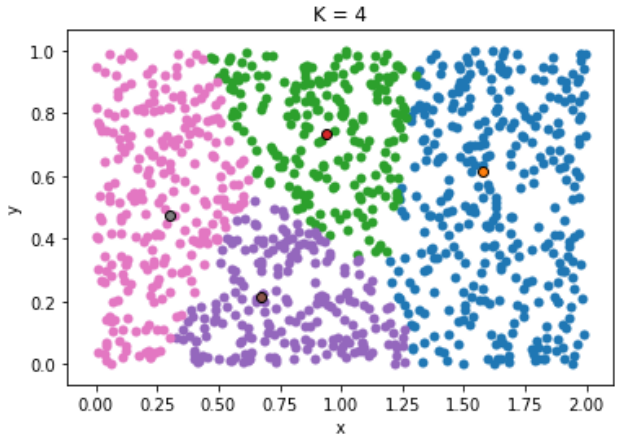
Sheet 10 - Self-Organising Maps

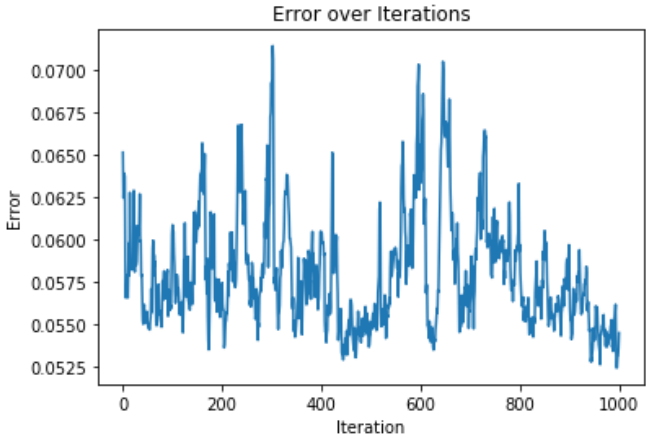
Team name: DataFun Members: Fabian Frank Jan Botsch David Munkacsi

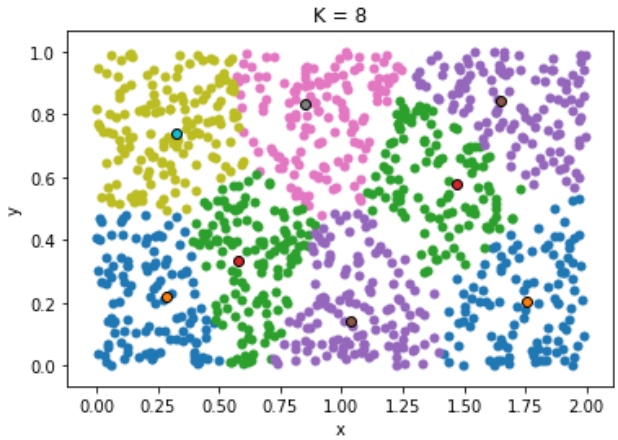
10.1 1d Self-Organizing Map for 2D data

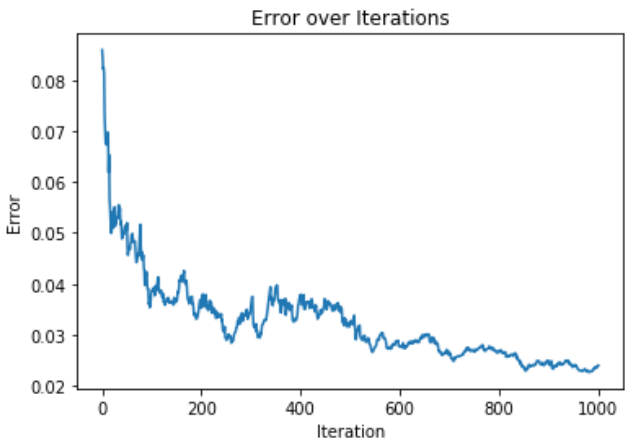
In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from pandas.tools.plotting import scatter_matrix
from mpl_toolkits.mplot3d import Axes3D
```

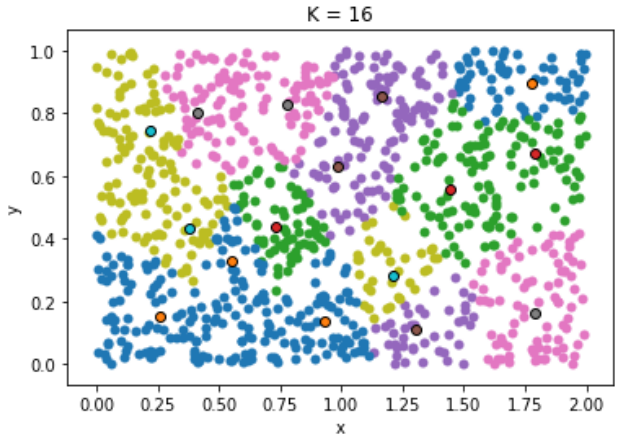
In [3]:
```
## 10.1 Self-organizing maps 1D

#create random data set
dataSet = np.random.uniform(size=(2,1000))
dataSet[0,:] = dataSet[0,:] * 2.0
dataMean = dataSet.mean(axis=1).reshape((2,1))
p = dataSet.shape[1]

# Init
def init(k):
    wqInit = ((np.random.rand(2,k) - 0.5) + dataMean)
    return wqInit

# Optimization
def assignDatapoint(x, wq):
    diff = x - wq.T
    nor = np.linalg.norm(diff, axis=1)
    mqIdx = np.argmin(nor, axis=0)
    return mqIdx

def updatePrototypes(wq, x, mqIdx, eps, sig):
    diff = (x.T - wq.T).T
    nbf = calcNeighborhoodFunction(wq,np.arange(float(wq.shape[1])), mqIdx,
sig)
    dw = eps * nbf * diff
    wq = wq + dw
    return wq


def calcNeighborhoodFunction(w,q,p,sig):
    diff = q - float(p)
    squ = np.square(diff)
    inp = - squ / (2.0 * sig * sig);
    return np.exp(inp)

def annealParam(eps,tau, tcurr, tmax):
    if t > tmax/4:
        eps = tau * eps
    return eps

def getError(wq):
    mq = assignDatapoints(wq, wq.shape[1])
    #print(mq.shape)
    diff = dataSet[np.newaxis,...] - wq[np.newaxis,...].T
    nor = np.square(np.linalg.norm(diff, axis=1))
    tmp = nor[np.nonzero(mq.T)]
    err = (1.0 / (2.0 * p)) * tmp.sum()
    return err


# Optimization
def assignDatapoints(wq, k):
    diff = dataSet[np.newaxis,...] - wq[np.newaxis,...].T
    nor = np.linalg.norm(diff, axis=1)
    mqIdx = np.argmin(nor, axis=0)
    mq = np.zeros((dataSet.shape[1], k))
    rows = list(range(dataSet.shape[1]))
    mq[rows, mqIdx] = 1
    return mq


# Plot dataset
def plot(k, t):
    plt.figure()
    plt.scatter(dataSet[0], dataSet[1])
    plt.scatter(wqInit[0], wqInit[1], color='black')
    plt.scatter(wq[0], wq[1], color='red')
    plt.xlabel('x')
```

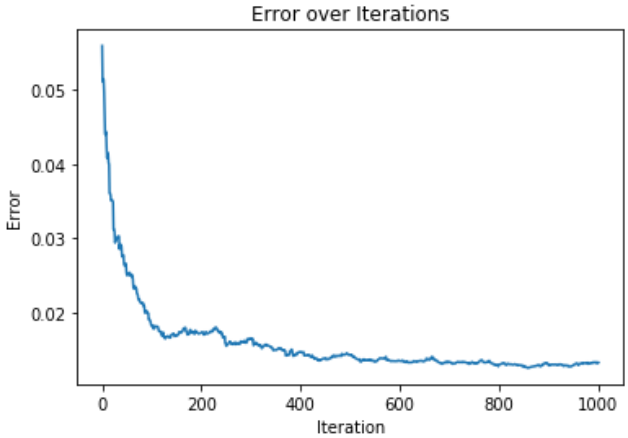

K = 4
y
x
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

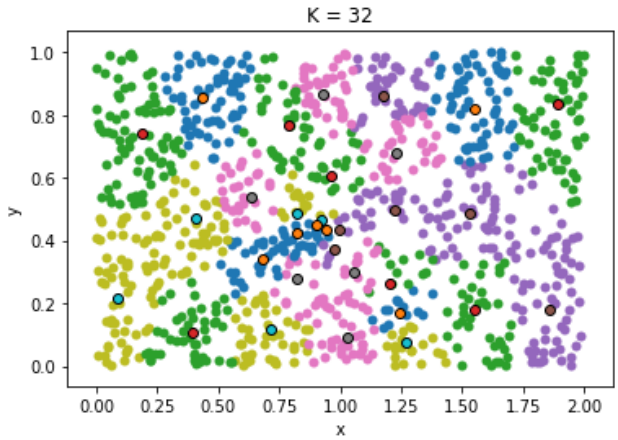

Error over Iterations
Error
Iteration
0.0525
0.0550
0.0575
0.0600
0.0625
0.0650
0.0675
0.0700
0
200
400
600
800
1000

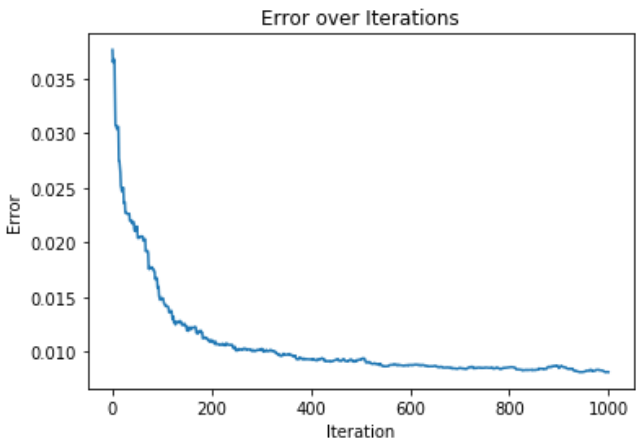

K = 8
y
x
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

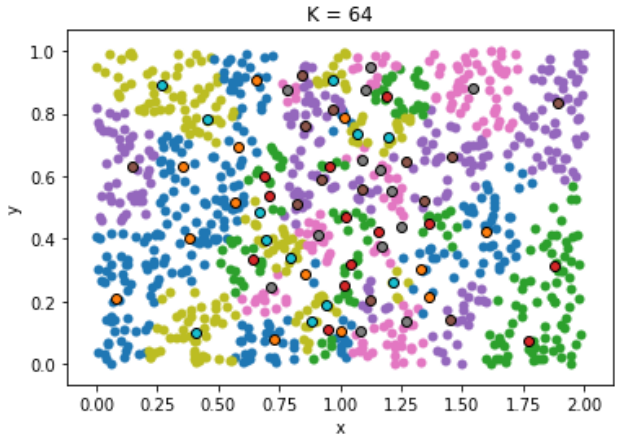

Error over Iterations
Error
Iteration
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0
200
400
600
800
1000

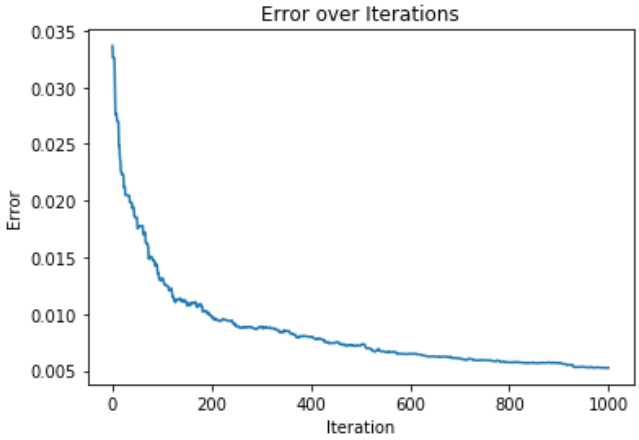

K = 16
y
x
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

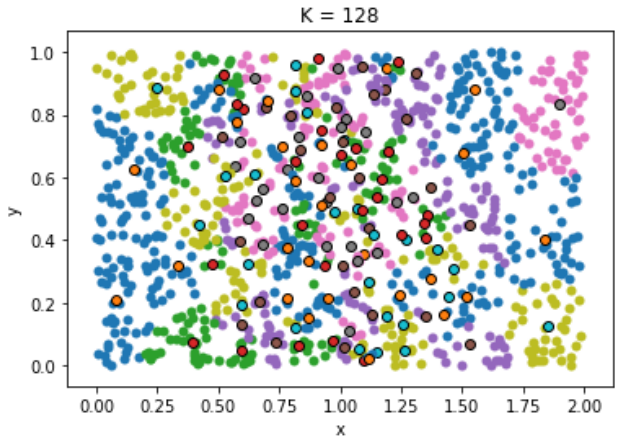

Error over Iterations
Error
Iteration
0.02
0.03
0.04
0.05
0
200
400
600
800
1000

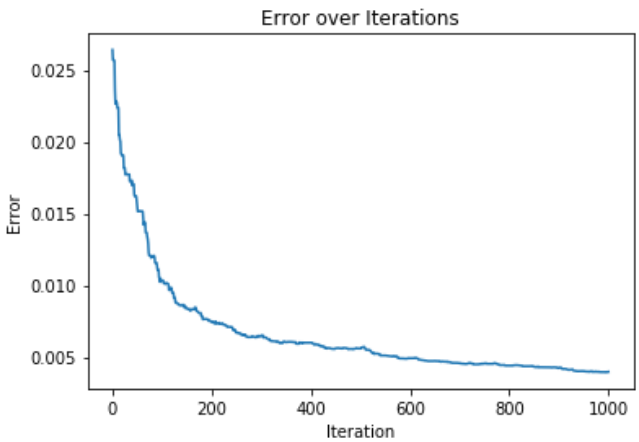

K = 32
y
x
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Error over Iterations
Error
Iteration
0.010
0.015
0.020
0.025
0.030
0.035
0
200
400
600
800
1000


K = 64
y
x
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

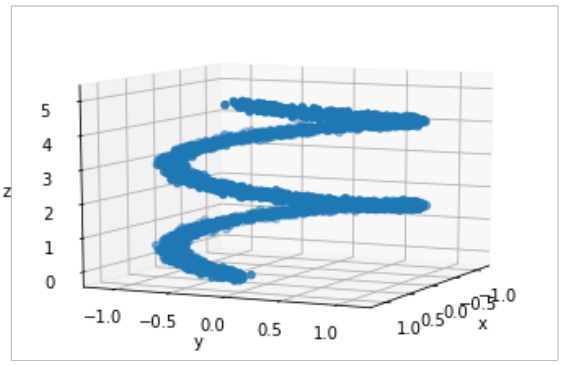

Error over Iterations
Error
Iteration
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0
200
400
600
800
1000

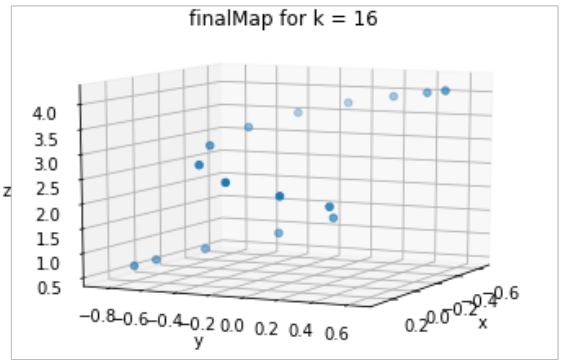

K = 128
y
x
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

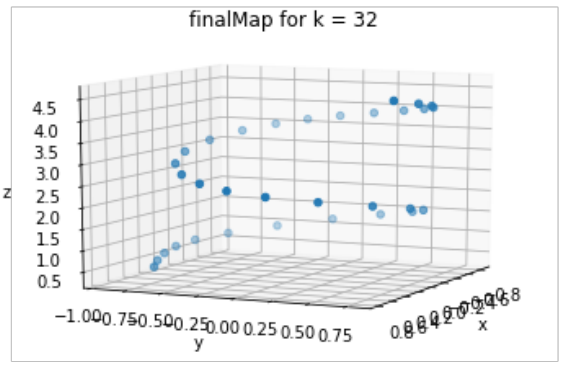

Error over Iterations
Error
Iteration
0.005
0.010
0.015
0.020
0.025
0
200
400
600
800
1000

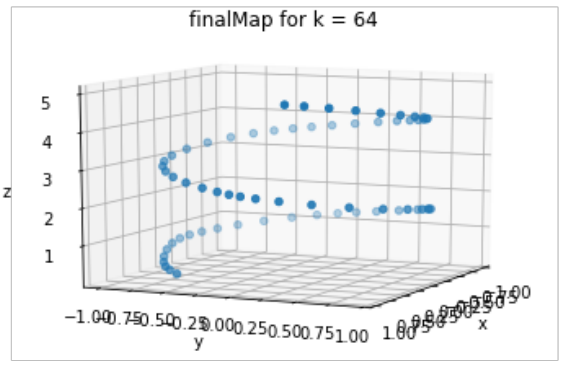
10.2 1d Self-Organizing Map for 3D data

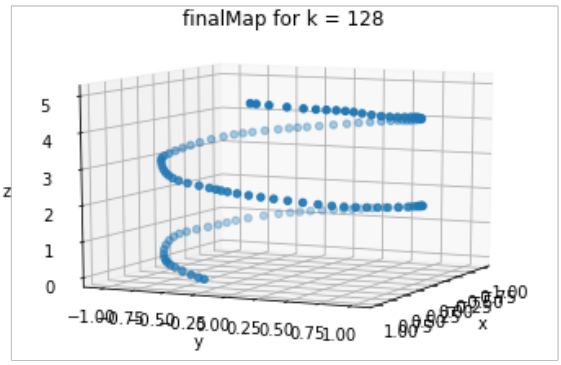
In [4]:
```
# 10.2
# a) Load and show
df = pd.read_csv("spiral.csv")
#print(df.head())
vis = plt.figure().gca(projection='3d')
vis.scatter(df['x'], df['y'], df['z'])
vis.view_init(elev=10., azim=25.)
vis.set_xlabel('x')
vis.set_ylabel('y')
vis.set_zlabel('z')
plt.show()

# b)
# adapt SOM
K = np.asarray([16, 32, 64, 128])
eps = 0.25
tau = 0.9
sig = 2

tmax = df.shape[0]
sfdIndices = np.random.permutation(tmax)
dataSet = np.asarray([df['x'], df['y'], df['z']])

# c)
# init map as line
def init_line(k):
    wq = np.zeros([3, k])
    wq[2] = np.linspace(0.0, 5.0, num=k)
    return wq

# d)
# do annealing and plot final maps
def doAnnealing(k, initMap, eps, sig):
    wq = initMap
    for t in range(tmax):
        #assign datapoint to prototype
        idx = assignDatapoint(dataSet[:,sfdIndices[t]], wq)

        #update prototype locations
        wq = updatePrototypes(wq, dataSet[:,sfdIndices[t]], idx, eps, sig)

        #anneal epsilon and sigma
        eps = annealParam(eps, tau, t, tmax)
        sig = annealParam(sig, tau, t, tmax)
    return wq

def plotMap(finalMap, k):
    fig = plt.figure().gca(projection='3d')
    fig.scatter(finalMap[0,:], finalMap[1,:], finalMap[2,:])
    fig.view_init(elev=10., azim=25.)
    fig.set_xlabel('x')
    fig.set_ylabel('y')
    fig.set_zlabel('z')
    fig.set_title('finalMap for k = %d' % k)

    plt.show()

for k in K:
    initMap = init_line(k)
    finalMap = doAnnealing(k, initMap, eps, sig)
    plotMap(finalMap, k)
```


z
y
x


finalMap for k = 16
z
y
x


finalMap for k = 32
z
y
x

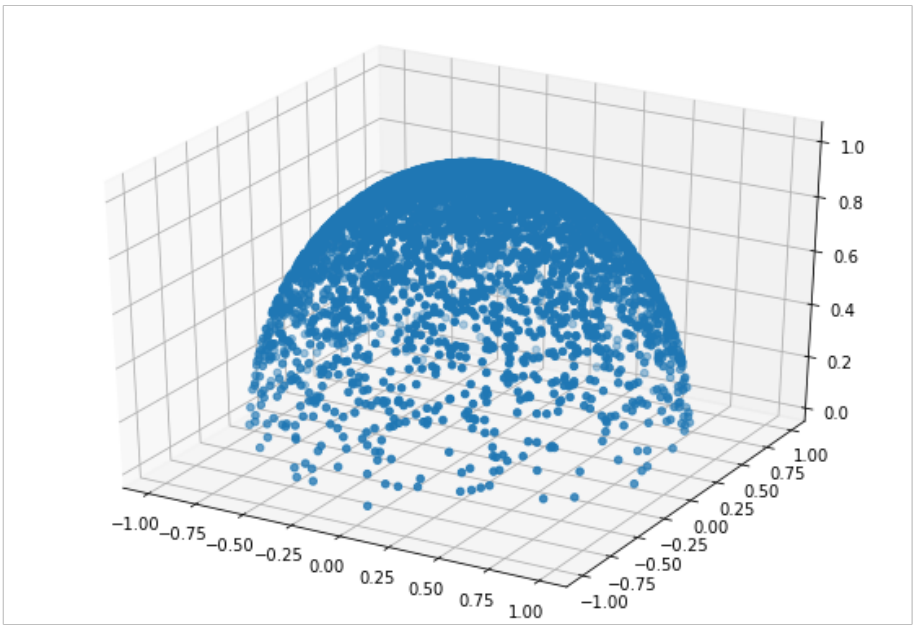

finalMap for k = 64
z
y
x



10.3 2d Self-Organizing Map for 3D data

Visualizing the 3D data

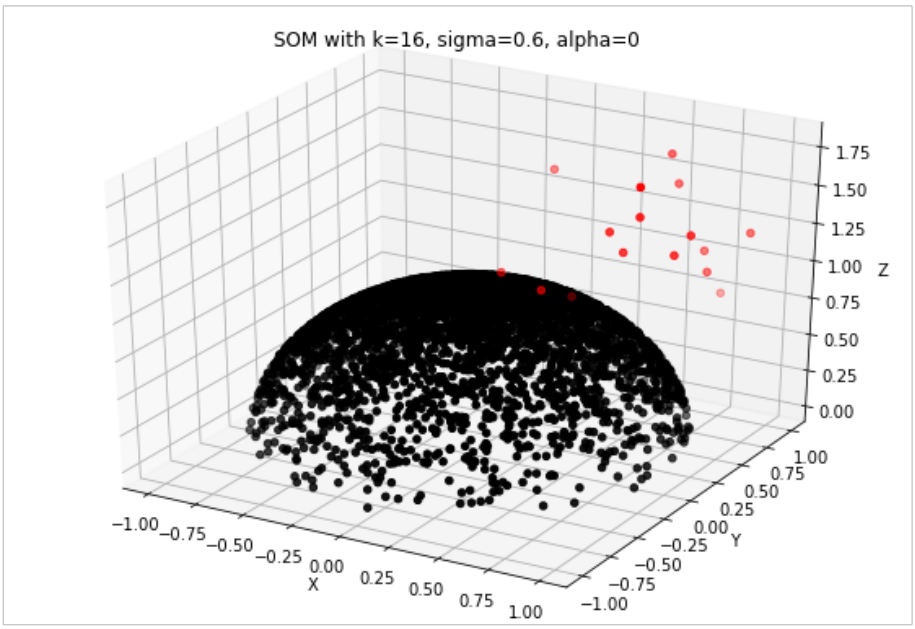
In [3]:
```
data3D = pd.read_csv("bowl.csv")
data3D = data3D.iloc[:,1:4].values
fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs = data3D[:,0],ys = data3D[:,1],zs = data3D[:,2])
plt.suptitle("Visualizing bowl.csv")
plt.show()
```

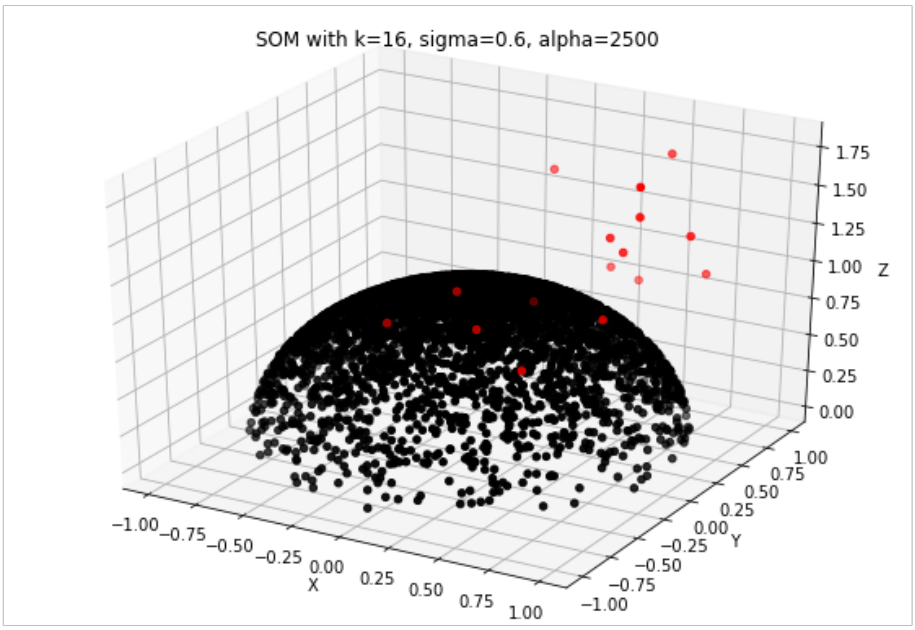


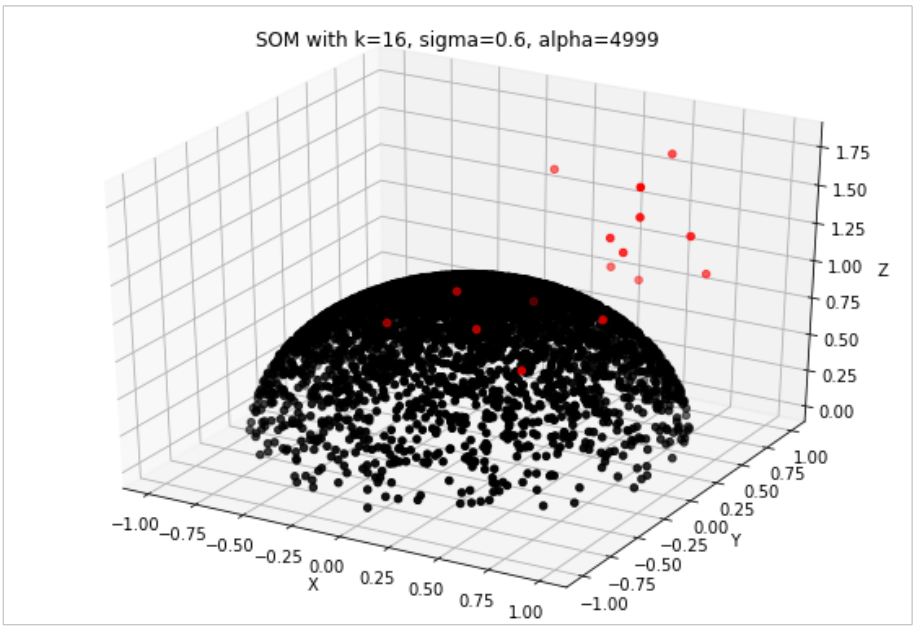
In [22]:
```
class SOM3D:
    def __init__(self, k, epsilon, sigma, tau):
        self.k = k*k
        self.epsilon = epsilon
        self.sigma = sigma
        self.tau = tau

    def run(self, X, W):
        # Initializing class variables X (data set) and W (initial protos).
        self.W = W
        self.X = X
        # Current index for plotting.
        self.currentIndex = 0
        # Plotting initial state.
        self.plot();
        # Iterating through every data point.
        for alpha in range(X.shape[0]):
            self.currentIndex = alpha;
            if (alpha == X.shape[0]/2):
                self.plot()
            if(alpha > X.shape[0]/4):
                self.epsilon *= self.tau

            p = np.argmin([np.linalg.norm(self.X[alpha] - self.W[q]) for q i
n range(self.k)])
            dw = self.epsilon * np.array([np.exp(-(q-p)**2/(2*self.sigma**2)
)*(self.X[alpha]-self.W[q]) for q in range(self.k)])
            self.W += dw
        self.plot()
    def plot(self):
        fig = plt.figure(figsize=(10,7))
        plot = fig.add_subplot(111, projection='3d')
        plot.scatter(self.X[:,0], self.X[:,1], self.X[:,2], color='black')
        plot.scatter(self.W[:,0], self.W[:,1], self.W[:,2], color='red')
        plot.set_title('SOM with k=' + str(self.k) + ', sigma=' + str(self.s
igma) + ', alpha=' + str(self.currentIndex))
        plot.set_xlabel('X')
        plot.set_ylabel('Y')
        plot.set_zlabel('Z')
        plt.show()
```

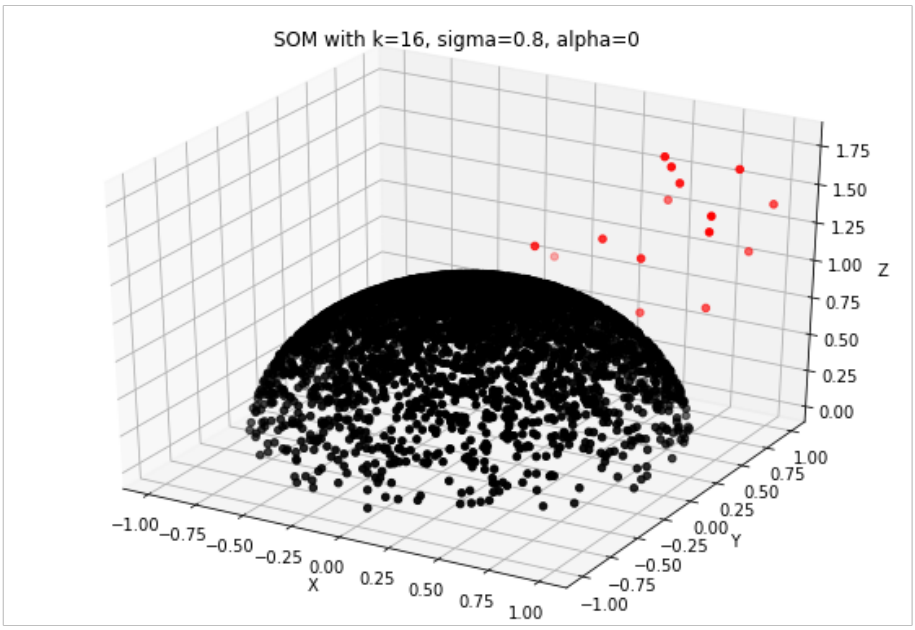
SOM with 3D data

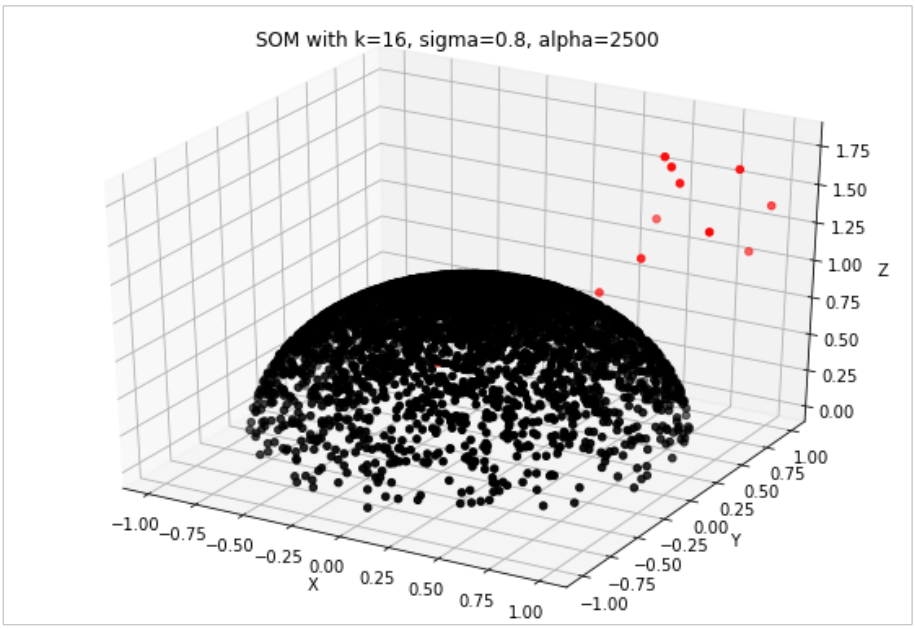
In [23]:
```
kArray = [4, 8]
sigArray = [0.6, 0.8]

for k in kArray:
    for sigma in sigArray:
        som3D = SOM3D(k=k, sigma=sigma, epsilon=0.01, tau=0.5)
        mean = np.mean(data3D, axis=0)
        W = np.array([mean + np.random.rand(data3D.shape[1]) for _ in range(
k*k)])
        som3D.run(data3D, W)
```

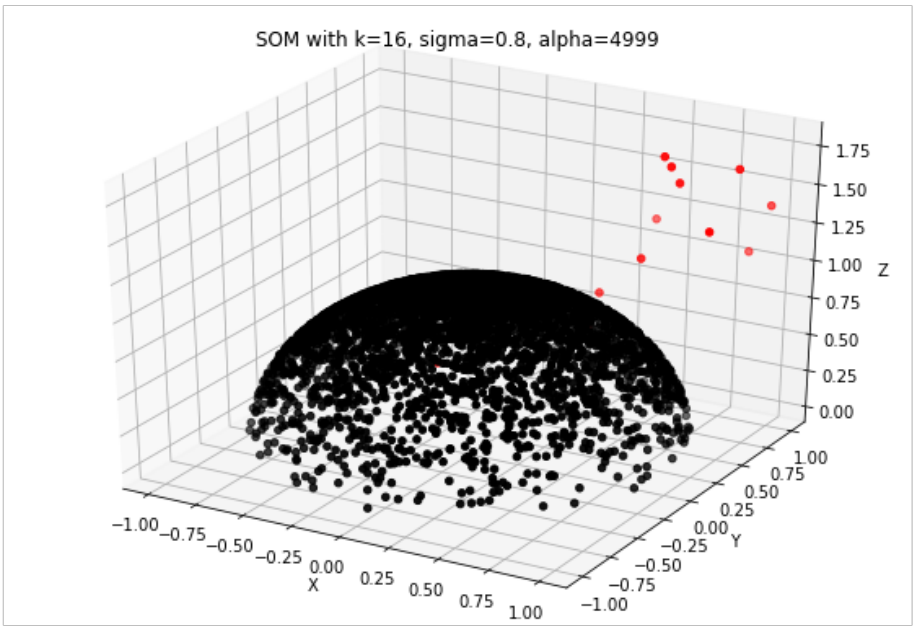

SOM with k=16, sigma=0.6, alpha=0
X
Y
Z

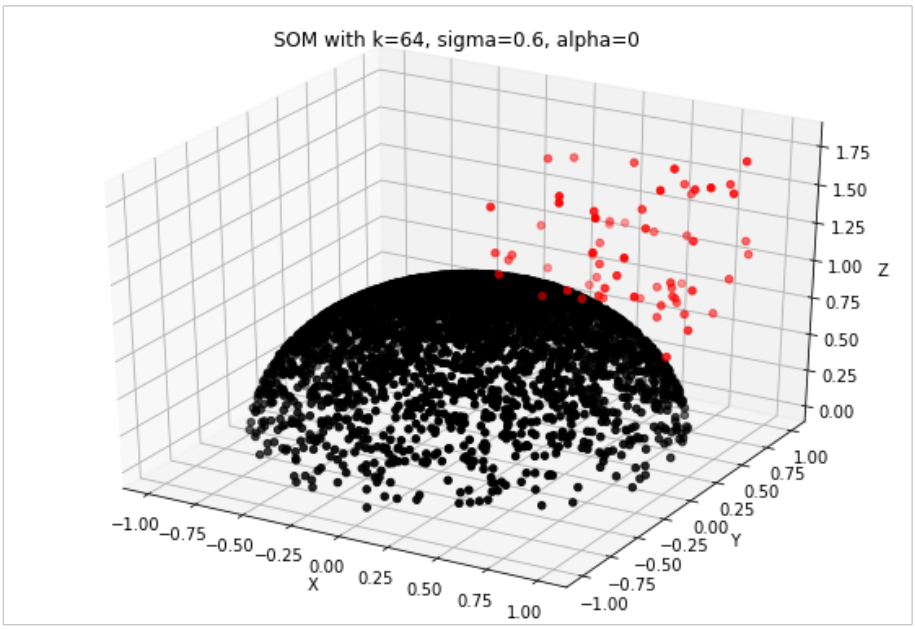

SOM with k=16, sigma=0.6, alpha=2500
X
Y
Z

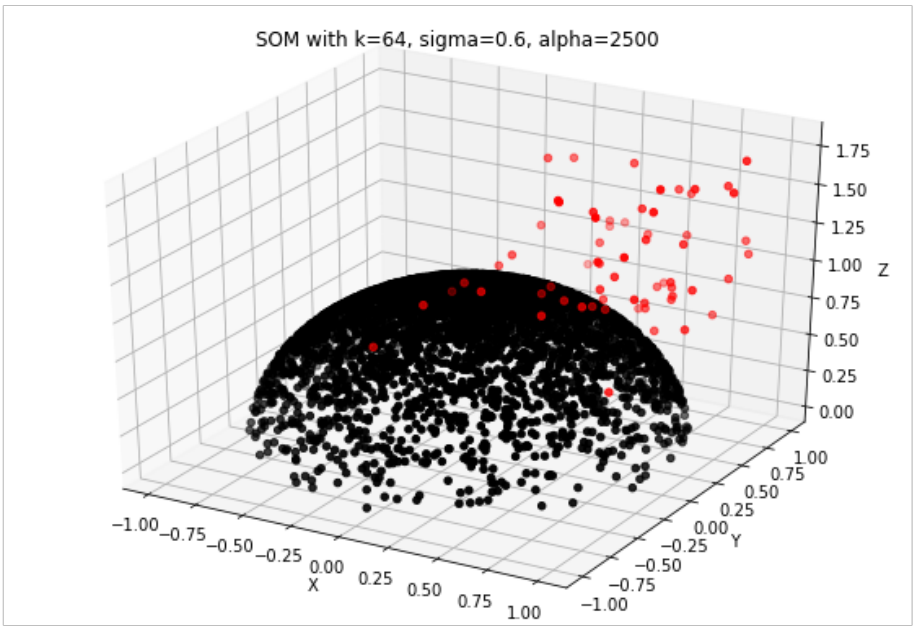

SOM with k=16, sigma=0.6, alpha=4999
X
Y
Z

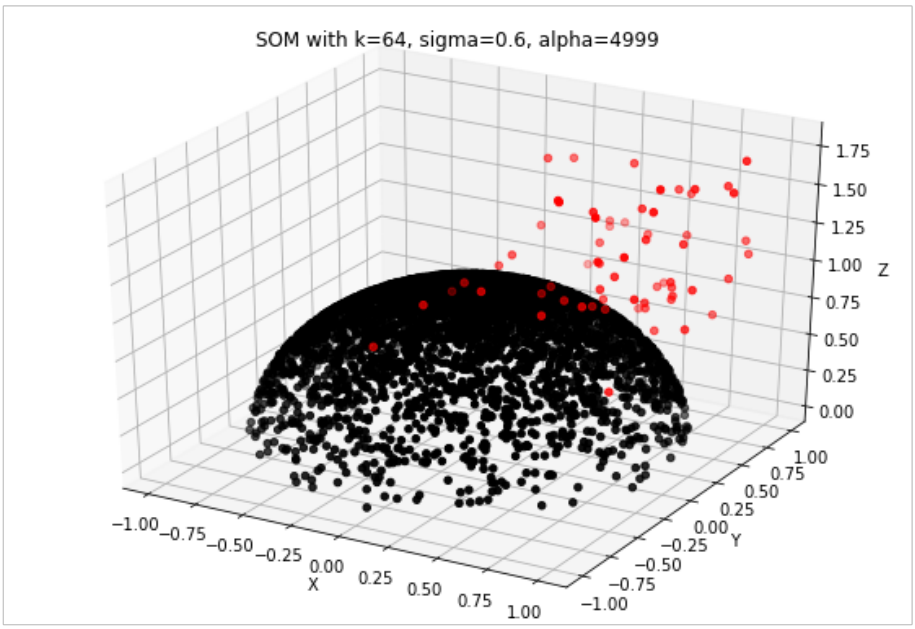

SOM with k=16, sigma=0.8, alpha=0
X
Y
Z

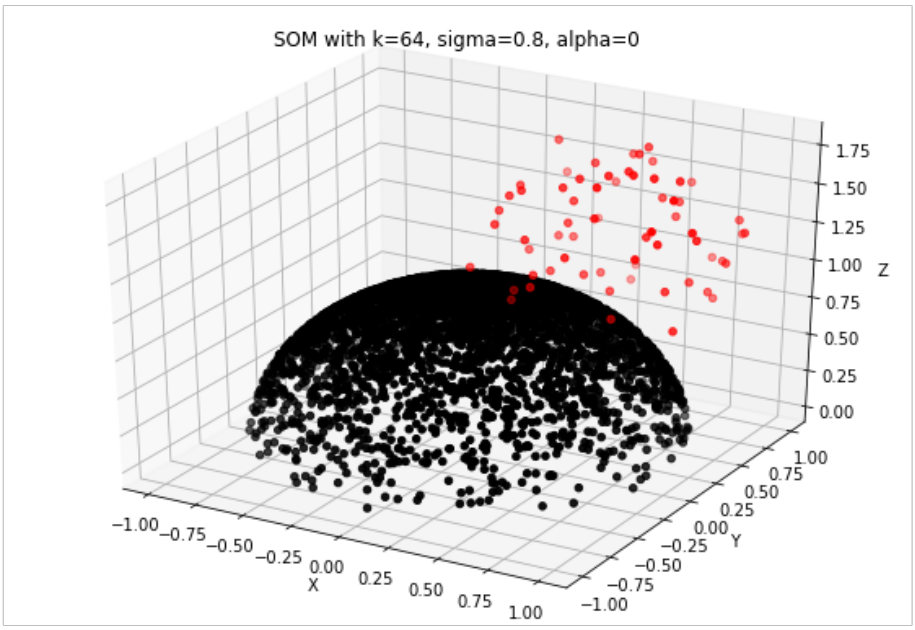

SOM with k=16, sigma=0.8, alpha=2500
X
Y
Z

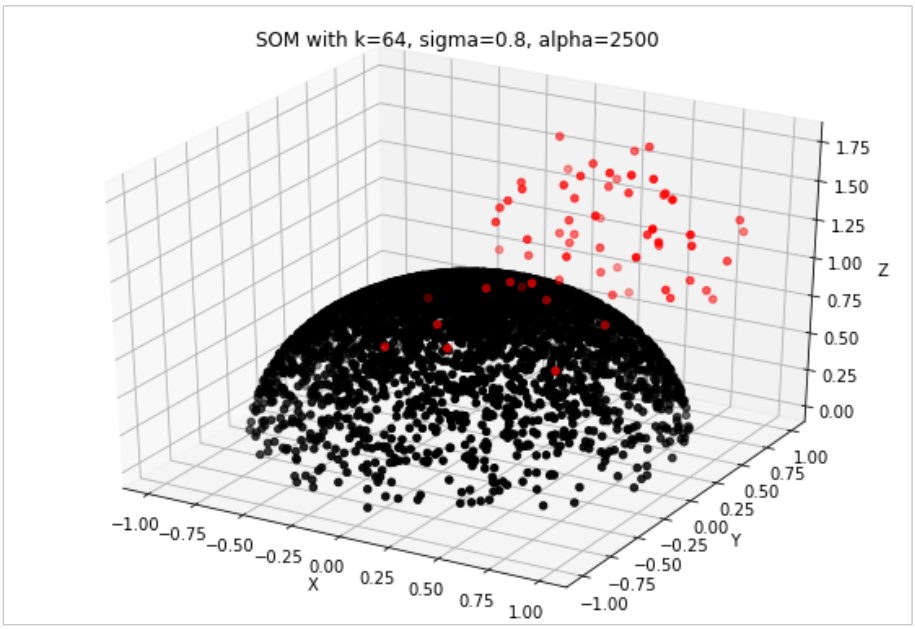

SOM with k=16, sigma=0.8, alpha=4999
X
Y
Z

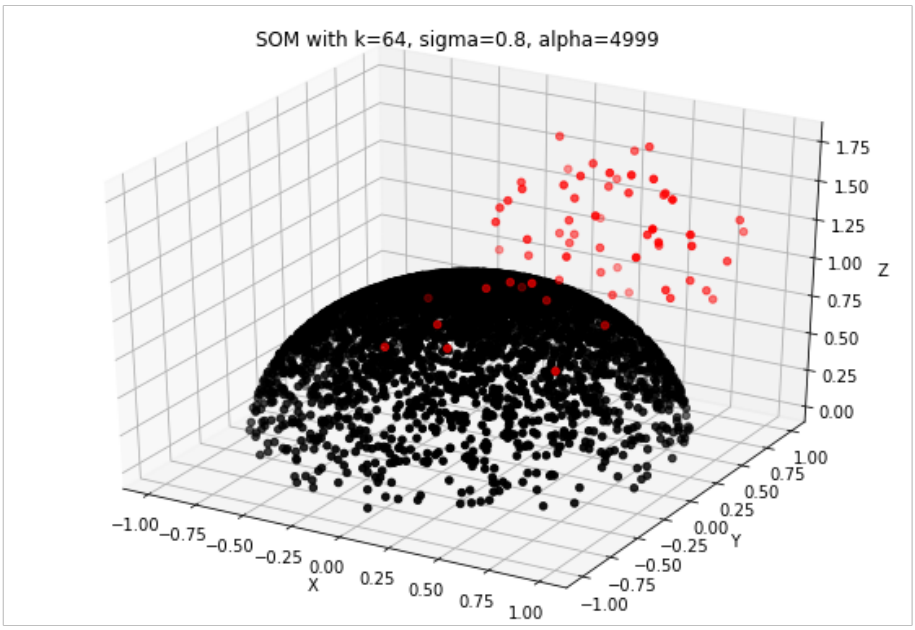

SOM with k=64, sigma=0.6, alpha=0
1.75
1.50
1.25
1.00
Z
0.75
0.50
0.25
0.00
1.00
0.75
0.50
0.25
0.00
−0.25
Y
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
X
0.25
0.50
0.75
1.00


SOM with k=64, sigma=0.6, alpha=2500
1.75
1.50
1.25
1.00
Z
0.75
0.50
0.25
0.00
1.00
0.75
0.50
0.25
0.00
−0.25
Y
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
X
0.25
0.50
0.75
1.00


SOM with k=64, sigma=0.6, alpha=4999
X
Y
Z


SOM with k=64, sigma=0.8, alpha=0
X
Y
Z


SOM with k=64, sigma=0.8, alpha=2500
X
Y
Z


SOM with k=64, sigma=0.8, alpha=4999
X
Y
Z