Homework 03

一、 实验目标：

实现混合高斯模型算法

二、 实验过程：

主要思路是估计数据由某个高斯模型生成的概率。对于每个数据 x_i，其由第 k 个 Component 生成的概率为

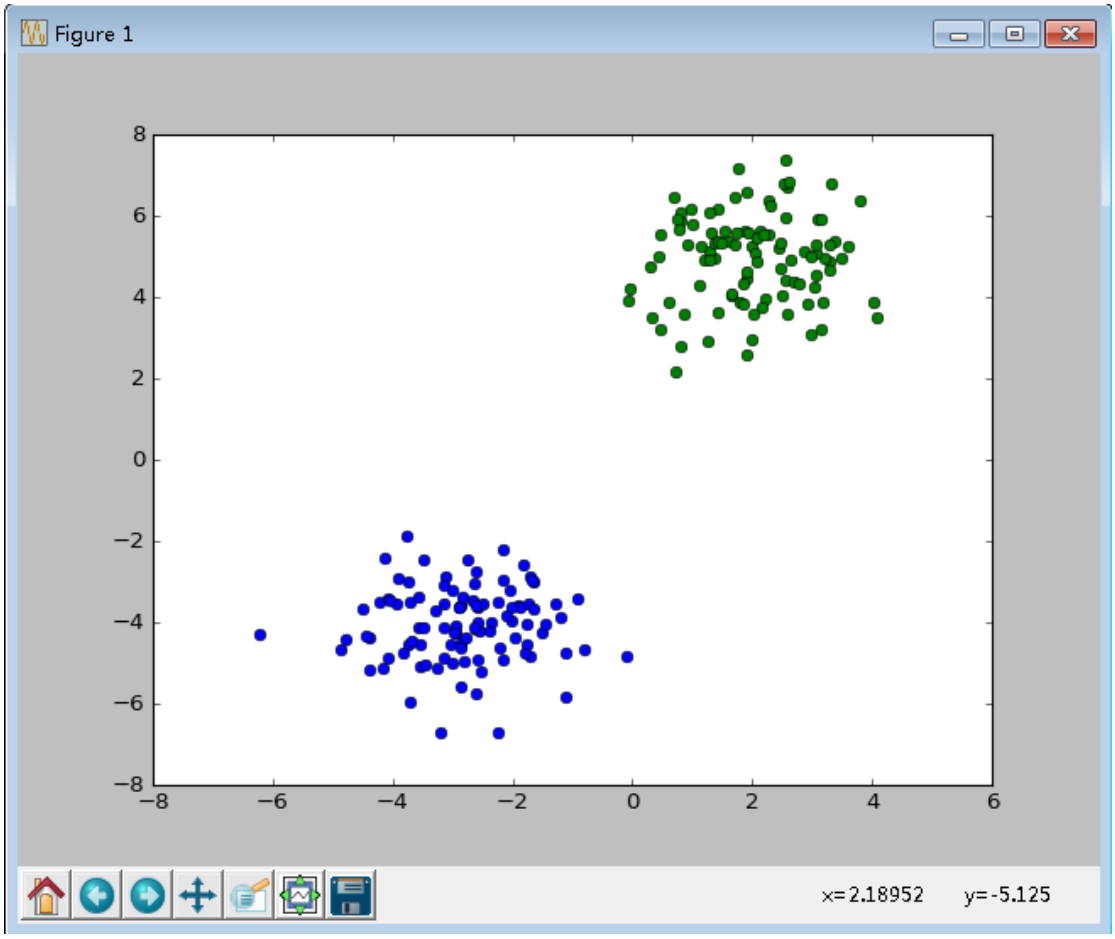
$$\gamma(i,k)=\frac{\pi_k\mathcal{N}(x_i|\mu_k,\Sigma_k)}{\sum_{j=1}^{K}\pi_j\mathcal{N}(x_i|\mu_j,\Sigma_j)}$$

第一步，基于高斯分布，随机生成采样点。其中 100 个采样点基于（-3，-4），协方差（1，1）的 2D 高斯分布获得，另外 100 个采样点基于（2，5），协方差（1，1）的 2D 高斯分布获得。对 200 个采样点进行合并，获得数据集。

第二步，初始化各项参数，给定随机值。开始迭代过程。在每次迭代过程中，首先求各个高斯分布单独生成样本的概率，然后计算出每个样本由各个 component 生成的概率，由此更新各项参数，包括最重要的混合高斯模型的系数。最后重新计算每个 component 的均值和协方差，根据新生成的各项参数，进入下一次迭代过程，直到设定的 200 次迭代完成。

第三步，将完成分类的各点用不同颜色绘制。

三、 实验结果：



从图中可以明显看出，分类效果良好

四、 实验代码及注释：

```
import matplotlib.pyplot as plt
import numpy as np

def Gaussian2D(mean, var, num):
    points = [];
    points.append(mean[0] + np.random.randn(num) * var[0])
    points.append(mean[1] + np.random.randn(num) * var[1])
    return points
def initial(classes, dim):
    miu = np.zeros((classes, dim))
    Sigma = np.zeros((classes, dim, dim))
    detSigma = np.zeros(classes)
    invSigma = Sigma.copy()
```

```
    Pi = np.zeros(classes)
    for cl in range(classes):
        Pi[cl] = np.random.rand()
        miu[cl] = np.random.rand(dim)
        Sigma[cl] = np.eye(dim, dim)
        invSigma[cl] = np.linalg.inv(Sigma[cl])
        detSigma[cl] = np.linalg.det(Sigma[cl])
        Pi=Pi/sum(Pi)
    return (miu, Sigma, detSigma, invSigma, Pi)
samples1 = Gaussian2D([-3,-4], [1,1], 100)
samples2 = Gaussian2D([2,5], [1,1], 100)
samples = np.vstack([np.hstack([samples1[0], samples2[0]]) , np.hstack([samples1[1], samples2[1]])])
samples = samples.T
classes = 2
rows, dim = samples.shape
#initial values
miu, Sigma, detSigma, invSigma, Pi = initial(classes, dim)
gamma=np.zeros((rows,dim))
iterator = 0
while iterator < 200:
    iterator += 1
    for row in range(rows):
        for cl in range(classes):
            #calculate the coefficient of MOG's exponent
            coef = (2 * np.pi) ** (-len(samples[row]) * 1.0 / 2) * (detSigma[cl] ** (-0.5))
            #calculete the probability of one Gaussian distribution
            vsum = coef * np.exp(-0.5 * np.dot(np.dot(samples[row] - miu[cl], invSigma[cl]), samples[row] - miu[cl]))
            gamma[row, cl] = Pi[cl] * vsum
        #calculate every sample's probibility generated by each Gaussian distribution
        gamma[row] = gamma[row] / sum(gamma[row])
    Nk = sum(gamma, 0)
    #update Pi
    Pi = Nk / rows
    miu = np.zeros((classes, dim))
    Sigma = np.zeros((classes, dim, dim))
    for cl in range(classes):
        #recalculate the miu of each component
        for row in range(rows):
            miu[cl] += gamma[row][cl]*samples[row]
        miu[cl] = miu[cl] / Nk[cl]
        #recalculate the cov of each component
        for row in range(rows):
```

```
                vsum = np.mat(samples[row] - miu[cl])
                Sigma[cl] += gamma[row][cl] * np.dot(vsum.T, vsum)
            Sigma[cl] = Sigma[cl]/Nk[cl]
            detSigma[cl] = np.linalg.det(Sigma[cl])
            invSigma[cl] = np.linalg.inv(Sigma[cl])
idx = np.zeros(rows)
for row in range(rows):
    idx[row] = np.nonzero(gamma[row] == max(gamma[row]))[0][0]
for cl in range(classes):
    t = np.nonzero(idx == cl)[0]
    plt.plot(samples[t, 0], samples[t, 1], 'o', label = 'Group ' + str(row + 1))
plt.show()
```