Question 1:

a) Number of datasets: 10

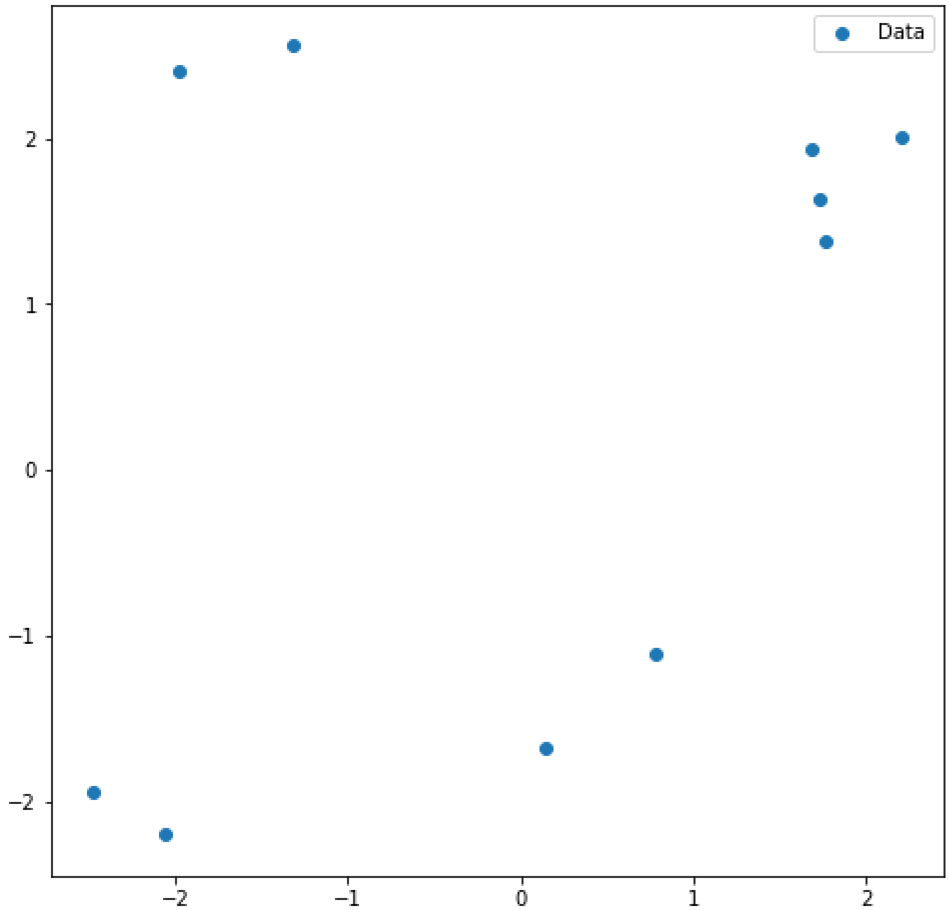
In [548]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
from sklearn.mixture import GaussianMixture
```

In [600]:
```
t_mu = [[2, 2], [-2, 2], [-2, -2], [1, -1]]
t_var_1 = [[0.1, 0], [0, 0.1]]
t_var_2 = [[0.2, 0.1], [0.1, 0.3]]
t_var_3 = [[0.3, 0], [0, 0.2]]
t_var_4 = [[0.2, 0], [0, 0.3]]
prior = [0.30, 0.25, 0.28, 0.17]

M = 6
N = 10
l_1 = 0
l_2 = 0
l_3 = 0
l_4 = 0

likelihood = []
bic = []
```

In [601]:
```
#generating number of sample for the GMMs
for i in range(N):
    temp = np.random.uniform(0, 1, 1)
    if temp <= prior[0]:
        l_1 = l_1 + 1
    elif temp <= prior[0] + prior[1]:
        l_2 = l_2 + 1
    elif temp <= prior[0] + prior[1] + prior[2]:
        l_3 = l_3 + 1
l_4 = N - l_1 - l_2 - l_3
```

In [602]:
```
#generating data according to component
data = []

for i in range(l_1):
    data.append(np.random.multivariate_normal(t_mu[0], t_var_1, 1))
for i in range(l_2):
    data.append(np.random.multivariate_normal(t_mu[1], t_var_2, 1))
for i in range(l_3):
    data.append(np.random.multivariate_normal(t_mu[2], t_var_3, 1))
for i in range(l_4):
    data.append(np.random.multivariate_normal(t_mu[3], t_var_4, 1))
```

In [603]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(data).reshape(N,2)[:, 0], np.array(data).reshape(N,2)[:, 1
], alpha=1, label='Data')
ax.legend()
plt.show()
```

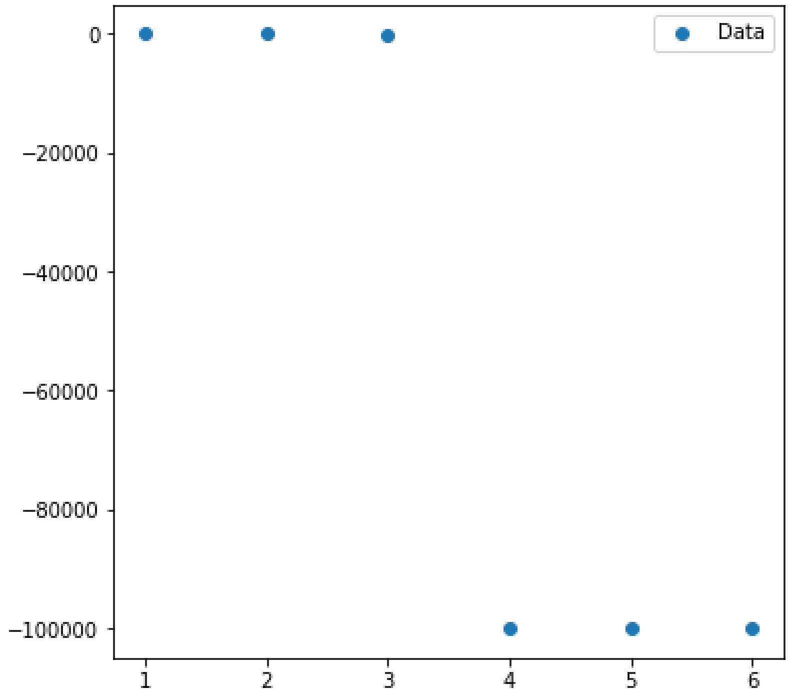


In [604]:

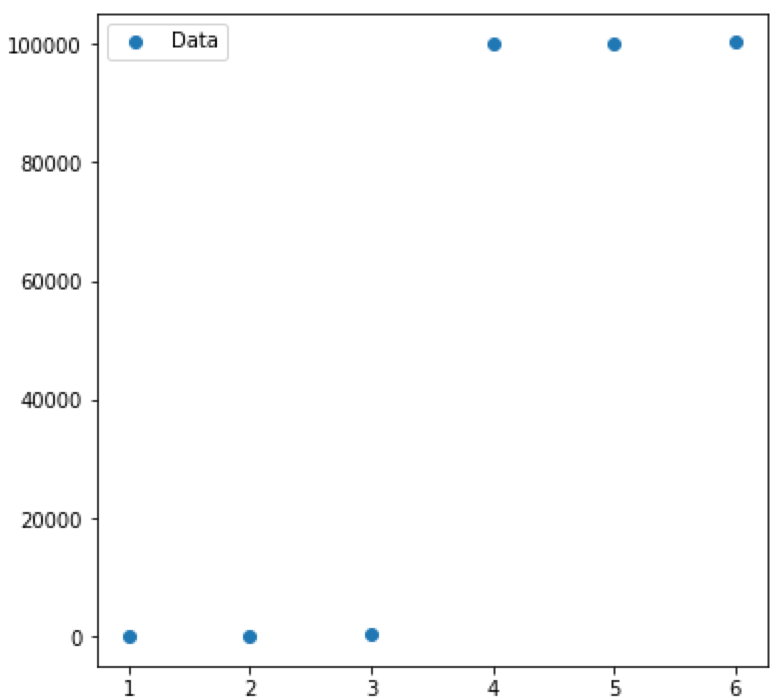
```
for h in range(M):
    sum_3 = 0
    for c in range(10):
        sum_2 = 0
        new_data = []
        for d in range(0, int(0.1*N*c), 1):
            new_data.append(np.array(data).reshape(N,2)[int(d), :])
        for d in range(int(0.1*N*(c+1)), N, 1):
            new_data.append(np.array(data).reshape(N,2)[int(d), :])
        gmm = GaussianMixture(n_components = h+1)
        label = gmm.fit_predict(np.array(new_data).reshape(int(0.9*N), 2))
        alpha = gmm.weights_
        mean = gmm.means_
        covariance = gmm.covariances_
        for i in range(int(0.1*N*c), int(0.1*N*(c+1)), 1):
            sum_1 = 0
            for j in range(h+1):
                p = math.exp(-0.5*np.matmul(np.matmul((np.array(data).reshape(
N,2)[i,:]) - mean[j,:],
                    np.linalg.inv(covariance[j])), (np.array(data).reshape(N,2
)[i,:]) - mean.reshape(h+1,2)[j,:]))/(2*math.pi*np.linalg.det(covariance[j]))
                sum_1 = sum_1 + alpha[j]*p
            sum_2 = sum_2 + np.log(sum_1)
        sum_3 = sum_3 +sum_2
    if np.isinf(sum_3) == True:
        likelihood.append(-10**5)
        bic.append(10**5 + h**3*np.log(N))
    else:
        likelihood.append(sum_3/N)
        bic.append(-2*sum_3 + h**3*np.log(N))
```

```
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
C:\Users\nanda\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: divide by zero encountered in log
```

In [605]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.scatter([1,2,3,4,5,6], np.array(likelihood), alpha=1, label='Data')
ax.legend()
plt.show()
```



In [606]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.scatter([1,2,3,4,5,6], np.array(bic), alpha=1, label='Data')
ax.legend()
plt.show()
```



In [607]:
```
print(likelihood)
```

```
[-5.543328967054174, -3.6569408331213458, -21.66188236435191, -100000, -10000
0, -100000]
```

In [608]:
```
print(bic)
```

```
[110.8665793410835, 75.44140175542097, 451.65832803099056, 100062.1697975108
4, 100147.36544595163, 100287.82313662425]
```

In [609]:
```
print("The GMM order that gets selected based on the likelihood values is : {}
".format(np.argmax(likelihood)+1))
print("The GMM order that gets selected based on the BIC values is : {}".forma
t(np.argmin(bic)+1))
```

```
The GMM order that gets selected based on the likelihood values is : 2
The GMM order that gets selected based on the BIC values is : 2
```

In [ ]:

Question 1:

b) Number of datasets: 100

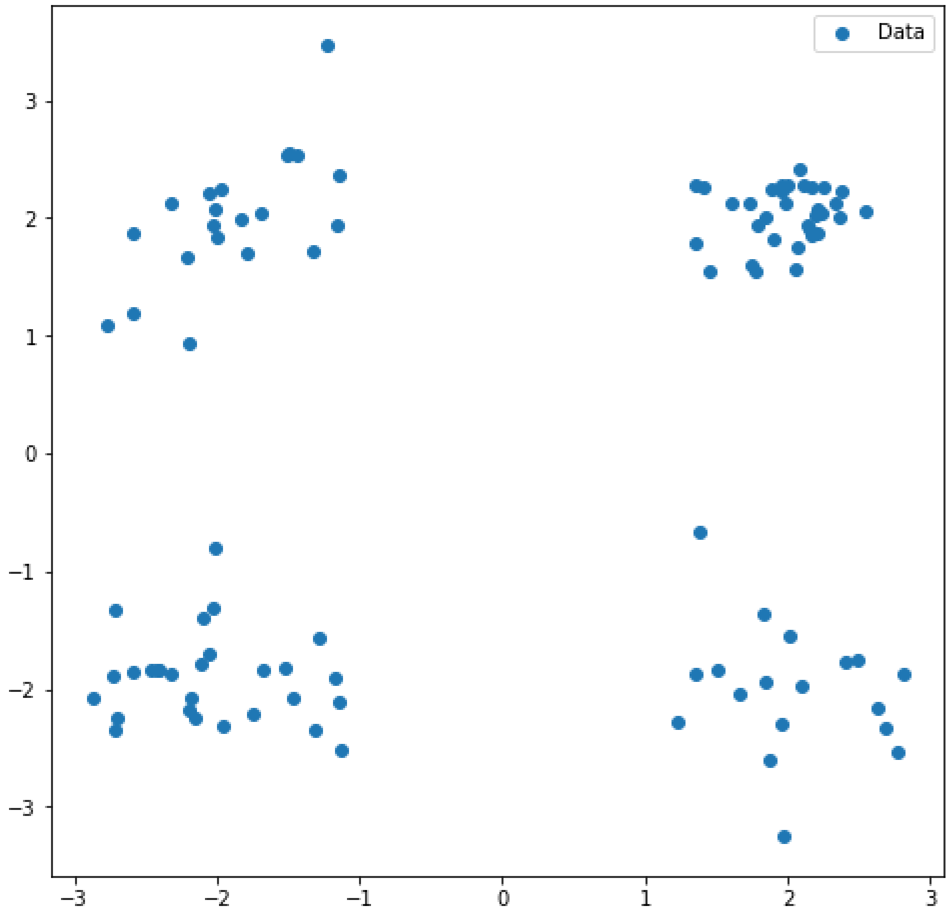
In [49]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
from sklearn.mixture import GaussianMixture
```

In [111]:
```
t_mu = [[2, 2], [-2, 2], [-2, -2], [2, -2]]
t_var_1 = [[0.1, 0], [0, 0.1]]
t_var_2 = [[0.2, 0.1], [0.1, 0.3]]
t_var_3 = [[0.3, 0], [0, 0.2]]
t_var_4 = [[0.2, 0], [0, 0.3]]
prior = [0.30, 0.25, 0.28, 0.17]

M = 6
N = 100
l_1 = 0
l_2 = 0
l_3 = 0
l_4 = 0

likelihood = []
bic = []
```

In [112]:
```
#generating number of sample for the GMMs
for i in range(N):
    temp = np.random.uniform(0, 1, 1)
    if temp <= prior[0]:
        l_1 = l_1 + 1
    elif temp <= prior[0] + prior[1]:
        l_2 = l_2 + 1
    elif temp <= prior[0] + prior[1] + prior[2]:
        l_3 = l_3 + 1
l_4 = N - l_1 - l_2 - l_3
```

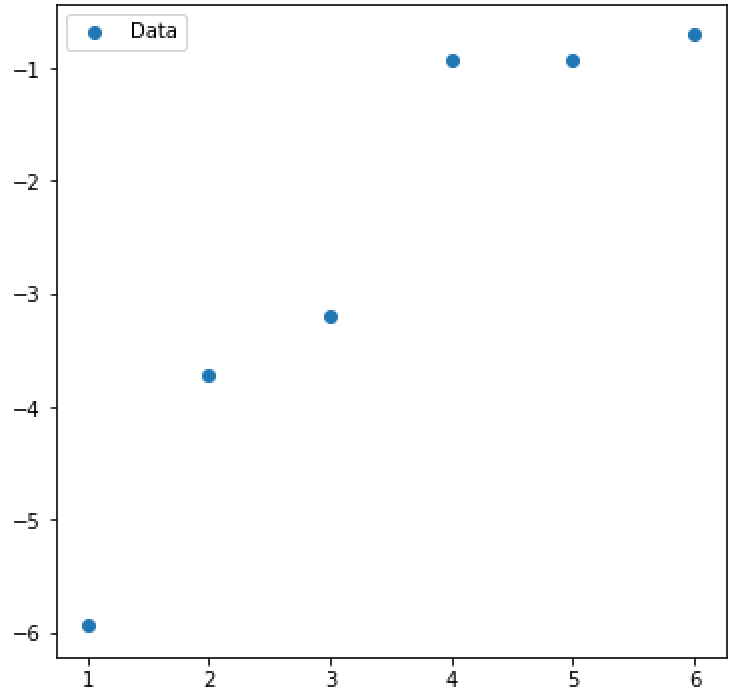
In [113]:
```
#generating data according to component
data = []

for i in range(l_1):
    data.append(np.random.multivariate_normal(t_mu[0], t_var_1, 1))
for i in range(l_2):
    data.append(np.random.multivariate_normal(t_mu[1], t_var_2, 1))
for i in range(l_3):
    data.append(np.random.multivariate_normal(t_mu[2], t_var_3, 1))
for i in range(l_4):
    data.append(np.random.multivariate_normal(t_mu[3], t_var_4, 1))
```

In [114]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(data).reshape(N,2)[:, 0], np.array(data).reshape(N,2)[:, 1
], alpha=1, label='Data')
ax.legend()
plt.show()
```

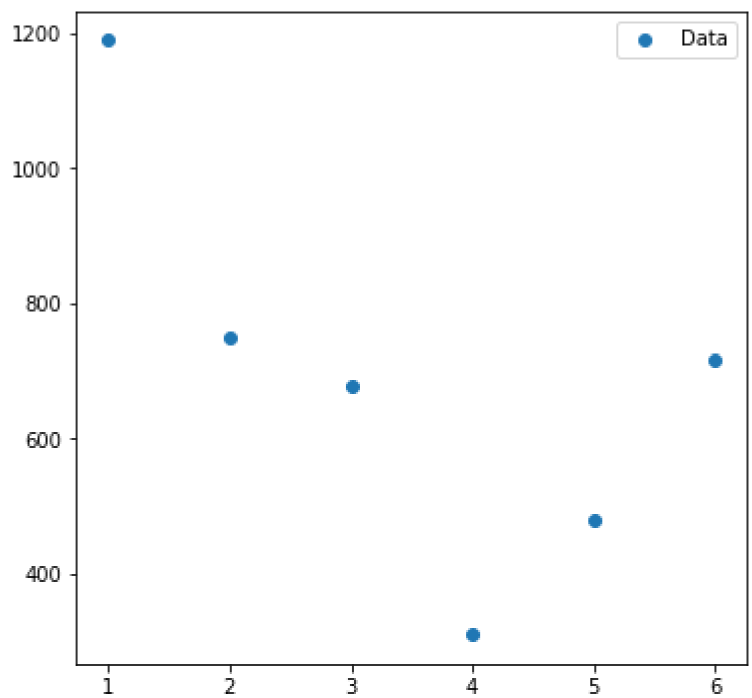


In [115]:
```
for h in range(M):
    sum_3 = 0
    for c in range(10):
        sum_2 = 0
        new_data = []
        for d in range(0, int(0.1*N*c), 1):
            new_data.append(np.array(data).reshape(N,2)[int(d), :])
        for d in range(int(0.1*N*(c+1)), N, 1):
            new_data.append(np.array(data).reshape(N,2)[int(d), :])
        gmm = GaussianMixture(n_components = h+1)
        label = gmm.fit_predict(np.array(new_data).reshape(int(0.9*N), 2))
        alpha = gmm.weights_
        mean = gmm.means_
        covariance = gmm.covariances_
        for i in range(int(0.1*N*c), int(0.1*N*(c+1)), 1):
            sum_1 = 0
            for j in range(h+1):
                p = math.exp(-0.5*np.matmul(np.matmul((np.array(data).reshape(
N,2)[i,:]) - mean[j,:],
                    np.linalg.inv(covariance[j])), (np.array(data).reshape(N,2
)[i,:]) - mean.reshape(h+1,2)[j,:]))/(2*math.pi*np.linalg.det(covariance[j]))
                sum_1 = sum_1 + alpha[j]*p
            sum_2 = sum_2 + np.log(sum_1)
        sum_3 = sum_3 +sum_2
    if np.isinf(sum_3) == True:
        likelihood.append(-10**5)
        bic.append(10**5 + h**3*np.log(N))
    else:
        likelihood.append(sum_3/N)
        bic.append(-2*sum_3 + h**3*np.log(N))
```

In [116]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.scatter([1,2,3,4,5,6], np.array(likelihood), alpha=1, label='Data')
ax.legend()
plt.show()
```



In [117]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.scatter([1,2,3,4,5,6], np.array(bic), alpha=1, label='Data')
ax.legend()
plt.show()
```



In [118]:
```
print(likelihood)
```

```
[-5.942509754929267, -3.7140544837987135, -3.203501957389327, -0.931701790152
4847, -0.9212217334601267, -0.6975922646844022]
```

In [119]:
```
print(bic)
```

```
[1188.5019509858535, 747.4160669457308, 677.5417529657701, 310.6799530521754,
478.97523859526325, 715.1647261853918]
```

In [120]:
```
print("The GMM order that gets selected based on the likelihood values is : {}
".format(np.argmax(likelihood)+1))
print("The GMM order that gets selected based on the BIC values is : {}".forma
t(np.argmin(bic)+1))
```

```
The GMM order that gets selected based on the likelihood values is : 6
The GMM order that gets selected based on the BIC values is : 4
```

In [ ]:

Question 1:

c) Number of datasets: 1000

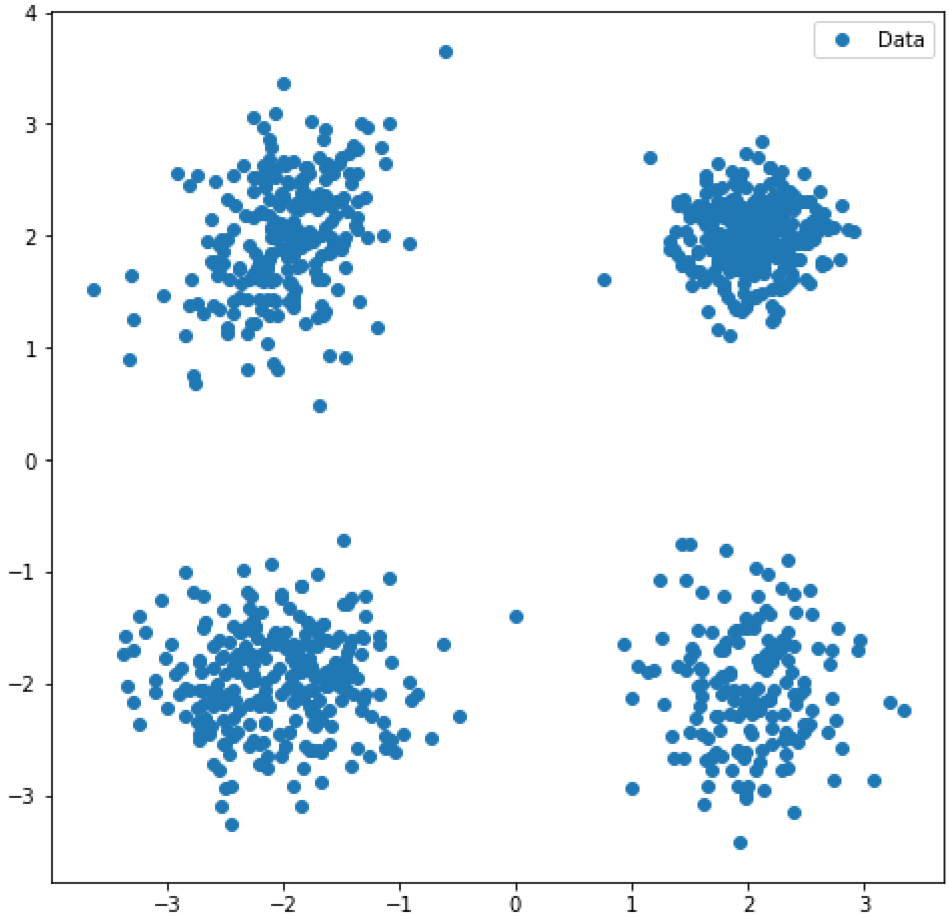
In [45]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
from sklearn.mixture import GaussianMixture
```

In [98]:
```
t_mu = [[2, 2], [-2, 2], [-2, -2], [2, -2]]
t_var_1 = [[0.1, 0], [0, 0.1]]
t_var_2 = [[0.2, 0.1], [0.1, 0.3]]
t_var_3 = [[0.3, 0], [0, 0.2]]
t_var_4 = [[0.2, 0], [0, 0.3]]
prior = [0.30, 0.25, 0.28, 0.17]

M = 6
N = 1000
l_1 = 0
l_2 = 0
l_3 = 0
l_4 = 0

likelihood = []
bic = []
```

In [99]:
```
#generating number of sample for the GMMs
for i in range(N):
    temp = np.random.uniform(0, 1, 1)
    if temp <= prior[0]:
        l_1 = l_1 + 1
    elif temp <= prior[0] + prior[1]:
        l_2 = l_2 + 1
    elif temp <= prior[0] + prior[1] + prior[2]:
        l_3 = l_3 + 1
l_4 = N - l_1 - l_2 - l_3
```

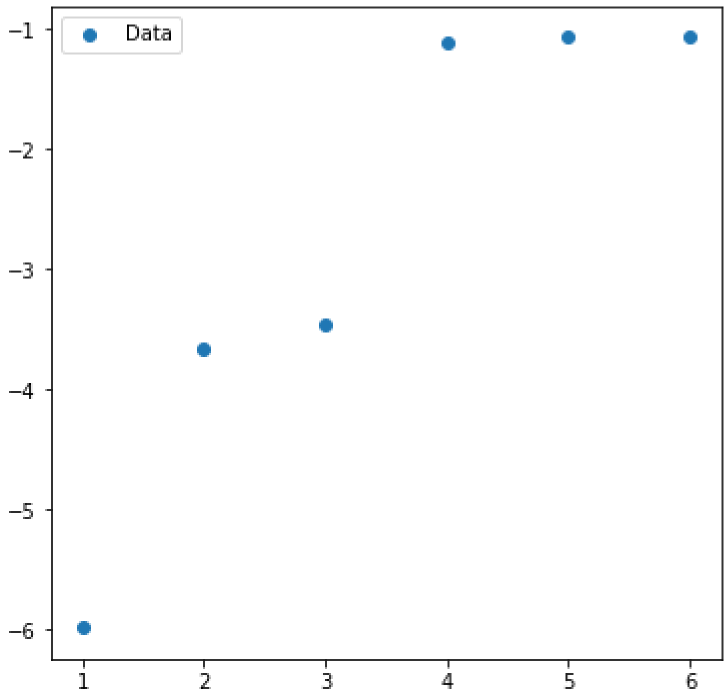
In [100]:
```
#generating data according to component
data = []

for i in range(l_1):
    data.append(np.random.multivariate_normal(t_mu[0], t_var_1, 1))
for i in range(l_2):
    data.append(np.random.multivariate_normal(t_mu[1], t_var_2, 1))
for i in range(l_3):
    data.append(np.random.multivariate_normal(t_mu[2], t_var_3, 1))
for i in range(l_4):
    data.append(np.random.multivariate_normal(t_mu[3], t_var_4, 1))
```

In [101]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(data).reshape(N,2)[:, 0], np.array(data).reshape(N,2)[:, 1
], alpha=1, label='Data')
ax.legend()
plt.show()
```

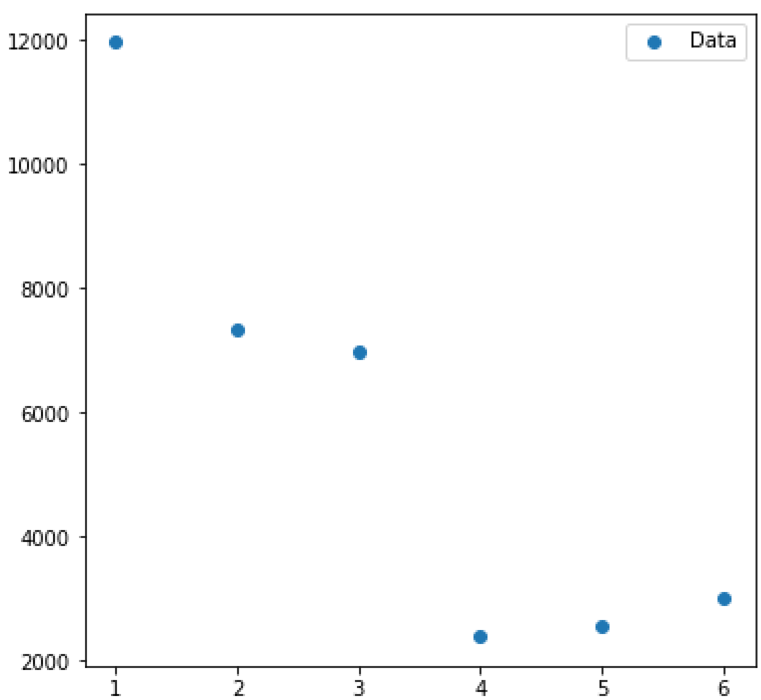


In [102]:
```
for h in range(M):
    sum_3 = 0
    for c in range(10):
        sum_2 = 0
        new_data = []
        for d in range(0, int(0.1*N*c), 1):
            new_data.append(np.array(data).reshape(N,2)[int(d), :])
        for d in range(int(0.1*N*(c+1)), N, 1):
            new_data.append(np.array(data).reshape(N,2)[int(d), :])
        gmm = GaussianMixture(n_components = h+1)
        label = gmm.fit_predict(np.array(new_data).reshape(int(0.9*N), 2))
        alpha = gmm.weights_
        mean = gmm.means_
        covariance = gmm.covariances_
        for i in range(int(0.1*N*c), int(0.1*N*(c+1)), 1):
            sum_1 = 0
            for j in range(h+1):
                p = math.exp(-0.5*np.matmul(np.matmul((np.array(data).reshape(
N,2)[i,:]) - mean[j,:],
                    np.linalg.inv(covariance[j])), (np.array(data).reshape(N,2
)[i,:]) - mean.reshape(h+1,2)[j,:]))/(2*math.pi*np.linalg.det(covariance[j]))
                sum_1 = sum_1 + alpha[j]*p
            sum_2 = sum_2 + np.log(sum_1)
        sum_3 = sum_3 +sum_2
    if np.isinf(sum_3) == True:
        likelihood.append(-10**5)
        bic.append(10**5 + h**3*np.log(N))
    else:
        likelihood.append(sum_3/N)
        bic.append(-2*sum_3 + h**3*np.log(N))
```

In [103]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.scatter([1,2,3,4,5,6], np.array(likelihood), alpha=1, label='Data')
ax.legend()
plt.show()
```



In [104]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.scatter([1,2,3,4,5,6], np.array(bic), alpha=1, label='Data')
ax.legend()
plt.show()
```



In [105]:
```
print(likelihood)
```

```
[-5.98407221463578, -3.660885017828124, -3.456659136157701, -1.10505786342573
78, -1.061587467910792, -1.065019395486585]
```

In [106]:
```
print(bic)
```

```
[11968.144429271559, 7328.6777909352295, 6968.580314547258, 2396.62511938399
3, 2565.2712736764406, 2993.5082008459367]
```

In [107]:
```
print("The GMM order that gets selected based on the likelihood values is : {}
".format(np.argmax(likelihood)+1))
print("The GMM order that gets selected based on the BIC values is : {}".forma
t(np.argmin(bic)+1))
```

```
The GMM order that gets selected based on the likelihood values is : 5
The GMM order that gets selected based on the BIC values is : 4
```

In [ ]:

Question 1:

c) Number of datasets: 10000

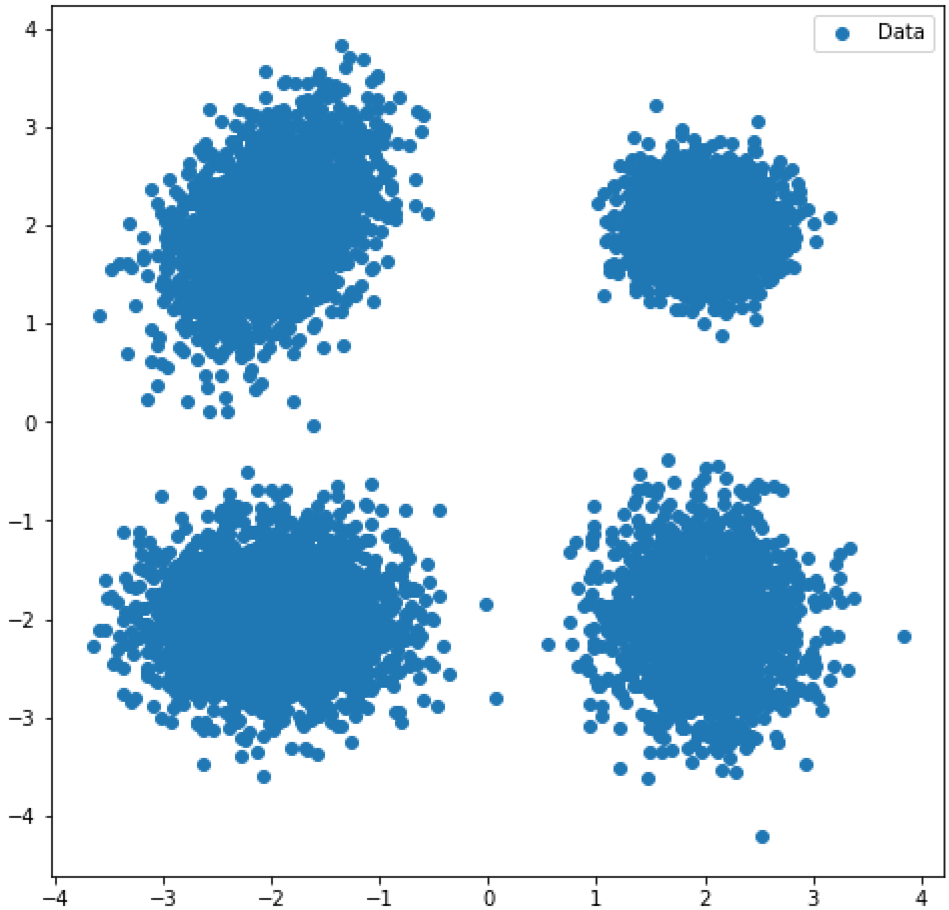
In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
from sklearn.mixture import GaussianMixture
```

In [14]:
```
t_mu = [[2, 2], [-2, 2], [-2, -2], [2, -2]]
t_var_1 = [[0.1, 0], [0, 0.1]]
t_var_2 = [[0.2, 0.1], [0.1, 0.3]]
t_var_3 = [[0.3, 0], [0, 0.2]]
t_var_4 = [[0.2, 0], [0, 0.3]]
prior = [0.30, 0.25, 0.28, 0.17]

M = 6
N = 10000
l_1 = 0
l_2 = 0
l_3 = 0
l_4 = 0

likelihood = []
bic = []
```

In [15]:
```
#generating number of sample for the GMMs
for i in range(N):
    temp = np.random.uniform(0, 1, 1)
    if temp <= prior[0]:
        l_1 = l_1 + 1
    elif temp <= prior[0] + prior[1]:
        l_2 = l_2 + 1
    elif temp <= prior[0] + prior[1] + prior[2]:
        l_3 = l_3 + 1
l_4 = N - l_1 - l_2 - l_3
```

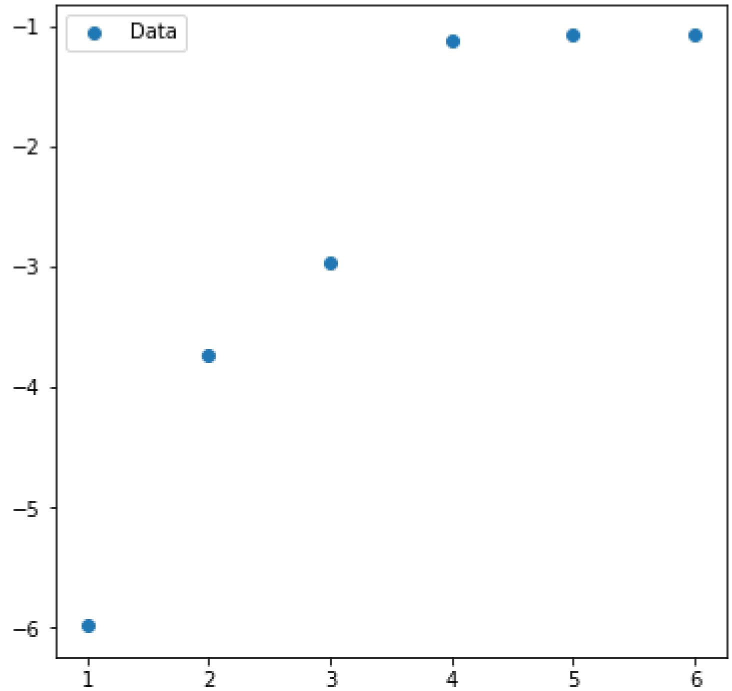
In [16]:
```
#generating data according to component
data = []

for i in range(l_1):
    data.append(np.random.multivariate_normal(t_mu[0], t_var_1, 1))
for i in range(l_2):
    data.append(np.random.multivariate_normal(t_mu[1], t_var_2, 1))
for i in range(l_3):
    data.append(np.random.multivariate_normal(t_mu[2], t_var_3, 1))
for i in range(l_4):
    data.append(np.random.multivariate_normal(t_mu[3], t_var_4, 1))
```

In [17]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(data).reshape(N,2)[:, 0], np.array(data).reshape(N,2)[:, 1
], alpha=1, label='Data')
ax.legend()
plt.show()
```

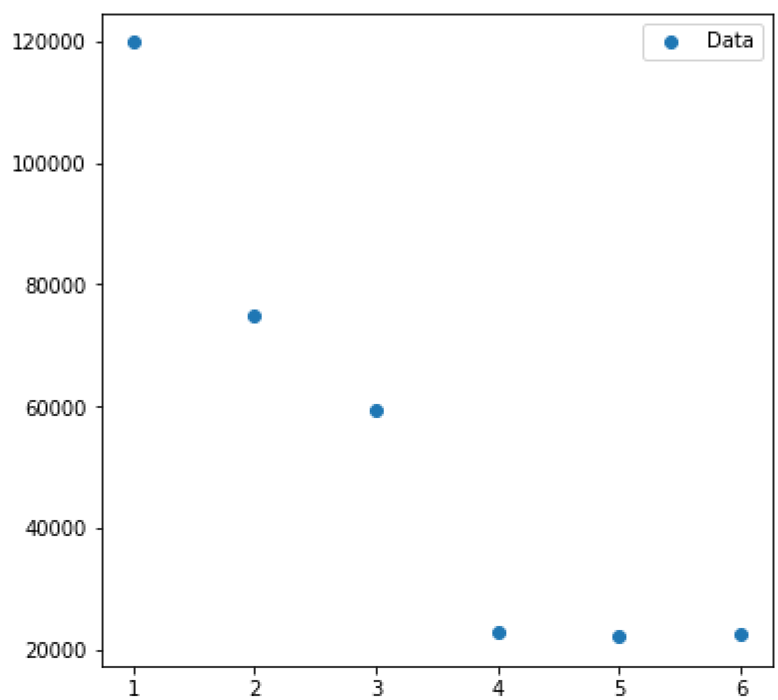


In [18]:
```
for h in range(M):
    sum_3 = 0
    for c in range(10):
        sum_2 = 0
        new_data = []
        for d in range(0, int(0.1*N*c), 1):
            new_data.append(np.array(data).reshape(N,2)[int(d), :])
        for d in range(int(0.1*N*(c+1)), N, 1):
            new_data.append(np.array(data).reshape(N,2)[int(d), :])
        gmm = GaussianMixture(n_components = h+1)
        label = gmm.fit_predict(np.array(new_data).reshape(int(0.9*N), 2))
        alpha = gmm.weights_
        mean = gmm.means_
        covariance = gmm.covariances_
        for i in range(int(0.1*N*c), int(0.1*N*(c+1)), 1):
            sum_1 = 0
            for j in range(h+1):
                p = math.exp(-0.5*np.matmul(np.matmul((np.array(data).reshape(
N,2)[i,:]) - mean[j,:],
                    np.linalg.inv(covariance[j])), (np.array(data).reshape(N,2
)[i,:]) - mean.reshape(h+1,2)[j,:]))/(2*math.pi*np.linalg.det(covariance[j]))
                sum_1 = sum_1 + alpha[j]*p
            sum_2 = sum_2 + np.log(sum_1)
        sum_3 = sum_3 +sum_2
    if np.isinf(sum_3) == True:
        likelihood.append(-10**5)
        bic.append(10**5 + h**3*np.log(N))
    else:
        likelihood.append(sum_3/N)
        bic.append(-2*sum_3 + h**3*np.log(N))
```

In [19]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.scatter([1,2,3,4,5,6], np.array(likelihood), alpha=1, label='Data')
ax.legend()
plt.show()
```



In [20]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.scatter([1,2,3,4,5,6], np.array(bic), alpha=1, label='Data')
ax.legend()
plt.show()
```



In [21]:
```
print(likelihood)
```

```
[-5.987828619339828, -3.7359827415346336, -2.9685575382937595, -1.12379815922
4201, -1.0801666638839804, -1.073491947764249]
```

In [22]:
```
print(bic)
```

```
[119756.57238679656, 74728.86517106465, 59444.833488851, 22724.642374527375,
22192.795061486086, 22621.131501782]
```

In [23]:
```
print("The GMM order that gets selected based on the likelihood values is : {}
".format(np.argmax(likelihood)+1))
print("The GMM order that gets selected based on the BIC values is : {}".forma
t(np.argmin(bic)+1))
```

```
The GMM order that gets selected based on the likelihood values is : 6
The GMM order that gets selected based on the BIC values is : 5
```

Results:

From the results obtained from the above experiment, we can summarise the following:

1) The likelihood of data is maximum around M = 4, 5, and 6. Hence, we could state that either of these GMM orders can be selected for the given dataset since they are very close to each other, even though the true number of components is 4. The selected component order by the program might vary slightly based on the number of samples and the parameters of the datasets generated.

2) On penalising over-fitting (using Bayesian Information Criterion), we can see that the model order selection tends towards M = 4 as the most preferred GMM order for the dataset. This is due to penalising over-fitting by giving higher values for higher orders, even though their likelihood might be comparable to some other order numbers.

In [ ]:

Question 2

Given: $y(x) = \dfrac{1}{1+e^{(w^T x+b)}}$

$P(L=+1|x) = y(x)$

$P(L=-1|x) = 1-y(x)$

Taking $X = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$, $\theta = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}$, $y(x) = \dfrac{1}{1+e^{\theta^T . X}}$

Now, using logistic regression, the log likelihood is given by:

$$\ln(L(\theta)) = \sum_{i=1}^{N} \sum_{j=1}^{2} \ln p(x_i | L=j)^{L_j}$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{2} \ln p(L=j|x_i)^{L_j} . p(x_i)^{L_j} \quad \text{(Bayes Rule)}$$

$$= \sum_{L=1}^{N} \sum_{j=1}^{2} \ln p(L=j|x_i)^{L_j} \quad (p(x_i) \text{ remains constant})$$

$$= \sum_{i=1}^{N} \ln p(L=1|x_i)^{L_1} + \sum_{i=1}^{N} \ln p(L=2|x_i)^{L_2}$$

$$= \sum_{i=1}^{N} L_1 \ln p(L=1|x_i) + \sum_{i=1}^{N} L_2 \ln p(L=2, x_i)$$

$$= \sum_{i=1}^{N} L_1 \ln y(x_i) + \sum_{i=1}^{N} L_2 . \ln(1-y(x_i)) = L_1 \ln(y(x)) + L_2 \ln(1-y(x))$$

$\therefore$ Gradient of the function,

$$\nabla \ln(L(\theta)) = \frac{d}{d\theta}(\ln(L(\theta)))$$

$$= \frac{d}{dy}(\ln(L(\theta))) \times \frac{dy}{d\theta} \cancel{(\ln(L(\theta)))} y$$

$$= \left(\frac{L_1}{y(x)} - \frac{L_2}{1-y(x)}\right) \times \frac{dy}{d\theta}\left(\frac{1}{1+e^{\theta^T . X}}\right)$$

$$\therefore \nabla \ln(L(\theta)) = \left(\frac{L_1}{y(x)} - \frac{1-L_1}{1-y(x)}\right) \times \left(y(x)(1-y(x)\right) X$$

$$= \left(\frac{L_1(1-y(x)) - y(x)(1-L_1)}{y(x)(1-y(x))}\right)\left(y(x)(1-y(x))\right) X$$

$$= \left[L_1 - y(x)\right] X$$

$$= X^T\left[L_1 - y(x)\right]$$

Gradient descent:

$$\theta_{new} = \theta_{old} - \alpha \cdot \nabla \ln(L(\theta))$$

Question 2:

In [1102]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
from sklearn.mixture import GaussianMixture
```

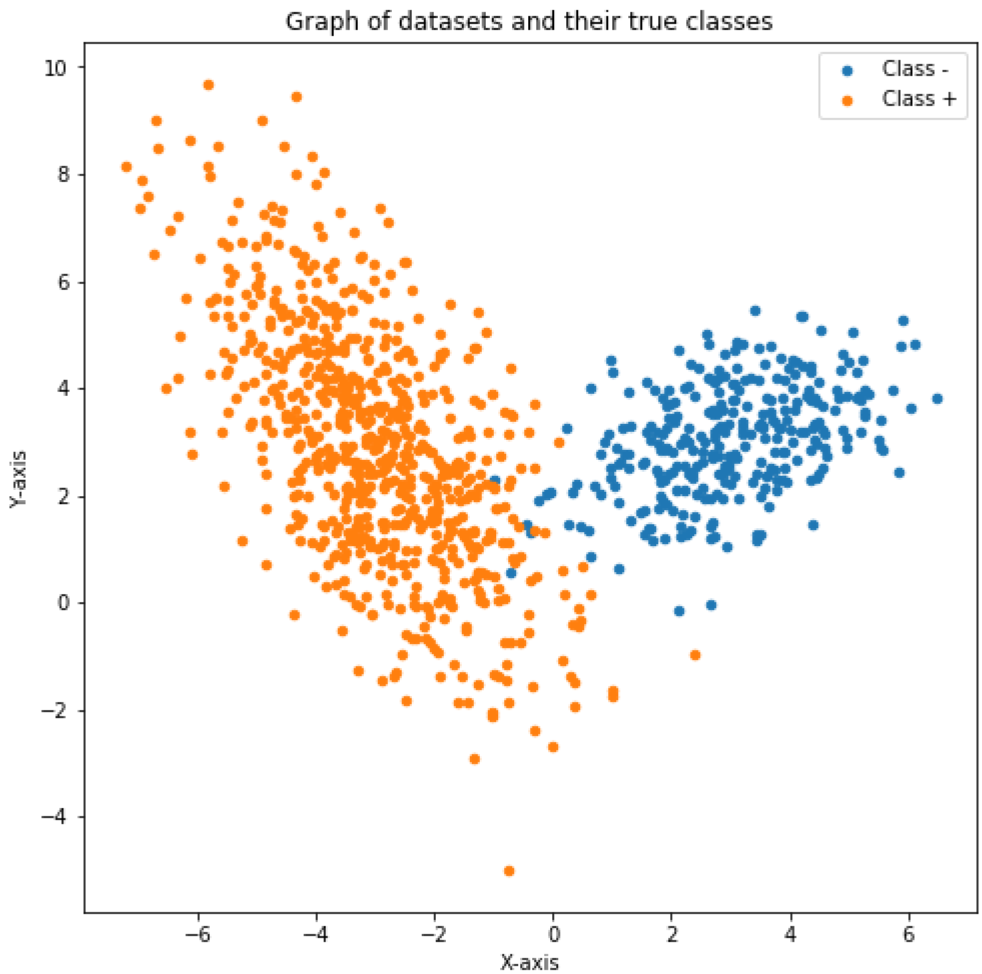
Fisher LDA Classifier

In [1103]:
```
s_b = []
s_w = []

mu_1 = [3, 3]
mu_2 = [-3, 3]
variance_1 = [[2, 0.5], [0.5, 1]]
variance_2 = [[2, -1.9], [-1.9, 5]]

prior = [0.30, 0.70]
N = 999
l_1 = 0
l_2 = 0
```

In [1104]:
```
#generating number of sample for the GMMs
for i in range(N):
    if np.random.uniform(0, 1, 1) <= prior[0]:
        l_1 = l_1 + 1

l_2 = N - l_1
```

In [1105]:
```
#generating data according to component
data_1 = []
data_2 = []
data = []
l_i = []

for i in range(l_1):
    z = np.random.multivariate_normal(mu_1, variance_1, 1)
    data_1.append(z)
    data.append(z)
    l_i.append(0)
for i in range(l_2):
    z = np.random.multivariate_normal(mu_2, variance_2, 1)
    data_2.append(z)
    data.append(z)
    l_i.append(1)
```

In [1106]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(data_1).reshape(l_1,2)[:, 0], np.array(data_1).reshape(l_1
,2)[:, 1], s=20, alpha=1, label='Class -')
ax.scatter(np.array(data_2).reshape(l_2,2)[:, 0], np.array(data_2).reshape(l_2
,2)[:, 1], s=20, alpha=1, label='Class +')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph of datasets and their true classes')
ax.legend()
plt.show()
```



In [1107]:
```
new_mu_1 = [np.array(data_1).reshape(l_1, 2)[:, 0].mean(), np.array(data_1).re
shape(l_1, 2)[:, 1].mean()]
new_mu_2 = [np.array(data_2).reshape(l_2, 2)[:, 0].mean(), np.array(data_2).re
shape(l_2, 2)[:, 1].mean()]
data_1_t = np.reshape(data_1, (2,l_1))
data_2_t = np.reshape(data_2, (2,l_2))
new_var_1 = np.cov(data_1_t)
new_var_2 = np.cov(data_2_t)
```

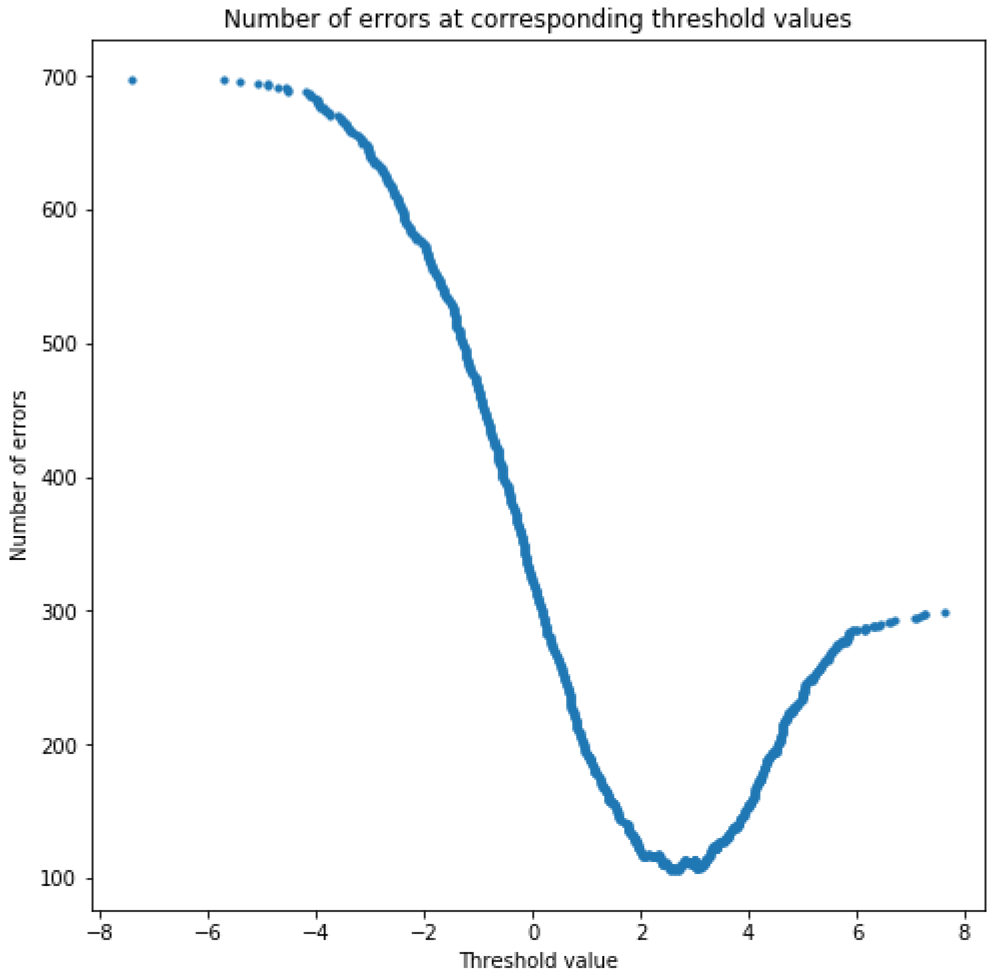
In [1108]:
```
s_b = np.matmul(np.subtract(new_mu_1, new_mu_2).reshape(2, 1), (np.subtract(ne
w_mu_1, new_mu_2)).reshape(1, 2))
s_w = np.add(new_var_1, new_var_2)
```

In [1109]:
```
V, D = np.linalg.eig(np.matmul((np.linalg.inv(s_w)), s_b))
```

In [1110]:
```
ind = np.argmax(V)

vec = D[:, ind]
new_ax_1 = np.matmul(vec, np.reshape(data_1, (2, l_1)))
new_ax_2 = np.matmul(vec, np.reshape(data_2, (2, l_2)))
```

In [1111]:
```
tr = 0
err = []
for i in range(l_1):
    count = 0
    tr = new_ax_1[i]
    for j in range(l_1):
        if new_ax_1[j] < tr:
            count = count + 1
    for j in range(l_2):
        if new_ax_2[j] > tr:
            count = count +1
    err.append([tr, count])

for i in range(l_2):
    count = 0
    tr = new_ax_2[i]
    for j in range(l_1):
        if new_ax_1[j] < tr:
            count = count + 1
    for j in range(l_2):
        if new_ax_2[j] > tr:
            count = count +1
    err.append([tr, count])
```

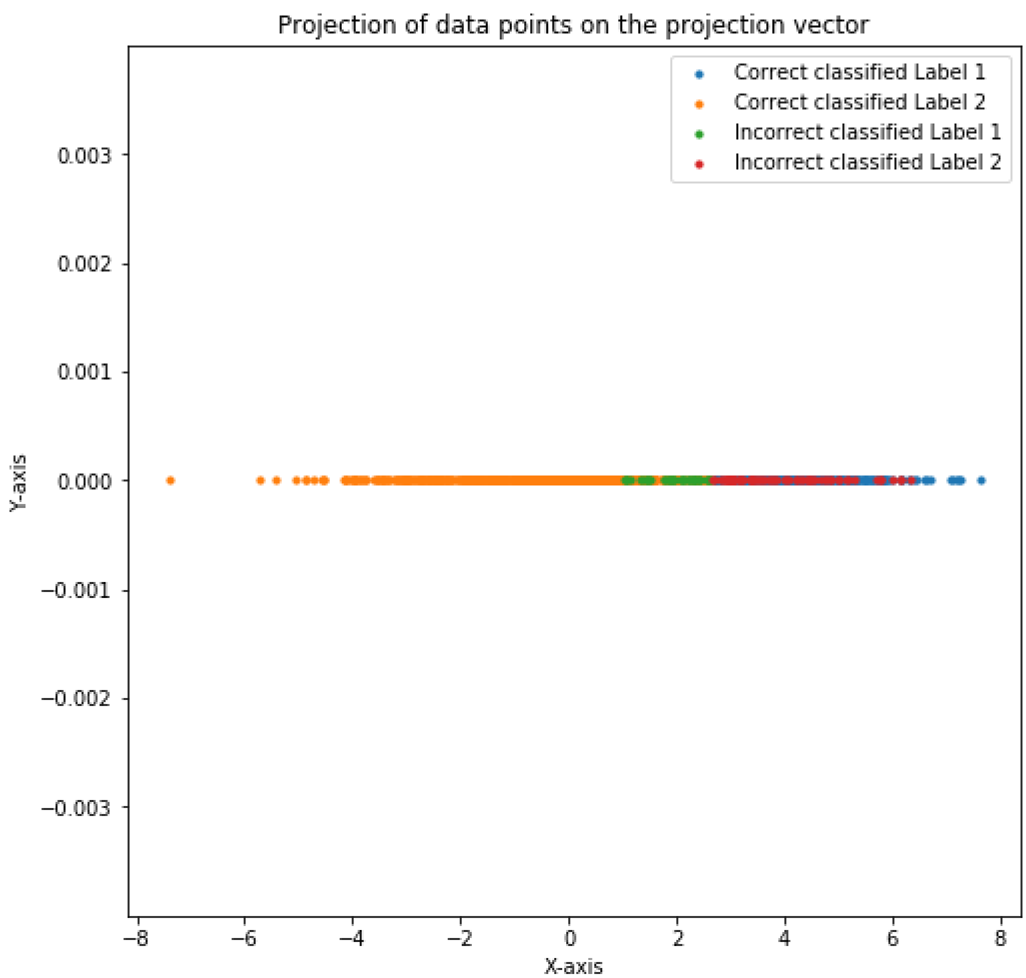
In [1112]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(err)[:, 0], np.array(err)[:, 1], s=10)
plt.xlabel('Threshold value')
plt.ylabel('Number of errors')
plt.title('Number of errors at corresponding threshold values')
plt.show()
```



In [1113]:
```
e = np.argmin(np.array(err)[:, 1])
thresh = err[e][0]
```

In [1114]:
```
right_1 = []
error_1 = []
right_2 = []
error_2 = []

for i in range(l_1):
    if new_ax_1[i] > thresh:
        right_1.append(np.array(new_ax_1)[i])
    else:
        error_1.append(np.array(new_ax_1)[i])

for i in range(l_2):
    if new_ax_2[i] < thresh:
        right_2.append(np.array(new_ax_2)[i])
    else:
        error_2.append(np.array(new_ax_2)[i])
```

In [1115]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(right_1), np.zeros(len(right_1)), s=10, alpha=1, label='Co
rrect classified Label 1')
ax.scatter(np.array(right_2), np.zeros(len(right_2)), s=10, alpha=1, label='Co
rrect classified Label 2')
ax.scatter(np.array(error_1), np.zeros(len(error_1)), s=10, alpha=1, label='In
correct classified Label 1')
ax.scatter(np.array(error_2), np.zeros(len(error_2)), s=10, alpha=1, label='In
correct classified Label 2')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Projection of data points on the projection vector')
ax.legend()
plt.show()
```



In [1116]:
```
print("The total number of errors are: {}".format(len(error_1) + len(error_2
)))
print("The error probability is {}.".format((len(error_1) + len(error_2))/N))
```

```
The total number of errors are: 107
The error probability is 0.10710710710710711.
```

Maximum Likelihood Estimation

In [1126]:
```
old_parameter = []
data_new = []
```

In [1127]:
```
l_rate = 0.000005

parameter = [vec[0], vec[1], thresh]

l_i = np.reshape(l_i, (N,1))

data = np.reshape(data, (N,2))
data_new = np.hstack((data, np.ones((N,1))))
```

In [1140]:
```
for i in range(150000):
    ex = np.exp(-1*np.matmul(data_new.reshape(N,3), np.array(parameter).reshap
e(3,1)))
    y_func = np.power(1 + ex, -1)
    deriv = np.matmul(np.reshape(data_new,(3,N)), l_i - y_func.reshape(N,1))
    new_parameter = np.array(parameter).reshape(3,1) - l_rate*deriv
    parameter = new_parameter
```

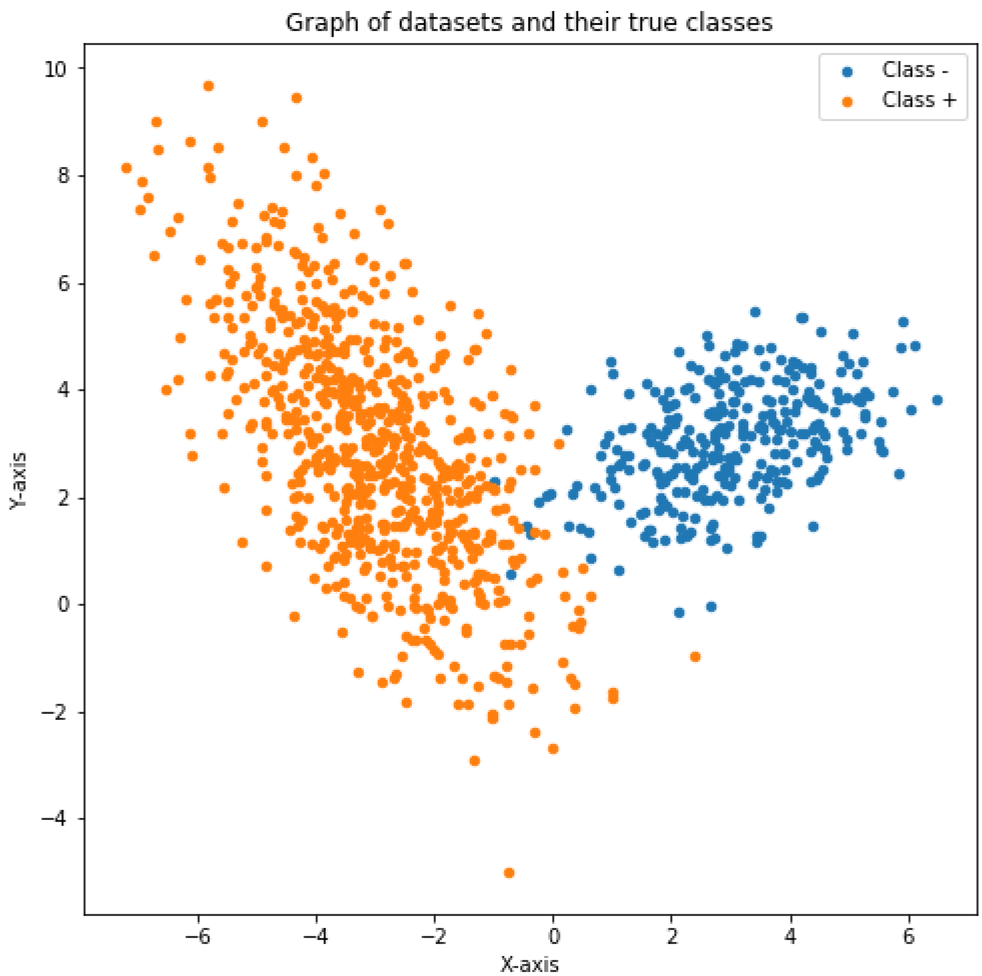
In [1141]:
```
new_l_i = np.round(np.array(y_func).reshape(N,1))
```

In [1142]:
```
mle_error_1 = []
mle_right_1 = []
mle_error_2 = []
mle_right_2 = []

for i in range(l_1):
    if new_l_i[i] == 1:
        mle_error_1.append(data[i])
    else:
        mle_right_1.append(data[i])

for i in range(l_2):
    if new_l_i[l_1+i] == 0:
        mle_error_2.append(data[l_1+i])
    else:
        mle_right_2.append(data[l_1+i])
```

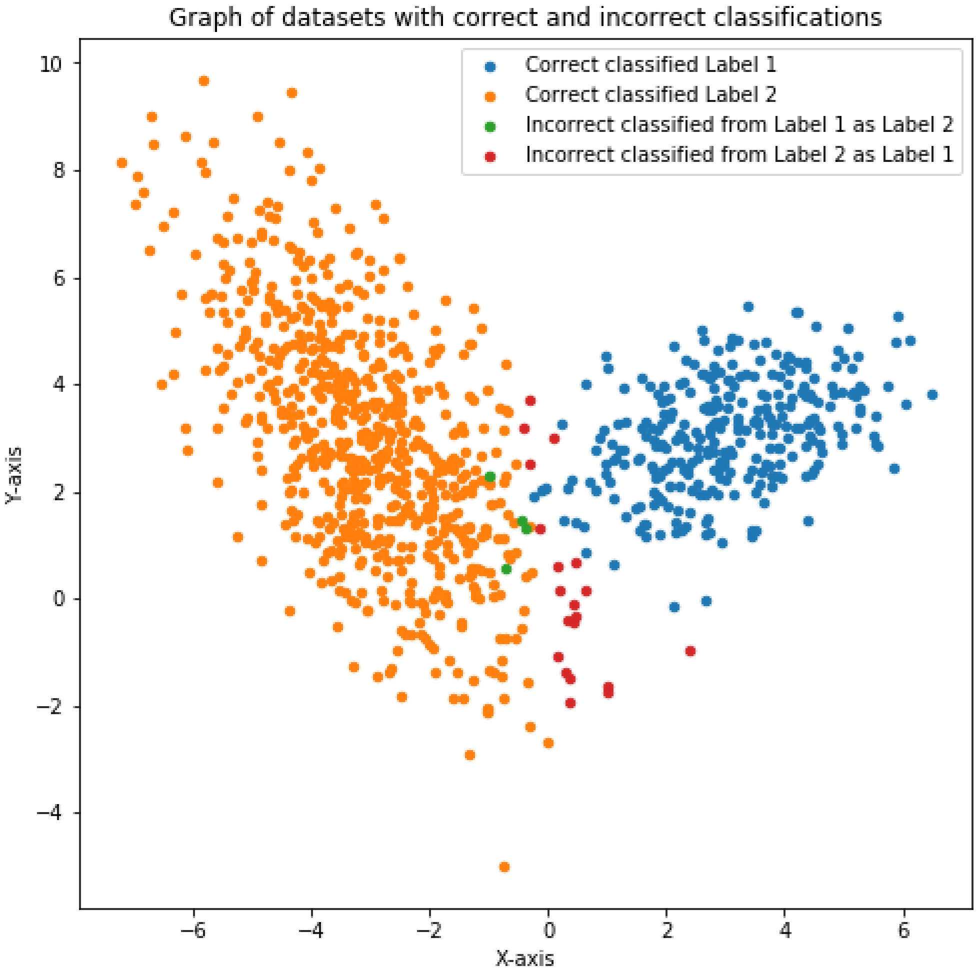
In [1143]:
```
len(mle_error_1), len(mle_right_1), len(mle_error_2), len(mle_right_2)
```

Out[1143]:
```
(4, 296, 20, 679)
```

In [1144]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(data_1).reshape(l_1,2)[:, 0], np.array(data_1).reshape(l_1
,2)[:, 1], s=20, alpha=1, label='Class -')
ax.scatter(np.array(data_2).reshape(l_2,2)[:, 0], np.array(data_2).reshape(l_2
,2)[:, 1], s=20, alpha=1, label='Class +')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph of datasets and their true classes')
ax.legend()
plt.show()
```



In [1145]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(mle_right_1)[:, 0], np.array(mle_right_1)[:, 1], s=20, alp
ha=1, label='Correct classified Label 1')
ax.scatter(np.array(mle_right_2)[:, 0], np.array(mle_right_2)[:, 1], s=20, alp
ha=1, label='Correct classified Label 2')
ax.scatter(np.array(mle_error_1)[:, 0], np.array(mle_error_1)[:, 1], s=20, alp
ha=1, label='Incorrect classified from Label 1 as Label 2')
ax.scatter(np.array(mle_error_2)[:, 0], np.array(mle_error_2)[:, 1], s=20, alp
ha=1, label='Incorrect classified from Label 2 as Label 1')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph of datasets with correct and incorrect classifications')
ax.legend()
plt.show()
```



In [1146]:
```
print("The total number of errors are: {}".format(len(mle_error_1) + len(mle_e
rror_2)))
print("The error probability is {}.".format((len(mle_error_1) + len(mle_error_
2))/N))
```

```
The total number of errors are: 24
The error probability is 0.024024024024024024.
```

MAP Classifier

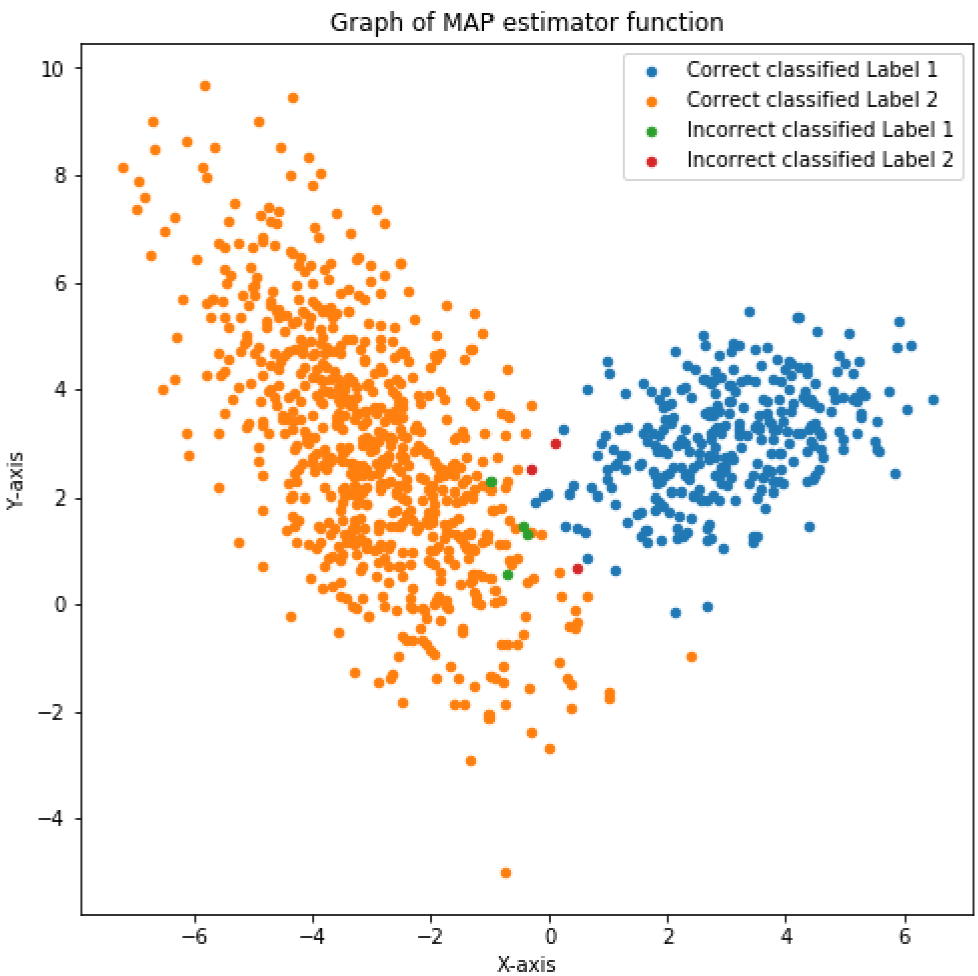
In [1147]:

```
def normal_prob(x, m, v):
    x_t = x
    m_t = m
    np.reshape(x, (2,1))
    np.reshape(m, (2,1))
    p = math.exp(-0.5*np.matmul(np.matmul((x_t-m_t), np.linalg.inv(v)), (x-m
)))/(2*math.pi*np.linalg.det(v))
    return p
```

In [1148]:

```
x_right = []
x_error = []
y_right = []
y_error = []

for i in range(l_1):
    p_1 = normal_prob(np.array(data)[i, :], mu_1, variance_1)
    p_2 = normal_prob(np.array(data)[i, :], mu_2, variance_2)
    if (p_1*prior[0] > p_2*prior[1]):
        x_right.append(np.array(data)[i, :])
    else:
        x_error.append(np.array(data)[i, :])

for i in range(l_2):
    p_1 = normal_prob(np.array(data)[l_1+i, :], mu_1, variance_1)
    p_2 = normal_prob(np.array(data)[l_1+i, :], mu_2, variance_2)
    if (p_1*prior[0] < p_2*prior[1]):
        y_right.append(np.array(data)[l_1+i, :])
    else:
        y_error.append(np.array(data)[l_1+i, :])
```

In [1149]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(x_right)[:, 0], np.array(x_right)[:, 1], s=20, alpha=1, la
bel='Correct classified Label 1')
ax.scatter(np.array(y_right)[:, 0], np.array(y_right)[:, 1], s=20, alpha=1, la
bel='Correct classified Label 2')
ax.scatter(np.array(x_error)[:, 0], np.array(x_error)[:, 1], s=20, alpha=1, la
bel='Incorrect classified Label 1')
ax.scatter(np.array(y_error)[:, 0], np.array(y_error)[:, 1], s=20, alpha=1, la
bel='Incorrect classified Label 2')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph of MAP estimator function')
ax.legend()
plt.show()
```



In [1150]:
```
print("The total number of errors are: {}".format(len(x_error) + len(y_error
)))
print("The error probability is {}.".format((len(x_error) + len(y_error))/N))
```

```
The total number of errors are: 7
The error probability is 0.007007007007007007.
```

Results:

1) The visual graphs of the results of all the three classifiers have been represnted above. Below the graphs, the respective error counts and the probabilities of error have also been mentioned.

2) For the above dataset, it can be observed here that the least number of errors are generated by the MAP classifier (7), followed by the MLE classifier (24), and finally the Fisher LDA classifier (107). This shows that the MAP classifier gives the best classification for the above dataset closely followed by the MLE classifier.

In [ ]:

NOTE: All the codes in **python** and in jupyter notebook (in their original form) are available on GitHub through the following link along with a zip folder on blackboard as well.

https://github.com/nandayvk/EECE5644_HW_3.git