```
function [label, model, llh] = emgm(X, init)
% Perform EM algorithm for fitting the Gaussian mixture model.
% X: d x n data matrix
% init: k (1 x 1) or label (1 x n, 1<=label(i)<=k) or center (d x k)
% Written by Michael Chen (sth4nth@gmail.com).
%% initialization
fprintf('EM for Gaussian mixture: running ... \n');
R = initialization(X,init);
[~,label(1,:)] = max(R,[],2);
R = R(:,unique(label));

tol = 1e-10;
maxiter = 500;
llh = -inf(1,maxiter);
converged = false;
t = 1;
while ~converged && t < maxiter                    // until the model converges or maximum iteration happens
t = t+1;
model = maximization(X,R);                    // run M step
[R, llh(t)] = expectation(X,model);                    // run E Step for each model

[~,label(:)] = max(R,[],2);
u = unique(label); % non-empty components
if size(R,2) ~= size(u,2)
R = R(:,u); % remove empty components
```

```
else
converged = llh(t)-llh(t-1) < tol*abs(llh(t));          // setting converged if change in objective is less than tolerance
end
figure(gcf); clf;                                        // plot the models
spread(X,label);
muA = model.mu;
SigmaA = model.Sigma;
wA = model.weight;
k = size(muA,2);
% figure(12); clf;
% for i=1:k
% mu1 =muA(i,:)
% Sigma1=SigmaA(i,:)
% w1=wA(i)
% xx= mvnrnd(mu1, Sigma1, 1000);
% yy= mvnpdf(xx,mu1,Sigma1);
% plot3(xx(:,1), xx(:,2), yy, '.b'); hold on;
% end

pause;



end
llh = llh(2:t);
```

```
if converged
fprintf('Converged in %d steps.\n',t-1);
else
fprintf('Not converged in %d steps.\n',maxiter);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function R = initialization(X, init)
[d,n] = size(X);
if isstruct(init) % initialize with a model
R = expectation(X,init);                    // Run E step
elseif length(init) == 1 % random initialization
k = init;
idx = randsample(n,k);                      // Generating random values for Zik
m = X(:,idx);
[~,label] = max(bsxfun(@minus,m'*X,dot(m,m,1)'/2),[],1);
[u,~,label] = unique(label);
while k ~= length(u)
idx = randsample(n,k);                      // random values for Zim
m = X(:,idx);
[~,label] = max(bsxfun(@minus,m'*X,dot(m,m,1)'/2),[],1);
[u,~,label] = unique(label);
end
R = full(sparse(1:n,label,1,n,k,n));
elseif size(init,1) == 1 && size(init,2) == n % initialize with labels
label = init;
```

```
k = max(label);
R = full(sparse(1:n,label,1,n,k,n));
elseif size(init,1) == d %initialize with only centers
k = size(init,2);
m = init;
[~,label] = max(bsxfun(@minus,m'*X,dot(m,m,1)'/2),[],1);
R = full(sparse(1:n,label,1,n,k,n));
else
error('ERROR: init is not valid.');
end

function [R, llh] = expectation(X, model)          // E Step
mu = model.mu;                                      // Geting mean of model
Sigma = model.Sigma;                                // Getting Covariance of model
w = model.weight;                                   // Getting weight of model

n = size(X,2);                                      // Size of data
k = size(mu,2);                                     // number of models
logRho = zeros(n,k);                                // initializing gaussian probabilities for all data points

for i = 1:k                                         // for all model
logRho(:,i) = loggausspdf(X,mu(:,i),Sigma(:,:,i));  // calculating gaussian probability for each point
end
logRho = bsxfun(@plus,logRho,log(w));               // adding log(probability for model)
T = logsumexp(logRho,2);                            // calculating the sum
```

```
llh = sum(T)/n; % loglikelihood          //
logR = bsxfun(@minus,logRho,T);          // substracting the normalization part
R = exp(logR);                           // taking the exponent


function model = maximization(X, R)      // M step
[d,n] = size(X);                         // Getting dimension and size of data
k = size(R,2);                           // Number of models

nk = sum(R,1);                           // sum of <Zim>
w = nk/n;
mu = bsxfun(@times, X*R, 1./nk);         // Recalculating mean

Sigma = zeros(d,d,k);                    // initializing sigma
sqrtR = sqrt(R);
for i = 1:k                              // for all the model
Xo = bsxfun(@minus,X,mu(:,i));           // (X - mean)
Xo = bsxfun(@times,Xo,sqrtR(:,i)');
Sigma(:,:,i) = Xo*Xo'/nk(i);             // (X-mean) * (X - mean)Tr
Sigma(:,:,i) = Sigma(:,:,i)+eye(d)*(1e-6); % add a prior for numerical stability
end

model.mu = mu;                           // setting new mean
model.Sigma = Sigma;                     // setting new covariance
model.weight = w;                        // setting new weights
```

```
function y = loggausspdf(X, mu, Sigma)          // Function to calculate log gaussian function
d = size(X,1);                                  // Dimension of data
X = bsxfun(@minus,X,mu);                        // Calculating (X - mean)
[U,p]= chol(Sigma);                             //
if p ~= 0
error('ERROR: Sigma is not PD.');
end
Q = U'\X;
q = dot(Q,Q,1); % quadratic term (M distance)
c = d*log(2*pi)+2*sum(log(diag(U))); % normalization constant  //
y = -(c+q)/2;
```