In [4]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
from scipy.stats import multivariate_normal as mn
from scipy.stats import chi2 as chi2
import random
import math
import time
```

In [5]:
```python
traindata=pd.read_table('/Users/adilrhiulam/Downloads/classification_data_HWK2-2/EMGaussiandat
a.txt',names=['x','y'],sep=' ')
testdata=pd.read_table('/Users/adilrhiulam/Downloads/classification_data_HWK2-2/EMGaussiantest.
txt',names=['x','y'],sep=' ')
```

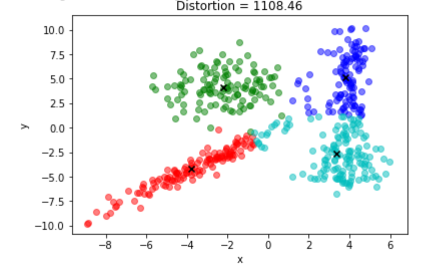
In [6]:
```python
train_data=traindata.values.tolist()
test_data=testdata.values.tolist()
```

In [5]:
```python
def K_means(data,k):
    new_classes=[[] for i in range(k)]
    previous_classes=[[0] for i in range(k)]
    np.random.shuffle(data)
    clusters_centers=np.array(data[0:k])
    while(new_classes!=previous_classes):
        previous_classes=new_classes
        new_classes=[[] for i in range(k)]
        for i in range(0,len(data)):
            L=np.array([np.linalg.norm(np.subtract(clusters_centers,data[i]),axis=1)])
            p=np.argmin(L)
            new_classes[p].append(data[i])
        clusters_centers=[np.mean(np.array(new_classes[j]),axis=0) for j in range(k)]
    return [np.array(new_classes),np.array(clusters_centers)]

def distortion(kmeans):
    distortion=np.sum([np.sum([np.linalg.norm(np.subtract(kmeans[0][j],kmeans[1][j]),axis=1)])
for j in range(len(kmeans[0]))])
    return [kmeans[1],distortion]
```

In [38]:
```python
def plt_kmeans(cluster,center,color):
    plt.scatter(np.array(cluster)[:,0],np.array(cluster)[:,1],c=color,alpha=0.5)
    plt.scatter(np.array(center)[0],np.array(center)[1],c='black',marker='x')
```

In [196]:
```python
solution=K_means(train_data,4)
plt_kmeans(solution[0][0],solution[1][0],'r')
plt_kmeans(solution[0][1],solution[1][1],'g')
plt_kmeans(solution[0][2],solution[1][2],'b')
plt_kmeans(solution[0][3],solution[1][3],'c')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Figure 1: K-means algorithm Result for a random intialization')
Distortion='Distortion = ' + str(round(distortion(solution)[1],2))
plt.title(Distortion )
plt.show()

print('Cluster centers are:', (np.round(solution[1],2)))
```

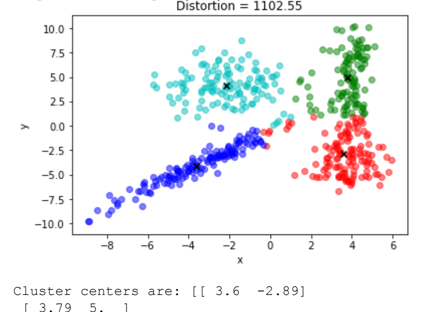
Figure 1: K-means algorithm Result for a random intialization
Distortion = 1108.46




```
Cluster centers are: [[-3.78 -4.22]
 [-2.24  4.16]
 [ 3.8   5.1 ]
 [ 3.36 -2.66]]
```

In [174]:
```python
solution=K_means(train_data,4)
plt_kmeans(solution[0][0],solution[1][0],'r')
plt_kmeans(solution[0][1],solution[1][1],'g')
plt_kmeans(solution[0][2],solution[1][2],'b')
plt_kmeans(solution[0][3],solution[1][3],'c')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Figure 2: K-means algorithm Result for a second random intialization')
Distortion='Distortion = ' + str(round(distortion(solution)[1],2))
plt.title(Distortion )
plt.show()
print('Cluster centers are:', (np.round(solution[1],2)))
```

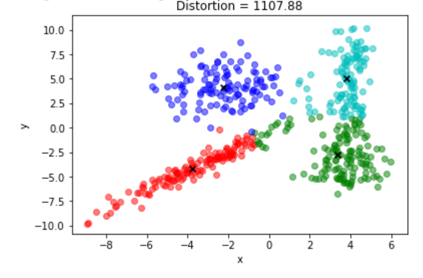
Figure 2: K-means algorithm Result for a second random intialization
Distortion = 1102.55




```
Cluster centers are: [[ 3.6  -2.89]
 [ 3.79  5.  ]
 [-3.64 -4.05]
 [-2.16  4.11]]
```

In [215]:
```python
solution=K_means(train_data,4)
plt_kmeans(solution[0][0],solution[1][0],'r')
plt_kmeans(solution[0][1],solution[1][1],'g')
plt_kmeans(solution[0][2],solution[1][2],'b')
plt_kmeans(solution[0][3],solution[1][3],'c')
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Figure 3: K-means algorithm Result for a third random intialization')
Distortion='Distortion = ' + str(round(distortion(solution)[1],2))
plt.title(Distortion )
plt.show()
print('Cluster centers are:', (np.round(solution[1],2)))
```

Figure 3: K-means algorithm Result for a third random intialization
Distortion = 1107.88




```
Cluster centers are: [[-3.78 -4.22]
 [ 3.36 -2.71]
 [-2.24  4.16]
 [ 3.8   5.03]]
```

In [413]:
```python
from sklearn.cluster import KMeans
import numpy as np
kmeans = KMeans(4,init='random', random_state=0).fit(train_data)
kmeans.cluster_centers_
```

Out[413]:
```
array([[ 3.80280826,  5.10467248],
       [ 3.36449672, -2.65646983],
       [-2.23856221,  4.16339661],
       [-3.78479953, -4.21639713]])
```

In [6]:
```python
def initializationStep(data,k):
    k_m=K_means(data,k)
    gaussian_means=k_m[1]              # Initialize the means by K_means
    covariances=np.random.random(k)    # Random strict positive initialization for the variance

    weights=[len(k_m[0][i])/len(data) for i in range(k)]    # Weights initialized by K_means
    return [gaussian_means,covariances,weights]
```

In [7]:
```python
def stepE(data,means,cov,weights,k):
    g_w=[[weights[j]*mn.pdf(data[i],means[j],cov[j]) for j in range(k)] for i in range(len(data
))]  #weight*Gaussian
    resp=np.array([[g_w[i][j]/np.sum(g_w[i]) for j in range(k)] for i in range(len(data))]) #Re
sponsabilities(Posteriors)
    log_llh=np.sum([math.log2(np.sum(g_w,axis=1)[i]) for i in range(len(data))]) #Loglikelihood
    return [resp,log_llh]
```

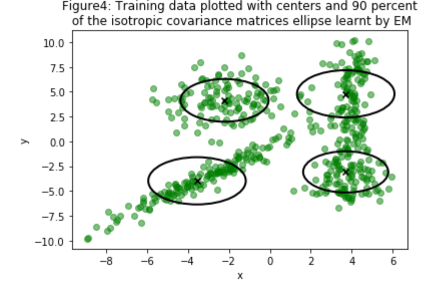
In [8]:
```python
def stepM(data,resp,k):
    resp_sum=np.sum(resp,axis=0) #Summation over data for each i dans {1,...,k}
    weights=resp_sum/len(data)
    means=np.divide(np.dot(resp.T,data),resp_sum[:,None])
    d=np.shape(means)[1]
    cov=np.divide(np.array([np.sum(resp[:,i]*np.linalg.norm(data-means[i],axis=1)) for i in ran
ge(4)]),d*resp_sum)
    return [means,cov,weights]
```

In [9]:
```python
def algorithmEM(data,k):
    [means,covariances,weights]=initializationStep(data,k)
    [responsabilities,log_llh]=stepE(data,means,covariances,weights,k)
    log_llh2=log_llh+1
    while(log_llh2!=log_llh):
        log_llh2=log_llh
        [means,covariances,weights]=stepM(data,responsabilities,k)
        [responsabilities,log_llh]=stepE(data,means,covariances,weights,k)
    return [means,covariances,weights,responsabilities,log_llh]
```

In [42]:
```python
resultEM=algorithmEM(train_data,4)
```

In [430]:
```python
L=[resultEM[1][i]*np.eye(len(resultEM[0][0])) for i in range(len(resultEM[0]))]#Transform the
 variances to cov matrices
fig = plt.figure()
ax = fig.add_subplot(111, aspect='auto')
ax.scatter(np.array(train_data)[:,0],np.array(train_data)[:,1],c='g',alpha=0.5)
ax.scatter(np.array(resultEM[0])[:,0],np.array(resultEM[0])[:,1],c='black',marker='x')
eigen_elements=[np.linalg.eig(L[i]) for i in range(4)]
quantile=chi2.ppf(0.90, 2)
ellipse_axis=[2*np.sqrt(quantile*eigen_elements[i][0]) for i in range(4)]
ellipse_angles=[math.degrees(math.atan2(eigen_elements[i][1][0][0],eigen_elements[i][1][0][1
])) for i in range(4)]
ellipses=[Ellipse(resultEM[0][i],ellipse_axis[i][1],ellipse_axis[i][0],ellipse_angles[i]) for
i in range(4)]
for e in ellipses:
    e.set_fill(False)
    e.set_linewidth(2)
    e.set_alpha(1)
    ax.add_artist(e)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Figure4: Training data plotted with centers and 90 percent\n of the isotropic covar
iance matrices ellipse learnt by EM')
plt.show()
```

Figure4: Training data plotted with centers and 90 percent
of the isotropic covariance matrices ellipse learnt by EM



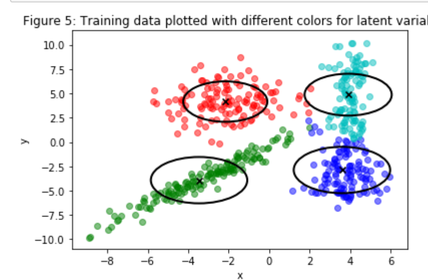
In [44]:
```python
print('Maximum Likelihood of Train data with isotropic covariances is =', resultEM[4])
```

```
Maximum Likelihood of Train data with isotropic covariances is = -4469.33658901
```

In [393]:
```python
def clusters(traindata,testdata,k):
    convergenceEM=algorithmEM(traindata,k)
    resp=convergenceEM[3]
    clusters_center=convergenceEM[0]
    clusters=[[] for i in range(k)]
    for i in range(np.shape(resp)[0]):
        p=np.argmax(resp[i])
        clusters[p].append(data[i])
    return [np.array(clusters),np.array(clusters_center),convergenceEM[1]]
```

In [428]:
```python
result=clusters(train_data,4)
fig = plt.figure()
ax = fig.add_subplot(111, aspect='auto')
ax.scatter(np.array(result[0][0])[:,0],np.array(result[0][0])[:,1],c='r',alpha=0.5)
ax.scatter(np.array(result[0][1])[:,0],np.array(result[0][1])[:,1],c='g',alpha=0.5)
ax.scatter(np.array(result[0][2])[:,0],np.array(result[0][2])[:,1],c='b',alpha=0.5)
ax.scatter(np.array(result[0][3])[:,0],np.array(result[0][3])[:,1],c='c',alpha=0.5)
ax.scatter(np.array(result[1])[:,0],np.array(result[1])[:,1],c='black',marker='x')
eigen_elements=[np.linalg.eig(L[i]) for i in range(4)]
quantile=chi2.ppf(0.90, 2)
ellipse_axis=[2*np.sqrt(quantile*eigen_elements[i][0]) for i in range(4)]
ellipse_angles=[math.degrees(math.atan2(eigen_elements[i][1][0][0],eigen_elements[i][1][0][1
])) for i in range(4)]
ellipses=[Ellipse(result[1][i],ellipse_axis[i][1],ellipse_axis[i][0],ellipse_angles[i]) for i
in range(4)]
for e in ellipses:
    e.set_fill(False)
    e.set_linewidth(2)
    e.set_alpha(1)
    ax.add_artist(e)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Figure 5: Training data plotted with different colors for latent variables ')
plt.show()
```

Figure 5: Training data plotted with different colors for latent variables



In [10]:
```python
def intializationStepGeneralCase(data,k):
    k_m=K_means(data,k)
    gaussian_means=k_m[1]
    cov=np.array([np.sum([list(np.dot(np.array([k_m[0][i][j]-gaussian_means[i]]).T,[k_m[0][i][j
]-gaussian_means[i]])) for j in range(len(k_m[0][i]))],axis=0)/len(k_m[0][i]) for i in range(k
)])
    weights=[len(k_m[0][i])/len(data) for i in range(k)]
    return [gaussian_means,cov,weights]
```

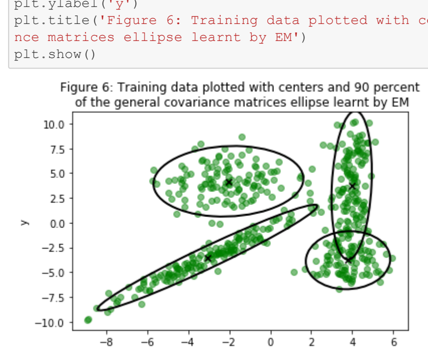
In [11]:
```python
def stepMGeneralCase(data,resp,k):
    resp_sum=np.sum(resp,axis=0) #Summation over data for each i dans {1,...,k}
    weights=resp_sum/len(data)
    means=np.divide(np.dot(resp.T,data),resp_sum[:,None])
    d=np.shape(means)[1]
    cov=np.array([np.sum([resp[:,i][j]*np.dot(np.array([data[j]-means[i]]).T,[data[j]-means[i
]]) for j in range(len(data))],axis=0)/resp_sum[i] for i in range(k)])
    return [means,cov,weights]
```

In [17]:
```python
def algorithmEMgeneralCase(data,k):
    [means,covariances,weights]=intializationStepGeneralCase(data,k)
    [responsabilities,log_llh]=stepE(data,means,covariances,weights,k)
    log_llh2=log_llh+1
    while(log_llh2!=log_llh):
        log_llh2=log_llh
        [means,covariances,weights]=stepMGeneralCase(data,responsabilities,k)
        [responsabilities,log_llh]=stepE(data,means,covariances,weights,k)
    return [means,covariances,weights,responsabilities,log_llh]
```

In [18]:
```python
resultEMgenCase=algorithmEMgeneralCase(train_data,4)
```

In [21]:
```python
fig = plt.figure()
ax = fig.add_subplot(111, aspect='auto')
ax.scatter(np.array(train_data)[:,0],np.array(train_data)[:,1],c='g',alpha=0.5)
ax.scatter(np.array(resultEMgenCase[0])[:,0],np.array(resultEMgenCase[0])[:,1],c='black',marker
='x')
eigenElements=[np.linalg.eig(resultEMgenCase[1][i]) for i in range(4)]
quantile=chi2.ppf(0.90, 2)
ellipse_axis=[2*np.sqrt(quantile*eigenElements[i][0]) for i in range(4)]
ellipse_angles=[math.degrees(math.atan2(eigenElements[i][1][0][0],eigenElements[i][1][0][1])) f
or i in range(4)]
ellipses=[Ellipse(resultEMgenCase[0][i],ellipse_axis[i][1],ellipse_axis[i][0],ellipse_angles[i
]) for i in range(4)]
for e in ellipses:
    e.set_fill(False)
    e.set_linewidth(2)
    e.set_alpha(1)
    ax.add_artist(e)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Figure 6: Training data plotted with centers and 90 percent\n of the general covaria
nce matrices ellipse learnt by EM')
plt.show()
```

Figure 6: Training data plotted with centers and 90 percent
of the general covariance matrices ellipse learnt by EM



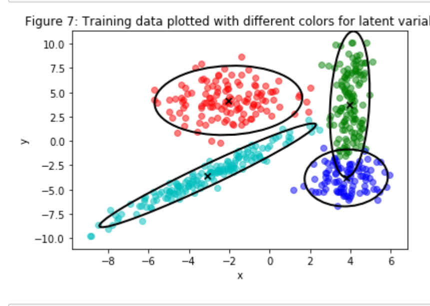
In [22]:
```python
print('Maximum Likelihood of Train data with general covariances is =', resultEMgenCase[4])
```

```
Maximum Likelihood of Train data with general covariances is = -3358.18386081
```

In [23]:
```python
def clustersEMgeneralCase(data,k):
    convergenceEM=algorithmEMgeneralCase(data,k)
    resp=convergenceEM[3]
    clusters_center=convergenceEM[0]
    clusters=[[] for i in range(k)]
    for i in range(np.shape(resp)[0]):
        p=np.argmax(resp[i])
        clusters[p].append(data[i])
    return [np.array(clusters),np.array(clusters_center),convergenceEM[1]]
```

In [24]:
```python
result=clustersEMgeneralCase(train_data,4)
fig = plt.figure()
ax = fig.add_subplot(111, aspect='auto')
ax.scatter(np.array(result[0][0])[:,0],np.array(result[0][0])[:,1],c='r',alpha=0.5)
ax.scatter(np.array(result[0][1])[:,0],np.array(result[0][1])[:,1],c='g',alpha=0.5)
ax.scatter(np.array(result[0][2])[:,0],np.array(result[0][2])[:,1],c='b',alpha=0.5)
ax.scatter(np.array(result[0][3])[:,0],np.array(result[0][3])[:,1],c='c',alpha=0.5)
ax.scatter(np.array(result[1])[:,0],np.array(result[1])[:,1],c='black',marker='x')
eigenElements=[np.linalg.eig(result[2][i]) for i in range(4)]
quantile=chi2.ppf(0.90, 2)
ellipse_axis=[2*np.sqrt(quantile*eigenElements[i][0]) for i in range(4)]
ellipse_angles=[math.degrees(math.atan2(eigenElements[i][1][0][0],eigenElements[i][1][0][1])) f
or i in range(4)]
ellipses=[Ellipse(result[1][i],ellipse_axis[i][1],ellipse_axis[i][0],ellipse_angles[i]) for i i
n range(4)]
for e in ellipses:
    e.set_fill(False)
    e.set_linewidth(2)
    e.set_alpha(1)
    ax.add_artist(e)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Figure 7: Training data plotted with different colors for latent variables')
plt.show()
```

Figure 7: Training data plotted with different colors for latent variables



In [25]:
```python
t=time.time()
resultTestData=algorithmEM(test_data,4)
elapsed=time.time()-t
```

In [47]:
```python
print('Maximum Likelihood of Test data with isotropic covariances is =', resultTestData[4])
print('time elapsed to compute the maximum log-likelihood by the EM algorithm is =',elapsed)
```