Problem 1

c) Gaussian Mixture Models Implementation

Importing libraries

In [49]:
```python
import matplotlib.pyplot as plt
import numpy as np

from matplotlib.patches import Ellipse
from PIL import Image
from sklearn import datasets
from sklearn.cluster import KMeans
```

Implementing the Gaussian density function.

In [50]:
```python
def gaussian(X, mu, cov):
    n = X.shape[1]
    diff = (X - mu).T
    return np.diagonal(1 / ((2 * np.pi) ** (n / 2) * np.linalg.det(cov) ** 0.5) * np.exp(-0.
5 * np.dot(np.dot(diff.T, np.linalg.inv(cov)), diff))).reshape(-1, 1)
```

Step 1

This is the initialization step of the GMM. At this point, we must initialise our parameters π_k, μ_k, and Σ_k. In this case, we are going to use the results of KMeans as an initial value for μ_k, set π_k to one over the number of clusters and Σ_k to the identity matrix. We could also use random numbers for everything, but using a sensible initialisation procedure will help the algorithm

In [51]:
```python
def initialize_clusters(X, n_clusters):
    clusters = []
    idx = np.arange(X.shape[0])

    # Using the KMeans centroids to initialise the GMM

    kmeans = KMeans(n_clusters).fit(X)
    mu_k = kmeans.cluster_centers_

    for i in range(n_clusters):
        clusters.append({
            'pi_k': 1.0 / n_clusters,
            'mu_k': mu_k[i],
            'cov_k': np.identity(X.shape[1], dtype=np.float64)
        })

    return clusters
```

Step 2 (Expectation step)

We should now calculate $\gamma(z_{nk})$

In [52]:
```python
def expectation_step(X, clusters):
    global gamma_nk, totals
    N = X.shape[0]
    K = len(clusters)
    totals = np.zeros((N, 1), dtype=np.float64)
    gamma_nk = np.zeros((N, K), dtype=np.float64)

    for k, cluster in enumerate(clusters):
        pi_k = cluster['pi_k']
        mu_k = cluster['mu_k']
        cov_k = cluster['cov_k']

        gamma_nk[:, k] = (pi_k * gaussian(X, mu_k, cov_k)).ravel()

    totals = np.sum(gamma_nk, 1)
    gamma_nk /= np.expand_dims(totals, 1)
```

Step 3 (Maximization step):

Let us now implement the maximization step.

In [53]:
```python
def maximization_step(X, clusters):
    global gamma_nk
    N = float(X.shape[0])

    for k, cluster in enumerate(clusters):
        gamma_k = np.expand_dims(gamma_nk[:, k], 1)
        N_k = np.sum(gamma_k, axis=0)

        pi_k = N_k / N
        mu_k = np.sum(gamma_k * X, axis=0) / N_k
        cov_k = (gamma_k * (X - mu_k)).T @ (X - mu_k) / N_k

        cluster['pi_k'] = pi_k
        cluster['mu_k'] = mu_k
        cluster['cov_k'] = cov_k
```

Log-likelihood of the model

In [54]:
```python
def get_likelihood(X, clusters):
    global gamma_nk, totals
    sample_likelihoods = np.log(totals)
    return np.sum(sample_likelihoods), sample_likelihoods
```

Model Training

Finally, let's put everything together! First, we are going to initialise the parameters, and then perform several expectation-maximization steps. In this case, we set the number of iterations of the training procedure to a fixed number. I have done this

In [55]:
```python
def train_gmm(X, n_clusters, n_epochs):
    clusters = initialize_clusters(X, n_clusters)
    likelihoods = np.zeros((n_epochs, ))
    scores = np.zeros((X.shape[0], n_clusters))
    history = []

    for i in range(n_epochs):
        clusters_snapshot = []

        # This is just for our later use in the graphs
        for cluster in clusters:
            clusters_snapshot.append({
                'mu_k': cluster['mu_k'].copy(),
                'cov_k': cluster['cov_k'].copy()
            })

        history.append(clusters_snapshot)

        expectation_step(X, clusters)
        maximization_step(X, clusters)

        likelihood, sample_likelihoods = get_likelihood(X, clusters)
        likelihoods[i] = likelihood

        print('Epoch: ', i + 1, 'Likelihood: ', likelihood)

    scores = np.log(gamma_nk)

    return clusters, likelihoods, scores, sample_likelihoods, history
```

d) GMM on Iris data

Train the model on Iris dataset.

In [56]:
```python
iris = datasets.load_iris()
X = iris.data
len(X)
```

Out[56]:
```
150
```

In [57]:
```python
n_clusters = 4
n_epochs = 50

clusters, likelihoods, scores, sample_likelihoods, history = train_gmm(X, n_clusters, n_epoc
hs)
```

```
Epoch:  1 Likelihood:  -733.0798788638429
Epoch:  2 Likelihood:  -221.96452566236437
Epoch:  3 Likelihood:  -195.538465692522
Epoch:  4 Likelihood:  -187.3543498804503
Epoch:  5 Likelihood:  -181.6091047273412
Epoch:  6 Likelihood:  -178.8996120059004
Epoch:  7 Likelihood:  -177.63890366232948
Epoch:  8 Likelihood:  -176.51766441640723
Epoch:  9 Likelihood:  -175.05535711847594
Epoch:  10 Likelihood:  -172.90979474388402
Epoch:  11 Likelihood:  -170.33026646156202
Epoch:  12 Likelihood:  -168.3672870039109
Epoch:  13 Likelihood:  -167.23164067997703
Epoch:  14 Likelihood:  -165.99797997744338
Epoch:  15 Likelihood:  -164.84178335905656
Epoch:  16 Likelihood:  -164.2783375733282
Epoch:  17 Likelihood:  -163.94619019319023
Epoch:  18 Likelihood:  -163.7897883141676
Epoch:  19 Likelihood:  -163.7318107381283
Epoch:  20 Likelihood:  -163.71229191048184
Epoch:  21 Likelihood:  -163.70575606403096
Epoch:  22 Likelihood:  -163.70349237669683
Epoch:  23 Likelihood:  -163.7026715618054
Epoch:  24 Likelihood:  -163.7023595290488
Epoch:  25 Likelihood:  -163.70223515270135
Epoch:  26 Likelihood:  -163.70218300719807
Epoch:  27 Likelihood:  -163.70215982684425
Epoch:  28 Likelihood:  -163.7021487545639
Epoch:  29 Likelihood:  -163.70214297775138
Epoch:  30 Likelihood:  -163.70213964211115
Epoch:  31 Likelihood:  -163.70213750644672
Epoch:  32 Likelihood:  -163.70213600869758
Epoch:  33 Likelihood:  -163.70213488214716
Epoch:  34 Likelihood:  -163.70213399286638
Epoch:  35 Likelihood:  -163.70213326880335
Epoch:  36 Likelihood:  -163.7021326679438
Epoch:  37 Likelihood:  -163.70213216358945
Epoch:  38 Likelihood:  -163.7021317373379
Epoch:  39 Likelihood:  -163.7021313756145
Epoch:  40 Likelihood:  -163.70213106788583
Epoch:  41 Likelihood:  -163.70213080568894
Epoch:  42 Likelihood:  -163.7021305820681
Epoch:  43 Likelihood:  -163.70213039122507
Epoch:  44 Likelihood:  -163.7021302282836
Epoch:  45 Likelihood:  -163.7021300891201
Epoch:  46 Likelihood:  -163.70212997023654
Epoch:  47 Likelihood:  -163.70212986865806
Epoch:  48 Likelihood:  -163.70212978185242
Epoch:  49 Likelihood:  -163.70212970766127
Epoch:  50 Likelihood:  -163.70212964424422
```

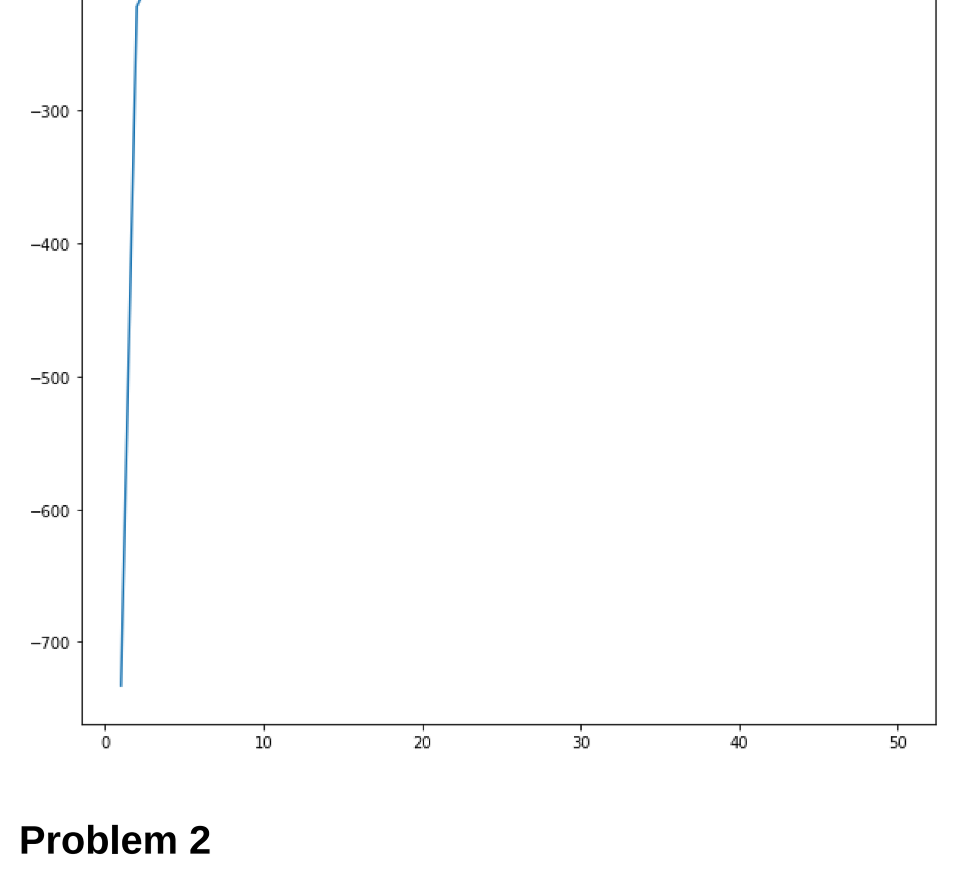
In [58]:
```python
# Last 5 values of parameters estimated by EM algo
history[-5]
```

Out[58]:
```
[{'mu_k': array([5.006, 3.428, 1.462, 0.246]),
  'cov_k': array([[0.121764, 0.097232, 0.016028, 0.010124],
         [0.097232, 0.140816, 0.011464, 0.009112],
         [0.016028, 0.011464, 0.029556, 0.005948],
         [0.010124, 0.009112, 0.005948, 0.010884]])},
 {'mu_k': array([6.67497375, 2.93848265, 5.71665471, 1.98040604]),
  'cov_k': array([[0.57818214, 0.14698734, 0.42432587, 0.06700327],
         [0.14698734, 0.1263782 , 0.09952827, 0.04372071],
         [0.42432587, 0.09952827, 0.34759906, 0.05567842],
         [0.06700327, 0.04372071, 0.05567842, 0.07258156]])},
 {'mu_k': array([5.92260269, 2.76649074, 4.11565459, 1.28815443]),
  'cov_k': array([[0.33698953, 0.1383726 , 0.24105188, 0.06901955],
         [0.1383726 , 0.10361799, 0.11410243, 0.0492038 ],
         [0.24105188, 0.11410243, 0.22448291, 0.06937506],
         [0.06901955, 0.0492038 , 0.06937506, 0.03167939]])},
 {'mu_k': array([6.21973526, 2.91055126, 4.93369615, 1.76759146]),
  'cov_k': array([[0.15579152, 0.03041092, 0.13698795, 0.11527049],
         [0.03041092, 0.08586382, 0.08019612, 0.07873263],
         [0.13698795, 0.08019612, 0.24297009, 0.18914566],
         [0.11527049, 0.07873263, 0.18914566, 0.17830542]])}]
```

Convergence Plot

Check the progess of log-likelihood.

In [59]:
```python
plt.figure(figsize=(10, 10))
plt.title('Log-Likelihood')
plt.plot(np.arange(1, n_epochs + 1), likelihoods)
plt.show()
```



Problem 2

Step-1 Expectatation

In [60]:
```python
def expectation_step(theta_t, x1):
    y2 = (x1 * theta_t)/(4 + theta_t)

    return y2
```

Step-2 Maximization

In [61]:
```python
def maximization_step(y2, x2, x3):
    theta_t = (y2 + x3)/(y2 + x3 + x2)
    return theta_t
```

Step-3 Model Training

In [62]:
```python
def training(x1, x2, x3):
    cur_theta = 0.5
    maxit = 100
    tol= 0.0001
    flag = 0

    for i in range(maxit):
        new_y2 = expectation_step(cur_theta, x1)
        new_theta = maximization_step(new_y2, x2, x3)

        if abs(cur_theta - new_theta)< tol:
            flag = 1
            break
        else:
            cur_theta = new_theta

    if not flag:
        print("Warning: Didn't converge")

    final_theta = cur_theta
    print("Final theta", round(final_theta,4))
```

In [63]:
```python
training(42, 10, 15)
```

```
Final theta 0.6783
```

The above case is not a case where the algorithm didn't converge. Final theta value obtained after 100 iterations is 0.6783

In [ ]: