```
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
    t = t+1;
    model = maximization(X,R);
    [R, llh(t)] = expectation(X,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));
    end
    figure(gcf); clf;
    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);
elseif length(init) == 1  % random initialization
    k = init;
    idx = randsample(n,k);
    m = X(:,idx);
    [~,label] = max(bsxfun(@minus,m'*X,dot(m,m,1)'/2),[],1);
    [u,~,label] = unique(label);
```

While the function has not converged and the max number of iterations has not been reached, iteratively run the expectation-maximization steps

If init is a model, set it as the initial model
If init is a constant, initialize the model for init Gaussians using the given X data

```
    while k ~= length(u)
        idx = randsample(n,k);
        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)
mu = model.mu;
Sigma = model.Sigma;
w = model.weight;

n = size(X,2);
k = size(mu,2);
logRho = zeros(n,k);

for i = 1:k
    logRho(:,i) = loggausspdf(X,mu(:,i),Sigma(:,:,i));
end
logRho = bsxfun(@plus,logRho,log(w));
T = logsumexp(logRho,2);
llh = sum(T)/n; % loglikelihood
logR = bsxfun(@minus,logRho,T);
R = exp(logR);


function model = maximization(X, R)
[d,n] = size(X);
k = size(R,2);

nk = sum(R,1);
w = nk/n;
mu = bsxfun(@times, X*R, 1./nk);

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

model.mu = mu;
model.Sigma = Sigma;
model.weight = w;

function y = loggausspdf(X, mu, Sigma)
d = size(X,1);
X = bsxfun(@minus,X,mu);
[U,p]= chol(Sigma);
if p ~= 0
    error('ERROR: Sigma is not PD.');
```

If init is a label, use it as the label for the model

If init is a set of centers, use that as the initial set of centers

For each Gaussian, get the log probability that each row of data was generated by that Gaussian

Update the model means, covariances, and weights to maximize the expectation (this is the step that updates the model)

Get the log probability that X was generated by a multivariate Gaussian with mean mu and covariance Sigma

```
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;
```