HW3 Yifan Wu (yw515)

In [2]:
```
# -*- coding: utf-8 -*-
"""


In Class Exercise:  xx/100  (+ 10 extra credit if submitted before end of day
 of class)
*****************************************************


Goal: Implement EM, for Gaussian Mixture given data X.
mu, sigma, prior = EM(X, m, epsilon)

mu is lxm where l is dimensionality and m is number of mixture components
sigma is m (diagonal) covariance matrices (so really just m vectors), eg (lxm)
prior is prior for each component 1xm (one by m)
epsilon is the total change in updated parameter values and is used as a stopp
ing criterion

*****************************************************

The functions should compute Gaussian Mixture parameters using EM given data s
et X. Plot
the results of the learned parameters (using a contour like display given belo
w) after
each iteration (or so) of the learning process. This will help to visualize th
e
process of learning. Feel free  to use the code snippets provided below
or feel free to edit as you see fit. Assume covariance matrices are diagonal.

Construct synthetic 2-d data X or find a simple data set to use to test your c
ode.
*******************************************************




Take-Home Assignment:  xx/100
*****************************************************
To Do: Type-up responses and supporting figures and submit as a PDF.
*****************************************************

#1 Using EM algorithm to derive the update equation for the mean parameter for
each Gaussian Mixture Compoenent j.

#2 Assume data set as follows:

x[0] = [ 1, 1.5, 1.2, 1.2, .9, .8, 1, 2.3, 2.1, 2, 3, 2.5, 3]

x[1] = [ 1.5, 1.2, 1.2, .9, .7, .8, 2.3, 2.1, 2, 3, 2.5, 3, 2.1]

a) Run your code with 1 component, 2 components, and 3 components. What epsilo
n did you use?
Which component number "fits" the data best? How did you make this determinati
on?

b) To your existing code, add a function that plots (in a different figure)
the loglikelihood as a function  of iteration. Can you use this information as
```

```
a stopping criterion rather than epsilon? Can you use this information to
determine which number of components is best?

c) Given your analysis in Part b), what was the learned parameters of the mixt
ure
model that best fit the data

********************************************************

For supplemental help ...
See Resources:
    https://matplotlib.org/
    [TK], [Bp], [DHS]


@author: jerem
"""



import numpy as np
import matplotlib.pyplot as plt
from itertools import cycle
import os
from scipy.misc import imread

#from __future__ import absolute_import
#from __future__ import division
#from __future__ import print_function

from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.widgets import Slider

# Compute Gaussian Likelihoods
def gaussian_2d(x, y, x0, y0, xsig, ysig):
    return np.exp(-0.5*(((x-x0) / xsig)**2 + ((y-y0) / ysig)**2))


# Put your code here to generate or load some sample data
#
#
#



# Create appropriate grid for display purposes.
delta = 0.025
x = np.arange(-3.0, 3.0, delta)
y = np.arange(-2.0, 2.0, delta)
X, Y = np.meshgrid(x, y)



# Example mean and covariances to test the plot/display
mu0 = [0, 0]
mu1 = [1, 1]
sig0 = [1, 1]
sig1 = [1.5, 0.5]
```

```
# Code to create a figure and repeatedly plot the newly learned mixture.
fig1 = plt.ioff()
plt.figure(figsize=(20,10))
for i in range(0, 10):

    plt.clf()

    # Plot your sample Data Here
    #CS0 = plt.plot(data[0][0], data[1][0], "r*")



    # Put your code here to define the EM algorithm
# Goal: Implement EM, for Gaussian Mixture given data X.
# def  EM(X, m, epsilon):
#
#    return mu, sigma, prior
#

# This is here to simulate newly learned params to test display
    simulateUpdateM = np.random.rand(2, 2) -.5
    simulateUpdateC = np.random.rand(2, 2) -.5
    mu0[0]+=simulateUpdateM[0][0]
    mu0[1]+=simulateUpdateM[1][0]
    sig0[0] += simulateUpdateC[0][0]
    sig0[1] += simulateUpdateC[1][0]
    #mu1[0]+=simulateUpdate[1][0]
    #mu1[1]+=simulateUpdate[1][1]


    # Plot your learned mixture components here:
    # for each component ...
    Z1 = gaussian_2d(X, Y, mu0[0], mu0[1], sig0[0],sig0[1])
    #Z2 = gaussian_2d(X, Y, mu1[0], mu1[1],  sig1[0],sig1[1])
    CS1 = plt.contour(X, Y, Z1)
    plt.clabel(CS1, inline=1, fontsize=10)
    #CS2 = plt.contour(X, Y, Z1)
    #plt.clabel(CS2, inline=1, fontsize=10)
    plt.pause(1)
    plt.title('Learned Gaussian Contours')
    plt.show()

   # plt.xlim(0, 1)
   # plt.ylim(0, 1)
   # plt.xlabel('x')
   # plt.title('test')
    #fig1.savefig("foo"+str(i)+".png")


#img_files = [ os.listdir('.')]
```

Out[2]:
```
'\n\n\nIn Class Exercise:  xx/100  (+ 10 extra credit if submitted before end
of day of class)\n******************************************************\n\nG
oal: Implement EM, for Gaussian Mixture given data X.\nmu, sigma, prior = EM
(X, m, epsilon)\n\nmu is lxm where l is dimensionality and m is number of mix
ture components\nsigma is m (diagonal) covariance matrices (so really just m
vectors), eg (lxm)\nprior is prior for each component 1xm (one by m)\nepsilon
is the total change in updated parameter values and is used as a stopping cri
terion\n\n******************************************************\n\nThe funct
ions should compute Gaussian Mixture parameters using EM given data set X. Pl
ot\nthe results of the learned parameters (using a contour like display given
below) after \neach iteration (or so) of the learning process. This will help
to visualize the \nprocess of learning. Feel free  to use the code snippets p
rovided below\nor feel free to edit as you see fit. Assume covariance matrice
s are diagonal. \n\nConstruct synthetic 2-d data X or find a simple data set
to use to test your code.\n***************************************************
******\n'
```

In Class

In [109]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# Compute Gaussian Likelihoods
def gaussian_2d(x, y, x0, y0, xsig, ysig):
    return np.exp(-0.5*(((x-x0) / xsig)**2 + ((y-y0) / ysig)**2))


def plt_2dGaussians(mu0, mu1, sig0, sig1):
    delta = 0.025
    x = np.arange(-2.0, 4.0, delta)
    y = np.arange(-1.0, 4.0, delta)
    X, Y = np.meshgrid(x, y)
    Z1 = gaussian_2d(X, Y, mu0[0], mu0[1], sig0[0], sig0[1])
    Z2 = gaussian_2d(X, Y, mu1[0], mu1[1], sig1[0], sig1[1])

    # Create a contour plot with labels using default colors.  The
    # inline argument to clabel will control whether the labels are draw
    # over the line segments of the contour, removing the lines beneath
    # the label

    # For use help, see https://matplotlib.org/
    plt.clf()
    plt.figure(figsize=(10,5))
    CS1 = plt.contour(X, Y, Z2)
    plt.clabel(CS1, inline=1, fontsize=10)
    CS2 = plt.contour(X, Y, Z1)
    plt.clabel(CS2, inline=1, fontsize=10)
    plt.scatter(X1, X2)

    plt.title('Learned Gaussian Contours')
    plt.show()


def EM(X, m = 2, epsilon = 0.000001):
    # let's assume dim l = 2

    theta_pre = np.random.random(4*m)
    mu = theta_pre[0:2*m]
    sigma = theta_pre[2*m:4*m]

    #if m == 2:
        #plt_2dGaussians(mu[0:2], mu[2:4], sigma[0:2], sigma[2:4])

    N = X.shape[0]
    prior_track = []
    mu_track = []
    sigma_track = []

    prior = np.array([1/m]*m)

    f = np.zeros((m, N))
    for k in range(m):
        f[k] = multivariate_normal.pdf(X, mean=mu[2*k:2*(k+1)], cov=sigma[2*k:
2*(k+1)])
```

```
    T = np.zeros((m, N))
    for k in range(m):
        T[k] = prior[k]*f[k]/np.matmul(prior, f)

    mu_after = np.zeros(2*m)
    sigma_after = np.zeros(2*m)

    for k in range(m):
        mu_after[2*k:2*(k+1)] = np.sum(T[k].reshape(N,1)*X, axis = 0)/np.sum(T
[k])
        sigma_after[2*k:2*(k+1)] = np.sqrt(np.sum(T[k].reshape(N,1)*(X-mu_afte
r[2*k:2*(k+1)])**2, axis = 0)/np.sum(T[k]))

    #if m == 2:
        #plt_2dGaussians(mu_after[0:2], mu_after[2:4], sigma_after[0:2], sigma
_after[2:4])

    theta = np.concatenate([mu_after, sigma_after])

    for k in range(m):
        prior[k] = np.sum(T[k])/N

    prior_track.append(prior)
    mu_track.append(mu_after)
    sigma_track.append(sigma_after)

    i = 1

    while max(abs(theta-theta_pre)) > epsilon:
        print("now iteration " + str(i))
        theta_pre = theta
        mu = theta_pre[0:2*m]
        sigma = theta_pre[2*m:4*m]

        for k in range(m):
            f[k] = multivariate_normal.pdf(X, mean=mu[2*k:2*(k+1)], cov=sigma[
2*k:2*(k+1)])

        for k in range(m):
            T[k] = prior[k]*f[k]/np.matmul(prior, f)

        mu_after = np.zeros(2*m)
        sigma_after = np.zeros(2*m)
        for k in range(m):
            mu_after[2*k:2*(k+1)] = np.sum(T[k].reshape(N,1)*X, axis = 0)/np.s
um(T[k])
            sigma_after[2*k:2*(k+1)] = np.sqrt(np.sum(T[k].reshape(N,1)*(X-mu_
after[2*k:2*(k+1)])**2, axis = 0)/np.sum(T[k]))
        print(mu_after)
        #if m == 2:
            #plt_2dGaussians(mu_after[0:2], mu_after[2:4], sigma_after[0:2], s
igma_after[2:4])

        theta = np.concatenate([mu_after, sigma_after])

        prior = np.zeros(m)
```

```
        for k in range(m):
            prior[k] = np.sum(T[k])/N

        i += 1
        prior_track.append(prior)
        mu_track.append(mu_after)
        sigma_track.append(sigma_after)

    print('Total iterations: ', i)

    return mu_after, sigma, prior, mu_track


X1 = [ 1, 1.5, 1.2, 1.2, .9, .8, 1, 2.3, 2.1, 2, 3, 2.5, 3]
X2 = [ 1.5, 1.2, 1.2, .9, .7, .8, 2.3, 2.1, 2, 3, 2.5, 3, 2.1]
X = np.column_stack([X1, X2])

mu, sigma, prior, mu_track = EM(X, m = 2, epsilon = 0.000001)
```

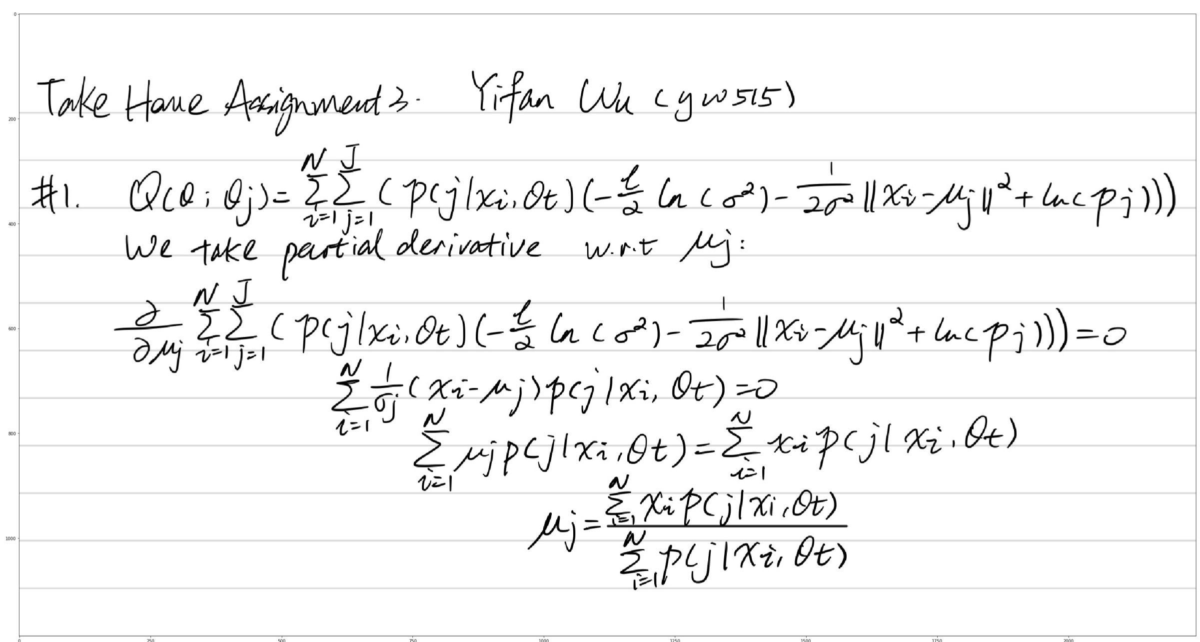
```
now iteration 1
[2.04023922 2.16811543 1.06492523 0.98373389]
now iteration 2
[2.13840797 2.26158872 1.08230421 1.04578312]
now iteration 3
[2.21570255 2.32226208 1.09227853 1.0945397 ]
now iteration 4
[2.27425687 2.35830791 1.0996885  1.13508606]
now iteration 5
[2.31623345 2.38051293 1.10616383 1.16477802]
now iteration 6
[2.34463948 2.39471393 1.11183604 1.18493306]
now iteration 7
[2.36322455 2.40389848 1.11654346 1.19834506]
now iteration 8
[2.37521192 2.40983906 1.12025461 1.20728759]
now iteration 9
[2.38291536 2.41368113 1.12307066 1.21328448]
now iteration 10
[2.38787314 2.41617157 1.12514944 1.21732358]
now iteration 11
[2.39107617 2.41779211 1.12665316 1.22005081]
now iteration 12
[2.39315544 2.41885129 1.12772432 1.22189436]
now iteration 13
[2.3945119  2.41954659 1.12847836 1.22314094]
now iteration 14
[2.39540098 2.42000483 1.12900426 1.22398372]
now iteration 15
[2.39598618 2.4203079  1.12936835 1.22455325]
now iteration 16
[2.39637281 2.42050894 1.12961896 1.22493792]
now iteration 17
[2.39662906 2.42064263 1.12979066 1.22519758]
now iteration 18
[2.39679934 2.42073173 1.12990785 1.22537276]
now iteration 19
[2.39691275 2.4207912  1.1299876  1.22549088]
now iteration 20
[2.39698843 2.42083095 1.13004174 1.22557049]
now iteration 21
[2.397039   2.42085756 1.13007842 1.22562413]
now iteration 22
[2.39707283 2.42087538 1.13010324 1.22566025]
now iteration 23
[2.39709549 2.42088733 1.13012002 1.22568457]
now iteration 24
[2.39711067 2.42089534 1.13013134 1.22570094]
now iteration 25
[2.39712086 2.42090072 1.13013897 1.22571195]
now iteration 26
[2.39712769 2.42090433 1.13014412 1.22571936]
now iteration 27
[2.39713228 2.42090676 1.13014759 1.22572435]
now iteration 28
[2.39713536 2.42090839 1.13014993 1.22572771]
now iteration 29
```

```
[2.39713743 2.42090948 1.1301515  1.22572996]
now iteration 30
[2.39713882 2.42091022 1.13015256 1.22573148]
now iteration 31
[2.39713975 2.42091071 1.13015327 1.2257325 ]
now iteration 32
[2.39714038 2.42091104 1.13015375 1.22573319]
Total iterations:  33
```

Take Home

(1).

In [81]:
```python
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img1 = mpimg.imread('1.jpg')
plt.figure(figsize = (50, 40))
plt.imshow(img1)
plt.show()
```



Take Home Assignment 3. Yifan Wu (yw515)

#1. $Q(\theta;\theta_j) = \sum_{i=1}^{N}\sum_{j=1}^{J}\left(p(j|x_i,\theta_t)\left(-\frac{1}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\|x_i-\mu_j\|^2 + \ln(p_j)\right)\right)$

We take partial derivative w.r.t μ_j:

$$\frac{\partial}{\partial \mu_j}\sum_{i=1}^{N}\sum_{j=1}^{J}\left(p(j|x_i,\theta_t)\left(-\frac{1}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\|x_i-\mu_j\|^2 + \ln(p_j)\right)\right) = 0$$

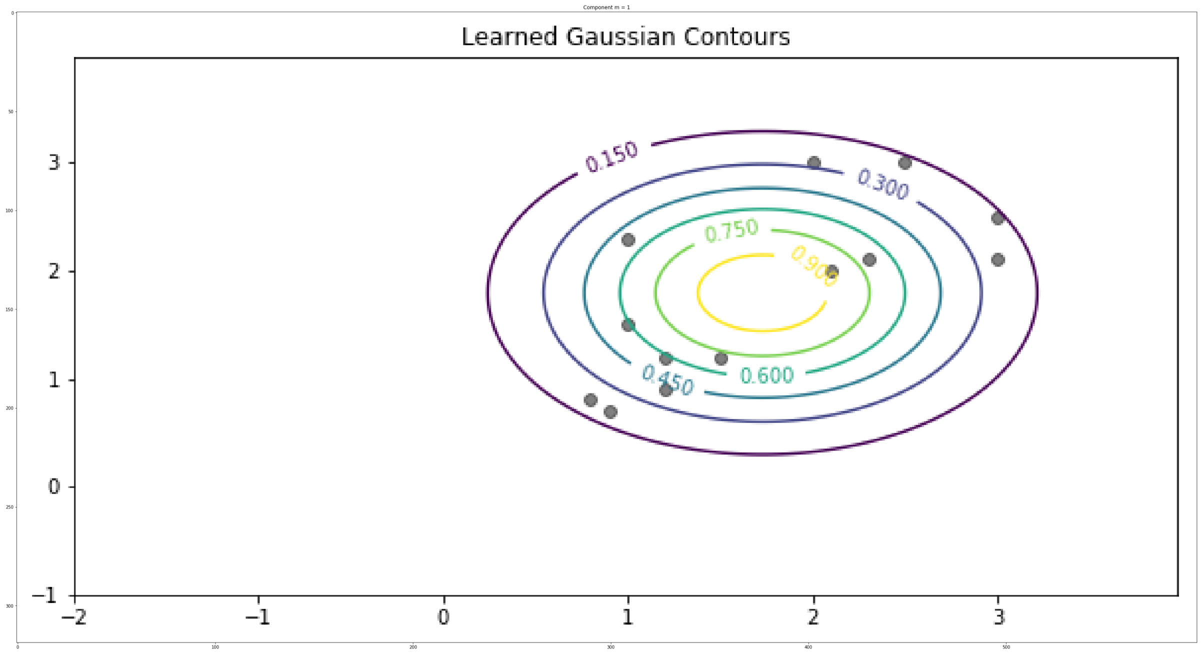
$$\sum_{i=1}^{N}\frac{1}{\sigma_j}(x_i-\mu_j)p(j|x_i,\theta_t) = 0$$

$$\sum_{i=1}^{N}\mu_j p(j|x_i,\theta_t) = \sum_{i=1}^{N} x_i p(j|x_i,\theta_t)$$

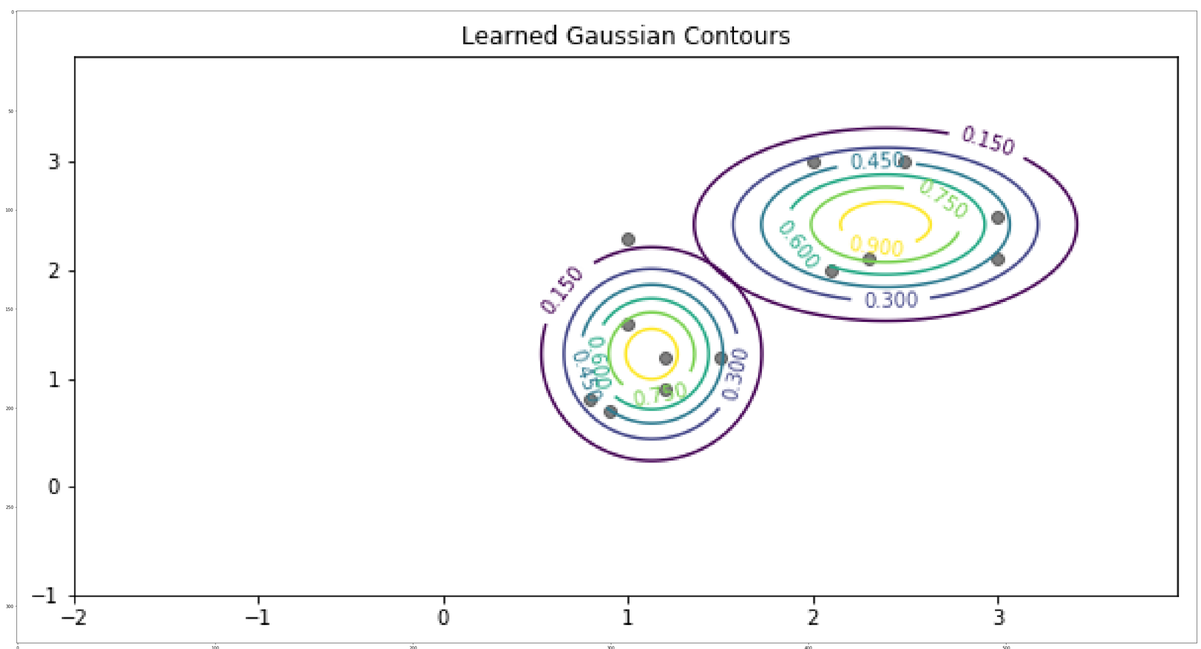
$$\mu_j = \frac{\sum_{i=1}^{N} x_i p(j|x_i,\theta_t)}{\sum_{i=1}^{N} p(j|x_i,\theta_t)}$$

(2)(a).

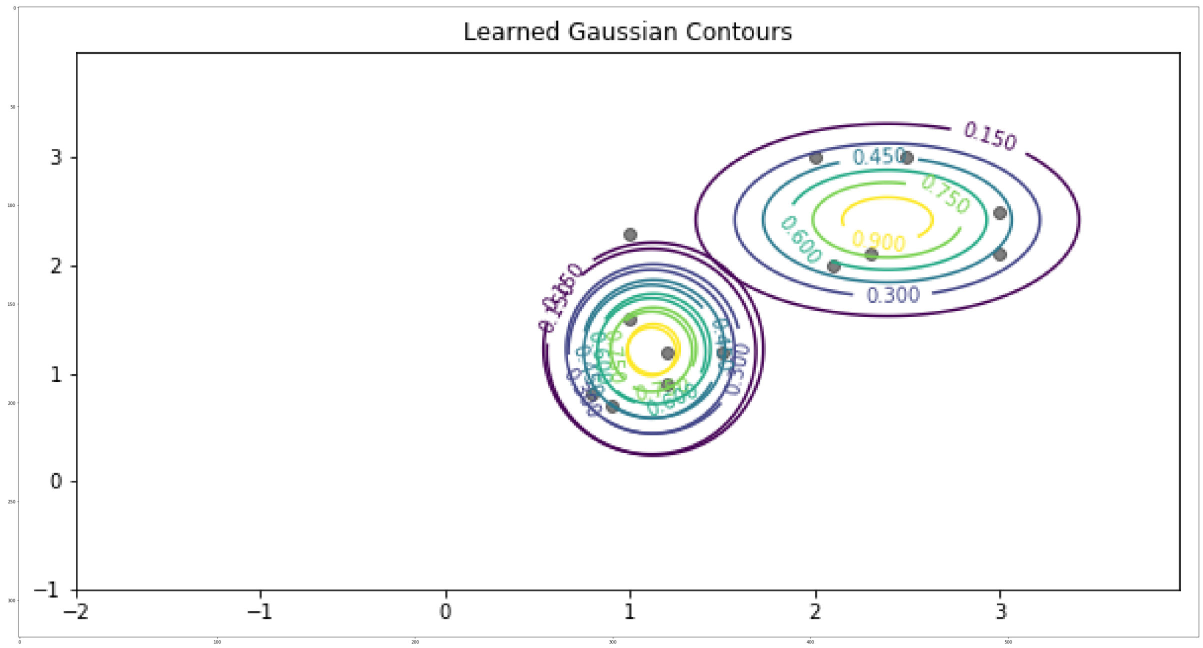
In [98]:
```
img1 = mpimg.imread('m_is_1.png')
plt.figure(figsize = (50, 40))
plt.imshow(img1)
plt.title("Component m = 1")
plt.show()
```



In [83]:
```
img1 = mpimg.imread('m_is_2.png')
plt.figure(figsize = (50, 40))
plt.imshow(img1)
plt.show()
```



In [84]:
```
img1 = mpimg.imread('m_is_3.png')
plt.figure(figsize = (50, 40))
plt.imshow(img1)
plt.show()
```



From the above coontour plot we can see that, when m = 1, the distribution is too generalized for the data set; When m = 3, two of the components actually have the same distribution. m = 2 is the best number of component for this data set. The epsilon I choose is 10^{-4}.

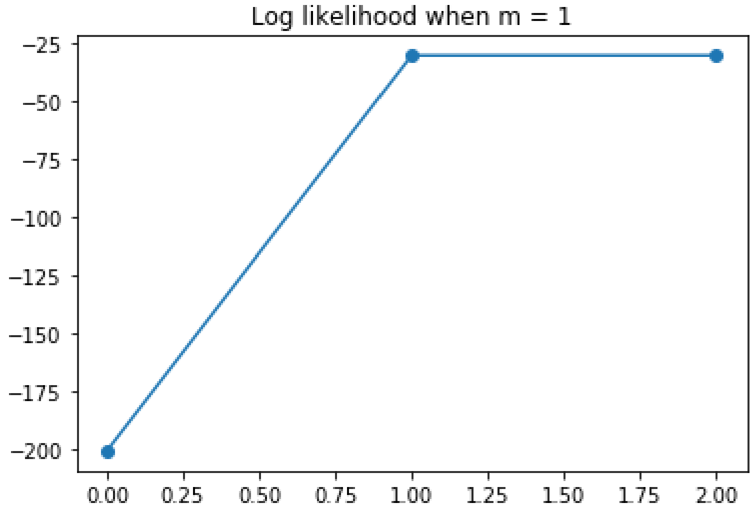
(2)(b).

In [108]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
import random

# Compute Gaussian Likelihoods
def gaussian_2d(x, y, x0, y0, xsig, ysig):
    return np.exp(-0.5*(((x-x0) / xsig)**2 + ((y-y0) / ysig)**2))


def plt_2dGaussians(mu0, mu1, sig0, sig1):
    delta = 0.025
    x = np.arange(-2.0, 4.0, delta)
    y = np.arange(-1.0, 4.0, delta)
    X, Y = np.meshgrid(x, y)
    Z1 = gaussian_2d(X, Y, mu0[0], mu0[1], sig0[0], sig0[1])
    Z2 = gaussian_2d(X, Y, mu1[0], mu1[1], sig1[0], sig1[1])

    # Create a contour plot with labels using default colors.  The
    # inline argument to clabel will control whether the labels are draw
    # over the line segments of the contour, removing the lines beneath
    # the label

    # For use help, see https://matplotlib.org/
    plt.clf()
    plt.figure(figsize=(10,5))
    CS1 = plt.contour(X, Y, Z2)
    plt.clabel(CS1, inline=1, fontsize=10)
    CS2 = plt.contour(X, Y, Z1)
    plt.clabel(CS2, inline=1, fontsize=10)
    plt.scatter(X1, X2)

    plt.title('Learned Gaussian Contours')
    plt.show()


def EM(X, m = 2, epsilon = 0.000001):
    # let's assume dim l = 2
    random.seed(4000)
    theta_pre = np.random.random(4*m)
    mu = theta_pre[0:2*m]
    sigma = theta_pre[2*m:4*m]

    #if m == 2:
        #plt_2dGaussians(mu[0:2], mu[2:4], sigma[0:2], sigma[2:4])

    N = X.shape[0]
    prior_track = []
    mu_track = []
    sigma_track = []

    prior = np.array([1/m]*m)
    prior_track.append(prior)
    mu_track.append(mu)
    sigma_track.append(sigma)
    f = np.zeros((m, N))
```

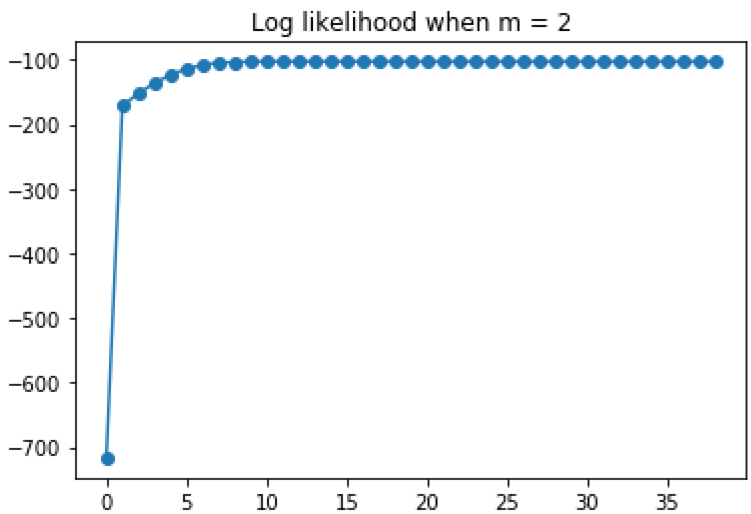
```
    for k in range(m):
        f[k] = multivariate_normal.pdf(X, mean=mu[2*k:2*(k+1)], cov=sigma[2*k:
2*(k+1)])

    T = np.zeros((m, N))
    for k in range(m):
        T[k] = prior[k]*f[k]/np.matmul(prior, f)

    mu_after = np.zeros(2*m)
    sigma_after = np.zeros(2*m)

    mu_after = np.zeros(2*m)
    sigma_after = np.zeros(2*m)
    for k in range(m):
        mu_after[2*k:2*(k+1)] = np.sum(T[k].reshape(N,1)*X, axis = 0)/np.sum(T
[k])
        sigma_after[2*k:2*(k+1)] = np.sqrt(np.sum(T[k].reshape(N,1)*(X-mu_afte
r[2*k:2*(k+1)])**2, axis = 0)/np.sum(T[k]))

    #if m == 2:
        #plt_2dGaussians(mu_after[0:2], mu_after[2:4], sigma_after[0:2], sigma
_after[2:4])

    theta = np.concatenate([mu_after, sigma_after])

    for k in range(m):
        prior[k] = np.sum(T[k])/N

    prior_track.append(prior)
    mu_track.append(mu_after)
    sigma_track.append(sigma_after)

    i = 1

    while max(abs(theta-theta_pre)) > epsilon:
        theta_pre = theta
        mu = theta_pre[0:2*m]
        sigma = theta_pre[2*m:4*m]

        for k in range(m):
            f[k] = multivariate_normal.pdf(X, mean=mu[2*k:2*(k+1)], cov=sigma[
2*k:2*(k+1)])

        for k in range(m):
            T[k] = prior[k]*f[k]/np.matmul(prior, f)


        for k in range(m):
            mu_after[2*k:2*(k+1)] = np.sum(T[k].reshape(N,1)*X, axis = 0)/np.s
um(T[k])
            sigma_after[2*k:2*(k+1)] = np.sqrt(np.sum(T[k].reshape(N,1)*(X-mu_
after[2*k:2*(k+1)])**2, axis = 0)/np.sum(T[k]))

        #if m == 2:
            #plt_2dGaussians(mu_after[0:2], mu_after[2:4], sigma_after[0:2], s
igma_after[2:4])
```

```
        theta = np.concatenate([mu_after, sigma_after])

        prior = np.zeros(m)
        for k in range(m):
            prior[k] = np.sum(T[k])/N

        i += 1
        prior_track.append(prior)
        mu_track.append(mu_after)
        sigma_track.append(sigma_after)

    print('Total iterations: ', i)


    ll_list = []
    for i in range(len(mu_track)):
        ll_sum = 0
        for n in range(X.shape[0]):
            for j in range(m):
                ll_sum += np.log(prior_track[i][j]*multivariate_normal.pdf(X[n
], mean=mu_track[i][2*j:2*(j+1)], cov=sigma_track[i][2*j:2*(j+1)]))
        ll_list.append(ll_sum)

    plt.plot(range(len(mu_track)), ll_list, '-o')
    plt.title("Log likelihood when m = " + str(m))
    plt.show()

    return mu_after, sigma, prior, prior_track, ll_list, mu_track


X1 = [ 1, 1.5, 1.2, 1.2, .9, .8, 1, 2.3, 2.1, 2, 3, 2.5, 3]
X2 = [ 1.5, 1.2, 1.2, .9, .7, .8, 2.3, 2.1, 2, 3, 2.5, 3, 2.1]
X = np.column_stack([X1, X2])

for i in [1,2,3]:
    mu, sigma, prior, prior_track, ll_list, mu_track = EM(X, m = i, epsilon = 
0.000001)
```

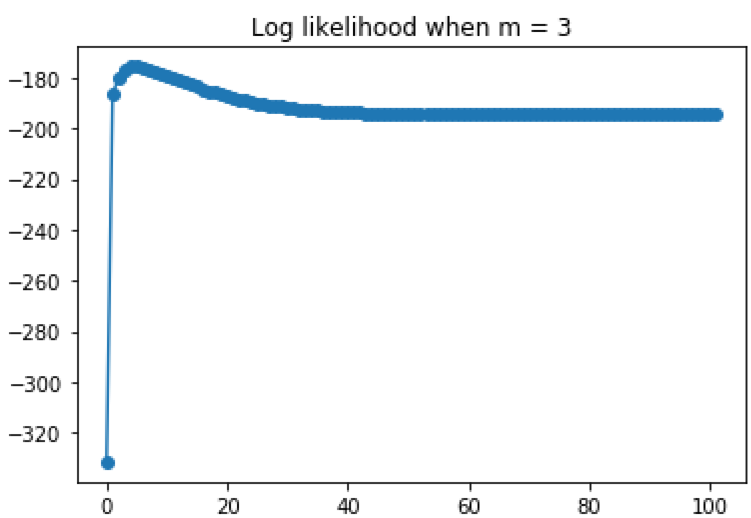
```
Total iterations:  2
```




```
Total iterations:  38
```




```
Total iterations:  101
```



To your existing code, add a function that plots (in a different figure) the loglikelihood as a function of iteration. Can you use this information as a stopping criterion rather than epsilon? Can you use this information to determine which number of components is best?

A: Similar result can be achieved by using log likeihood compare to stopping criterion. The log likelihood will finally converge and the parameters will be fixed. The log likelihood can be used to choose number of component. Looking from the graph we can conclude that 2 component is better for the dataset by acheieving higher loglikelihood.

```
In [ ]:
```