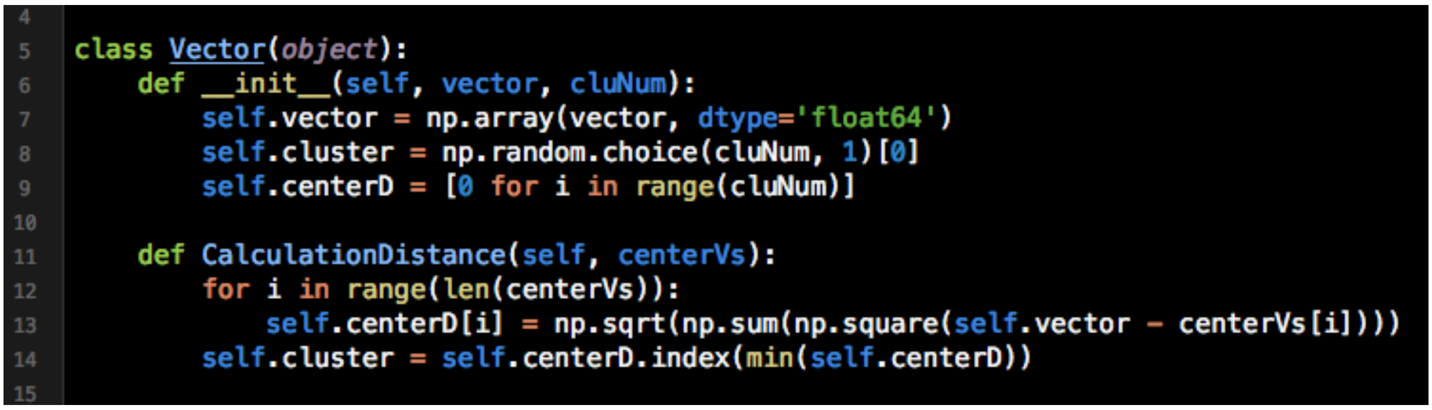
k-means&EM算法实现报告

周政

1.k-means算法

Vector类



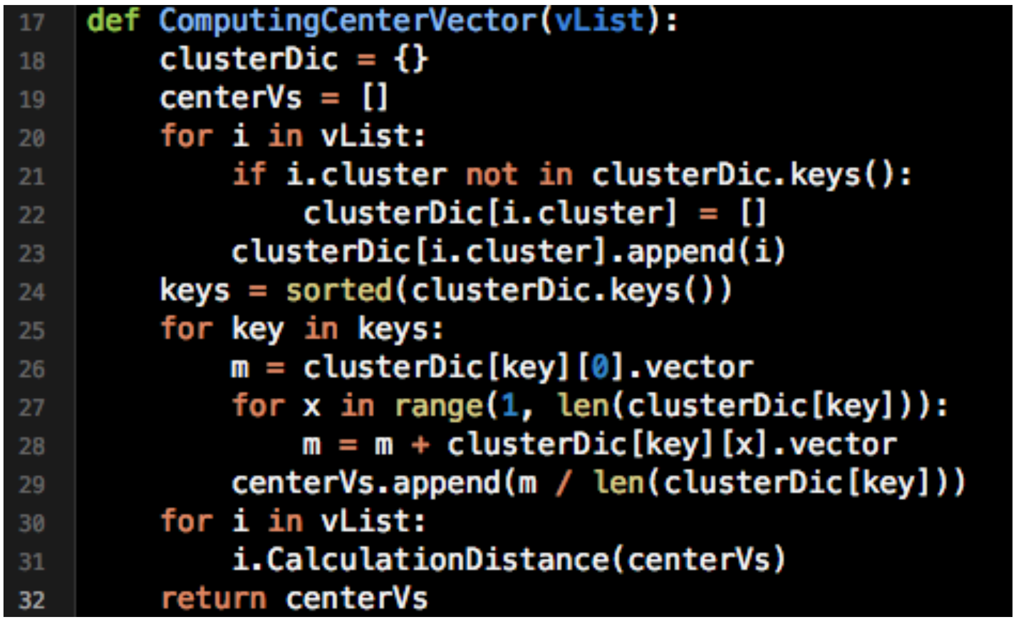
```python
class Vector(object):
    def __init__(self, vector, cluNum):
        self.vector = np.array(vector, dtype='float64')
        self.cluster = np.random.choice(cluNum, 1)[0]
        self.centerD = [0 for i in range(cluNum)]

    def CalculationDistance(self, centerVs):
        for i in range(len(centerVs)):
            self.centerD[i] = np.sqrt(np.sum(np.square(self.vector - centerVs[i])))
        self.cluster = self.centerD.index(min(self.centerD))
```

用 `vector` , `cluster` , `centerD` 分别来记录:

`被聚类向量的值`
`目前的聚类`
`到各聚类中心向量的距离`

用 `CalculationDistance()` 方法来计算目前向量到各聚类中心向量的距离并选择最近的聚类更新cluster的值.

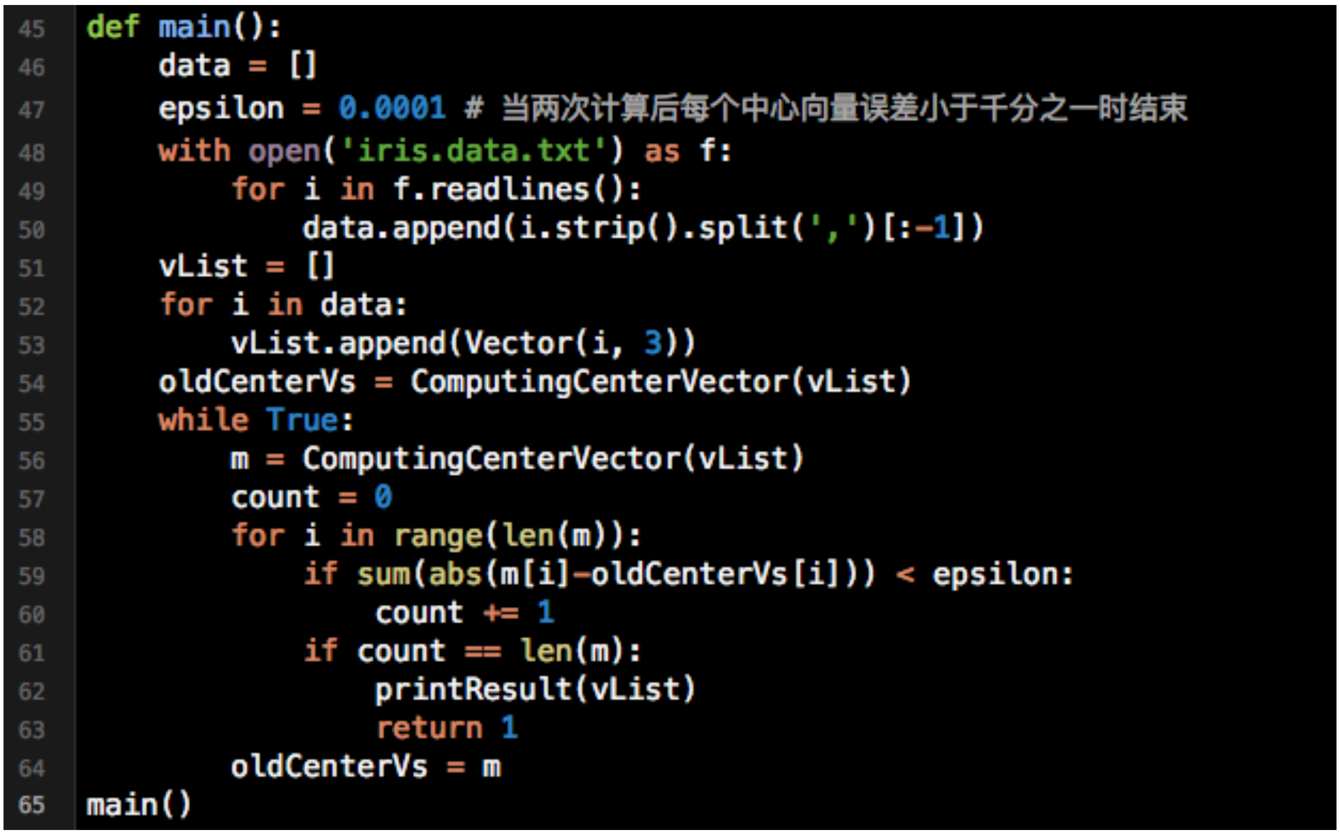
ComputingCenterVector(vList)函数



```python
def ComputingCenterVector(vList):
    clusterDic = {}
    centerVs = []
    for i in vList:
        if i.cluster not in clusterDic.keys():
            clusterDic[i.cluster] = []
        clusterDic[i.cluster].append(i)
    keys = sorted(clusterDic.keys())
    for key in keys:
        m = clusterDic[key][0].vector
        for x in range(1, len(clusterDic[key])):
            m = m + clusterDic[key][x].vector
        centerVs.append(m / len(clusterDic[key]))
    for i in vList:
        i.CalculationDistance(centerVs)
    return centerVs
```

传入装有Vector对象的list根据当前Vector.cluster来计算各个聚类的中心向量

主函数main()

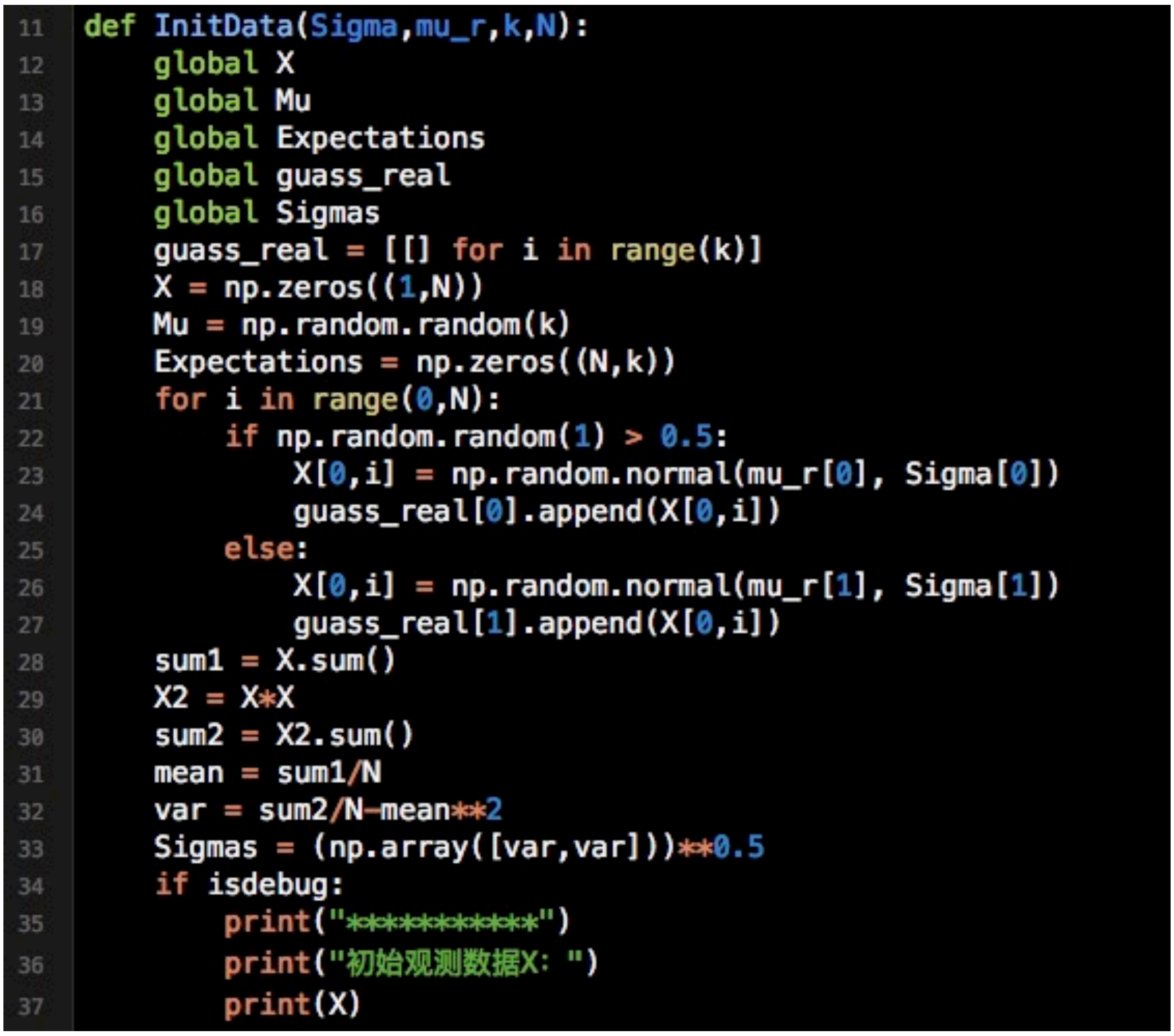


```python
def main():
    data = []
    epsilon = 0.0001 # 当两次计算后每个中心向量误差小于千分之一时结束
    with open('iris.data.txt') as f:
        for i in f.readlines():
            data.append(i.strip().split(',')[:-1])
    vList = []
    for i in data:
        vList.append(Vector(i, 3))
    oldCenterVs = ComputingCenterVector(vList)
    while True:
        m = ComputingCenterVector(vList)
        count = 0
        for i in range(len(m)):
            if sum(abs(m[i]-oldCenterVs[i])) < epsilon:
                count += 1
            if count == len(m):
                printResult(vList)
                return 1
        oldCenterVs = m
main()
```

读入iris.data.txt中的数据然后初始化Vector对象迭代执行聚类函数直到各聚类中心向量两次迭代后改变小于epsilon结束迭代并输出聚类结果

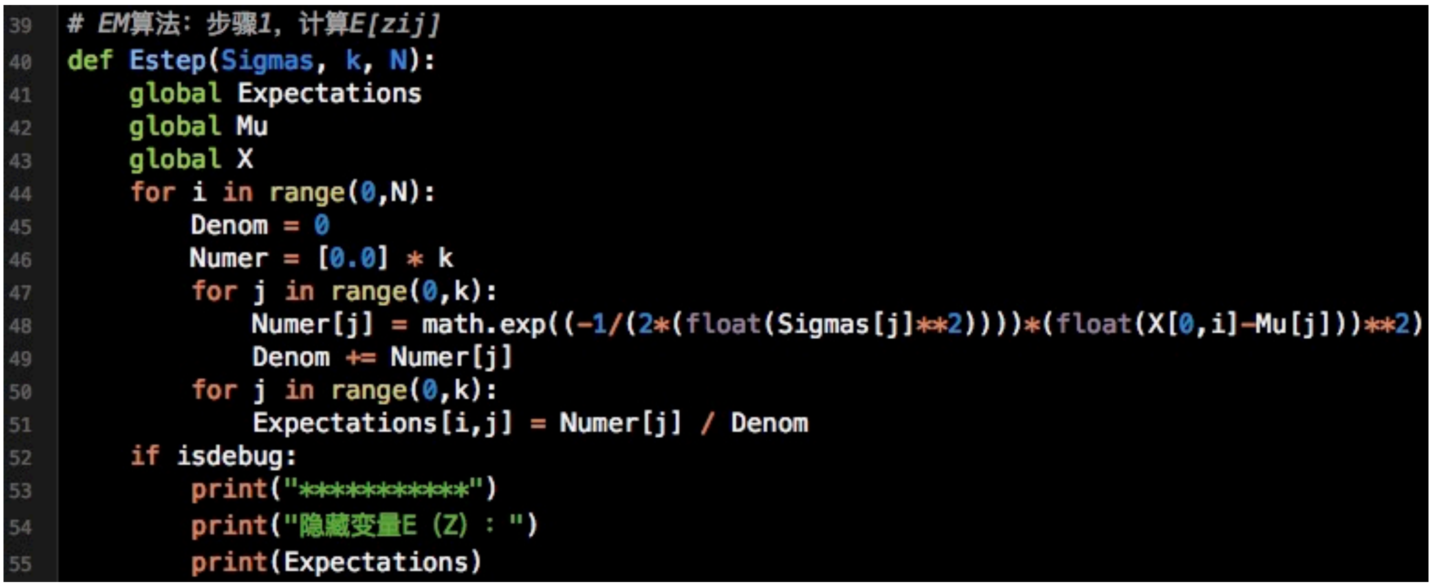
1.EM算法

InitData()函数


```
def InitData(Sigma,mu_r,k,N):
    global X
    global Mu
    global Expectations
    global guass_real
    global Sigmas
    guass_real = [[] for i in range(k)]
    X = np.zeros((1,N))
    Mu = np.random.random(k)
    Expectations = np.zeros((N,k))
    for i in range(0,N):
        if np.random.random(1) > 0.5:
            X[0,i] = np.random.normal(mu_r[0], Sigma[0])
            guass_real[0].append(X[0,i])
        else:
            X[0,i] = np.random.normal(mu_r[1], Sigma[1])
            guass_real[1].append(X[0,i])
    sum1 = X.sum()
    X2 = X*X
    sum2 = X2.sum()
    mean = sum1/N
    var = sum2/N-mean**2
    Sigmas = (np.array([var,var]))**0.5
    if isdebug:
        print("***********")
        print("初始观测数据X: ")
        print(X)
```

通过传入的sigma值和mu值初始化数据，并随机生成两个μ值类作为预测μ的初始值，用整个数据的标准差作为两个要预测的高斯函数的标准差

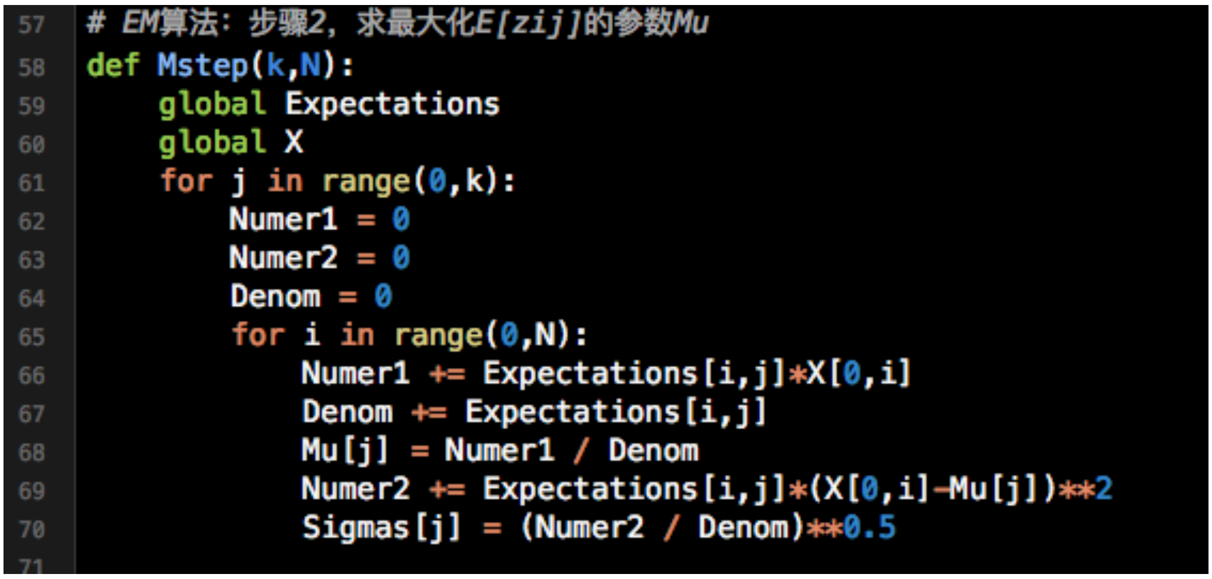
Estep()函数



```python
# EM算法：步骤1，计算E[zij]
def Estep(Sigmas, k, N):
    global Expectations
    global Mu
    global X
    for i in range(0,N):
        Denom = 0
        Numer = [0.0] * k
        for j in range(0,k):
            Numer[j] = math.exp((-1/(2*(float(Sigmas[j]**2))))*(float(X[0,i]-Mu[j]))**2)
            Denom += Numer[j]
        for j in range(0,k):
            Expectations[i,j] = Numer[j] / Denom
    if isdebug:
        print("***********")
        print("隐藏变量E（Z）：")
        print(Expectations)
```

通过预测的μ和sigma来计算各个点属于各个高斯分布的概率并存在Expectations里.

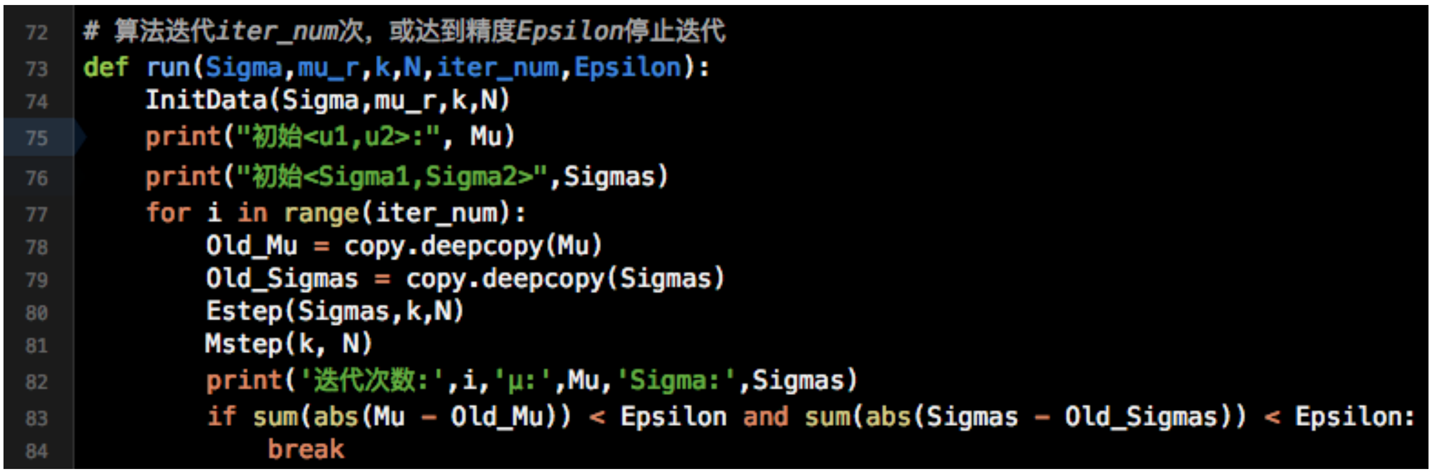
Mstep()函数



```python
# EM算法：步骤2，求最大化E[zij]的参数Mu
def Mstep(k,N):
    global Expectations
    global X
    for j in range(0,k):
        Numer1 = 0
        Numer2 = 0
        Denom = 0
        for i in range(0,N):
            Numer1 += Expectations[i,j]*X[0,i]
            Denom += Expectations[i,j]
            Mu[j] = Numer1 / Denom
            Numer2 += Expectations[i,j]*(X[0,i]-Mu[j])**2
            Sigmas[j] = (Numer2 / Denom)**0.5
```

通过每个点到不同高斯分布的Expectations来更新各个高斯分布的μ和sigma.

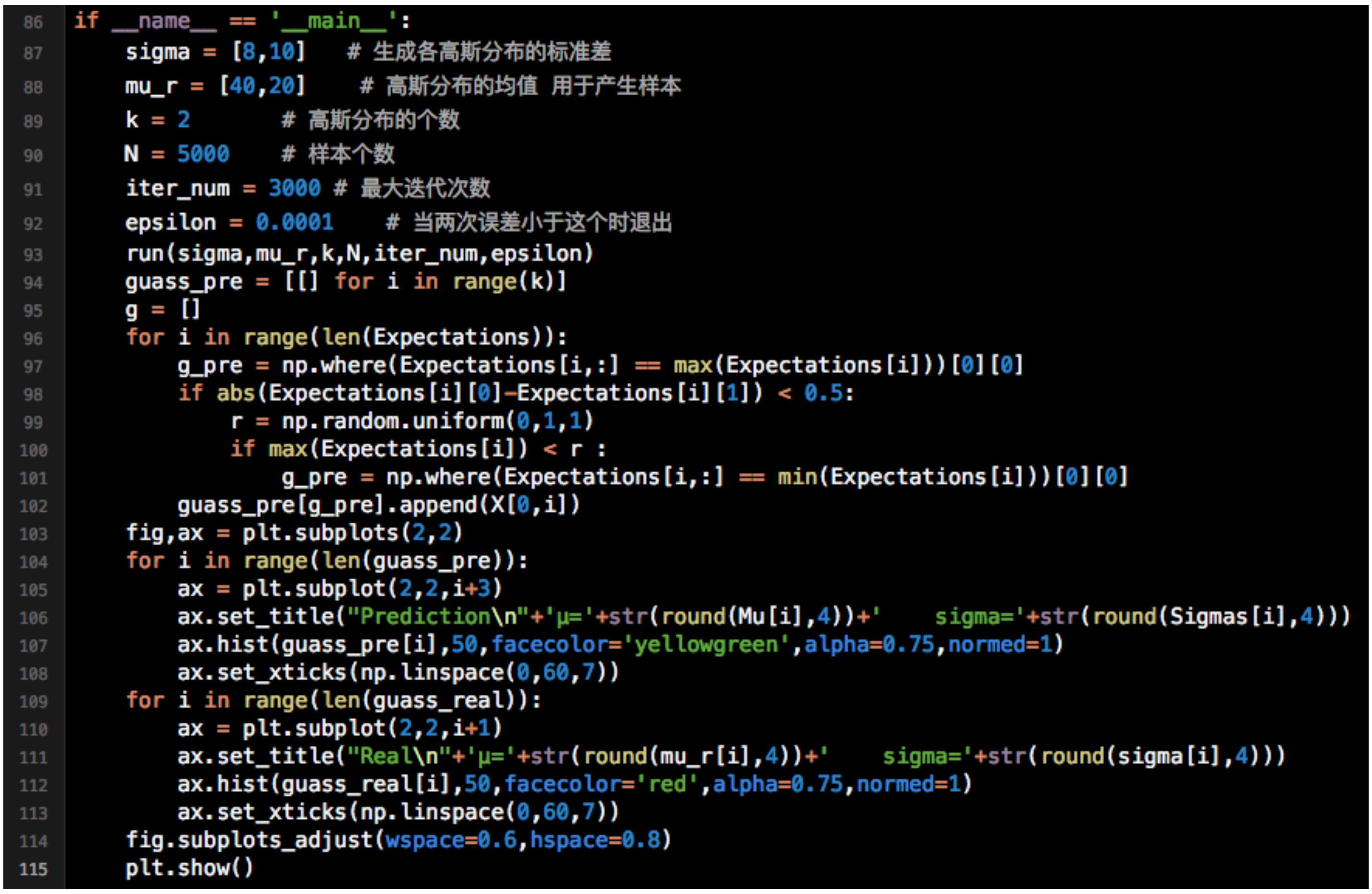
run()函数



```python
# 算法迭代iter_num次，或达到精度Epsilon停止迭代
def run(Sigma,mu_r,k,N,iter_num,Epsilon):
    InitData(Sigma,mu_r,k,N)
    print("初始<u1,u2>:", Mu)
    print("初始<Sigma1,Sigma2>",Sigmas)
    for i in range(iter_num):
        Old_Mu = copy.deepcopy(Mu)
        Old_Sigmas = copy.deepcopy(Sigmas)
        Estep(Sigmas,k,N)
        Mstep(k, N)
        print('迭代次数:',i,'μ:',Mu,'Sigma:',Sigmas)
        if sum(abs(Mu - Old_Mu)) < Epsilon and sum(abs(Sigmas - Old_Sigmas)) < Epsilon:
            break
```

执行迭代并通过设置好的迭代次数和精度来控制迭代的终止.

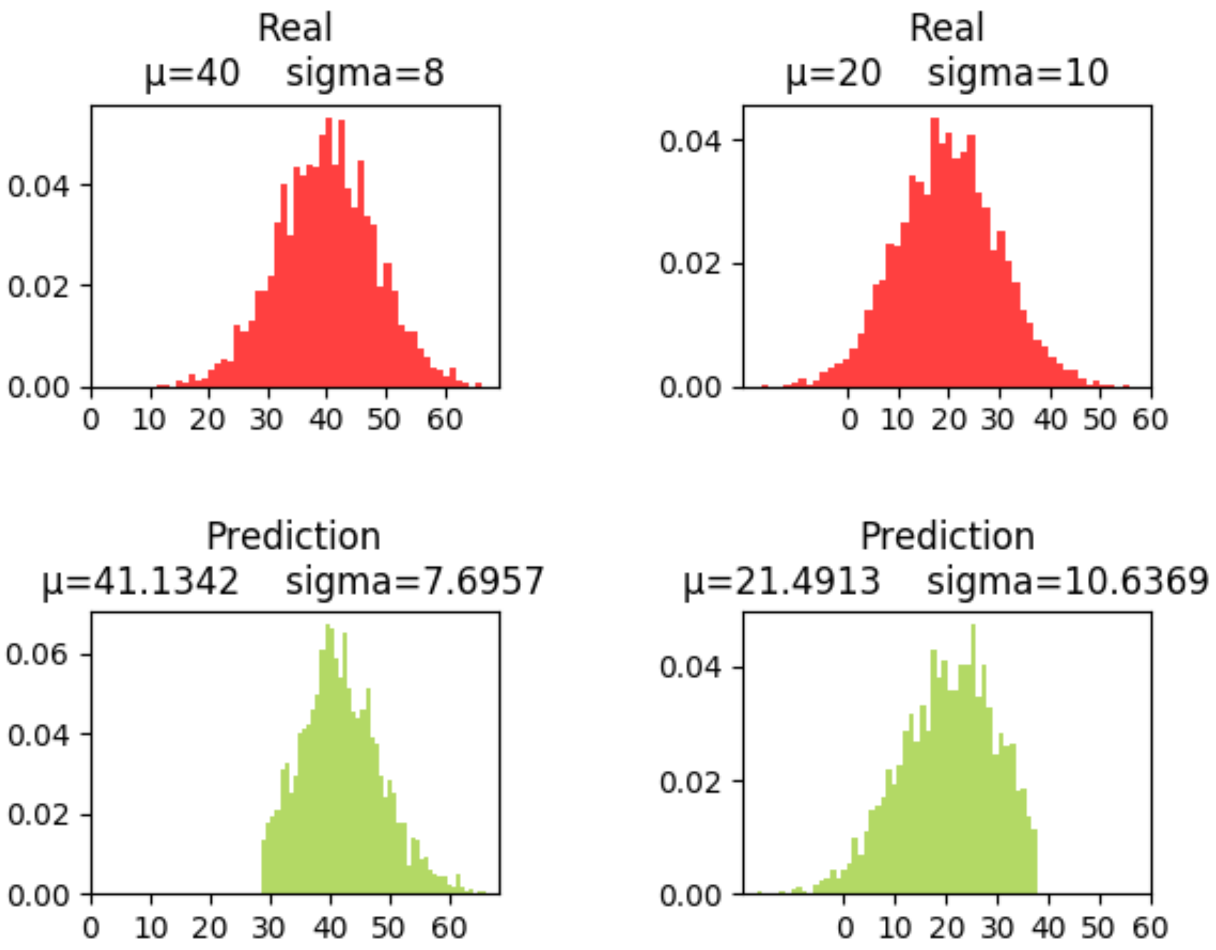
初始执行条件设置与画图



```python
if __name__ == '__main__':
    sigma = [8,10]    # 生成各高斯分布的标准差
    mu_r = [40,20]     # 高斯分布的均值 用于产生样本
    k = 2        # 高斯分布的个数
    N = 5000     # 样本个数
    iter_num = 3000 # 最大迭代次数
    epsilon = 0.0001     # 当两次误差小于这个时退出
    run(sigma,mu_r,k,N,iter_num,epsilon)
    guass_pre = [[] for i in range(k)]
    g = []
    for i in range(len(Expectations)):
        g_pre = np.where(Expectations[i,:] == max(Expectations[i]))[0][0]
        if abs(Expectations[i][0]-Expectations[i][1]) < 0.5:
            r = np.random.uniform(0,1,1)
            if max(Expectations[i]) < r :
                g_pre = np.where(Expectations[i,:] == min(Expectations[i]))[0][0]
        guass_pre[g_pre].append(X[0,i])
    fig,ax = plt.subplots(2,2)
    for i in range(len(guass_pre)):
        ax = plt.subplot(2,2,i+3)
        ax.set_title("Prediction\n"+'μ='+str(round(Mu[i],4))+'     sigma='+str(round(Sigmas[i],4)))
        ax.hist(guass_pre[i],50,facecolor='yellowgreen',alpha=0.75,normed=1)
        ax.set_xticks(np.linspace(0,60,7))
    for i in range(len(guass_real)):
        ax = plt.subplot(2,2,i+1)
        ax.set_title("Real\n"+'μ='+str(round(mu_r[i],4))+'     sigma='+str(round(sigma[i],4)))
        ax.hist(guass_real[i],50,facecolor='red',alpha=0.75,normed=1)
        ax.set_xticks(np.linspace(0,60,7))
    fig.subplots_adjust(wspace=0.6,hspace=0.8)
    plt.show()
```

设置好初始运行参数后开始执行，并将结果画图表现出来.

运行结果


Real
μ=40 sigma=8
Real
μ=20 sigma=10
Prediction
μ=41.1342 sigma=7.6957
Prediction
μ=21.4913 sigma=10.6369