    for point in points_List:
        xsum+= point[0]
        ysum+= point[1]
        x2+=point[0]**2
        xy+=point[1]*point[0]
    x = np.array([[x2,xsum],[xsum,n]])
    y = np.array([xy,ysum])
    d = np.linalg.solve(x,y)
#     d = solve((n*b + a*xsum - ysum, b*xsum + a*x2 - xy),a,b)

    return d[0],d[1]
```

Previsous function is a finding the best line of fit function, it use numpy linear algebra to find that line (easy and simple)

In [4]:
```
x = np.arange(0,1,0.05)
y = x*2+1
points=[]
for i in range(len(x)):
     points.append([x[i],y[i]])
     plt.scatter(x, y, color='red',s = 0.2)
# points
a, b = R(points)
plt.plot(x,x*a+b)
# plt.show()
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x116f75590>]
```



In [ ]:

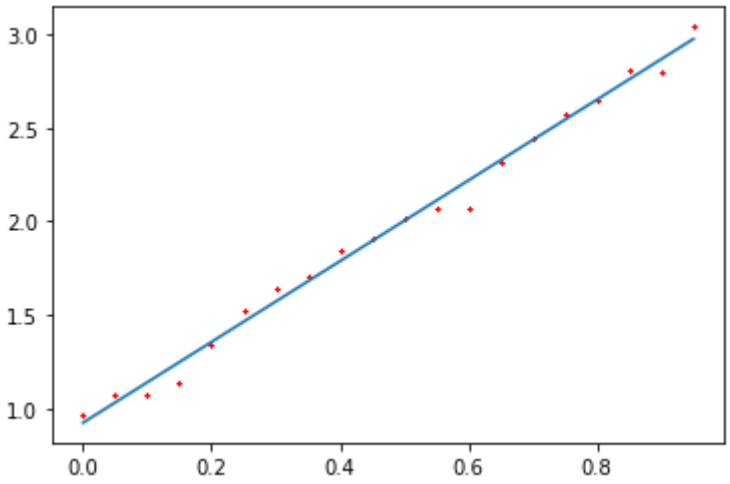
What if we add some noise to that data?

1: Random noise with normal distuibution

Our linear method still works fine!

In [8]:
```
x = np.arange(0,1,0.05)
y = x*2+1 + 0.1*np.random.normal(0,1,len(x))
points=[]
for i in range(len(x)):
     points.append([x[i],y[i]])
     plt.scatter(x, y, color='red',s = 0.2)
points
a, b = R(points)

plt.plot(x,x*a+b)
# plt.show()
# y=2*x+1+0.1*randn(size(x)).
```

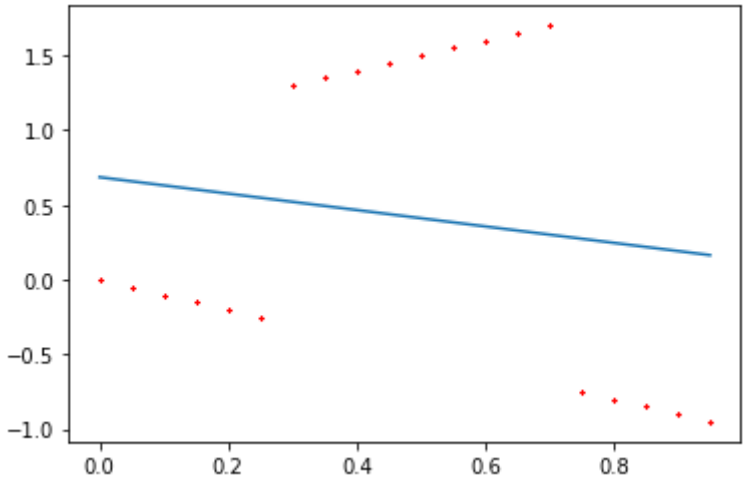
Out[8]:
```
[<matplotlib.lines.Line2D at 0x1173149d0>]
```



2 If the point are in two group, a simple line cannot fit to them

In [9]:
```
x = np.arange(0,1,0.05)
y = (abs(x-0.5) < 0.25) * (x+1) + (abs(x-0.5) >=0.25)*(-x)
points=[]
for i in range(len(x)):
     points.append([x[i],y[i]])
     plt.scatter(x, y, color='red',s = 0.2)
points
a, b = R(points)
plt.plot(x,x*a+b)
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x11741fe50>]
```



So we need to come up with an EM algorthium

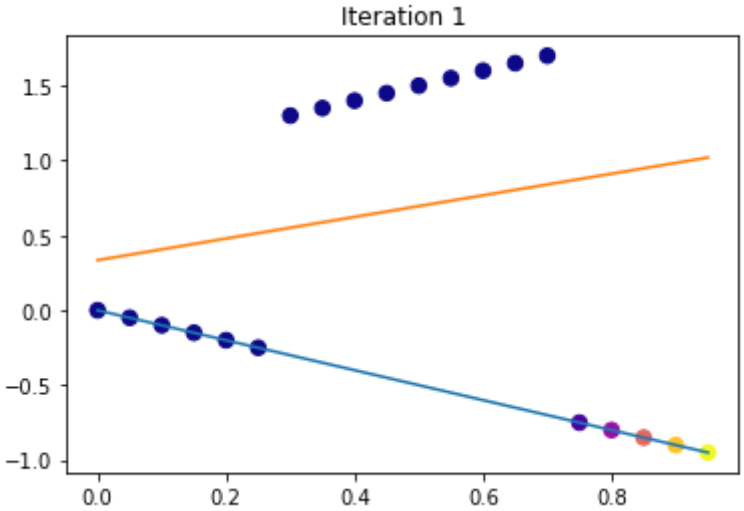
Basic idea, randomly initize two line, adjust their parameter in each iteraction

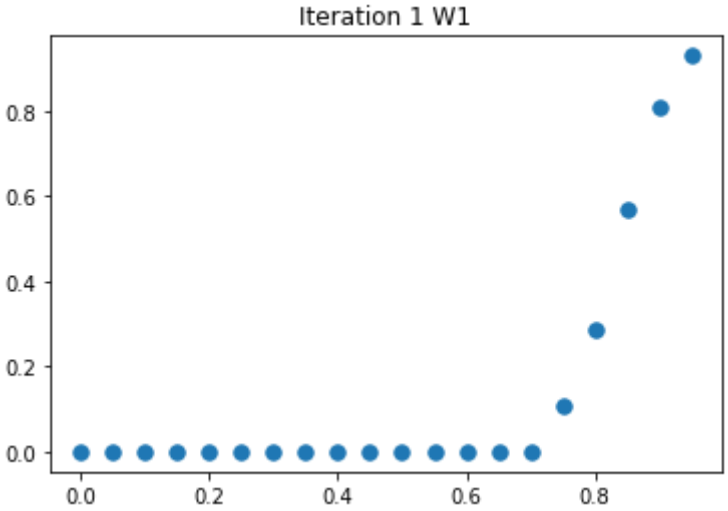
In [16]:
```
import numpy as np

def EM(points):

    a1,a2,b1,b2 = np.random.normal(0,1,4)
    for j in range(5):
        r1 = []
        r2 = []
        w1 = []
        w2 = []
        w1x2Sum = 0
        w1xSum = 0
        w1xySum = 0
        w1Sum = 0
        w1ySum = 0
        w2x2Sum = 0
        w2xSum = 0
        w2xySum = 0
        w2Sum = 0
        w2ySum = 0
        for i in range(len(points)):
            r1.append((a1*points[i][0] + b1 - points[i][1])**2)
            r2.append((a2*points[i][0] + b2 - points[i][1])**2)

            w1.append(np.exp(-r1[i]/0.1) / (np.exp(-r1[i]/0.1) + np.exp(
-r2[i]/0.1)))
            w2.append(1-w1[i])
            w1x2Sum += w1[i]*(points[i][0]**2)
            w1xSum += w1[i]*points[i][0]
            w1xySum += w1[i]*points[i][0]*points[i][1]
            w1Sum += w1[i]
            w1ySum += w1[i]*points[i][1]
            w2x2Sum += w2[i]*(points[i][0]**2)
            w2xSum += w2[i]*points[i][0]
            w2xySum += w2[i]*points[i][0]*points[i][1]
            w2Sum += w2[i]
            w2ySum += w2[i]*points[i][1]

        a1t = [[w1x2Sum,w1xSum],[w1xSum,w1Sum]]
        y1t = [w1xySum,w1ySum]
        a2t = [[w2x2Sum,w2xSum],[w2xSum,w2Sum]]
        y2t = [w2xySum,w2ySum]

        d1 = np.linalg.solve(a1t,y1t)
        d2 = np.linalg.solve(a2t,y2t)


        a1= d1[0]
        b1= d1[1]
        a2= d2[0]
        b2= d2[1]
        plt.scatter(x, y, c = w1, cmap = "plasma",s = 55)
        plt.plot(x,x*a1+b1)
        plt.plot(x,x*a2+b2)
        plt.title("Iteration " + str(j+1))
        plt.show()
```

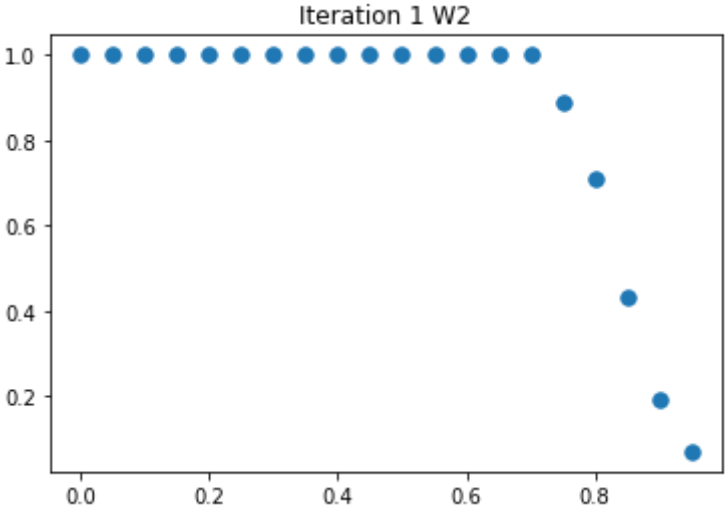
```
        plt.scatter(x, w1,s = 55)
        plt.title("Iteration " + str(j+1) + " W1")
        plt.show()

        plt.scatter(x, w2,s = 55)
        plt.title("Iteration " + str(j+1) + " W2")
        plt.show()

    return a1,b1,a2,b2
```

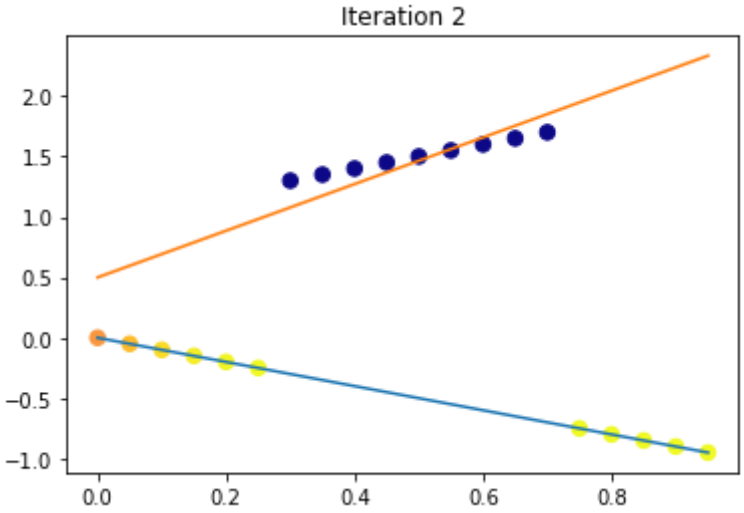
In [ ]:

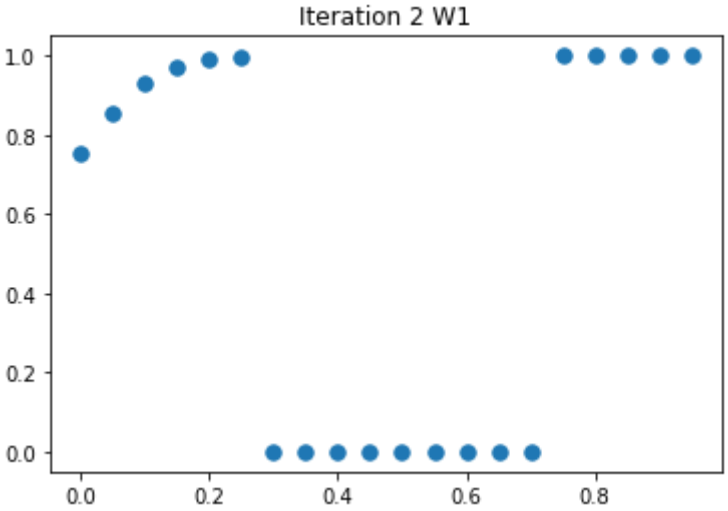
In [20]:
```
x = np.arange(0,1,0.05)
# y = (abs(x-0.5) < 0.25) * (x+1) + (abs(x-0.5) >=0.25)*(-x) + 0.1*np.ra
ndom.normal(1,20,len(x))
y = (abs(x-0.5) < 0.25) * (x+1) + (abs(x-0.5) >=0.25)*(-x)
points=[]
0.8*np.random.normal(1,6,len(x))
for i in range(len(x)):
     points.append([x[i],y[i]])
#      plt.scatter(x, y, color='red',s = 0.2)
EM(points)
# a1,b1,a2,b2 = EM(points)
# plt.plot(x,x*a1+b1)
# plt.plot(x,x*a2+b2)
```

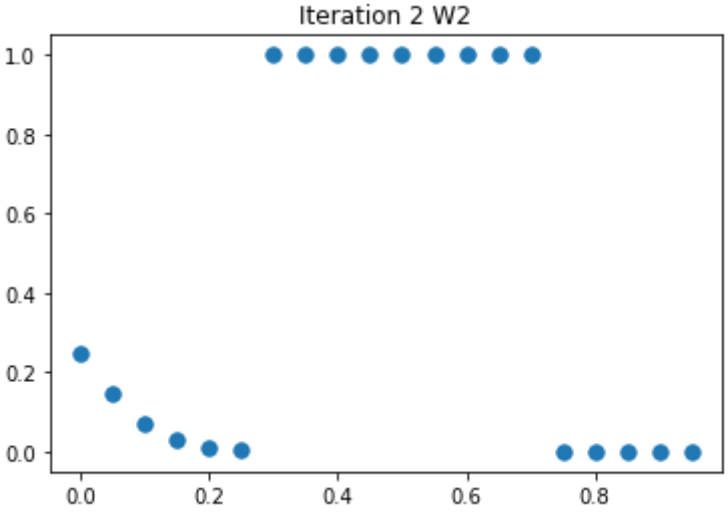

Iteration 1
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8

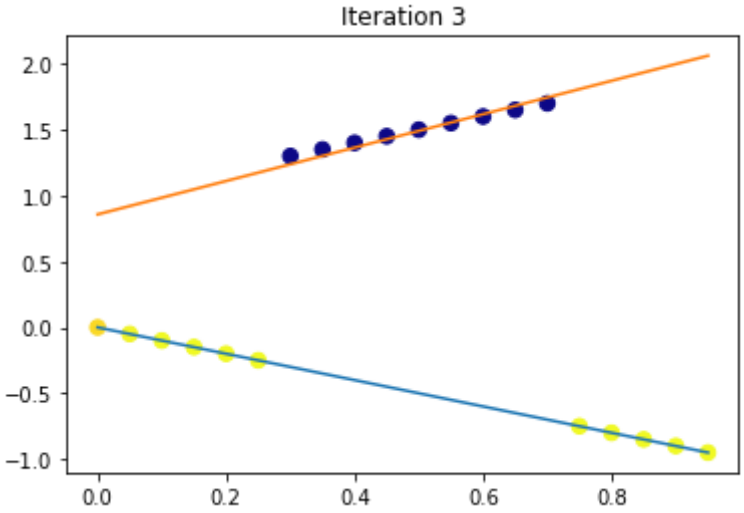

Iteration 1 W1
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8

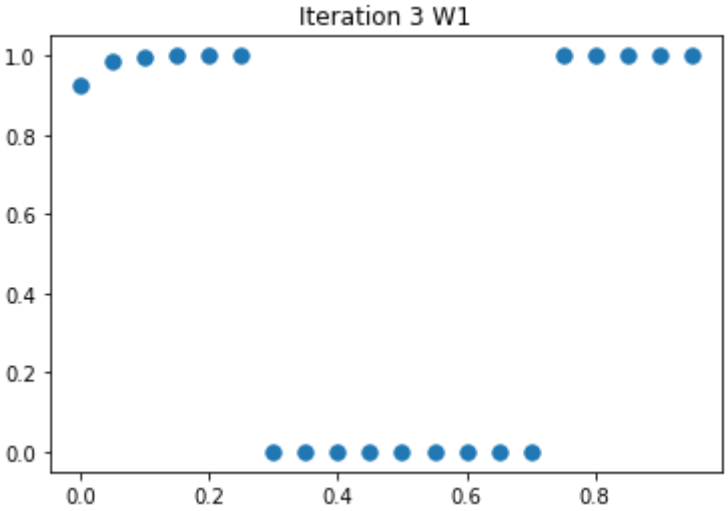

Iteration 1 W2
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8

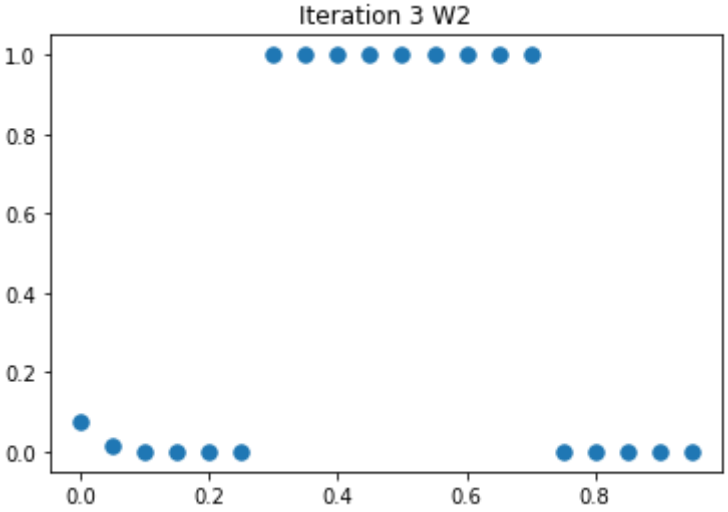

Iteration 2
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8

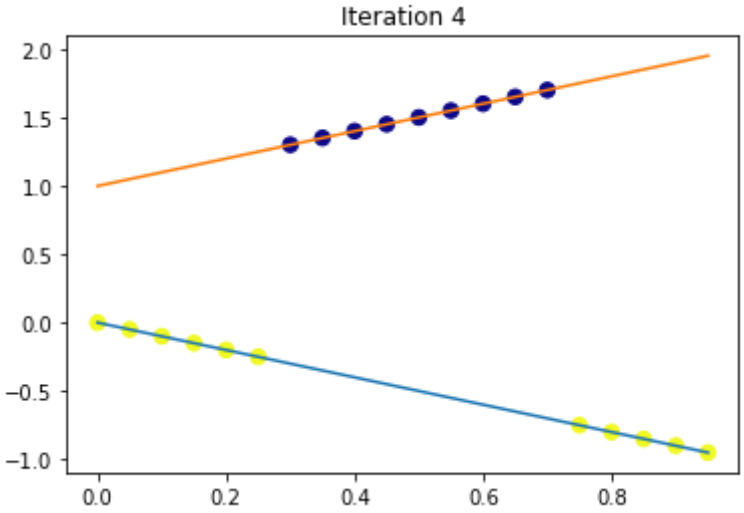

Iteration 2 W1
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8

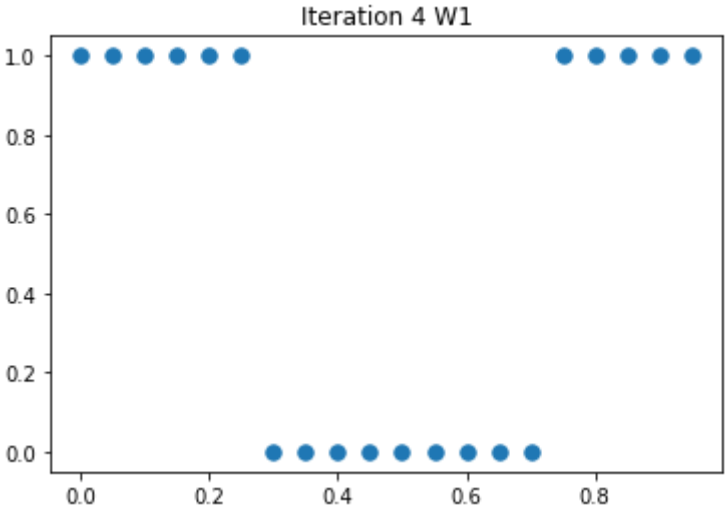

Iteration 2 W2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8

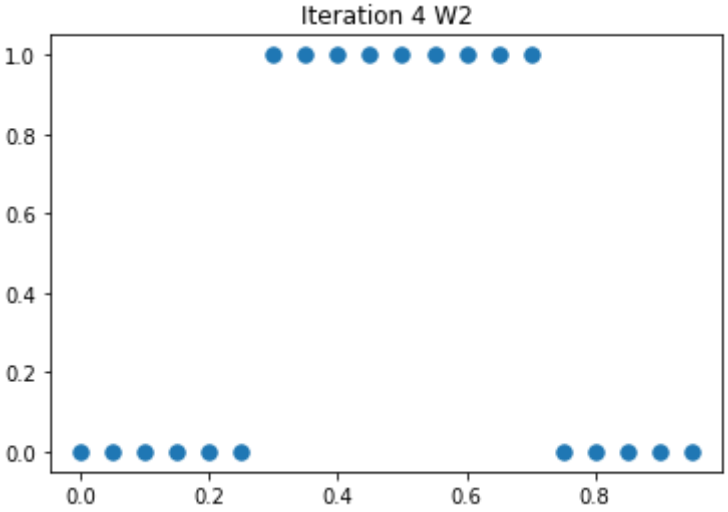

Iteration 3
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8

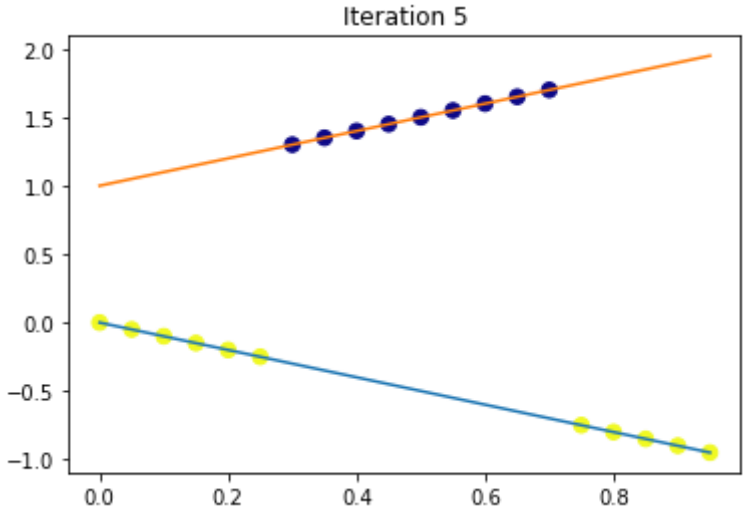

Iteration 3 W1
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8

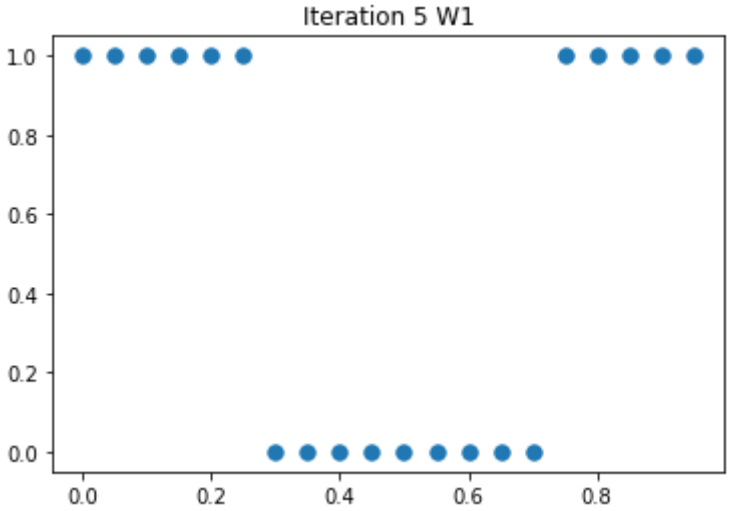

Iteration 3 W2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8

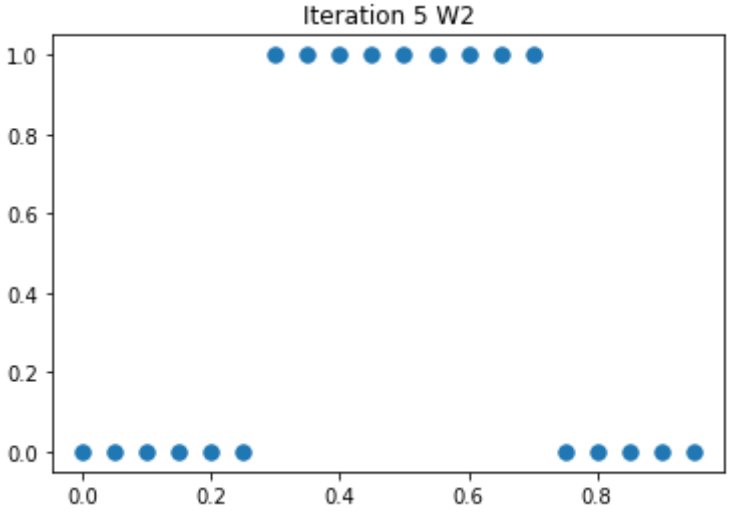

Iteration 4
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8


Iteration 4 W1
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8


Iteration 4 W2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8







Out[20]:
```
(-1.0000000000012892,
 1.739425955338704e-12,
 1.0001761079271911,
 0.999905988556335)
```

In [17]:
```
import numpy as np

def EM1(points,sigma):

    a1,a2,b1,b2 = np.random.normal(0,1,4)
    for j in range(5):
        r1 = []
        r2 = []
        w1 = []
        w2 = []
        w1x2Sum = 0
        w1xSum = 0
        w1xySum = 0
        w1Sum = 0
        w1ySum = 0
        w2x2Sum = 0
        w2xSum = 0
        w2xySum = 0
        w2Sum = 0
        w2ySum = 0
        for i in range(len(points)):
            r1.append((a1*points[i][0] + b1 - points[i][1])**2)
            r2.append((a2*points[i][0] + b2 - points[i][1])**2)

            w1.append(np.exp(-r1[i]/sigma) / (np.exp(-r1[i]/sigma) + np.
exp(-r2[i]/sigma)))
            w2.append(1-w1[i])
            w1x2Sum += w1[i]*(points[i][0]**2)
            w1xSum += w1[i]*points[i][0]
            w1xySum += w1[i]*points[i][0]*points[i][1]
            w1Sum += w1[i]
            w1ySum += w1[i]*points[i][1]
            w2x2Sum += w2[i]*(points[i][0]**2)
            w2xSum += w2[i]*points[i][0]
            w2xySum += w2[i]*points[i][0]*points[i][1]
            w2Sum += w2[i]
            w2ySum += w2[i]*points[i][1]

        a1t = [[w1x2Sum,w1xSum],[w1xSum,w1Sum]]
        y1t = [w1xySum,w1ySum]
        a2t = [[w2x2Sum,w2xSum],[w2xSum,w2Sum]]
        y2t = [w2xySum,w2ySum]

        d1 = np.linalg.solve(a1t,y1t)
        d2 = np.linalg.solve(a2t,y2t)


        a1= d1[0]
        b1= d1[1]
        a2= d2[0]
        b2= d2[1]
        plt.scatter(x, y, c = w1, cmap = "plasma",s = 55)
        plt.plot(x,x*a1+b1)
        plt.plot(x,x*a2+b2)
        plt.title("Iteration " + str(j+1))
        plt.show()
```

```
#         plt.scatter(x, w1,s = 55)
#         plt.title("Iteration " + str(j+1) + " W1")
#         plt.show()

#         plt.scatter(x, w2,s = 55)
#         plt.title("Iteration " + str(j+1) + " W2")
#         plt.show()

    return a1,b1,a2,b2
```

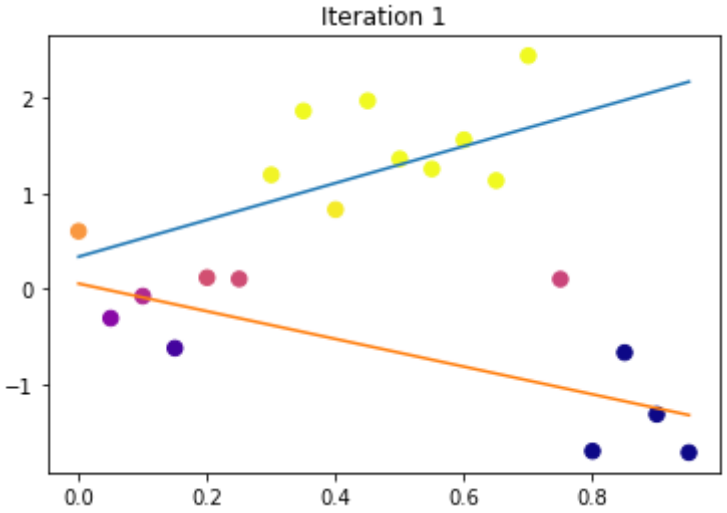
Noise sensitive?

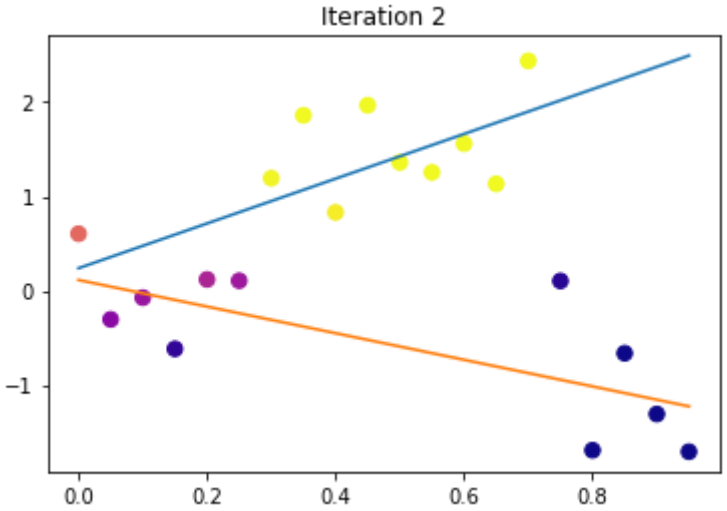
When noise varience * weight is larger than sigma^2 (the free parameter), algorthium will break.

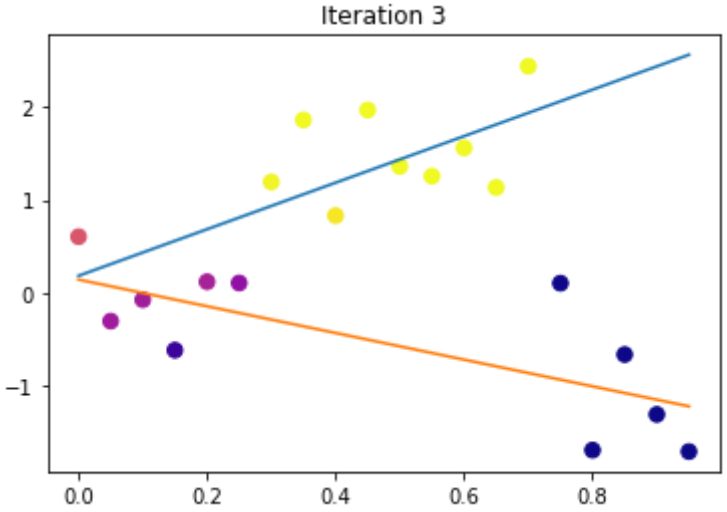
Below are three plot shows when noise have varience of 5, weight 0.1 and free parameter is 0.5. The result for points with noise is still idenitical

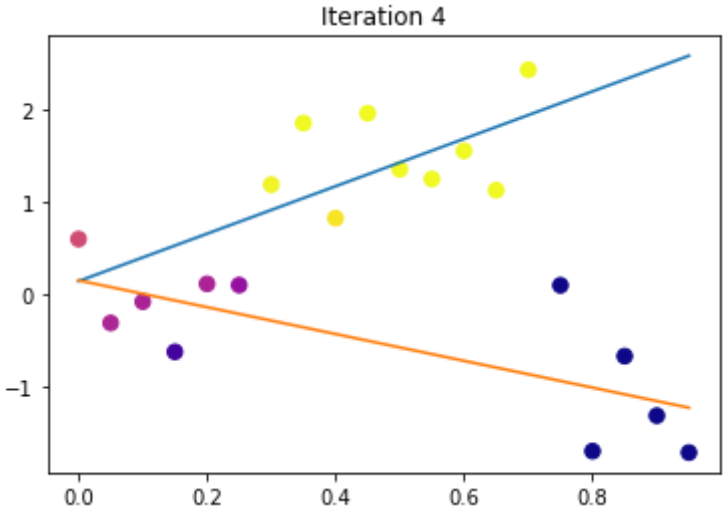
In [65]:

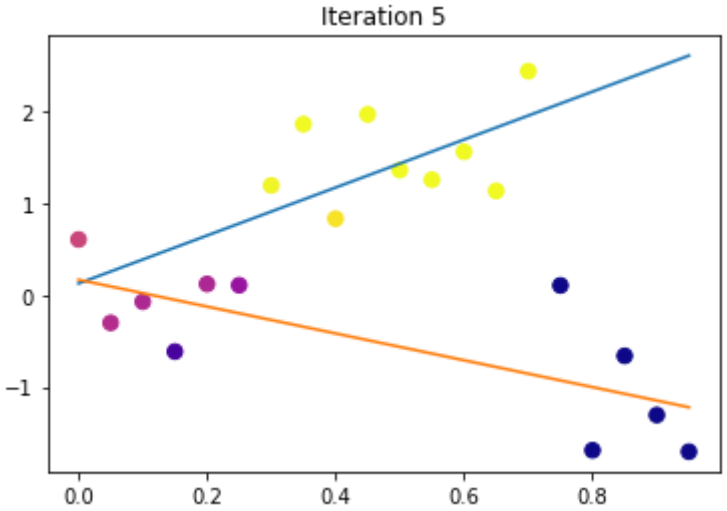
In [18]:
```
x = np.arange(0,1,0.05)
y = (abs(x-0.5) < 0.25) * (x+1) + (abs(x-0.5) >=0.25)*(-x) + 0.1*np.rand
om.normal(0,5,len(x))

points=[]
0.8*np.random.normal(1,6,len(x))
for i in range(len(x)):
     points.append([x[i],y[i]])

EM1(points,0.5)
```


Iteration 1
2
1
0
−1
0.0
0.2
0.4
0.6
0.8


Iteration 2
2
1
0
−1
0.0
0.2
0.4
0.6
0.8

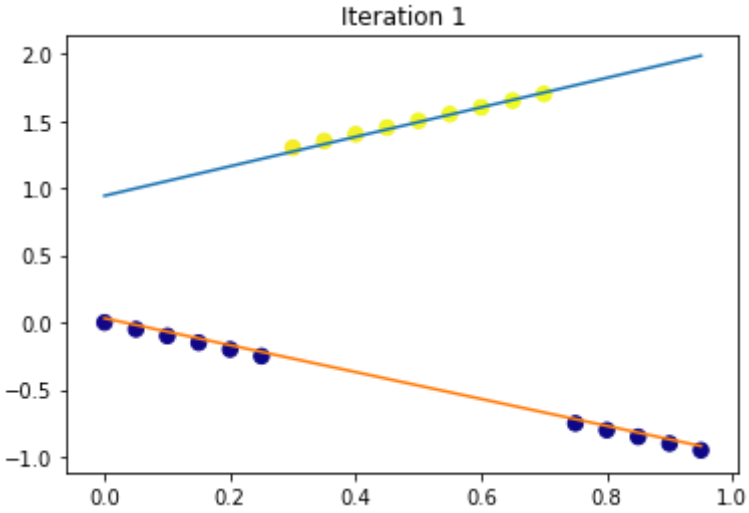

Iteration 3
2
1
0
−1
0.0
0.2
0.4
0.6
0.8

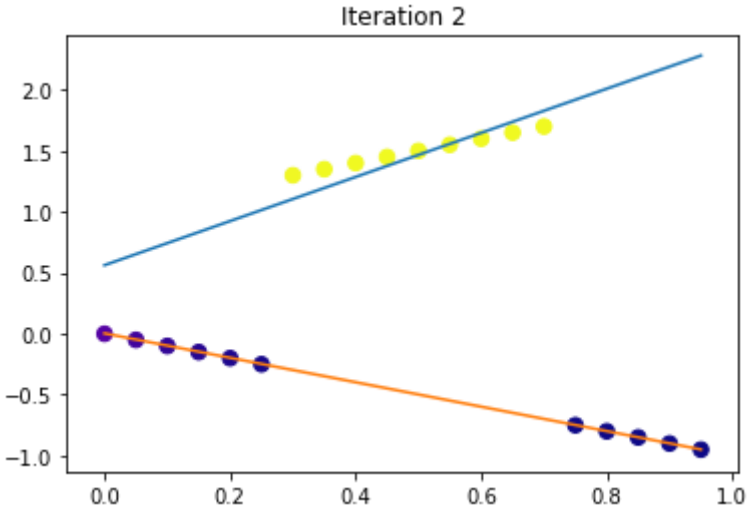


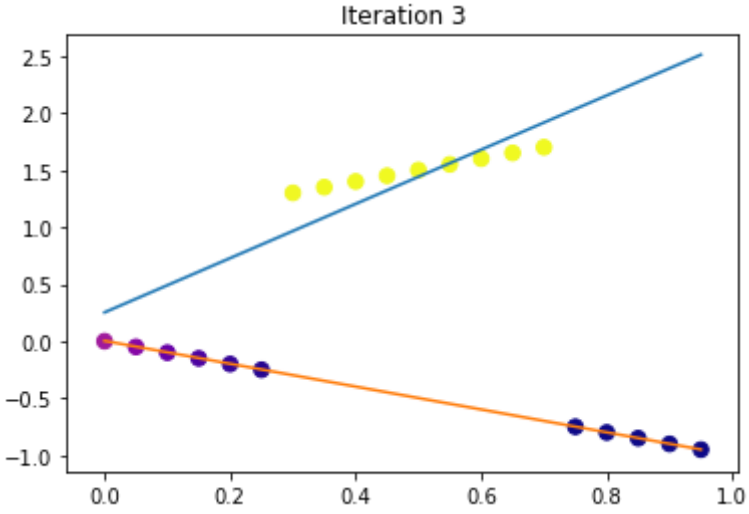


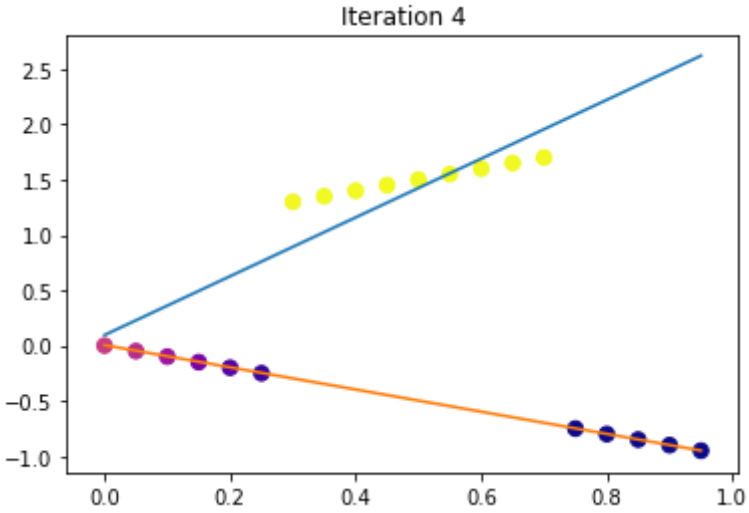
Out[18]:
```
(2.6117782470425492,
 0.12499332374328753,
 -1.4616822026429732,
 0.16566556312773686)
```

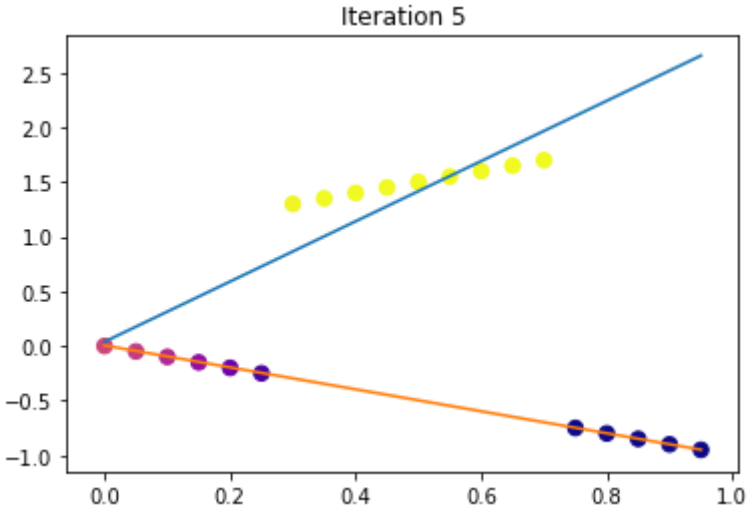
In [75]:
```
x = np.arange(0,1,0.05)
y = (abs(x-0.5) < 0.25) * (x+1) + (abs(x-0.5) >=0.25)*(-x)

points=[]
0.8*np.random.normal(1,6,len(x))
for i in range(len(x)):
     points.append([x[i],y[i]])

EM1(points,0.5)
```


Iteration 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0


Iteration 2
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0


Iteration 3
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

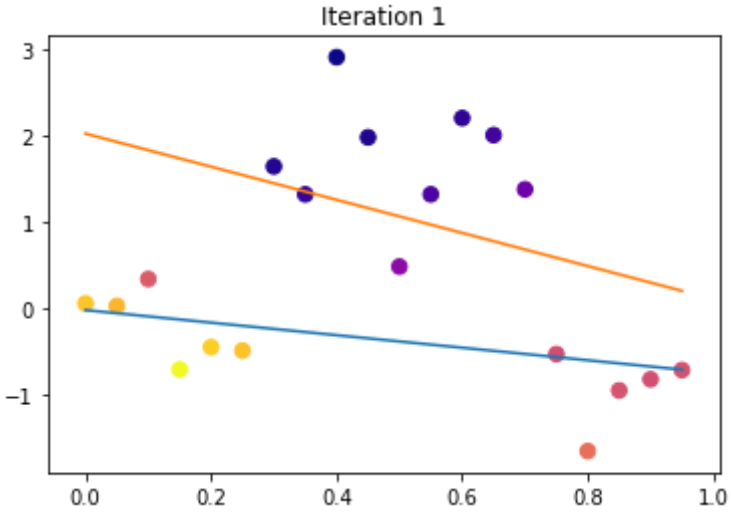


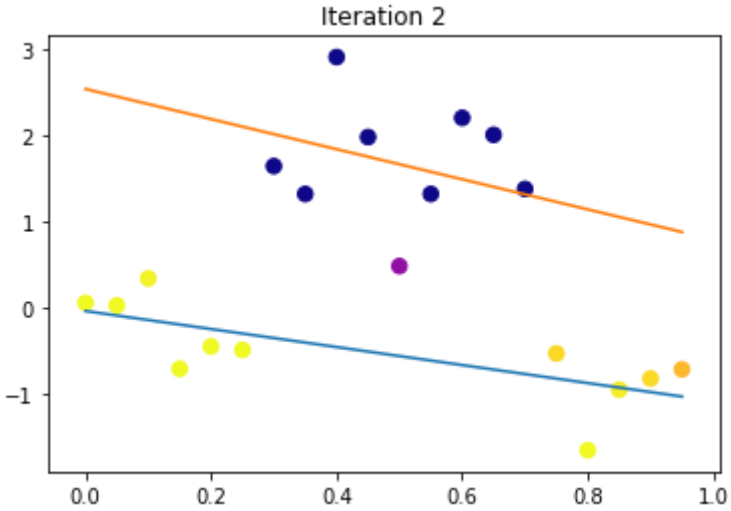


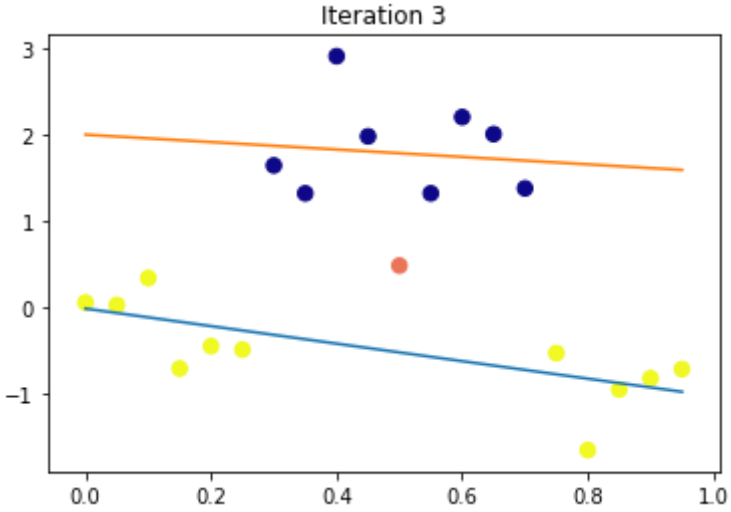
Out[75]:
```
(2.755923782376167,
 0.03883120389927329,
 -1.004194319411618,
 0.005113400633959798)
```

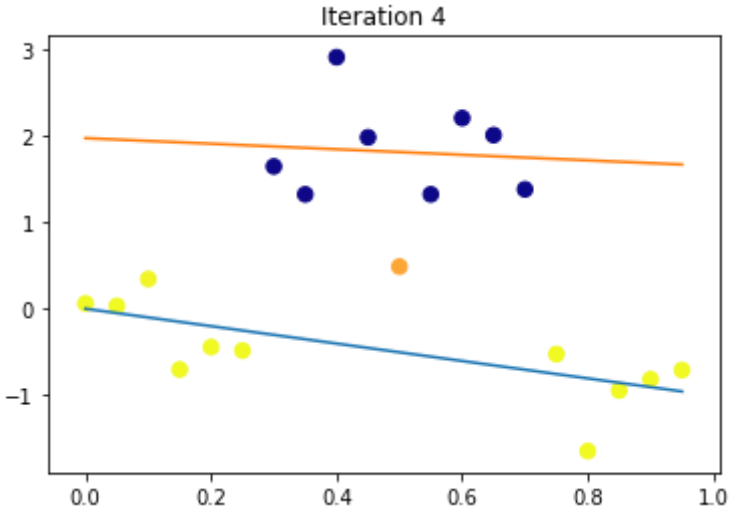
It will break when weight * varience > sigma^2

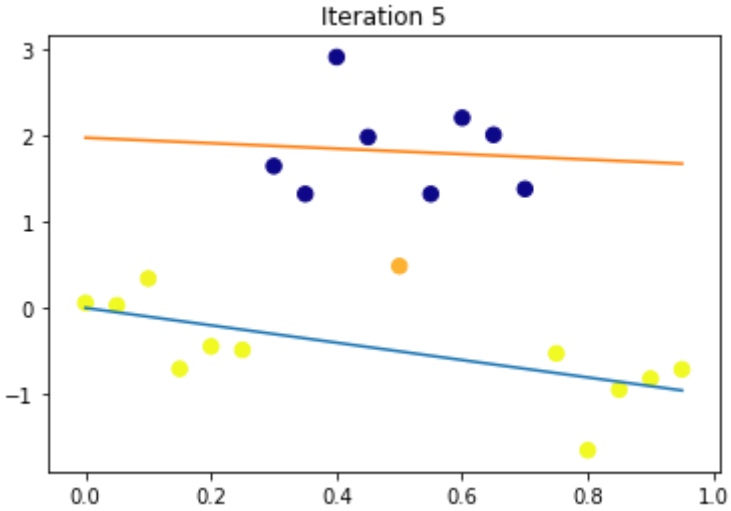
In [78]:
```
x = np.arange(0,1,0.05)
y = (abs(x-0.5) < 0.25) * (x+1) + (abs(x-0.5) >=0.25)*(-x) + 0.1*np.rand
om.normal(0,6,len(x))

points=[]
0.8*np.random.normal(1,6,len(x))
for i in range(len(x)):
     points.append([x[i],y[i]])

EM1(points,0.5)
```


Iteration 1
3
2
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0


Iteration 2
3
2
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0


Iteration 3
3
2
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0

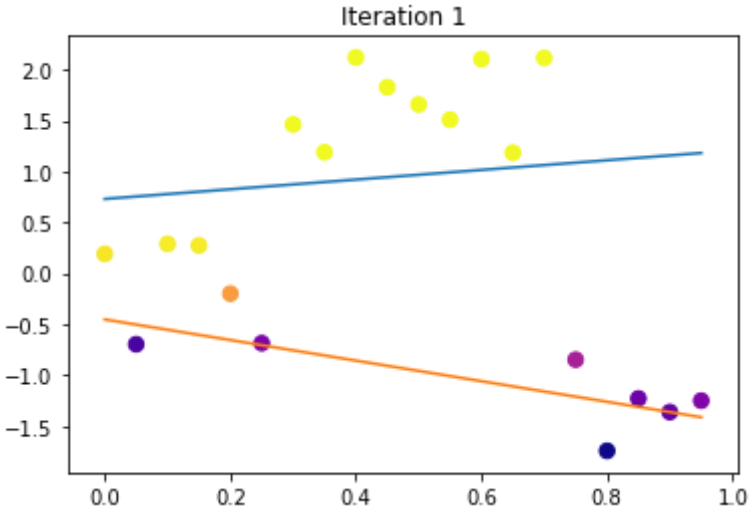


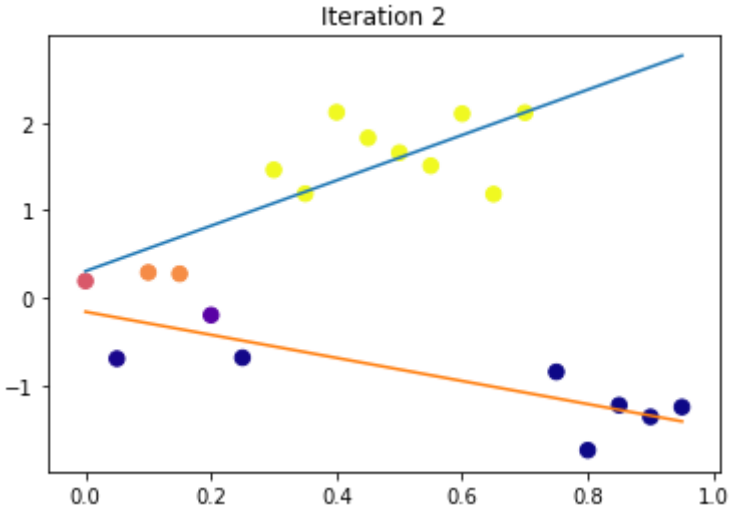


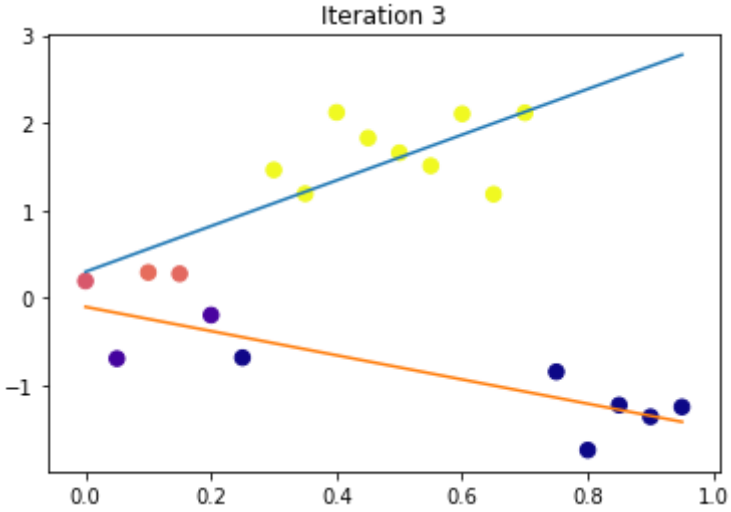
Out[78]:
```
(-1.0105326253477735,
 -0.00488759332370843,
 -0.31721099617654014,
 1.9763062186873503)
```

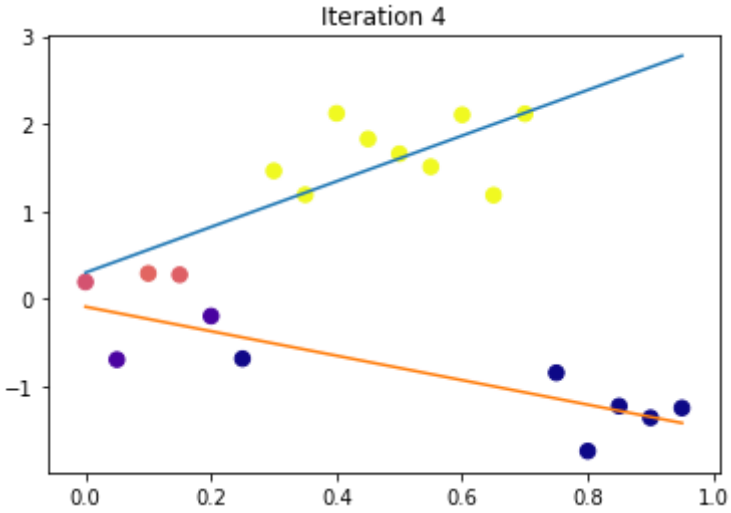
It will not break when weight * varience < sigma^2

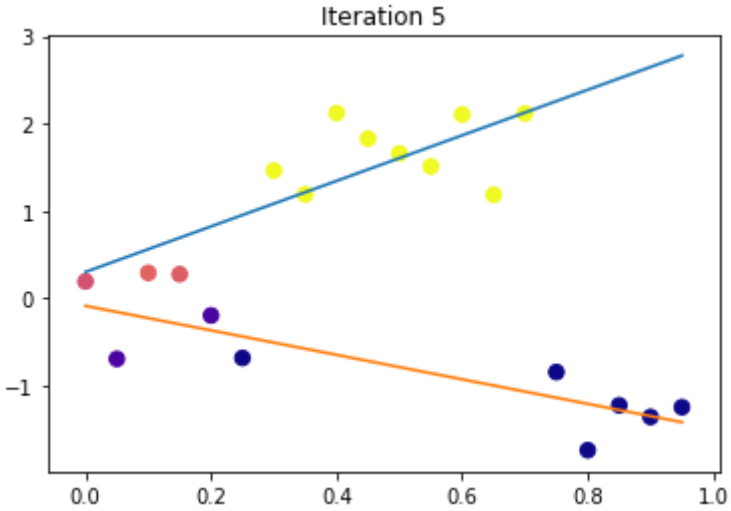
In [79]:
```
x = np.arange(0,1,0.05)
y = (abs(x-0.5) < 0.25) * (x+1) + (abs(x-0.5) >=0.25)*(-x) + 0.1*np.rand
om.normal(0,4,len(x))

points=[]
0.8*np.random.normal(1,6,len(x))
for i in range(len(x)):
     points.append([x[i],y[i]])

EM1(points,0.5)
```


Iteration 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.2
0.4
0.6
0.8
1.0


Iteration 2
2
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0


Iteration 3
3
2
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0





Out[79]:
```
(2.6085830158506536,
 0.29947329657742827,
 -1.4027800688072263,
 -0.08918314044731847)
```

In [21]:
```
3**3
```

Out[21]: 27

In [ ]:
```

```