```
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;            converge threshold 10^-10
maxiter = 500;          maximum 500 steps
llh = -inf(1,maxiter);      initialize loglikelihoods with -inf's
converged = false;      init converged
t = 1;                  init t
while ~converged && t < maxiter        loop while not converged and not reach maxiter
    t = t+1;
    model = maximization(X,R);          m-step
    [R, llh(t)] = expectation(X,model);        e-step

    [~,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));  converged if difference between current and previous likelihood less than threshold.
    end
end
llh = llh(2:t);     llh value starts from llh(2), first value not set in loop.
if converged          converged
    fprintf('Converged in %d steps.\n',t-1);
else                  not converged in maxiter steps
    fprintf('Not converged in %d steps.\n',maxiter);
end

function R = initialization(X, init)
[d,n] = size(X);      get X (d×n data) dimensions
if isstruct(init)  % initialize with a model
    R  = expectation(X,init);     return expectation for X from given model
elseif length(init) == 1  % random initialization
    k = init;
    idx = randsample(n,k);     randomly pick sample k indexes from 1 to n
    m = X(:,idx);             get sample data
    [~,label] = max(bsxfun(@minus,m'*X,dot(m,m,1)'/2),[],1);
    [u,~,label] = unique(label);
    while k ~= length(u)
        idx = randsample(n,k);      pick random samples
        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));   make a full (1xn) sparse matrix which is a random membership matrix
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     otherwise, init is not valid.
```

```
    error('ERROR: init is not valid.');
end

function [R, llh] = expectation(X, model)
mu = model.mu;
Sigma = model.Sigma;        } get parameters from model
w = model.weight;

n = size(X,2);              get size parameters
k = size(mu,2);
logRho = zeros(n,k);        initialize log gaussian pdf values

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

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

nk = sum(R,1);              number of points for each gaussian
w = nk/n;                   weights for each gaussian
mu = bsxfun(@times, X*R, 1./nk);    means for each gaussian

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

model.mu = mu;
model.Sigma = Sigma;        } update parameters.
model.weight = w;

function y = loggausspdf(X, mu, Sigma)
d = size(X,1);
X = bsxfun(@minus,X,mu);    x-μ
[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;               return pdf value.
```