Machine Intelligence 2

Exercise 11

Self Organizing Maps and Embedding

Group Members:
Xugang Zhou
Fangzhou Yang

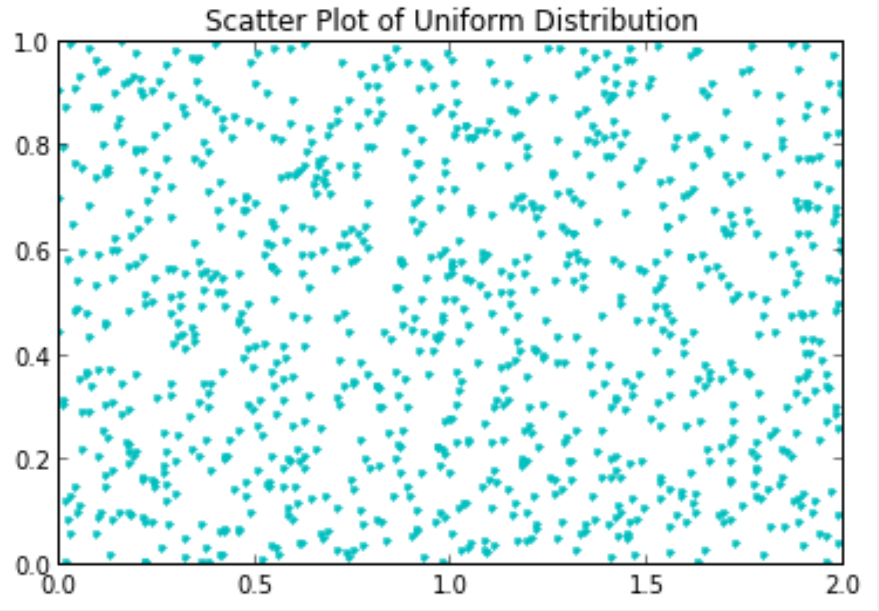
Tutor:
Timm Lochmann

July 11, 2013

1 1d Self-Organizing Map for 2d data

```
#functions
def plotScatter(X,Y,title,c):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    #ax.axis([-ran,ran,-ran,ran])
    ax.plot(X,Y,c+'.')
    ax.set_title(title)
    plt.show()

def plotW(X,Y,W,title,c):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    #ax.axis([-ran,ran,-ran,ran])
    ax.plot(X,Y,c+'.')
    ax.plot(W[:,0],W[:,1],"rD")
    ax.plot(W[:,0],W[:,1],"r-")
    ax.set_title(title)
    plt.show()
```

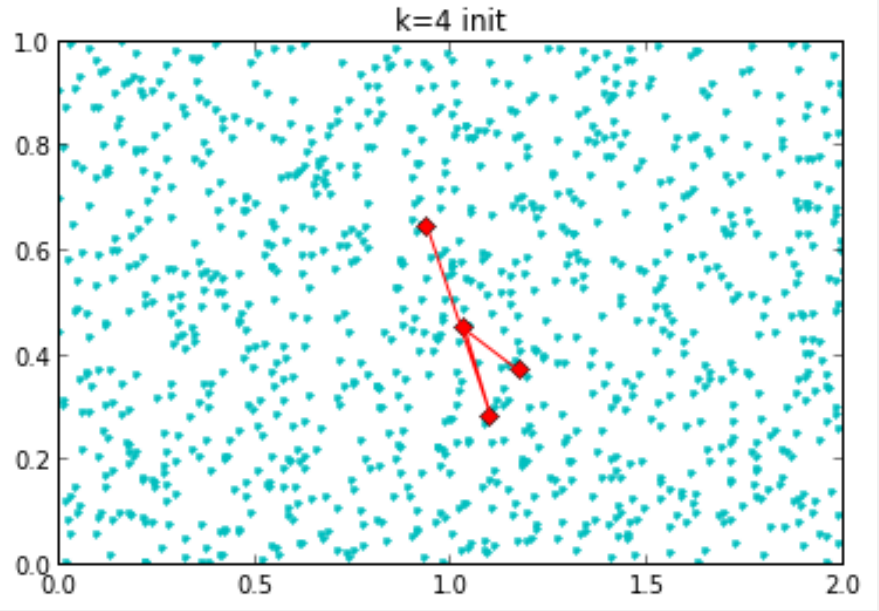
```
#data generator
random.seed(100)
X = random.uniform(0,2,1000)
Y = random.uniform(0,1,1000)
data = [[0 for j in range(2)] for i in range (1000)]
for i in range(1000):
    data[i][0] = X[i]
    data[i][1] = Y[i]
data = array(data)
plotScatter(X,Y,"Scatter Plot of Uniform Distribution","c")
```

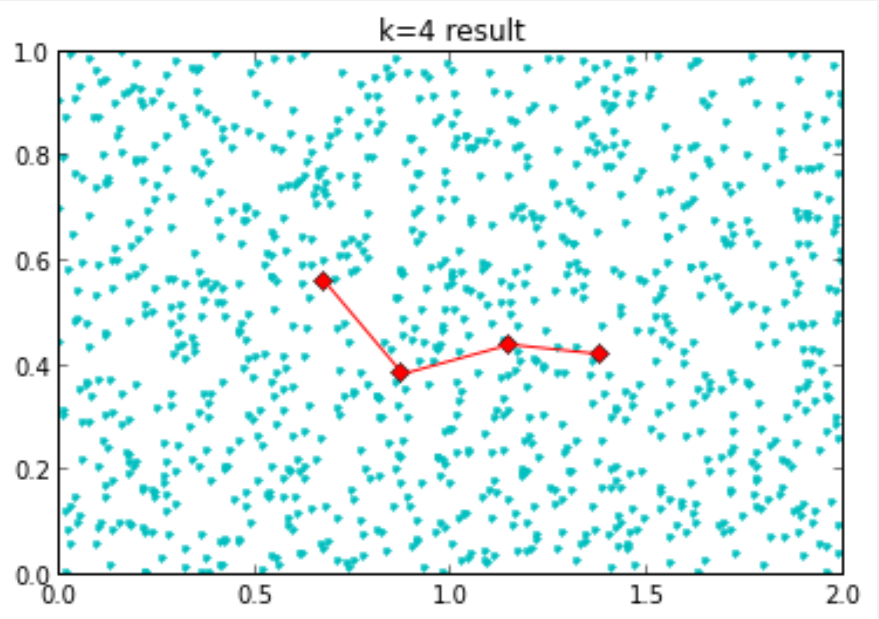



```
#init_W
def init_W(data,k):
    d = len(data[0])
    center = [0 for i in range (d)]
    for i in range(d):
        center[i] = sum(data[:,i])/len(data)
    random.seed(200)
    W = [[0 for j in range(d)] for i in range(k)]
    for i in range(k):
        for j in range(d):
            W[i][j] = center[j] + random.random() * 0.4 -0.2
    return W

#1d-SOM
def SOM_1d(data,k,W_init,xi,sigma):
    sigma0 = sigma
    xi0 = xi
    W = array(W_init)
    delta_W = array(W_init)
    d = len(data[0])
    size = len(data)

    flag = True
    alpha =0
    iteration = 0.
    while(flag):
        #choose the closest P
        for p in range (k):
            if (p==0):
                dis_min = distance(data[alpha],W[p])
```

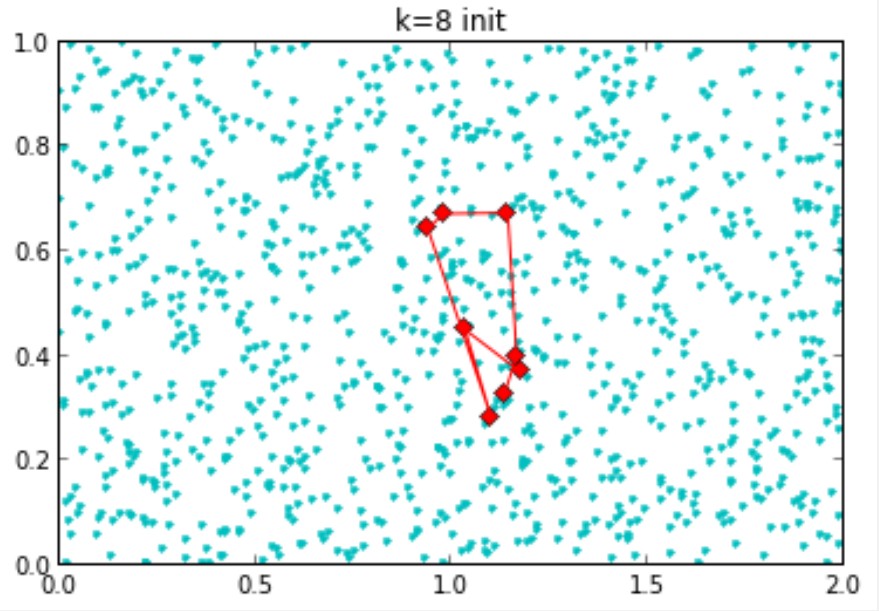
```
                p_min = p
            else:
                dis = distance(data[alpha],W[p])
                if(dis<dis_min):
                    dis_min=dis
                    p_min = p
        #change prototypes
        for q in range (k):
            delta_W[q] = xi * h_qp([q],[p_min],sigma) * (data[alpha] - W[q])
        for i in range (k):
            W[i] = W[i] + delta_W[i]

        if(sigma<0.00000001):
            flag = False

        alpha = (alpha+1)%size
        iteration +=1
        if(iteration > size ):
            sigma = sigma * (size/iteration)
            xi = xi * (size/iteration)

    return W

def h_qp(w1,w2,sigma):
    sqsum = 0.
    for i in range (len(w1)):
        sqsum += (w1[i] - w2[i])**2
    #print sigma
    tmp = -sqsum/(2* sigma**2)
    return math.exp(tmp)


def distance(x,w):
    tmpsum = 0.
    for i in range (len(x)):
        tmpsum += (x[i] - w[i])**2
    dis = math.sqrt(tmpsum)
    return dis
```

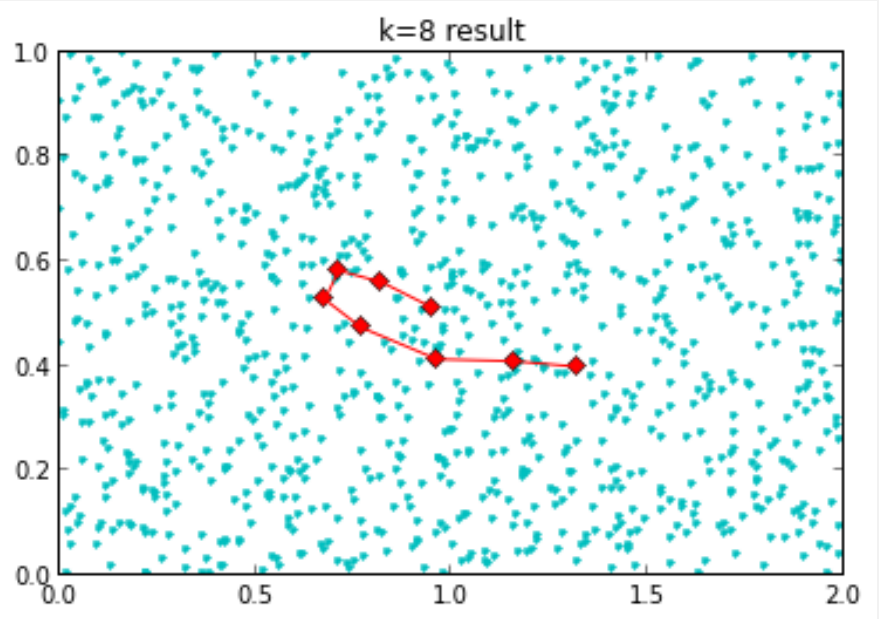
```
W = init_W(data,4)
plotW(X,Y,array(W),"k=4 init","c")
W_final = SOM_1d(data,4,W,0.002,1.0)
plotW(X,Y,W_final,"k=4 result","c")
```



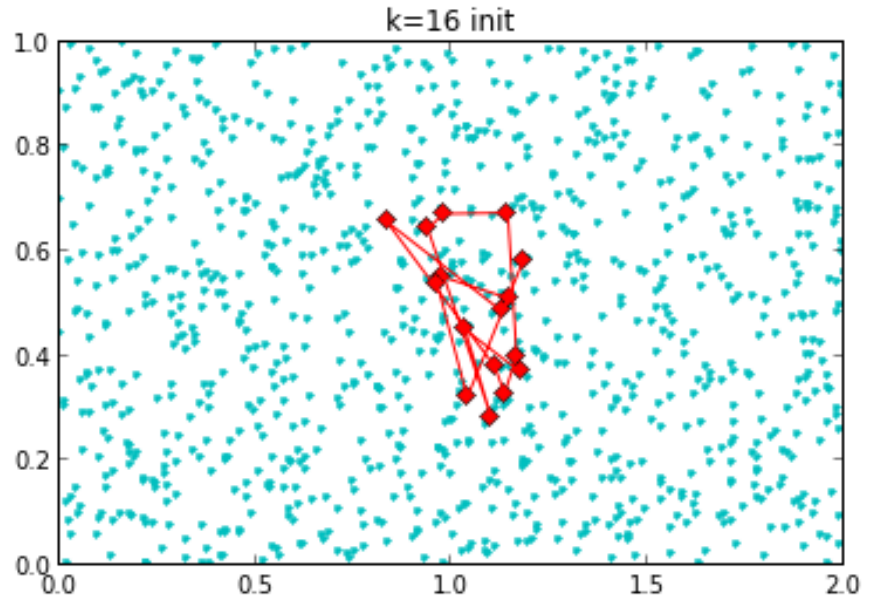


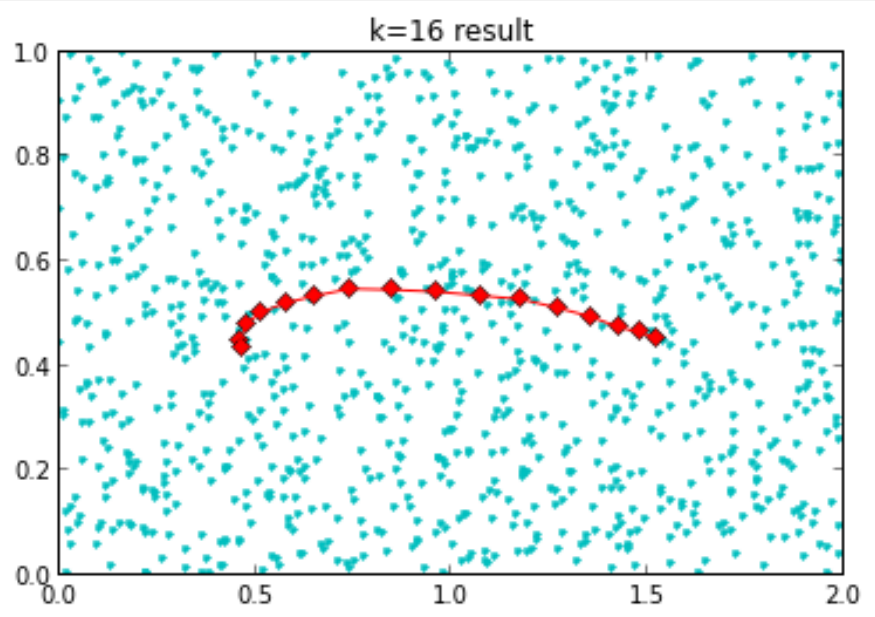

```
W = init_W(data,8)
plotW(X,Y,array(W),"k=8 init","c")
W_final = SOM_1d(data,8,W,0.004,2.)
plotW(X,Y,W_final,"k=8 result","c")
```



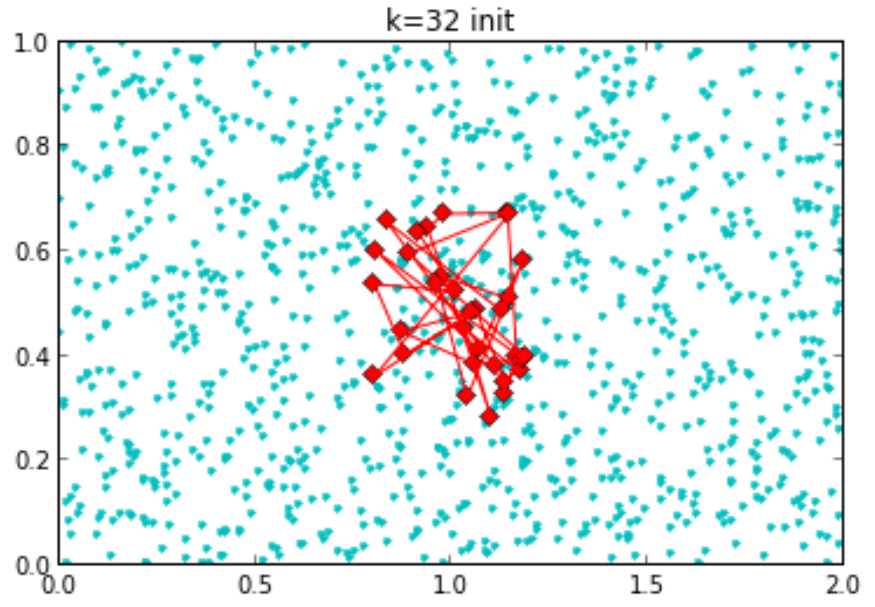


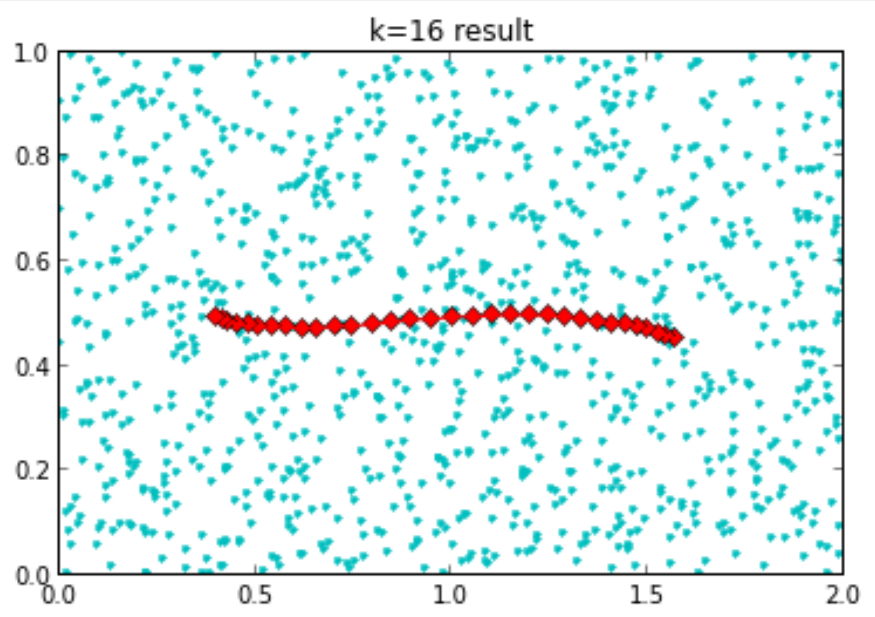

```
W = init_W(data,16)
plotW(X,Y,array(W),"k=16 init","c")
W_final = SOM_1d(data,16,W,0.008,4.)
plotW(X,Y,W_final,"k=16 result","c")
```



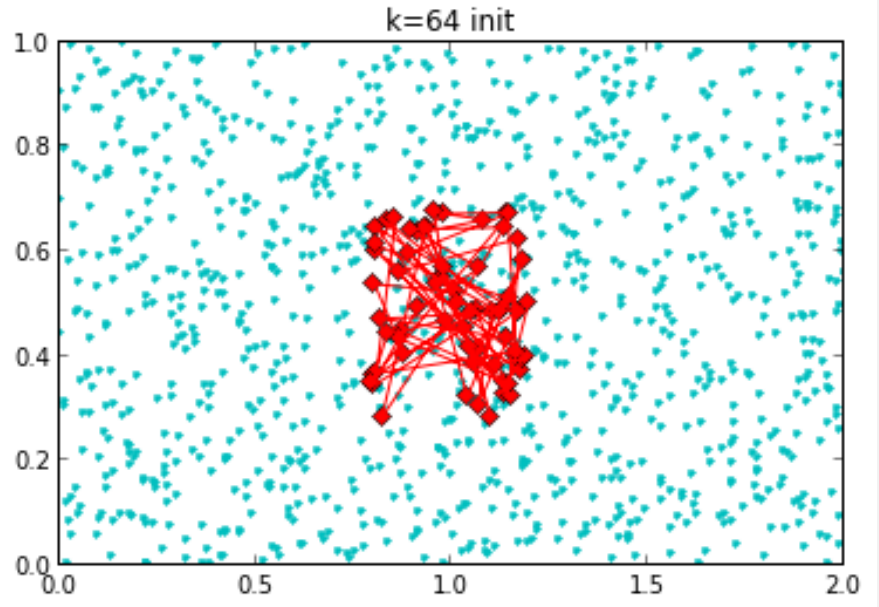


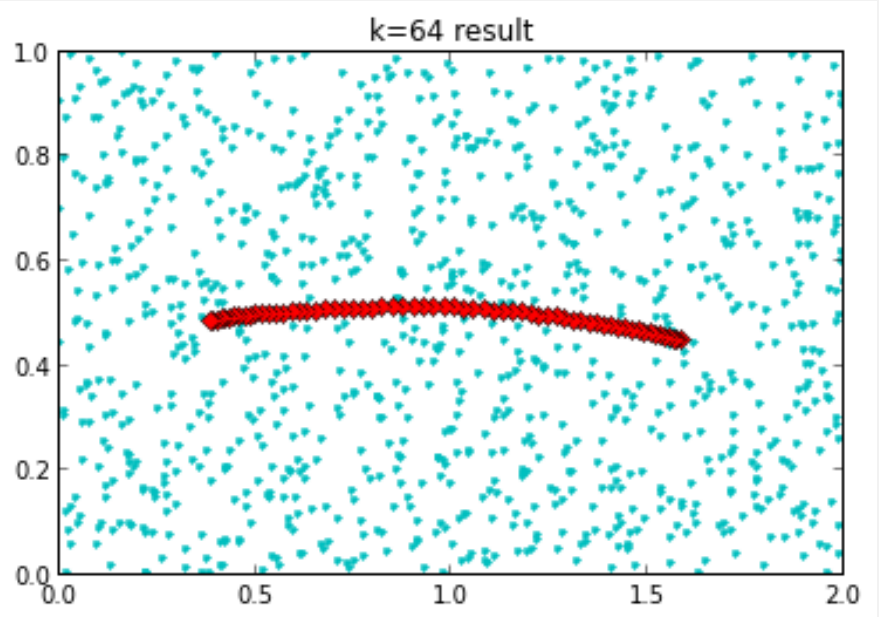

```
W = init_W(data,32)
plotW(X,Y,array(W),"k=32 init","c")
W_final = SOM_1d(data,32,W,0.01,8.)
plotW(X,Y,W_final,"k=16 result","c")
```






```
W = init_W(data,64)
plotW(X,Y,array(W),"k=64 init","c")
W_final = SOM_1d(data,64,W,0.012,16.)
plotW(X,Y,W_final,"k=64 result","c")
```

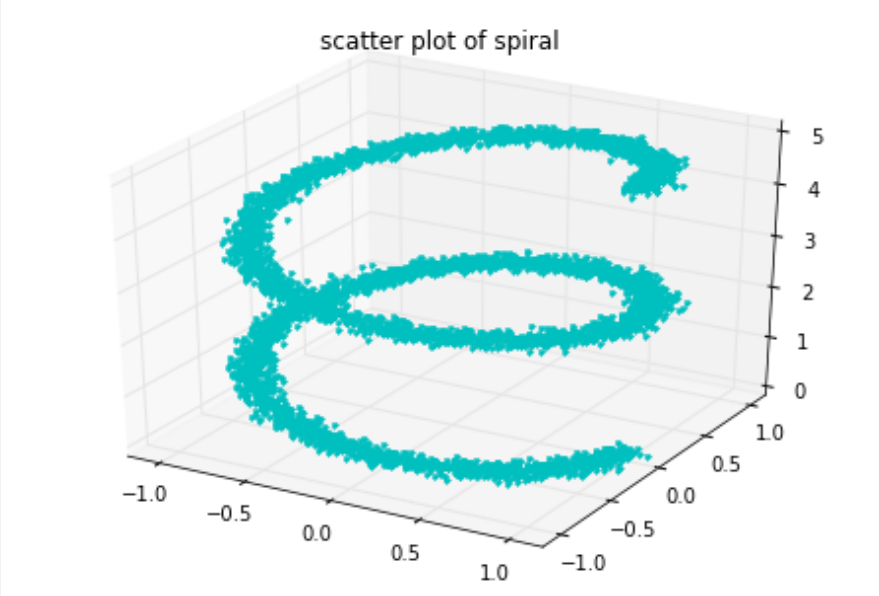




2 1d Self-Organizing Maps for 3d data

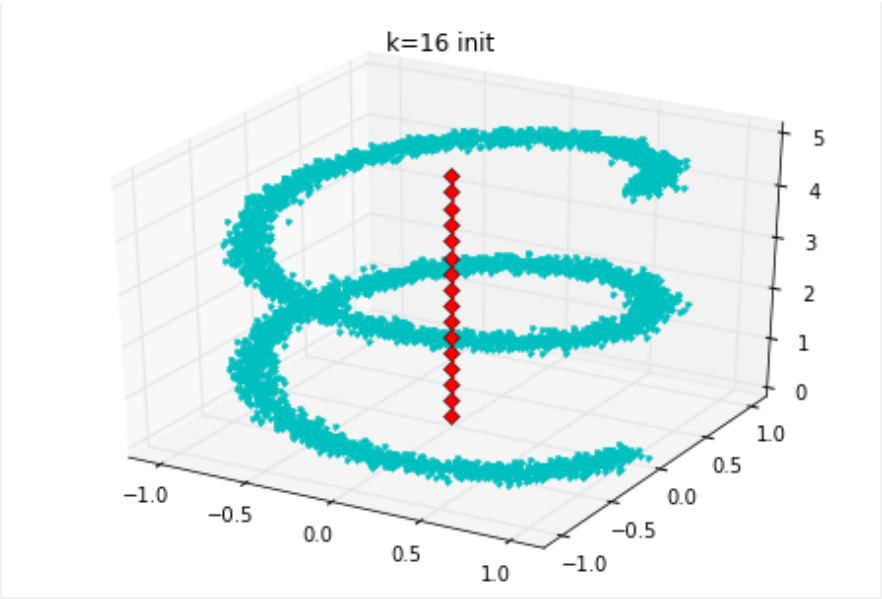
```
from mpl_toolkits.mplot3d import Axes3D
def plot3dScatter(X,Y,Z,title,c):
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot(X,Y,Z,c)
    ax.set_title(title)
    plt.show()
```

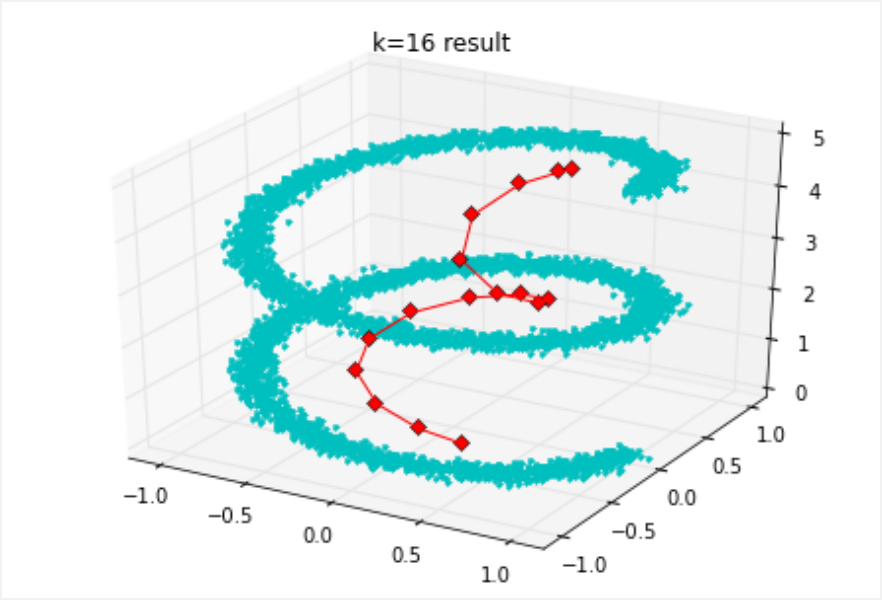
```
#read data
spiral = loadtxt("spiral.csv",skiprows=1,delimiter=",",usecols=(1,2,3))
plot3dScatter(spiral[:,0],spiral[:,1],spiral[:,2],"scatter plot of spiral","c.")
```



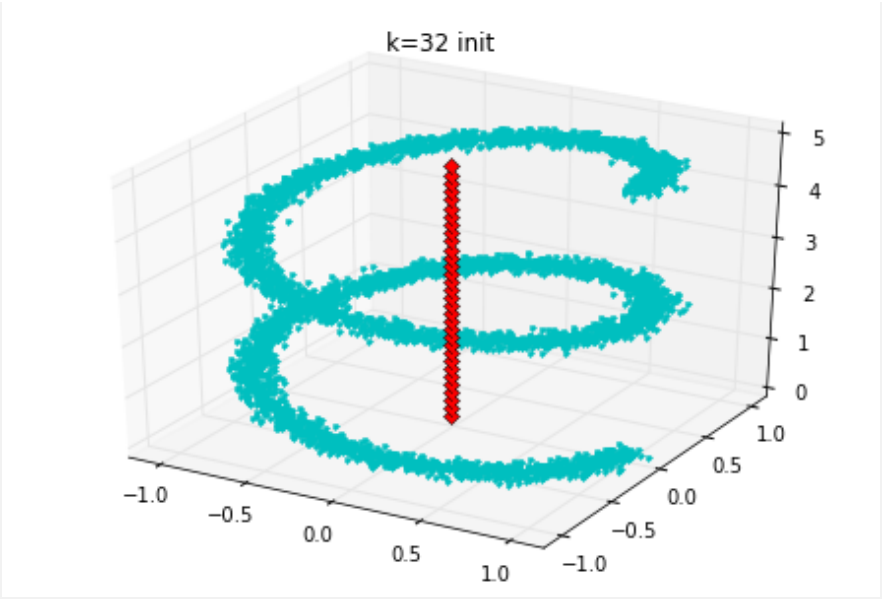

```
#init_W
def init_W_3d(data,k):

    W = [ [0,0, float(i)*5/k ]for i in range (k)]

    return W

def plotW_3d(data,W,title,c):
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot(data[:,0],data[:,1],data[:,2],c+'.')
    ax.plot(W[:,0],W[:,1],W[:,2],"rD")
    ax.plot(W[:,0],W[:,1],W[:,2],"r-")
    ax.set_title(title)
    plt.show()
```

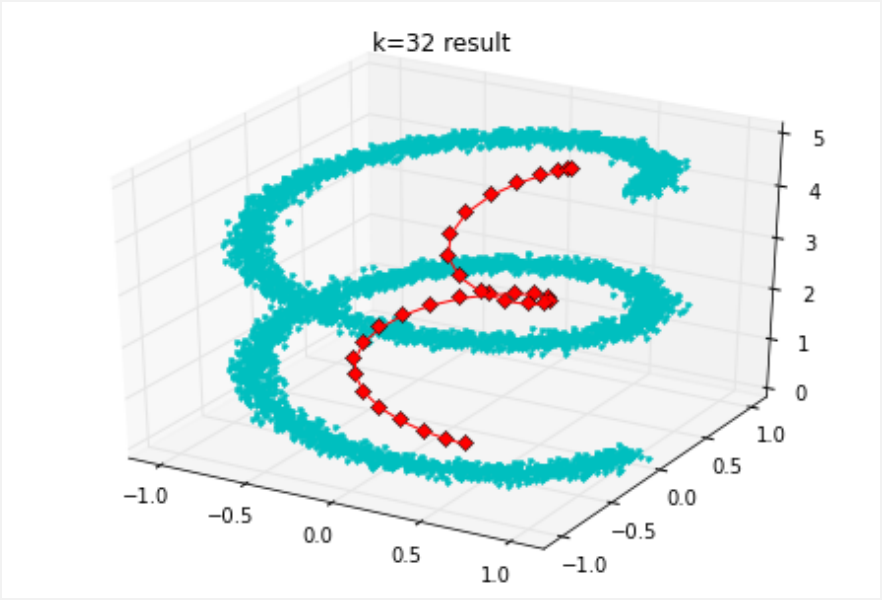
```
W = init_W_3d(spiral,16)
plotW_3d(spiral,array(W),"k=16 init","c")
W_final = SOM_1d(spiral,16,W,0.002,2.)
plotW_3d(spiral,W_final,"k=16 result","c")
```



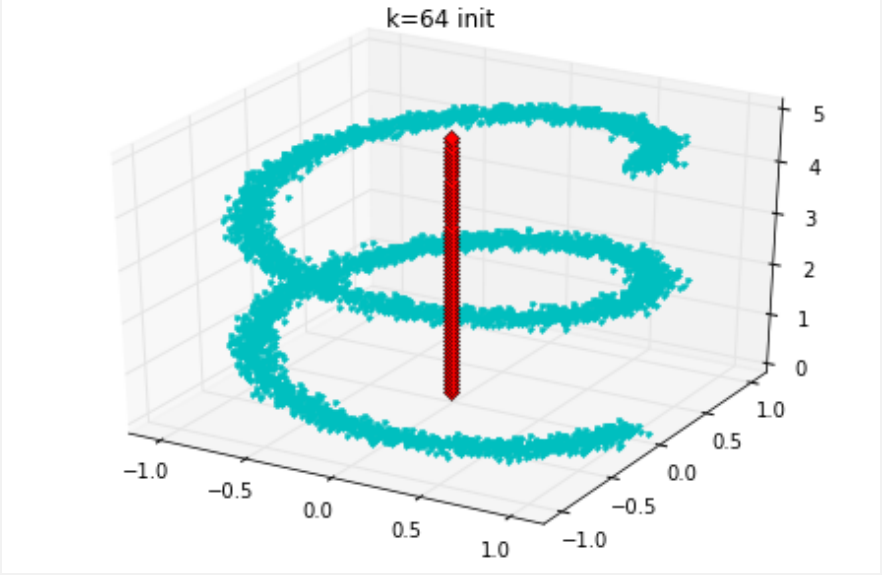


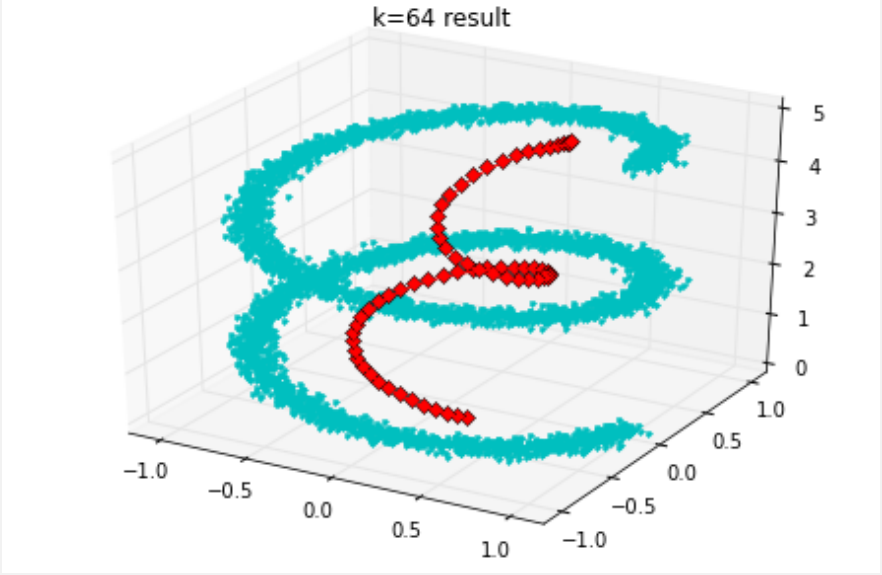

```
W = init_W_3d(spiral,32)
plotW_3d(spiral,array(W),"k=32 init","c")
W_final = SOM_1d(spiral,32,W,0.002,4.)
plotW_3d(spiral,W_final,"k=32 result","c")
```



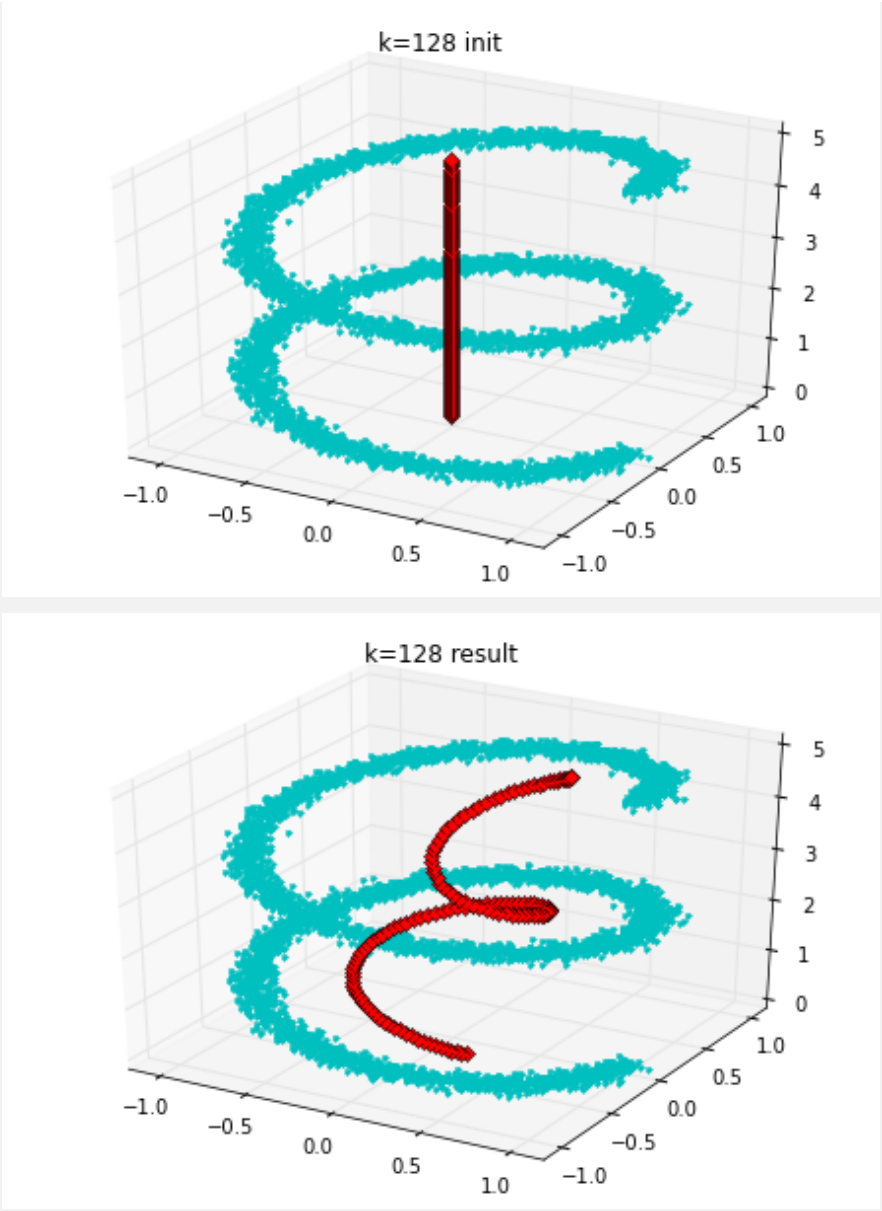



```
W = init_W_3d(spiral,64)
plotW_3d(spiral,array(W),"k=64 init","c")
W_final = SOM_1d(spiral,64,W,0.002,8.)
plotW_3d(spiral,W_final,"k=64 result","c")
```






```
W = init_W_3d(spiral,128)
plotW_3d(spiral,array(W),"k=128 init","c")
W_final = SOM_1d(spiral,128,W,0.002,16.)
plotW_3d(spiral,W_final,"k=128 result","c")
```

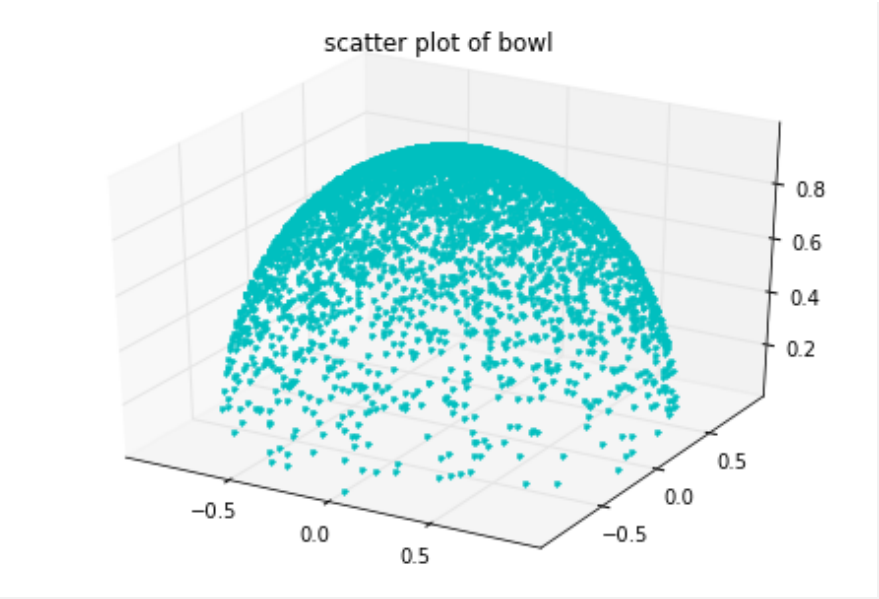


3 2d Self-Organizing Maps for 3d data

```
#2d-SOM
def SOM_2d(data,k,W_init,xi,sigma):
    W = array(W_init)
    delta_W = array(W_init)
    d = len(data[0])
    size = len(data)

    flag = True
    alpha =0
```

```
    iteration = 0.
    while(flag):
        #choose the closest P
        for pi in range (k):
            for pj in range (k):
                if (pi==0 and pj==0):
                    dis_min = distance(data[alpha],W[pi][pj])
                    pi_min = pi
                    pj_min = pj
            else:
                dis = distance(data[alpha],W[pi][pj])
                if(dis < dis_min):
                    dis_min = dis
                    pi_min = pi
                    pj_min = pj
        for qi in range (k):
            for qj in range (k):
                delta_W[qi][qj] = xi * h_qp([qi,qj],[pi_min,pj_min],sigma) * (data[alpha]
                    - W[qi][qj])

        for i in range (k):
            for j in range (k):
                W[i][j] = W[i][j] + delta_W[i][j]

        if(sigma<0.0000000000001):
            flag = False
            break;
        alpha = (alpha+1)%size
        iteration +=1
        if(iteration > size):
            sigma = sigma * (size/iteration)
            #xi = xi * (size/iteration)

    return W
```

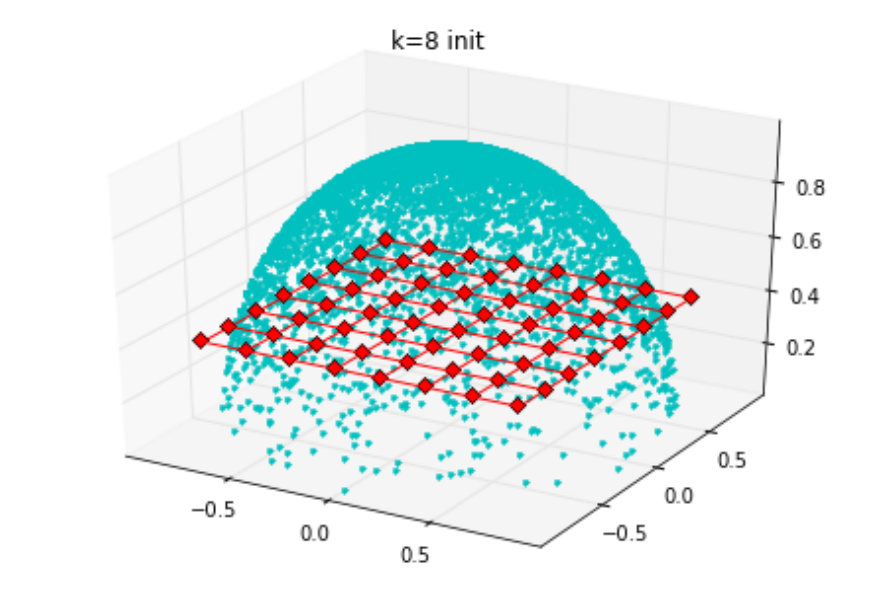
```
#read data
bowl = loadtxt("bowl.csv",skiprows=1,delimiter=",",usecols=(1,2,3))
plot3dScatter(bowl[:,0],bowl[:,1],bowl[:,2],"scatter plot of bowl","c.")
```

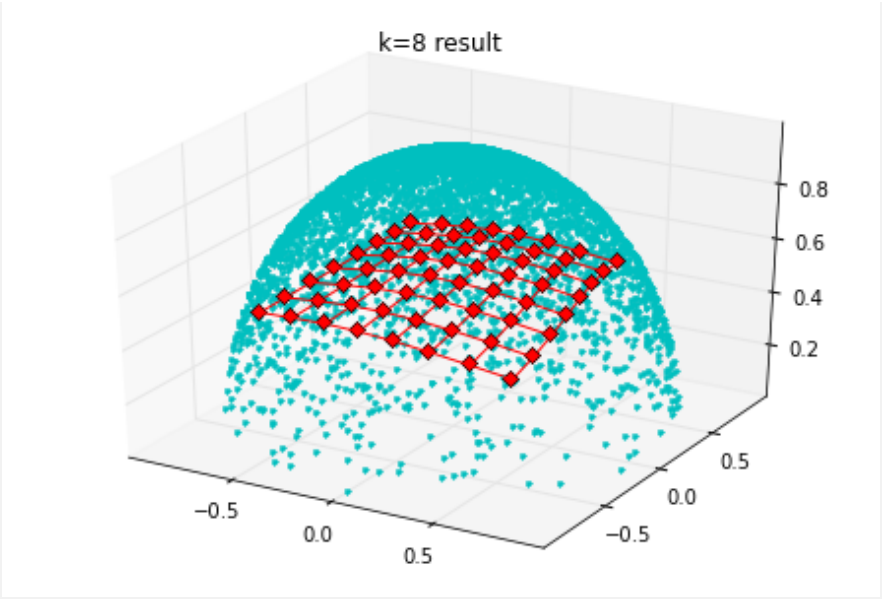



```
#init_W
def init_W_bowl(data,k):

    W = [[ [float(i+1)/(k+1)*2-1.,float(j+1)/(k+1)*2-1., 0.4 ]for j in range (k)]for i in
         range (k)]

    return W

def plotW_3d_bowl(data,W,title,c):
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot(data[:,0],data[:,1],data[:,2],c+'.')
    k = len(W)
    for i in range (k):
        ax.plot(W[i,:,0],W[i,:,1],W[i,:,2],"rD")
        ax.plot(W[i,:,0],W[i,:,1],W[i,:,2],"r-")
    for i in range (k):
        ax.plot(W[:,i,0],W[:,i,1],W[:,i,2],"rD")
        ax.plot(W[:,i,0],W[:,i,1],W[:,i,2],"r-")
    ax.set_title(title)
    plt.show()

def plotW_map(data,W):
    k = len(W)
    size = len(data)
    alpha = 0
    C = [[0 for i in range(k)] for j in range (k)]
    for alpha in range (size):
        #choose the closest P
        for pi in range (k):
            for pj in range (k):
```

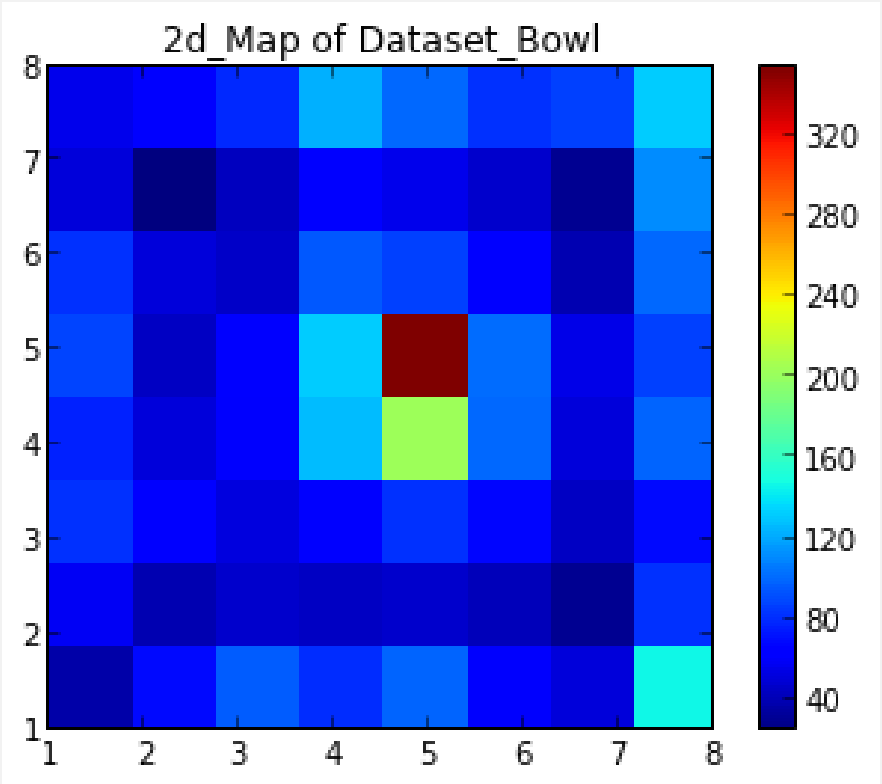
```
                if (pi==0 and pj==0):
                    dis_min = distance(data[alpha],W[pi][pj])
                    pi_min = pi
                    pj_min = pj
                else:
                    dis = distance(data[alpha],W[pi][pj])
                    if(dis < dis_min):
                        dis_min = dis
                        pi_min = pi
                        pj_min = pj
        C[pi_min][pj_min] += 1

    plt.title("2d_Map of Dataset_Bowl")
    plt.imshow(C,interpolation="nearest",extent=[1,k,1,k])
    plt.colorbar()
    plt.show()
```

```
W = init_W_bowl(bowl,8)
plotW_3d_bowl(bowl,array(W),"k=8 init","c")
W_final1 = SOM_2d(bowl,8,W,0.0002,4.)
plotW_3d_bowl(bowl,W_final1,"k=8 result","c")
```



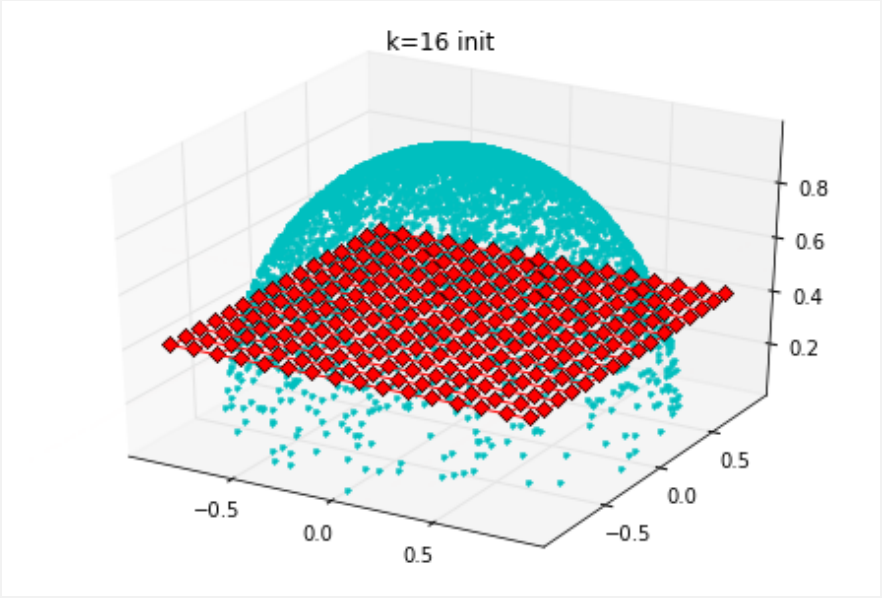


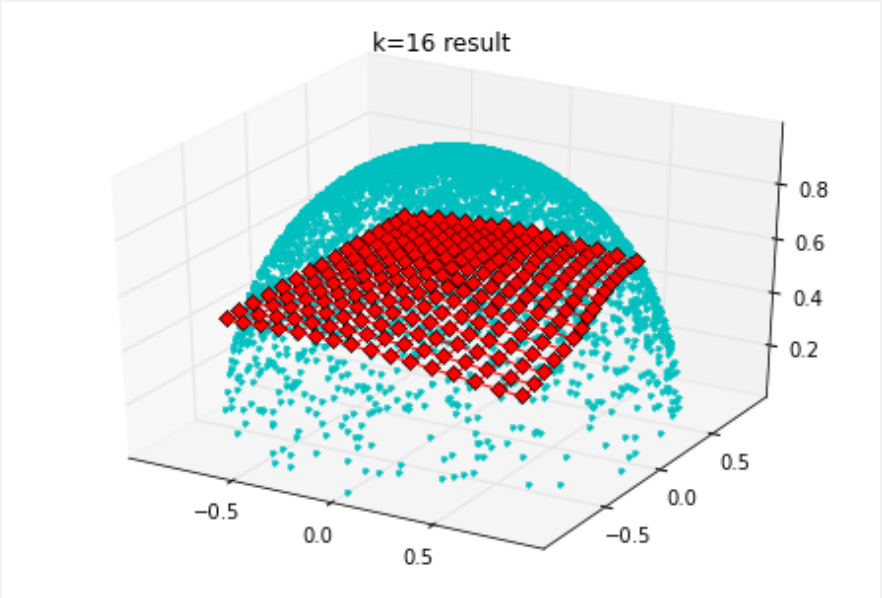

```
plotW_map(bowl,W_final1)
```



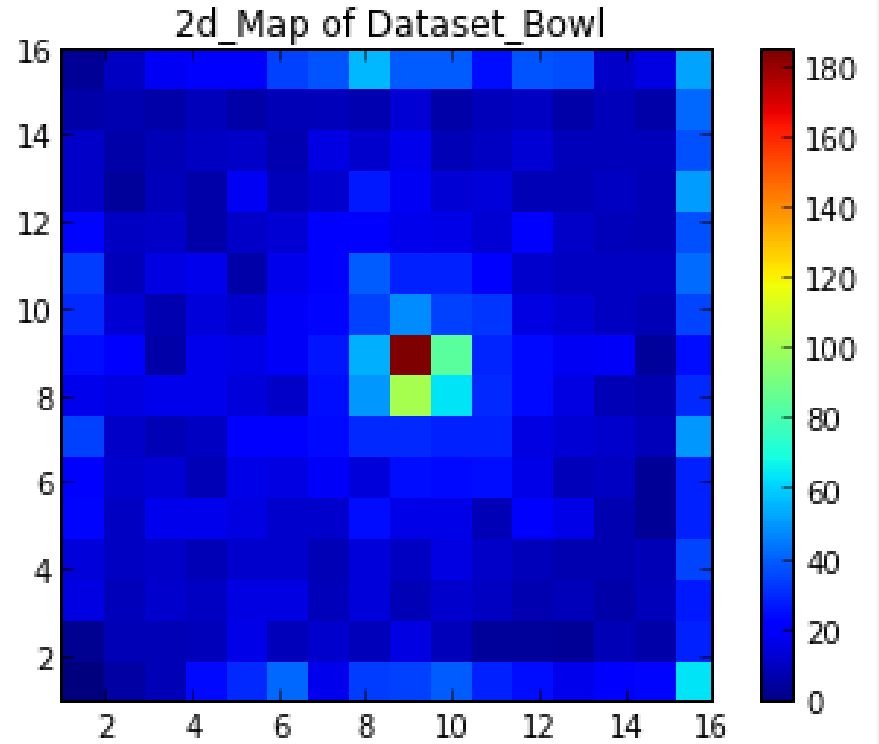

```
W = init_W_bowl(bowl,16)
plotW_3d_bowl(bowl,array(W),"k=16 init","c")
W_final2 = SOM_2d(bowl,16,W,0.0002,8.)
```

```
plotW_3d_bowl(bowl,W_final2,"k=16 result","c")
```

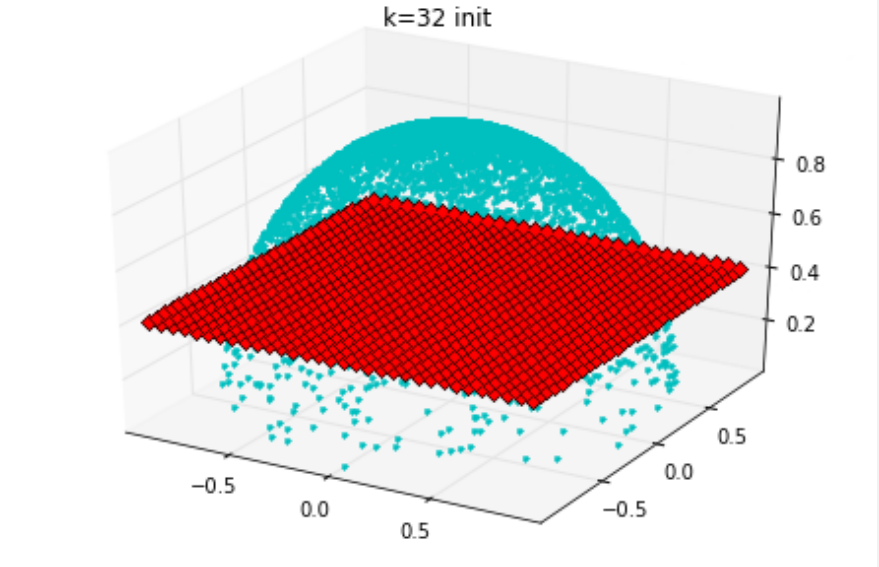


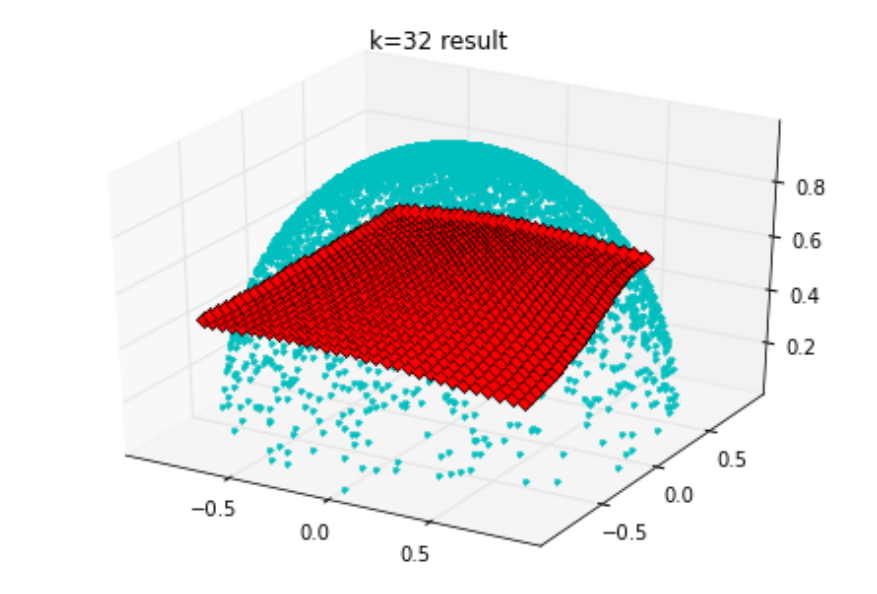



```
plotW_map(bowl,W_final2)
```




```
W = init_W_bowl(bowl,32)
plotW_3d_bowl(bowl,array(W),"k=32 init","c")
W_final3 = SOM_2d(bowl,32,W,0.0002,16.)
plotW_3d_bowl(bowl,W_final3,"k=32 result","c")
```






```
plotW_map(bowl,W_final3)
```