```python
from hw4code.DataPoints import DataPoints
from hw4code.KMeans import KMeans, compute_purity,compute_NMI
import math
from scipy.stats import multivariate_normal

# =============================================================================
class GMM:
    # ---------------------------------------------------------------------
    def __init__(self):
        self.dataSet = []
        self.K = 0
        self.mean = [[0.0 for x in range(2)] for y in range(3)]
        self.stdDev = [[0.0 for x in range(2)] for y in range(3)]
        self.coVariance = [[[0.0 for x in range(2)] for y in range(2)] for z in range(3)]
        self.W = None
        self.w = None
    # ---------------------------------------------------------------------
    def main(self, dataname):

        self.dataname = dataname[5:-4]
        print("\nFor " + self.dataname)
        self.dataSet = KMeans.readDataSet(dataname)
        self.K = DataPoints.getNoOFLabels(self.dataSet)
        # weight for pair of data and cluster
        self.W = [[0.0 for y in range(self.K)] for x in range(len(self.dataSet))]
        # weight for pair of data and cluster
        self.w = [0.0 for x in range(self.K)]
        self.GMM()

    # ---------------------------------------------------------------------
    def GMM(self):
        clusters = []
        # [num_clusters,2]
        self.mean = [[0.0 for y in range(2)] for x in range(self.K)]
        # [num_clusters,2]
        self.stdDev = [[0.0 for y in range(2)] for x in range(self.K)]
        # [num_clusters,2]
        self.coVariance = [[[0.0 for z in range(2)] for y in range(2)] for x in range(self.K)]
        k = 0
        while k < self.K:
            cluster = set()
            clusters.append(cluster)
            k += 1

        # Initially randomly assign points to clusters
        i = 0
        for point in self.dataSet:
            clusters[i % self.K].add(point)
            i += 1

        # Initially assign equal prior weight for each cluster
        for m in range(self.K):
            self.w[m] = 1.0 / self.K

        # Get Initial mean, std, covariance matrix
        DataPoints.getMean(clusters, self.mean)
        DataPoints.getStdDeviation(clusters, self.mean, self.stdDev)
        DataPoints.getCovariance(clusters, self.mean, self.stdDev, self.coVariance)
```

```
59
60          length = 0
61          while True:
62              mle_old = self.Likelihood()
63              self.Estep()
64              self.Mstep()
65              length += 1
66              mle_new = self.Likelihood()
67
68              # convergence condition
69              if abs(mle_new - mle_old) / abs(mle_old) < 0.000001:
70                  break
71
72          print("Number of Iterations = " + str(length))
73          print("\nAfter Calculations")
74          print("Final mean = ")
75          self.printArray(self.mean)
76          print("\nFinal covariance = ")
77          self.print3D(self.coVariance)
78
79          # Assign points to cluster depending on max prob.
80          for j in range(self.K):
81              clusters[j] = set()
82
83          i = 0
84          for point in self.dataSet:
85              index = -1
86              prob = 0.0
87              for j in range(self.K):
88                  if self.W[i][j] > prob:
89                      index = j
90                      prob = self.W[i][j]
91              temp = clusters[index]
92              temp.add(point)
93              i += 1
94
95          # Calculate purity and NMI
96          compute_purity(clusters,len(self.dataSet))
97          compute_NMI(clusters,self.K)
98
99          # write clusters to file for plotting
100         f = open("GMM_" + self.dataname + ".csv", "w")
101         for w in range(self.K):
102             print("Cluster " + str(w) + " size :" + str(len(clusters[w])))
103             for point in clusters[w]:
104                 f.write(str(point.x) + "," + str(point.y) + "," + str(w) + "\n")
105         f.close()
106     # ------------------------------------------------------------------
107     def Estep(self):
108         # Update self.W
109         for i in range(len(self.dataSet)):
110             denominator = 0.0
111             for j in range(self.K):
112                 gaussian = multivariate_normal(self.mean[j], self.coVariance[j])
113                 # Compute numerator for self.W[i][j] below
114                 numerator = 0.0
115                 # ========================#
116                 # STRART YOUR CODE HERE  #
117                 # ========================#
118
```

```
                # w_ij = w_j*f(xi)
                numerator = self.w[j]*gaussian.pdf([self.dataSet[i].x,
self.dataSet[i].y])
                # =======================#
                #   END YOUR CODE HERE   #
                # =======================#
                self.W[i][j] = numerator
                denominator += numerator

            # normalize W[i][j] into probabilities
            # =======================#
            # STRART YOUR CODE HERE  #
            # =======================#
            self.W[i] = self.W[i] / denominator
            # =======================#
            #   END YOUR CODE HERE   #
            # =======================#
    # ---------------------------------------------------------------------
    def Mstep(self):
        for j in range(self.K):
            denominator = 0.0
            numerator_x = 0.0
            numerator_y = 0.0
            cov_xy = 0.0
            updatedMean_x = 0.0
            updatedMean_y = 0.0

            # update self.w[j] and self.mean
            for i in range(len(self.dataSet)):
                denominator += self.W[i][j]
                updatedMean_x += self.W[i][j] * self.dataSet[i].x
                updatedMean_y += self.W[i][j] * self.dataSet[i].y

            self.w[j] = denominator / len(self.dataSet)

            #update self.mean
            # =======================#
            # STRART YOUR CODE HERE  #
            # =======================#
            self.mean[j][0] = updatedMean_x / denominator
            self.mean[j][1] = updatedMean_y / denominator
            # =======================#
            #   END YOUR CODE HERE   #
            # =======================#

            # update covariance matrix
            for i in range(len(self.dataSet)):
                numerator_x += self.W[i][j] * pow((self.dataSet[i].x - self.mean[j]
[0]), 2)
                numerator_y += self.W[i][j] * pow((self.dataSet[i].y - self.mean[j]
[1]), 2)
                # Compute conv_xy +=?
                # =======================#
                # STRART YOUR CODE HERE  #
                # =======================#

                covar = (self.dataSet[i].x - self.mean[j][0]) * (self.dataSet[i].y -
self.mean[j][1])
                cov_xy += self.W[i][j] * covar
                # =======================#
```

```
                #   END YOUR CODE HERE   #
                # ========================#

            self.stdDev[j][0] = numerator_x / denominator
            self.stdDev[j][1] = numerator_y / denominator


            self.coVariance[j][0][0] = self.stdDev[j][0]
            self.coVariance[j][1][1] = self.stdDev[j][1]
            self.coVariance[j][0][1] = self.coVariance[j][1][0] = cov_xy /
denominator
    # ------------------------------------------------------------------
    def Likelihood(self):
        likelihood = 0.0
        for i in range(len(self.dataSet)):
            numerator = 0.0
            for j in range(self.K):
                gaussian = multivariate_normal(self.mean[j], self.coVariance[j])
                numerator += self.w[j] * gaussian.pdf([self.dataSet[i].x,
self.dataSet[i].y])
            likelihood += math.log(numerator)
        return likelihood
    # ------------------------------------------------------------------
    def printArray(self, mat):
        for i in range(len(mat)):
            for j in range(len(mat[i])):
                print(str(mat[i][j]) + " "),
            print("")
    # ------------------------------------------------------------------
    def print3D(self, mat):
        for i in range(len(mat)):
            print("For Cluster : " + str((i + 1)))
            for j in range(len(mat[i])):
                for k in range(len(mat[i][j])):
                    print(str(mat[i][j][k]) + " "),
                print("")
            print("")

# ======================================================================
if __name__ == "__main__":
    g = GMM()
    dataname = "dataset1.txt"
    g.main(dataname)
```