On the methodology and comparison of the graph-cut, normalized graph-cut, and expectation maximization algorithms for segmentation

Luis E. Tirado*
l tirado@gmail.com

NUID 000185039

March 12, 2009

*Electrical and Computer Engineering Department, Northeastern University
Abstract

The present paper will detail the graph-cut, normalized graph-cut, and expectation maximization algorithms as segmentation techniques. The aim of these methods is to partition a set of data points into clusters by differing approaches.

The paper is organized as follows: the first section will introduce the segmentation and clustering problem as a whole, and define background information as required. The next subsections go into detail describing and comparing the graph-cut and normalized graph-cut algorithms. Section 3 deals with the Expectation Maximization (EM) algorithm. Section 4 uses MATLAB as a framework for computer experiments to generate results and discuss the graph-cut and EM algorithms. Lastly, conclusions are formulated in section 5.
1 Introduction

Segmentation is a field of computer vision which arises from partitioning an image in order to extract useful information, such as a structure or objects of interest. Applications of segmentation include medical imaging, face and fingerprint recognition, and locating objects in aerial images, among many others. One may choose to view segmentation as clustering, in which case image partitions may be clustered by measures of distance, intensity, color, texture, or any combination of these.

The discussions that continue will be limited to segmentation by distance for 80 independent two-dimensional samples forming four clusters with different means and equal standard deviations. The elements in the clusters can be grouped using three different partitioning algorithms. Two of these algorithms are called graph-cut and normalized graph-cut, which fall in the graph-theoretical category. The basic idea of segmentation by graph-cut is that every element is viewed as a graph vertex. The similarity between two elements is determined using the Euclidean distance between them, which constitutes a weighting factor. Segmentation is done via edge-cutting the graph to form sets of connected components with high weights, which imply good clustering.

In addition, the elements will be clustered using the expectation maximization (EM) probabilistic mixture-resolving algorithm. This algorithm recursively generates estimates for the mean, covariance matrix, and probabilities of cluster membership starting from a set of initial conditions. Given that the algorithm can converge to a local maximum, a good initialization method is important in order to obtain good clustering results.

The next section will expand on the theory for the spectral clustering based algorithms. In the two subsections that follow, we discuss the graph-cut and normalized graph-cut clustering approaches. EM clustering is explained in detail in section 3 and MATLAB simulations using graph-cut and EM are in section 4. Lastly, we conclude in section 5.
2 Graph-theoretic based segmentation

We begin by discussing and introducing the terminology and theory for graph-theoretic clustering. A graph is a set of vertices $V$ and edges $E$ that connect various pairs of vertices, and can be written as $G = \{V, E\}$. In the particular case of the problem at hand, the graph is a weighted undirected graph, which means that a weight is associated with each edge and there is no distinction drawn between edges $(a, b)$ and $(b, a)$.[1]

The approach to segmentation is representing the weighted graph by a square matrix $A$ named the affinity matrix. The element $a_{ij}$ represents the weight (affinity measure) on the edge from vertex $i$ to vertex $j$. For an undirected graph, we use a symmetric matrix and place half the weight in each of the $i$, $j$th, and $j$, $i$th elements.[1] The weights between edges correspond to similarity between components. We define the cut of a graph as the sum of the weights of the edges joining a set of vertices as:

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$$ (1)

For the case of the set of points in 2 dimensions, a Gaussian decay affinity distance metric is defined as:

$$w_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_d^2}\right)$$ (2)

where $\|\cdot\|^2$ is the Euclidean distance between the two vertices (2-norm), and $\sigma_d^2$ is the parameter for Gaussian decay. It is small if only closely-spaced points are grouped, and large if further-spaced points are grouped.

2.1 Graph-cut segmentation

It turns out that there is a strong link between the eigenvectors of the affinity matrix. If the affinity matrix is block-diagonalized, the summation of the weights of the off-diagonal block matrices is minimized, or the summation of the diagonal block matrices is maximized.[2] We
define a vector of weights \( \mathbf{W}_n \) for each of the \( 1, \ldots, N \) data points as \( \mathbf{W}_n = [w_{n1} w_{n2} \ldots w_{nN}]^T \), such that \( \mathbf{W}_n^T \mathbf{W}_n = 1 \) and construct the matrix \( \mathbf{A} \) based on Eq. (2). The block-diagonalization problem is solved by maximizing the quantity \( \mathbf{W}_n^T \mathbf{A} \mathbf{W}_n \). To solve this problem, following the approach in [1], the Lagranian is: \( \mathbf{W}_n^T \mathbf{A} \mathbf{W}_n + \lambda (\mathbf{W}_n^T \mathbf{W}_n - 1) \) where \( \lambda \) is a Lagrange multiplier. After differentiating and ignoring a factor of 2, we get: \( \mathbf{A} \mathbf{W}_n = \lambda \mathbf{W}_n \), which implies that \( \mathbf{W}_n \) is an eigenvector of \( \mathbf{A} \).

This presents a very simple solution to clustering. After computing the eigenvalues of \( \mathbf{A} \), we can use them to compute the most stable clustering configuration by using the value of \( k \) that maximizes the expression: \( \Delta_k = |\Delta_k - \Delta_{k+1}| \), called the eigengap.[3] Next, we find the eigenvectors \( \mathbf{e}_k \) that correspond to the \( k \) largest eigenvalues of the affinity matrix \( \mathbf{A} \). We notice that \( e_{ki} \) is an indicator of the \( i \)th element belonging to the \( k \)th cluster. If \( e_{ki} \) is close to zero, the \( i \)th element does not belong to the \( k \)th cluster, otherwise it does.[2]

This graph-based cut approach mentioned above is not without its flaws, however. For example, if the data set consists of copies of the same set of points, the affinity matrix exhibits a repeated block structure. This means that we would have very similar eigenvalues, which implies that the eigenvectors would not split clusters, because any linear combination of eigenvectors with the same eigenvalue is also an eigenvector.[2] To overcome this issue, we proceed in discussing the normalized graph-cut approach to clustering.

### 2.2 Normalized graph-cut segmentation

An alternative and, in fact, recommended approach for spectral clustering is to always use normalized clustering algorithms[4]. One such popular algorithm developed by Shi-Malik[5] is called the normalized cut. Shi and Malik propose a new segmentation criteria that avoids partitioning out small sets of points and defines it as:

\[
Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)},
\]

(3)
where \( \text{cut}(A,B) \) is defined in (1), and \( \text{assoc}(A,V) = \sum_{u \in A, t \in V} w_{ut} \), which is the connection from all nodes in the set \( A \) to the nodes in the graph, and \( \text{assoc}(B,V) \) is defined correspondingly. It is shown in [5] that minimizing the disassociation between the groups is equivalent to maximizing the association within the groups, and both conditions can be satisfied simultaneously. Even though the problem is NP-complete, it turns out that an approximation to the solution can be found discretely as follows: [5]

**Step 1:** Compute \( N \times N \) matrices: \( A \) with Eq. (2), and \( D = \text{diag}(d_i) = D(i,i) = \sum_j w_{ij} \)

**Step 2:** Solve generalized eigensystem \((D-A)y = \lambda Dy\). \((D-A)\) is also called the Laplacian matrix \( L \). Obtain eigenvector of \( L \) with second smallest eigenvalue.

**Step 3:** Use eigenvector from previous step to bipartition the graph. Shi and Malik ascertain that we may choose 0, the median, or the point where Ncut is minimized. The point which minimizes Ncut is shown to be:

\[
y^T(D-A)y \over y^TDy, \text{where: } b = \frac{k}{1-k}, \quad k = \frac{\sum_{x_i>0} d_i}{\sum_i d_i}, \quad (4)
\]

\( y = (1 + x) - b(1 - x) \), and \( x \) is a vector depicting cluster membership as \( x_i = 1 \) if point \( i \) is in \( A \), and \( -1 \), otherwise.

**Step 4:** Partition can be subdivided if the stability of the cut is good and if Ncut is below a certain threshold. If necessary, repartition graph.

It turns out that the graph cut method mentioned in section 2.1 can also be applied to the Laplacian matrix \( L \) defined above, as a recursive way to bi-partition the data and produce similar results. The recursive bi-partitioning is done with the eigenvector corresponding to the second smallest eigenvalue of the matrix, also called the Fiedler vector.

### 3 Expectation-maximization (EM) segmentation

Expectation maximization is an iterative algorithm that is used find maximum likelihood (ML) estimates when observations can be viewed as incomplete data. [6] The algorithm
consists of an expectation step (E) and a minimization step (M). The E step computes the expectation of the likelihood function via the use of a latent variable. The M step then maximizes the log-likelihood of the complete data to estimate the parameters $\Theta$.

In order to understand how EM works for clustering, we first start with a generic model of Gaussian mixtures as: $p(x) = \sum_{j=1}^{g} \pi_j N(x|\mu_j, \Sigma_j)$, where $x$, $g$, $\mu_j$, $\Sigma_j$, and $\pi_j$, represent the data point, number of clusters, means, covariance matrix, and mixing coefficients of the Gaussian mixtures, respectively. Given that $\sum_{j=1}^{g} \pi_j = 1; 0 \leq \pi_j \leq 1$, we can interpret these as prior probabilities of a data point belonging to a cluster. We wish to find the mixture coefficients, means, and covariances for each Gaussian, thus being able to assign probabilities to the individual data points.

The problem of segmentation can, in fact, be reformulated as a missing data problem.[2] The missing data in this case is which component generated each data point. If we knew this labeling information, we could estimate the parameters $\Theta$ of the distribution function that generated the set of $X = \{x_k, k = 1, \ldots, N\}$ data points based on ML theory. Our solution is introducing a latent variable $Z$, an $N \times g$ matrix initialized uniformly at random, such that it contains a 1 in the $k$th column if the data point is in a cluster, and 0 otherwise, which is the labeling part we were missing. The probability distribution function (PDF) for the bivariate normal distribution which applies to our data set is:

$$p(x_k|\theta_j) = \frac{1}{2\pi|\Sigma_j|^{1/2}} \exp \left[ -\frac{1}{2} (x_k - \mu_j)^T \Sigma_j^{-1} (x_k - \mu_j) \right]$$

(5)

The parameters $\Theta$ of the function above can be estimated by maximizing the likelihood function, defined as:

$$p(X|\Theta) = \prod_{n=1}^{N} \left( \sum_{j=1}^{g} p(x_n|\theta_j)\pi_j \right)$$

(6)

We wish to maximize the function above[7]; in order to do so we take its logarithm and end
up with the log-likelihood:

$$\log p(X|\Theta) = \sum_{n=1}^{N} \log \left( \sum_{j=1}^{g} p(x_n|\theta_j)\pi_j \right)$$  \hfill (7)

The problem with Eq. (7) is that there is no closed form solution for the maximum likelihood, unless we introduce the latent variable $Z$, as defined earlier, in order to account for the missing values. Given these values, we can write the complete data as $\mathbf{Y} = [X^T, Z^T]^T$ [8] and the log-likelihood becomes:

$$\log p(Y|\Theta) = \sum_{n=1}^{N} \log \left( \sum_{j=1}^{g} z_{nj}p(x_n|\theta_j) \right) = \sum_{n=1}^{N} \sum_{j=1}^{g} \log z_{nj}p(x_n|\theta_j)$$ \hfill (8)

ML estimation is easy now, and we denote the ML estimate as: $\hat{\Theta} = \arg \max_{\Theta} \log p(Y|\Theta)$

Once we have our initial hypotheses of class membership, we start iterating between the maximization and expectation steps. We compute the prior probabilities $\pi_i$, the means $\mu_i$, and covariance $\Sigma_i$, (where $i = 1, \ldots, g$) defined in [8] as:

$$\pi_i = \frac{1}{N} \sum_{n=1}^{N} p(i|x_n, \Theta)$$ \hfill (9a)

$$\mu_i = \frac{\sum_{n=1}^{N} x_n p(i|x_n, \Theta)}{\sum_{n=1}^{N} p(i|x_n, \Theta)}$$ \hfill (9b)

$$\Sigma_i = \frac{\sum_{n=1}^{N} p(i|x_n, \Theta)[(x_n - \mu_i)(x_n - \mu_i)^T]}{\sum_{n=1}^{N} p(i|x_n, \Theta)}$$ \hfill (9c)

After this is done, we calculate the distribution in Eq. (5) with the new measurements calculated in Eq. (9), and compute the log-likelihood function as the term in the middle of Eq. (8), since we have already calculated the inner terms of the summation. Finally, we do the E step, which updates $Z$ with the calculated posterior probabilities as the input to the
M step of the next iteration of the EM algorithm as:

\[ E[z_{ki}] = \frac{\pi_i p(x_k|\theta_i)}{\sum_{j=1}^{g} \pi_j p(x_k|\theta_j)} \]  \hspace{1cm} (10)

One drawback against the EM algorithm is that we need to know the number of components to cluster in advance\(^2\). However, an advantage is that the algorithm is guaranteed to converge given that the likelihood function increases with each iteration until it reaches a local maximum (or in some cases, a saddle point), and does not subsequently decrease. Thus, it makes sense to impose a stopping condition based on the difference of the value of the current and previous log-likelihood value. A more precise convergence statement is given in \([9]\). Because EM converges to a local rather than a global maximum, it depends on the initial estimates of the parameters \(\Theta\). If prior knowledge of optimal parameters is available, convergence can be accelerated, and the algorithm is likelier to converge to a global maximum. In general, EM is guaranteed to be stable and to converge to a ML estimate.\(^9\)

4 Computer simulations

In this section we present results from MATLAB simulations of two-dimensional \((x,y)\) data point clustering using the graph-cut and expectation maximization approaches. The MATLAB code that generated the simulations and results is attached in appendix A.

We define a configuration where there are a total of \(N = 80\) independent two dimensional samples forming four clusters in the following way (where \(I\) is the identity matrix):

(a) 20 points from a Gaussian distribution \(N[(0,0),0.04I]\)
(b) 20 points from a Gaussian distribution \(N[(0,1),0.04I]\)
(c) 20 points from a Gaussian distribution \(N[(1,0),0.04I]\)
(d) 20 points from a Gaussian distribution \(N[(1,1),0.04I]\)
4.1 Graph-cut simulations

Our first approach was to segment the data set in section 4 via the approach mentioned in section 2.1, which uses the \( k \) largest eigenvectors of the affinity matrix \( A \). The affinity matrices and their eigenvalues for the data set are shown below in Figure 1:

![Figure 1: Distance affinity matrices and their eigenvalues](image)

We can see that the affinity matrix \( A \) has a block-diagonal structure when the data is grouped by clusters. However, the eigenvalues and performance of the clustering by the four largest eigenvectors are unaffected by the order in which the data is organized in the matrix. Figure 2(a) shows that the four eigenvectors corresponding to the four largest eigenvalues of \( A \) for \( \sigma_d = 0.2 \) have produced large values indicating the grouping of data points into clusters. The clustering results for different values of \( \sigma_d \) is shown in Figure 2(b). For \( \sigma_d = 0.1 \), the affinity of the blocks in \( A \) is small, thus only some points are clustered together. For \( \sigma_d = 0.2 \), we can see four clear blocks, and clustering is good, while for \( \sigma_d = 1 \) blocks are less apparent, and all the data gets grouped in one cluster.
4.2 EM simulations

Because the likelihood is multimodal, some runs of the algorithm might tend to a lower mode with a smaller likelihood if the starting points are in the domain of attraction of a lower mode. This is why it is important to run the algorithm using different starting conditions to escape such lower modes. Figures 3 and 4 show the log-likelihood plots and the EM algorithm clustering results given two randomly initialized runs. As can be seen, the run with the maximum log-likelihood value correctly classified every data point, whereas the estimated parameters for the next run deviated from the true values.

Figure 3: EM log-likelihood function for two randomly initialized runs
Figure 4: EM clustering results for two randomly initialized runs

5 Conclusion

This paper implemented simple examples of clustering via the graph-cut and EM algorithms. It was shown that spectral clustering is an effective method of data segmentation, given a good choice of Gaussian decay parameter $\sigma_d$, and number of clusters $k$, especially when considering normalized cut or other spectral methods which optimize these parameter values for a given data set.[11] The limited number of data points in our experiment does not fare well for EM convergence, although this algorithm is quite an effective method of clustering, offering the advantage of outputting soft decisions which can be greatly useful for some fields such as radar discrimination, an area in which the author is currently working professionally.
References


A MATLAB code

A.1 Graph-cut method

```matlab
%% spectral_clustering.m
close all; clear all; clc;
% Computes distance affinity matrix for data set using Gaussian decay with
% sigma_d = 0.1, 0.2, and 1. Also computes the eigenvalues and eigenvectors
% of the affinity matrices and uses them to segment the data.

% creates random data for experiment and saves it for later use
% var_n = 0.04;
% var = [var_n 0; 0 var_n];
% data = [mvnrnd([0 0],var,20); mvnrnd([0 1],var,20); ... %
%         mvnrnd([1 0],var,20); mvnrnd([1 1],var,20)];
% plot(data(:,1),data(:,2),'.')
% save('data.mat','data')

% loads data
load data

% if DATA_PERM is true, creates random permutation of data points to plot
% the affinity matrices w/ data in random order instead of generated order
DATA_PERM = 0;
% if(DATA_PERM)
%     perm = randperm(length(data));
%     data = data(perm,:);
% end

sigma_d = [0.1 0.2 1]; % Gaussian decay values
num_clusters = 4; % # of clusters determined in advance for this case

% initialize data holders
A = cell(1,length(sigma_d)); D = A; L = A; img_norm = A; % matrices
eigVc = A; eigVl = A; eigVc_new = A; % eigenvectors & eigenvalues
%
% delta_l = zeros(1,size(data,1)); % eigenvalues delta to calculate eigengap
ideal_num_clusters = A; % eigengap
ind_idx = A; ind_idx1 = A; % indexing variables for clustering

for ii = 1:1:length(sigma_d)
    for i = 1:1:size(data,1)
        for j = i:1:size(data,1)
            % code
        end
    end
end
```

15
% not practical to plot entire graph; not very distinguishable
% figure;
% hold on
% plot(data(i,:),data(j,:));

% euclidean distance (2-norm)
dist = norm(data(i,:)-data(j,:),2);

if(i == j)
    A{ii}(i,j) = 0;
else
    A{ii}(i,j) = exp((-dist)/(2*(sigma_d(ii)^2)));
end
clear dist;
end
end
% hold off

% since distance metric is symmetric, populate the rest of A matrix
clear i j
for i = 1:1:size(data,1)
    for j = i:1:size(data,1)
        A{ii}(j,i) = A{ii}(i,j);
    end
end
% normalize affinity matrix for plotting
img_norm{ii} = (A{ii}-min(min(A{ii}))) / (max(max(A{ii})));
end
clear i ii j jj

% compute eigenvalues and eigenvectors of distance affinity matrices
for i = 1:1:length(A)
    [V,D] = eig(A{i});
    eigVc{i} = V; eigVl{i} = D;
    clear V D ideal_clusters;
end

clear i;
% compute Degree matrices using affinity matrices
for j = 1:1:length(A)
    for i = 1:1:size(A{j},1)
        D{j}(i,i) = sum(A{j}(i,:));
    end
end
clear j i; f1 = figure;
% compute Laplacian matrices using Degree and affinity matrices
for j = 1:1:length(A) % A is sometimes called W
    L{j} = D{j} - A{j};

    % plots eigenvalues of the affinity matrix
d_diag = eig(A{j}); cnt = 1;
    subplot(2,3,j+3)
    plot(1:length(data),d_diag,’.’)
    title(['EigVl of affinity matrix ’ ...
        ‘(\sigma_d = ’ num2str(sigma_d(j)) ‘)’])
    xlabel(‘index’); ylabel(‘value’);
    for kkk = length(d_diag):-1:1
        if(kkk == 1)
            break;
        else
            delta_l(cnt) = abs(d_diag(kkk)-d_diag(kkk-1)); % eigengap
            cnt = cnt + 1;
        end
    end
end
[val ideal_clusters] = max(delta_l);
% for this case, sigma_d = 0.1 does not fare well with this metric
if(j == 1)
    ideal_num_clusters{j} = NaN;
else
    ideal_num_clusters{j} = ideal_clusters; % ideal # of clusters
end
% plots affinity matrices and their corresponding eigenvalues
subplot(2,3,j)
colormap(gray(255))
% plots image based on the negative image of normalized affinity matrix
image(ceil((1-img_norm{j})*255))
title(['Negative of affinity matrix ’ ...
        ‘(\sigma_d = ’ num2str(sigma_d(j)) ‘)’])
end
% suptitle(['Distance affinity matrices and their ’ ...
% ’eigenvalues for \sigma_d = 0.1,0.2,1.0’])
clear j ii jj kkk
% data segmentation

% eigenvalue plots
for ii = 1:1:length(sigma_d)
    % need to keep count of the points that
    % have already have been clustered
    ind_idx{ii} = zeros(size(data,1),1);
ind_idx1{ii} = zeros(size(data,1),1);
f2 = figure;
for jj = 1:1:num_clusters
    subplot(2,2,jj);
    % zero out components that have already been clustered
    for kk = 1:1:length(sigma_d)
        eigVc_new{ii} = eigVc{ii};
        eigVc_new{ii}(find(ind_idx{ii}),end+1-jj) = 0;
    end
    % threshold set in advance
    idx = find( abs(eigVc_new{ii}(:,end+1-jj) ) > 0.015);
    % set the ones we will cluster as clustered
    ind_idx{ii}(idx) = 1;  % 1 means clustered
    ind_idx1{ii}(idx) = jj;
    plot(idx,eigVc_new{ii}(idx,end+1-jj),'ro');
    hold on
    plot(eigVc{ii}(:,end+1-jj),'.')
    xlabel('index'); ylabel('value');
    axis([0 length(data) -1 1])
    hold off
    if(jj==1)
        title(['eigVc for 1st largest eigVl (\sigma_d = ' ...
              num2str(sigma_d(ii)) ')']);
    elseif(jj==2)
        title(['eigVc for 2nd largest eigVl (\sigma_d = ' ...
              num2str(sigma_d(ii)) ')']);
    elseif(jj==3)
        title(['eigVc for 3rd largest eigVl (\sigma_d = ' ...
              num2str(sigma_d(ii)) ')']);
    elseif(jj==4)
        title(['eigVc for 4th largest eigVl (\sigma_d = ' ...
              num2str(sigma_d(ii)) ')']);
    end
end
print_eps(['..\eigvc_' num2str(ii) '_' num2str(DATA_PERM) '.eps'],f2)
end

f3 = figure;
cluster_color = 'rgby';
for ii = 1:1:length(sigma_d)
    subplot(2,2,ii)
    % plot all points
    plot(data(:,1),data(:,2),'.','k.' ,...
'MarkerFaceColor','k','MarkerSize',6); hold on;

for jj=1:1:num_clusters
    id = find(ind_idx1{ii}==jj);
    % Plot points with color depending on cluster.
    %
    % Some points might not be identified depending on the fixed
    % threshold. The algorithm should proceed until all points are
    % identified. However, since we know the number of clusters in
    % advance for this simple example, we use it as prior knowledge.
    plot(data(id,1),data(id,2),[cluster_color(jj),'o'], ...
    'MarkerFaceColor',cluster_color(jj),'MarkerSize',5);
    title([num2str(num_clusters) ' clusters; (\sigma_d = ' ...
    num2str(sigma_d(ii)) ')']);
end

end
hold off;
% suptitle('Clustering Results')
print_eps('..\..esults.eps',f3) % results are same for both

% this is the framework for graph-cut bi-partitioning
[vv1,dd1] = eig(A{2}); % find eigenvalues & eigenvectors
ddd1 = diag(dd1);

vv2 = vv1(:,2); % looks at the 2nd eigenvector (Fiedler vector)
plot(1:length(data),vv2,'.'); % can see bi-partitions

% this is the framework for setting up normalized cut bi-partitioning
D12 = diag(1./sqrt(sum(A{2})))
[vv,dd] = eig(D12*L{2}*D12);
%[vv,dd] = eigs(A{2},num_clusters);

v2 = vv(:,2); % looks at the 2nd eigenvector (Fiedler vector)
plot(1:length(data),v2,'.'); % can see bi-partitions
median(v2) % one choice of splitting point, other is zero or Ncut value
A.2 EM method

%% EM_clustering.m
% Implementation of EM algorithm (for segmentation of 2D data points)

% rand('state',0) % uncomment for deterministic results
g = 4; % number of classes
USE_KMEANS = 0; % if true, uses MATLAB's kmeans to initialize EM algorithm

% loads same data set as used in graph-theoretical clustering approach
load data % size is N x 2 -- x and y coordinate are in separate columns
[N, xy] = size(data); % number of data points to cluster

% As an initialization, we start with an indicator variable, which
% assigns random numbers chosen uniformly (indicating cluster membership)
% to all the data points
ind_var=unidrnd(g,N,1);

% the Z matrix contains a 1 in each column if the data point is in a
% cluster, and zero otherwise using the indicator variable ind_var above
% Note: Z later becomes the matrix of posterior probabilities
% as the input to the M step of the EM algorithm
Z = zeros(N,g);
for k = 1:N
    Z(k,ind_var(k)) = 1;
end
% Now that we have our initial hypotheses (Z), we can start
% iterating between the maximization and expectation steps.

% initialize a starting counter at 2, because we need to
% account for the bogus values of the likelihood function so
% that the while loops proceeds accordingly until convergence
cnt = 2;
ll(1)= 999999; ll(2)= 999998;
while abs(ll(cnt)-ll(cnt-1)) > 10^(-5); % decent enough margin
    % throughout we are assuming Gaussian mixture models, so we calculate
    % the means and covariances of the Normal distribution along w/ prior
    % and posterior probabilities
    cnt = cnt + 1;
    iter = cnt - 2; % iteration number
    disp(['Running EM: iteration ' num2str(iter)]);

    % nj is an estimate of the number of points
% assigned randomly to the j_th cluster below
nj = sum(Z,1);

% The prior probabilities are established here, otherwise known as the
% estimate of each element to belong to the j_th cluster below
priors=nj./N;

% Next, compute a matrix of means, assuming that
% the k_th data point belongs to the j_th cluster
sum_data_cluster = data'*Z; % sum of x and y values for each cluster
diag_nj = diag(nj); % computes estimate of initial covariance matrix
diagI_nj = inv(diag_nj); % computes initial estimate of inverse cov mtx

% if desired, we can initialize the estimates for the means with
% the k-means algorithm, which speeds up convergence and increases
% the likelihood of converging to global maximum vs a local maximum
%
% Note: The results in the paper were not obtained by using k-means.
if(USE_KMEANS)
    if(iter == 1)
        [cidx, means] = kmeans(data,g); % only use means
        means = means.'; % need a transpose to match the format below
    else
        % estimate of means for x and y coordinates per cluster
        means = sum_data_cluster*diagI_nj;
    end
else
    means = sum_data_cluster*diagI_nj; % same as above
end

% holder for covariance matrix
cov_mtx = cell(1,g); % pre-allocate for speed
meansT = means.'; % used frequently
for j = 1:1:g
    XX = data - ones(N,1)*meansT(j,:); % data - estimated means
    XXT = XX'; % faster than doing transpose every time later

    % initialize a holder for the sum of the covariance matrices
    sum_cov = zeros(xy,xy); % because of x and y components

    for k = 1:1:N
        % create un-normalized covariance matrix by
        % summing all x and y point coordinates
        sum_cov = sum_cov + [Z(k,j); Z(k,j)].*XXT(:,k)*XX(k,:);
    end
    cov_mtx{j}=sum_cov./nj(j); % covariance matrix
% This next loop computes f_x(x_k | theta_hat) e.g.,
% the normal PDF given the estimated means and covariances
% as well as the likelihood function
f_x = zeros(N,g); l = zeros(N,g);
for j = 1:1:g
    for k = 1:1:N
        XX = data(k,:) - meansT(j,:);  % x_k - estimated mean
        XXT = XX';  % transpose of above
        f_x(k,j) = ((1/(2*pi)) * (det(cov_mtx{j}))^(-1/2)) ...
            * exp(-(1/2)*(XX*(cov_mtx{j}^-1))*XXT);
% likelihood function for each data point and class
        l(k,j)=priors(j)*f_x(k,j);
    end
end

% next, we compute the log-likelihood function for each iteration
ll(cnt)=sum(log(sum(l,2)));
% calculate transposes used frequently to speed up calculations
f_xT = f_x.';  % used in loop below
priorsT = priors.';  % used in loop below

% To finish up, we do the E step, which calculates the posterior
% probabilities and updates the Z matrix to be used in next iteration
for j = 1:1:g
    for k = 1:1:N
        Z(k,j) = (priors(j)*f_x(k,j)) / sum(priorsT.*f_xT(:,k));
    end
end
% z{iter} = Z;  % saves posterior probabilities
% MEANS{iter} = means;  % saves estimated means
% COVS{iter} = cov_mtx;  % saves estimated covariance matrix
end  % end of EM algorithm
sum(Z)  % sum of priors for each class to look at algorithm results

disp('Press any key to continue for plots'); pause;

% nice + novel way to plot EM soft decisions!
% (albeit it runs slower than the algorithm)
%
% ensure the posterior probabilities vector Z does not have any zero
% entries by adding eps, otherwise the colors will not be consistent
Z = Z + eps;
a = figure; hold on;
% use red, green, blue, yellow colors
colormap([[1 0 0; 0 1 0; 0 0 1; 1 1 0]])
for k = 1:1:N
    disp(['Plotting pie graph #' num2str(k) ' out of ' num2str(N)])
    h = pie(Z(k,:),{'','','',''}); % don't want any labels
    movepieto(h, data(k,1), data(k,2)); % scales/moves pie chart to x,y
    axis equal; % scales axis
end
for j = 1:1:g
    draw_ellipse(means(:,j),cov_mtx{j},j); % draws ellipse using means/cov
end
title('EM clustering results','fontsize',18);
xlabel('x coordinate value','fontsize',18);
ylabel('y coordinate value','fontsize',18);
set(gca,'fontsize',16); hold off;

% plots log-likelihood function
b = figure; % plots the first real value of ll -- ll(3) until last iter
plot((1:iter-2)',ll(3:iter),'.'); set(gca,'fontsize',16);
title('Log-likelihood function','fontsize',18)
xlabel('iteration','fontsize',18);
ylabel('log-likelihood value','fontsize',18);
axis([1 iter ll(3)-1 ll(iter)+1])
keyboard % stop before saving data plots
saveas(a,'EM1_I','eps'); saveas(b,'EM1_L','eps'); % saves data
%
%% draw_ellipse.m
% function h = draw_ellipse(x, c, iter, outline_color, fill_color)
% DRAW_ELLIPSE(x, c, outline_color, fill_color)
% Draws ellipses at centers x with covariance matrix c.
% x is a matrix of columns. c is a positive definite matrix.
% outline_color and fill_color are optional.
%
% helper function taken from Tom Minka’s Lightspeed Matlab toolbox v2.2
% and modified to plot rgby colors; available from:
% http://research.microsoft.com/en-us/um/people/minka/software/lightspeed
%
n = 40; % resolution
radians = (0:(2*pi)/(n-1):2*pi);
unitC = [sin(radians); cos(radians)];
r = chol(c)';
if nargin < 4
    outline_color = 'rgby';
end

h = []; for i = 1:1:size(x,2)
    y = r*unitC + repmat(x(:, i), 1, n);
    if nargin < 5
        h = [h line(y(1,:), y(2,:), 'Color', outline_color(iter))];
    else
        h = [h fill(y(1,:), y(2,:), fill_color,
            'EdgeColor', outline_color(iter))];
    end
end
% return
%
%%
% function piehandles = movepieto(piehandles, newx, newy)
% moves and scales pie charts to x,y coordinates
% used for EM algorithm plotting purposes to represent
% points with soft decisions, rather than hard decisions
%
% Author: Walter Robertson; available from:
% http://www.mathworks.com/matlabcentral/newsreader/view_original/601933
%
% assume pairs, patch first then text
for K = 1:2:length(piehandles)
    set(piehandles(K), 'Vertices', ... 
        bsxfun(@plus, get(piehandles(K), 'Vertices')*0.015, [newx newy]));
    set(piehandles(K+1), 'Position', ...
        get(piehandles(K+1),'Position')*0.015 + [newx, newy, 0]);
end

%end