Probabilistic Algorithms for Aerospace Autonomy
ASEN 6519 - Homework 2
Maximum Likelihood Parameter Estimation

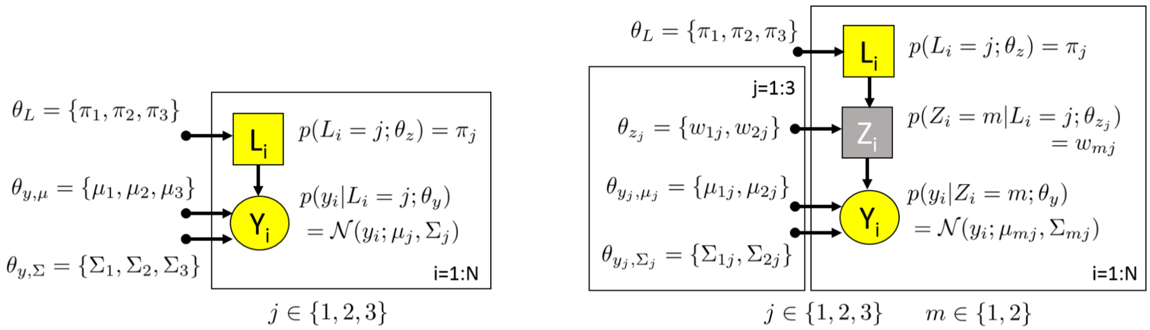
Carl Stahoviak
Carl Mueller

March 13, 2019

Contents

Problem 1 - Generative Classification

Estimate the unknown model parameters for model (a) via maximum likelihood and model (b) using ML or EM if necessary.



(a) (b)

Figure 1: CLL is known for model (a) whereas only the ICLL is known for (b).

Maximum Likelihood for Model (a)

Below outlines the justification for the chosen implementation for ML the model:

$$L(y_{1:N}; \theta_j, \theta_y) = \prod_{i=1}^{N} p(L_i = j; \theta_L) p(y_i | L_i = j; \theta_y)$$

$$l(y_{1:N}; \theta_j, \theta_y) = log[\prod_{i=1}^{N} p(L_i = j; \theta_L) p(y_i | L_i = j; \theta_y)]$$

$$l(y_{1:N}; \theta_j, \theta_y) = log[p(L_i = j; \theta_L)] + \sum_{i=1}^{N} log[p(y_i | L_i = j; \theta_y)]$$

$$l(y_{1:N}; \theta_j, \theta_y) = log[\prod_{j=1}^{3} p(\pi_j^{I(L_i=j)}] + \sum_{i=1}^{N} log[\prod_{j=1}^{3} \mathcal{N}(y_i; \mu_j, \Sigma_j)^{I(L_i=j)}]$$

$$l(y_{1:N}; \theta_j, \theta_y) = \sum_{j=1}^{3} I(L_i = j) log[\pi_j] + \sum_{i=1}^{N} I(L_i = j) \sum_{j=1}^{3} log[\mathcal{N}(y_i; \mu_j, \Sigma_j)]$$

$$l(y_{1:N}; \theta_j, \theta_y) = \sum_{j=1}^{3} I(L_i = j) log[\pi_j] + \sum_{i=1}^{N} I(L_i = j) \sum_{j=1}^{3} log[\frac{-\frac{1}{2}(y - \mu_j)^T \Sigma^{-1} (y - \mu_j)}{\sqrt{(2\pi)^d |\Sigma_j|}}]$$

When maximizing the log-likelihood for any of the individual parameters θ_j, the indicator function $I(L_i = j)$ filters out all the data such that we can effectively maximize μ_j and Σ_j using data labeled as class j. Maximizing the partial derivative w.r.t π_j is unnecessary as π_j is simply constant.

Maximizing the partial derivative w.r.t μ_j sifts all data out that is not labeled as j, leaving:

$$\frac{\partial I(L_i = j) log[p(L_i = j; \theta_L) j] + \sum_{i=1}^{N} I(L_i = j) \sum_{j=1}^{3} log[\frac{-\frac{1}{2}(y-\mu_j)^T \Sigma^{-1}(y-\mu_j)}{\sqrt{(2\pi)^d |\Sigma_j|}}]}{\partial \mu_j} = 0$$

$$\hat{\mu_j} = \frac{1}{N_j} \sum_j y_j$$

Maximizing the partial derivative w.r.t Σ and setting to zero:

$$\frac{\partial I(L_i = j) log[p(L_i = j; \theta_L) j] + \sum_{i=1}^{N} I(L_i = j) \sum_{j=1}^{3} log[\frac{-\frac{1}{2}(y-\mu_j)^T \Sigma^{-1}(y-\mu_j)}{\sqrt{(2\pi)^d |\Sigma_j|}}]}{\partial \Sigma_j} = 0$$

$$\hat{\Sigma} = \frac{1}{N_j} \sum_j (y_j - \mu_j)(y_j - \mu_j)^T$$

Estimated Parameters for Model (a):

$\mu_1 = \begin{bmatrix} -1.3650 \\ -2.6720 \end{bmatrix} \Sigma_1 = \begin{bmatrix} 2.5780 & 0.5999 \\ 0.5999 & 2.2415 \end{bmatrix}, \mu_2 = \begin{bmatrix} -1.6550 \\ 2.5280 \end{bmatrix} \Sigma_2 = \begin{bmatrix} 1.9302 & -0.8914 \\ -0.8914 & 2.6287 \end{bmatrix}, \mu_3 = \begin{bmatrix} 4.0812 \\ 4.3114 \end{bmatrix} \Sigma_3 = \begin{bmatrix} 7.7501 & 3.7525 \\ 3.7525 & 10.3827 \end{bmatrix}$

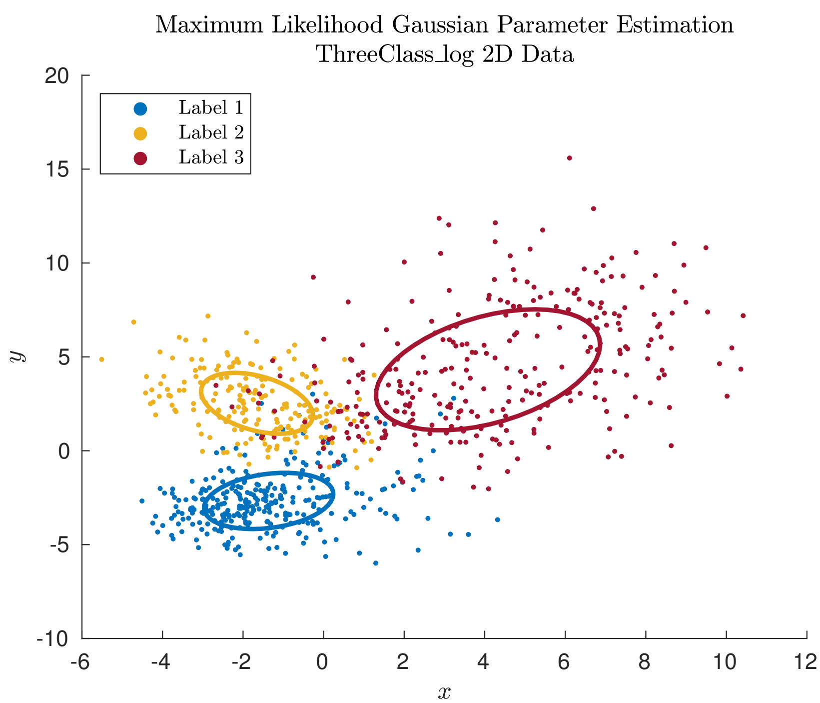


Figure 2: Estimated distribution using ML for model (a).

Maximum Likelihood for Model (b)

Since we do not know z_i, we must marginalized over z to obtain the marginal log-likelihood of $y_{1:N}$. The resulting log-likelihood is therefore the incomplete log-likelihood. We cannot efficiently optimize this function directly as it is a sum of a log of a sum where the $p(y_i|z_i = m; \theta_y)$ distribution is coupled to the unknown distribution of the R.V. Z_i. Therefore we have opted to used the expected complete log-likelihood to perform Expectation Maximization.

$$L(y_{1:N};\theta_j,\theta_y) = p(y_{1:N}|L_i=j;\theta_y) = \prod_{i=1}^{N}\sum_{m=1}^{2} p(L_i=j;\theta_L)p(z_i=m|L_i=j;\theta_y)p(y_i|z_i=m;\theta_y)$$

$$l(y_{1:N};\theta_j,\theta_y) = log[\prod_{i=1}^{N}\sum_{m=1}^{2} p(L_i=j;\theta_L)p(z_i=m|L_i=j;\theta_y)p(y_i|z_i=m;\theta_y)]$$

$$l(y_{1:N};\theta_j,\theta_y) = log[p(L_i=j;\theta_L)] + \sum_{i=1}^{N} log[\sum_{m=1}^{2} p(L_i=j;\theta_L)p(z_i=m|L_i=j;\theta_y)p(y_i|z_i=m;\theta_y)]$$

For expectation maximization, we infer $p(z_i = m|y_i;\theta_z,\theta_y)$ via the following:

$$\begin{aligned}
p(z_i=m|y_i;\theta_z,\theta_y) &= \frac{p(z_i=m;\theta_z,\theta_y)p(y_i|z_i;\theta_z,\theta_y)}{\sum_{n=1}^{2} p(z_i=n;\theta_z)p(z_i=n|y_i;\theta_z,\theta_y)} \\
&= \frac{w_{mj}p(y_i|z_i=m;\theta_y)}{\sum_{n=1}^{2} w_{nj}p(y_i|z_i=n)} \\
&= \frac{w_{mj}\mathcal{N}(y_i;\mu_m,\Sigma_m)}{\sum_{n=1}^{2} w_{nj}\mathcal{N}(y_i;\mu_n,\Sigma_n)} \\
&= \gamma(z_{im})
\end{aligned}$$

γ represents the responsibility, or the probability that data y_i within the label set j belongs to the cluster / component z_m. The calculation of γ is used as the expectation step for the EM algorithm. It is then used to update our parameters for the j^th class via the following equations:

$$\begin{aligned}
\hat{\mu}_{mj}^{\text{new}} &= \frac{1}{N_m}\sum_{i=1}^{N}\gamma(z_{im})y_i \\
\hat{\Sigma}_{mj}^{\text{new}} &= \frac{1}{N_m}\sum_{i=1}^{N}\gamma(z_{im})(y_i-\hat{\mu}_m j)(y_i-\hat{\mu}_{mj})^T \\
\hat{w}_{mj}^{\text{new}} &= \frac{\sum_{i=1}^{N}\gamma(z_{im})}{N} = \frac{N_m}{N}
\end{aligned}$$

Note: We can update the j^{th} parameters independently using the training data associated with class j because the ICLL functions are associated with the same one-hot encoding that sifts out all data not associated with the parameters of j.

Estimated Parameters for Model (b):

Parameters for Label 1

$\mu_{11} = \begin{bmatrix}-1.8687\\-2.9637\end{bmatrix}$, $\Sigma_{11} = \begin{bmatrix}1.3125 & 0.1612\\0.1612 & 1.1954\end{bmatrix}$, $w_2 = 0.8143$ $\mu_{21} = \begin{bmatrix}0.8447\\-1.3926\end{bmatrix}$, $\Sigma_{21} = \begin{bmatrix}2.1391 & -0.9447\\-0.9447 & 4.8213\end{bmatrix}$, $w_2 = 0.1857$

Parameters for Label 2

$\mu_{12} = \begin{bmatrix}-1.5592\\1.9603\end{bmatrix}$, $\Sigma_{12} = \begin{bmatrix}1.9298 & -0.7605\\-0.7605 & 1.6368\end{bmatrix}$, $w_2 = 0.7698$ $\mu_{22} = \begin{bmatrix}-1.9752\\4.4272\end{bmatrix}$, $\Sigma_{22} = \begin{bmatrix}1.7986 & -0.5394\\-0.5394 & 1.2613\end{bmatrix}$, $w_2 = 0.2302$

Parameters for Label 3

$\mu_{13} = \begin{bmatrix} 5.8549 \\ 6.3759 \end{bmatrix}$, $\Sigma_{13} = \begin{bmatrix} 4.5488 & -0.3473 \\ -0.3473 & 8.1516 \end{bmatrix}$, $w_2 = 0.5277$ $\quad$ $\mu_{23} = \begin{bmatrix} 2.0991 \\ 2.0044 \end{bmatrix}$, $\Sigma_{23} = \begin{bmatrix} 3.8830 & -0.3309 \\ -0.3309 & 2.7911 \end{bmatrix}$, $w_2 = 0.4723$

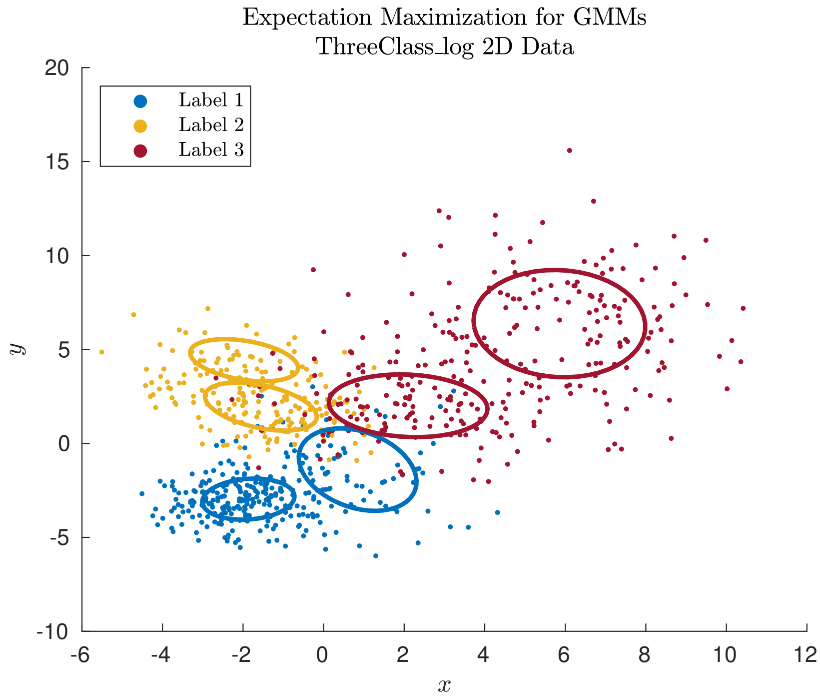


Figure 3: Estimated distribution using ML for model (a).

Problem 2 - Baum-Welch for Supervisory Operator HMM

Parameter Comparisons:

True	.25	.25	.25	.25
Estimated	.2078	.0453	.3376	.4093

10 Sequences

True	.25	.25	.25	.25
Estimated	.5314	.0762	.3210	.0714

50 Sequences

True	.25	.25	.25	.25
Estimated	0.3528	0.1352	.3090	.2029

100 Sequences

Table 1: Initial state probability $p(x_0)$ estimates for 10, 50, and 100 observation sequences with 50 iterations.

True	.0200	.0190	0	.6660
	0	.0250	.5170	0
	.1630	.7690	.4660	0
	.8170	.1870	.0170	.3340
Estimated	.0897	.0252	0	.7669
	0	.0072	.4979	0
	.1586	.7843	.4832	0
	.7517	.1834	.0189	.2331

10 Sequences

True	.0200	.0190	0	.6660
	0	.0250	.5170	0
	.1630	.7690	.4660	0
	.8170	.1870	.0170	.3340
Estimated	.0546	.0235	0	.6971
	0	.0436	.5747	0
	.1672	.7588	.4120	0
	.7782	.1741	.0133	.3029

50 Sequences

True	.0200	.0190	0	.6660
	0	.0250	.5170	0
	.1630	.7690	.4660	0
	.8170	.1870	.0170	.3340
Estimated	.0686	.0317	0	.6801
	0	.0637	.5855	0
	.1681	.7336	.4033	0
	.7633	.1709	.0112	.3199

100 Sequences

Table 2: State transition probability $p(x_k|x_{k-1})$ estimates for 10, 50, and 100 observation sequences with 50 iterations.

True	.0338	0	0	.3273
	.0934	0	0	.0949
	.1356	0	0	.0311
	.1031	0	0	.0125
	.1350	0	0	.0113
	.0289	0	0	.3354
	.0968	0	0	.1094
	.1409	0	0	.0488
	.1117	0	0	.0149
	.1208	0	0	.0144
	0	.0842	.7353	0
	0	.2048	.1869	0
	0	.2774	.0195	0
	0	.3336	.0267	0
	0	0	.0316	0
Estimated	0	0	0	.3796
	.0626	0	0	.0631
	.1405	0	0	.0080
	.1499	0	0	.0000
	.1090	0	0	.0000
	.0284	0	0	.3728
	.0775	0	0	.1029
	.1673	0	0	.0610
	.1163	0	0	.0055
	.1484	0	0	.0071
	0	.0637	.7721	0
	0	.1392	.1538	0
	0	.3087	.0581	0
	0	.4884	.0000	0
	0	0	.0160	0

10 Sequences

True	.0338	0	0	.3273
	.0934	0	0	.0949
	.1356	0	0	.0311
	.1031	0	0	.0125
	.1350	0	0	.0113
	.0289	0	0	.3354
	.0968	0	0	.1094
	.1409	0	0	.0488
	.1117	0	0	.0149
	.1208	0	0	.0144
	0	.0842	.7353	0
	0	.2048	.1869	0
	0	.2774	.0195	0
	0	.3336	.0267	0
	0	0	.0316	0
Estimated	.0002	0	0	.3501
	.0978	0	0	.0912
	.1212	0	0	.0186
	.1106	0	0	.0145
	.1325	0	0	.0091
	.0452	0	0	.3232
	.0925	0	0	.1201
	.1328	0	0	.0523
	.1313	0	0	.0122
	.1359	0	0	.0087
	0	.1210	.7743	0
	0	.2015	.1533	0
	0	.3600	.0222	0
	0	.3175	.0145	0
	0	0	.0356	0

50 Sequences

True	.0338	0	0	.3273
	.0934	0	0	.0949
	.1356	0	0	.0311
	.1031	0	0	.0125
	.1350	0	0	.0113
	.0289	0	0	.3354
	.0968	0	0	.1094
	.1409	0	0	.0488
	.1117	0	0	.0149
	.1208	0	0	.0144
	0	.0842	.7353	0
	0	.2048	.1869	0
	0	.2774	.0195	0
	0	.3336	.0267	0
	0	0	.0316	0
Estimated	.0216	0	0	.3321
	.0878	0	0	.0947
	.1387	0	0	.0157
	.1180	0	0	.0111
	.1437	0	0	.0087
	.0295	0	0	.3409
	.0794	0	0	.1258
	.1338	0	0	.0524
	.1087	0	0	.0142
	.1387	0	0	.0044
	0	.1324	.7527	0
	0	.2094	.1690	0
	0	.3289	.0254	0
	0	.3292	.0170	0
	0	0	.0358	0

100 Sequences

Table 3: Emission probability $p(y_k|x_k)$ estimates for 10, 50, and 100 observation sequences with 50 iterations.

Log-likelihood Comparisons:

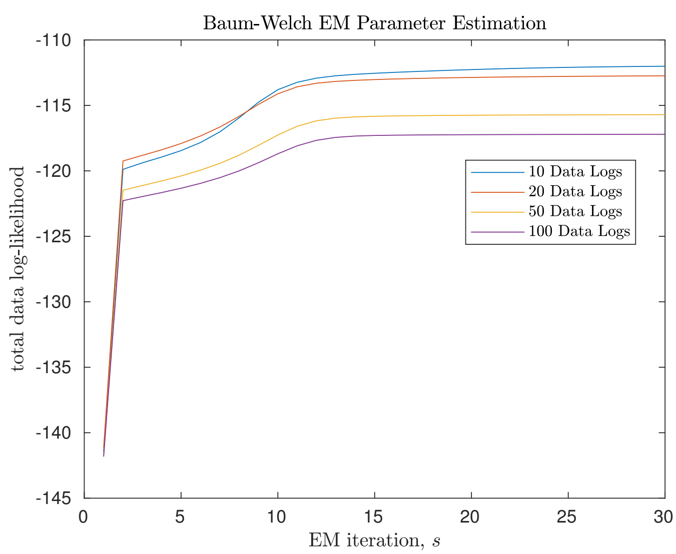


Figure 4: Data log-likelihood average per iteration for 10, 20, 50, 100 observation sequences.

We initialized our parameters using an "informed" approach. The idea is that we might not know the exact values for each entry in our initial CPT's, but we know which values ought to be zero. We used a uniform distribution on all remaining non-zero entries appropriately to maintain row or column stochasticity. However, our results show that the lower count of observation sequences sometimes have a slightly higher log-likelihood. This is unexpected and perhaps may be due to a bug or incorrect implementation.

Appendix - MATLAB Code

The following MATLAB code was used to generate the data presented above.

hw2_BaumWelch.m - main script

```
%% Header

% Filename:     hw2_BaumWelch.m
% Author:       Carl Stahoviak
% Date Created: 03/02/2019

clc;
clear;
close ALL;

%% Load Data

% load HMM CPT Parameters
load('nominal_hmm_params.mat')

%% Initialize CPT distributions for Baum-Welch

% load observation set for Baum-Welch implementation (y_obs)
% NOTE: each column of y_obs is an independent observation sequence
load('nominal_hmm_multi_logs.mat')

n = size(pxk_xkm1,1);   % number of states
T = size(y_obs,1);      % number of observations per data log
D = size(y_obs,2);      % numnber of unique data logs
p = size(pyk_xk,1);     % number of unique emmision symbols

% number of unique data logs to use in E-step (Berkeley notation)
N_logs = 10;
log_sz = [10, 20, 50, 100];

% number of times M-step will be done
N_mstep = 30;

% total data log-likelihood per EM iteration
data_ll_total = zeros(N_mstep,size(log_sz,2));

% total data log-likelihood per EM iteration
data_ll_mean = zeros(N_mstep,size(log_sz,2));

% scatter plot size
sz = 5;


%% Baum-Welch - Attempt 2

% In this attempt, the e-step (resulting in the calculation of {elnalpha,
% elnbeta, elngamma}) will be run for all D data logs, and a single m-step
% will be done at the end of this. So rather than performing D number of
% e-step/m-step iterations (which failed at the second iteration), D number
% of e-step iterations will be performed and then a single m-step will be
% done to update the CPT parameter estimates.

figure(1)
ax1 = gca;
title('Baum-Welch EM Parameter Estimation','Interpreter','latex');
xlabel('EM iteration, $s$','Interpreter','latex');
ylabel('data log-likelihood','Interpreter','latex');
hold on;

for i=1:size(log_sz,2)
```

```
    N_logs = log_sz(i);

    % initialize CPTs
    trans_prob = zeros(n,n,N_mstep+1);      % row-stochastic
    obs_prob   = zeros(p,n,N_mstep+1);      % column-stochastic (is this correct?)
    init_distr = zeros(n,N_mstep+1);        % column-stochastic

    % initialize Conditional Probability Tables (CPTs)
    type = 'informed';
    [ init_distr(:,1), trans_prob(:,:,1), obs_prob(:,:,1) ] = initCPTs( ...
        pxk_xkm1', pyk_xk, px0, type);

    % data log-likelihood (for each data log per EM iteration)
    data_ll = zeros(N_logs,N_mstep);

    for s=1:N_mstep  % number of times M-step will be done

        fprintf('EM iteration: %d\n', s);

        % init variables
        eln_alpha = zeros(n,T+1,N_logs);    % T+1 columns because alpha(0) is computed
        eln_beta  = zeros(n,T+1,N_logs);    % T+1 columns because beta(0) is computed
        eln_gamma = zeros(n,T+1,N_logs);    % log-posterior
        gamma     = zeros(n,T+1,N_logs);    % true posterior
        eln_xi    = zeros(n,n,T,N_logs);    % log of probability xi(i,j,k)
        xi        = zeros(n,n,T,N_logs);

        for d=1:N_logs  % iterate over D number of data logs
            obs_seq = y_obs(:,d);

            % do E-step for each data log
            [eln_alpha(:,:,d), eln_beta(:,:,d), eln_gamma(:,:,d), ...
                gamma(:,:,d), eln_xi(:,:,:,d), xi(:,:,:,d)] = baumwelch_Estep( ...
                init_distr(:,s), trans_prob(:,:,s), obs_prob(:,:,s), obs_seq);

            % calculate data log-likelihood for the e-step given the current
            % CPT parameter estimates
            nonNaN_idx = ¬isnan(eln_alpha(:,end,d));
            data_ll(d,s) = eln(sum(eexp(eln_alpha(nonNaN_idx,end,d))));
        end

        % do M-step given D iterations of the E-step
        [init_distr(:,s+1), trans_prob(:,:,s+1), obs_prob(:,:,s+1) ] = ...
            baumwelch_Mstep( p, eln_gamma, gamma, eln_xi, xi, y_obs);

        % plot the data log-likelihood of each data log at EM iteration s
        scatter(ax1,s*ones(N_logs,1),data_ll(:,s),sz,'filled');

        % compute the total data log-likelihood per EM iteration
        data_ll_total(s,i) = sum(data_ll(:,s));
        data_ll_mean(s,i) = mean(data_ll(:,s));
    end

end

fprintf('\nFinal (sum) data log-likelihood =\n')
disp([log_sz; data_ll_total(end,:)]);
fprintf('Final (avg) data log-likelihood =\n\n')
disp([log_sz; data_ll_mean(end,:)]);

fprintf('Estimated Initial Distribution\n')
disp(init_distr(:,end));
disp(sum(init_distr(:,end),1));

fprintf('True State Transition CPT\n')
disp(pxk_xkm1);
fprintf('%d) Estimated State Transition CPT\n', s)
```

```
disp(trans_prob(:,:,end)');
disp(1-sum(trans_prob(:,:,end)',1));

fprintf('True Emission CPT\n')
disp(pyk_xk);
fprintf('Estimated Emission CPT\n')
disp(obs_prob(:,:,end));
disp(1-sum(obs_prob(:,:,end),1));

figure(2)
for i=1:size(log_sz,2)
    plot(data_ll_mean(:,i));
    title('Baum-Welch EM Parameter Estimation','Interpreter','latex');
    xlabel('EM iteration, $s$','Interpreter','latex');
    ylabel('total data log-likelihood','Interpreter','latex');
    hold on;
    hdl = legend();
end
set(hdl,'Interpreter','latex');
```

initCPTs.m - Initialize CPT distributions

```
function [ init_distr, trans_prob, obs_prob ] = ...
    initCPTs( pxk_xkm1, pyk_xk, px0, type )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

% NOTE: in accordance with Rabiner/Mann notation, the trans_prob matrix
% must be a row-stochastic matrix

n = size(pxk_xkm1,1);   % number of states
p = size(pyk_xk,1);     % number of unique emmision symbols

if strcmp(type, 'truth')
    init_distr = px0;
    trans_prob = pxk_xkm1;          % row-stochastic
    obs_prob   = pyk_xk;

elseif strcmp(type, 'informed')
    % "informed" uniform initialization
    % initialize CPTs WITH knowledge of where zeros exist in the tables
    init_distr = (1/n)*ones(n,1);

    trans_prob = [1/3 0   1/3 1/3;
                  1/4 1/4 1/4 1/4;
                  0   1/3 1/3 1/3;
                  1/2 0   0   1/2];

    obs_prob   = [(1/10)*ones(10,1), zeros(10,2), (1/10)*ones(10,1);
                  zeros(4,1), (1/4)*ones(4,1), (1/5)*ones(4,1), zeros(4,1);
                  0,          0,               1/5,             0];

elseif strcmp(type, 'uninformed')
    % "un-informed" uniform initialization
    % initialize CPTs WITHOUT knowledge of where zeros exist in the tables
    init_distr = (1/n)*ones(n,1);
    trans_prob = (1/n)*ones(n);
    obs_prob   = (1/p)*ones(p,n);

elseif strcmp(type, 'rand')
    init_distr = rand(n,1);
    init_distr = init_distr./sum(init_distr);   % normalize

    trans_prob = rand(n);
    for i=1:n
        % normalize rows
        trans_prob(i,:) = trans_prob(i,:)./sum(trans_prob(i,:));
```

```
    end

    obs_prob = rand(p,n);
    for i=1:n
        % normalize columns
        obs_prob(:,i) = obs_prob(:,i)./sum(obs_prob(:,i));
    end

else
    error('Improperly specified initialization type')
end

end
```

baumwelch_Estep.m - Baum-Welch E-Step

```
function [ eln_alpha, eln_beta, eln_gamma, gamma, eln_xi, xi ] = ...
    baumwelch_Estep( init_distr, trans_prob, obs_prob, y_obs )

% Mann log-weighted (numerically-stable) forward-backward alg.
eln_alpha = forward_eln( init_distr, trans_prob, obs_prob, y_obs );
eln_beta = backward_eln( trans_prob, obs_prob, y_obs );

% get the log-posterior and posterior distributions (exact inference)
% [ eln_gamma, gamma ] = posterior_elnfb( eln_alpha(:,2:end), eln_beta );
[ eln_gamma, gamma ] = posterior_elnfb( eln_alpha, eln_beta );

% get log of probability xi(i,j,k)
[ eln_xi, xi ] = elnxi( eln_alpha, eln_beta, ...
    trans_prob, obs_prob, y_obs );

end
```

forward_eln.m - extended-logarithm forward pass

```
function [ eln_alpha ] = forward_eln( px0, trans_prob, obs_prob, y_obs )

% NOTE: in accordance with Rabiner/Mann notation, the trans_prob matrix
% must be a row-stochastic matrix

n = size(px0,1);
T = size(y_obs,1);
eln_alpha = zeros(n,T+1);

% initialization
for i=1:n
    eln_alpha(i,1) = eln(px0(i,1));
end

for k=2:T+1
    for j=1:n
        logalpha = NaN;
        for i=1:n
            logalpha = elnsum( logalpha, ...
                elnprod( eln_alpha(i,k-1), eln(trans_prob(i,j)) ));
        end
        eln_alpha(j,k) = elnprod( logalpha, ...
            eln(obs_prob(y_obs(k-1),j)) );
    end
end

end
```

backward_eln.m - extended-logarithm backward pass

```
function [ eln_beta ] = backward_eln( trans_prob, obs_prob, y_obs )

% NOTE: in accordance with Rabiner/Mann notation, the trans_prob matrix
% must be a row-stochastic matrix

n = size(trans_prob,1);
T = size(y_obs,1);

% initialization
eln_beta = zeros(n,T+1);

for k=T:-1:1
    for i=1:n
        logbeta = NaN;
        for j=1:n
            logbeta = elnsum( logbeta, ...
                elnprod( eln(trans_prob(i,j)), ...
                elnprod( eln(obs_prob(y_obs(k),j)), eln_beta(j,k+1) )));
        end
        eln_beta(i,k) = logbeta;
    end
end
```

elnfb_posterior.m - extended-logarithm posterior calculation

```
function [ eln_gamma, gamma ] = posterior_elnfb( eln_alpha, eln_beta )

if size(eln_alpha,2) ≠ size(eln_beta,2)
    error('eln_alpha, eln_beta size mismatch')

else
    n = size(eln_alpha,1);
    T = size(eln_alpha,2);

    % initialization
    eln_gamma = zeros(n,T);     % log-posterior
    gamma = zeros(n,T);         % true posterior

    for k=1:T
        normalizer = NaN;
        for i=1:n
            eln_gamma(i,k) = elnprod(eln_alpha(i,k),eln_beta(i,k));
            normalizer = elnsum(normalizer,eln_gamma(i,k));
        end

        for i=1:n
            eln_gamma(i,k) = elnprod(eln_gamma(i,k),-normalizer);
            gamma(i,k) = eexp(eln_gamma(i,k));
        end
        % sanity check
        fprintf('1 - sum(gamma(:,%d)) = %e\n', k, ...
        1-nansum(eexp(eln_gamma(:,k))))
    end
end
```

elnxi.m - extended-logarithm xi calculation

```
function [ eln_xi, xi ] = elnxi( eln_alpha, eln_beta, trans_prob, obs_prob, y_obs )
%ELNXI - Computes the log of xi(i,j,k)
%   xi(i,j,k) is the probability of being in state x_i and timestep k, and
%   state x_j at timestep k+1 given a model lambda and observation sequence
%   O (in Rabiner notation)

% NOTE: in accordance with Rabiner/Mann notation, the trans_prob matrix
% must be a row-stochastic matrix
```

```

n = size(trans_prob,1);
T = size(y_obs,1);

% initialization
eln_xi = zeros(n,n,T);
xi = zeros(n,n,T);

for k=1:T
    normalizer = NaN;
    for i=1:n
        for j=1:n
            eln_xi(i,j,k) = elnprod( eln_alpha(i,k), ...
                elnprod( eln(trans_prob(i,j)), ...
                elnprod( eln(obs_prob(y_obs(k),j)), eln_beta(j,k+1) )));
            normalizer = elnsum( normalizer, eln_xi(i,j,k) );
        end
    end

    for i=1:n
        for j=1:n
            eln_xi(i,j,k) = elnprod( eln_xi(i,j,k), -normalizer);
            xi(i,j,k) = eexp(eln_xi(i,j,k));
        end
    end
end

end
```

baumwelch_Mstep.m - Baum-Welch M-Step

```
function [ init_distr_hat, trans_prob_hat, obs_prob_hat ] = ...
    baumwelch_Mstep( p, eln_gamma, gamma, eln_xi, xi, y_obs )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

if (size(gamma,2) ≠ size(xi,3)+1) && (size(gamma,3) ≠ size(xi,4))
    error('gamma, xi size mismatch')

else
    % Implement equations from Berkeley paper
%     init_distr_hat = updateInitDistr( gamma );
%     trans_prob_hat = updateTransitionProb( gamma, xi);
%     obs_prob_hat   = updateEmissionProb( p, gamma, y_obs );

    % Implement extended-log versions of the parameter updates
    % from the Berkeley paper:
    % WORKING!! - Now need to update functions above to achieve similar
    % results!
    init_distr_hat = updateInitDistr_eln( eln_gamma );
    trans_prob_hat = updateTransitionProb_eln( eln_gamma, eln_xi);
    obs_prob_hat   = updateEmissionProb_eln( p, eln_gamma, y_obs );
end
```

updateInitDist_eln.m - Baum-Welch Initial Distribution Update

```
function [ init_distr_hat ] = updateInitDistr_eln( eln_gamma )

n = size(eln_gamma,1);
D = size(eln_gamma,3);

init_distr_hat = zeros(n,1);

% get initial distribution estimate
for d=1:D
    for i=1:n
```

```
        init_distr_hat(i,1) = init_distr_hat(i,1) + eexp(eln_gamma(i,1,d));
    end
end

init_distr_hat = init_distr_hat/D;

% normalize
% init_distr_hat = init_distr_hat./sum(init_distr_hat);

end
```

updateTransitionProb_eln.m - Baum-Welch Transition Probability Update

```
function [ trans_prob_hat ] = updateTransitionProb_eln( eln_gamma, eln_xi)

n = size(eln_gamma,1);
T = size(eln_gamma,2) - 1;
D = size(eln_gamma,3);

trans_prob_hat = zeros(n,n);

% get transition probability estimate
for i=1:n
    for j=1:n
        numerator = NaN;
        denominator = NaN;

        for d=1:D
            for k=2:T+1
                numerator = elnsum(numerator, eln_xi(i,j,k-1,d));
                denominator = elnsum(denominator, eln_gamma(i,k-1,d));
            end % end k
        end % end d

        trans_prob_hat(i,j) = eexp(elnprod(numerator, -denominator));
    end % end j
end % end i

end
```

updateEmissionProb_eln.m - Baum-Welch Emission Probability Update

```
function [ obs_prob_hat ] = updateEmissionProb_eln( p, eln_gamma, y_obs )
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here

n = size(eln_gamma,1);
T = size(eln_gamma,2) - 1;
D = size(eln_gamma,3);

obs_prob_hat = zeros(p,n);

% get emission probability estimate
for j=1:p
    for i=1:n
        numerator = NaN;
        denominator = NaN;

        for d=1:D
            for k=2:T+1
%                   disp(k)
                % symbol j observed at time step k in data log d
                if y_obs(k-1,d) == j
                    numerator = elnsum(numerator, eln_gamma(i,k,d));
                end
                denominator = elnsum(denominator, eln_gamma(i,k,d));
```

```
            end % end k
        end % end d

        obs_prob_hat(j,i) = eexp(elnprod(numerator, -denominator));
    end % end i
end % end j

end
```

hw2_MLE.m - Maximum Likelihood and Expectation Maximization

```
%% Header

% Filename:     hw2_MLE.m
% Author:       Carl Stahoviak
% Date Created: 03/02/2019

clc;
clear;
close ALL;

%% Load Data

% load MLE Data Log
load('ThreeClass_log.mat')
y_obs = ThreeClass_log;

%% Plot Data

colors = [0,      0.4470, 0.7410;
          0.9290, 0.6940, 0.1250;
          0.6350, 0.0780, 0.1840];

figure(1);
ax(1) = gca;
title({'Maximum Likelihood Gaussian Parameter Estimation', ...
    'ThreeClass\_log 2D Data'},'Interpreter','latex');
xlabel('$x$','Interpreter','latex');
ylabel('$y$','Interpreter','latex');
hold on;

for i=1:3
    figure(i+1);
    ax(i+1) = gca;
    title({'Expectation Maximization for GMMs', ...
        strcat('Data Label','$\;$',num2str(i))},'Interpreter','latex');
    xlabel('$x$','Interpreter','latex');
    ylabel('$y$','Interpreter','latex');
    hold on;
    pause on;
end

figure(5);
ax(5) = gca;
title({'Expectation Maximization for GMMs', ...
    'ThreeClass\_log 2D Data'},'Interpreter','latex');
xlabel('$x$','Interpreter','latex');
ylabel('$y$','Interpreter','latex');
hold on;
pause on;

figure(6);
ax(6) = gca;
title({'Expectation Maximization for GMMs', ...
    'Unknown Data Labels'},'Interpreter','latex');
xlabel('$x$','Interpreter','latex');
ylabel('$y$','Interpreter','latex');
```

```
hold on;
pause on;


%% Maximum Likelihood Estimation - Multivariate Gaussian

% For Model A, the labels {pi_j = 1,2,3} associated with each data point
% are known. In this case, it is straightforward to compute the guassian
% parameters theta = {mu_j, sigma_j} =  {mu1, sigma1, mu2, sigma2, mu3,
% sigma3} via Maximum Likelihood Estimation (MLE).

% get one-pass ML estimates of mean and variance of each distribution

mu = zeros(2,max(y_obs(:,1)));
sigma = zeros(2,2,max(y_obs(:,1)));

idx_one = (y_obs(:,1) == 1);
idx_two = (y_obs(:,1) == 2);
idx_three = (y_obs(:,1) == 3);

for i=1:length(ax)
    if (i == 1) || (i == 5)
        h1 = scatter(ax(i),y_obs(idx_one,2),y_obs(idx_one,3), ...
            5, colors(y_obs(idx_one,1),:),'filled');

        h2 = scatter(ax(i),y_obs(idx_two,2),y_obs(idx_two,3), ...
            5, colors(y_obs(idx_two,1),:),'filled');

        h3 = scatter(ax(i),y_obs(idx_three,2),y_obs(idx_three,3), ...
            5, colors(y_obs(idx_three,1),:),'filled');

        hdl = legend(ax(i),[h1(1), h2(1), h3(1)],'Label 1','Label 2','Label 3');
        set(hdl,'Interpreter','latex','Location','Northwest','Autoupdate','off');
    end
end

[mu(:,1),sigma(:,:,1)] = gaussian_mle( y_obs(idx_one,2:3)' );
plot_gaussian_ellipsoid(mu(:,1), sigma(:,:,1), colors(1,:), 1, [], ax(1));

[mu(:,2),sigma(:,:,2)] = gaussian_mle( y_obs(idx_two,2:3)' );
plot_gaussian_ellipsoid(mu(:,2), sigma(:,:,2), colors(2,:), 1, [], ax(1));

[mu(:,3),sigma(:,:,3)] = gaussian_mle( y_obs(idx_three,2:3)' );
plot_gaussian_ellipsoid(mu(:,3), sigma(:,:,3), colors(3,:), 1, [], ax(1));

%% Expectation Maximization - Gaussian Mixture Models (GMMs)

% For Model B, the data labels {pi_j = 1,2,3} are known, but the variable
% Z_i, the sub-distribution labels {m = 1,2} are unknown (latent
% variables), and we can use Expectation Maximization (EM) to learn the
% parameters of the GMM - {w1_j, w2_j, mu1_j, sigma1_j, mu2_l, sigma2_j}.

% Since the data labels {pi_j = 1,2,3} are known, we can use EM to
% determine the paramter set {w1_j, w2_j, mu1_j, sigma1_j, mu2_l, sigma2_j}
% associated with each label j. This process for a given label j is
% independent of every other label j.

n = 2;      % dimensionality of data
M = 2;      % number of sub-distributions within label j

% number of times EM algorithm will be done
N_iter = 100;

for j=1:3

    % get data with label j
    idx = (y_obs(:,1) == j);
    X = y_obs(idx,2:3)';
```

```
    Npoints = size(X,2);

    % init parameters
    mu      = zeros(n,N_iter,M);
    sigma   = zeros(n,n,N_iter,M);
    weights = zeros(M,N_iter);
    r       = zeros(M,Npoints,N_iter);

    % get intial parameter estimates
    lower = [min(X(1,:)), min(X(2,:))];   % min [x,y]
    upper = [max(X(1,:)), max(X(2,:))];   % max [x,y]
    for m=1:M
        for i=1:n
            mu(i,1,m) = (upper(i)-lower(i)).*rand(1) + lower(i);
        end
        sigma(:,:,1,m) = [5, 0.5; 0.5 5];
    end
    weights(:,1) = ones(M,1)/M;

    for s=1:N_iter
        % clear axes and replot data
        cla(ax(j+1));
        scatter(ax(j+1),y_obs(idx,2),y_obs(idx,3), 5, ...
                colors(j,:),'filled');

        % Gaussian EM - E-step
        [ r(:,:,s) ] = gaussian_em_estep( X, squeeze(mu(:,s,:)), ...
            squeeze(sigma(:,:,s,:)), weights(:,s) );

        % Guassian EM - M-step
        [ mu(:,s+1,:), sigma(:,:,s+1,:), weights(:,s+1) ] = ...
            gaussian_em_mstep( X, r(:,:,s), squeeze(mu(:,s,:)) );

        for m=1:M
            % plot guassian ellipsoid m
            plot_gaussian_ellipsoid(mu(:,s+1,m), sigma(:,:,s+1,m), ...
                colors(j,:), 1, [], ax(j+1));
        end
        pause(.1);

    end

    % plot m sub-distributions within label j
    for m=1:M
        % plot guassian ellipsoid m
        plot_gaussian_ellipsoid(mu(:,s+1,m), sigma(:,:,s+1,m), ...
            colors(j,:), 1, [], ax(5));
    end

    fprintf('data label %d:\n', j);
    disp([mu(:,end,1), mu(:,end,2)])
    disp(sigma(:,:,end,1))
    disp(sigma(:,:,end,2))
    disp(weights(:,end));

end
```

gaussian_mle.m - Maximum Likelihood Estimators

```
function [ mu, sigma ] = gaussian_mle( X )

n = size(X,1);  % number of variables in multivariate Gaussian
M = size(X,2);  % number of observations of each n-dim realization of X

mu = sum(X,2)/M;

sigma = zeros(n);
```

```
for i=1:size(X,2)
    sigma = sigma + (X(:,i) - mu)*(X(:,i) - mu)';
end

sigma = sigma/M;
```

gaussian_em_estep.m - Expectation Maximization E-Step

```
function [ r ] = gaussian_em_estep( X, mu, sigma, weights )

M = size(mu,2);          % number of sub-distributions within label j
Npoints = size(X,2);     % number of data points in label j

% fprintf('estep: [M, Npoints] = [%d, %d]\n', M, Npoints);

% init responsibility
r = zeros(M,Npoints);

% Gaussian EM - E-step
for m = 1:M
    for i = 1:Npoints
        numerator = weights(m,1) * ...
            mvnpdf(X(:,i),mu(:,m),sigma(:,:,m));

        denominator = 0;
        for j=1:M
            denominator = denominator + weights(j,1) * ...
                mvnpdf(X(:,i),mu(:,j),sigma(:,:,j));
        end

        % compute "responsibility" that data point i belongs to cluster m
        r(m,i) = numerator/denominator;
    end
end

end
```

gaussian_em_mstep.m - Expectation Maximization M-Step

```
function [ mu, sigma, weights ] = gaussian_em_mstep( X, r, mu )

n = size(mu,1);          % dimensionality of data
M = size(mu,2);          % number of sub-distributions within label j
Npoints = size(X,2);     % number of data points in label j

% fprintf('mstep: [M, Npoints] = [%d, %d]\n', M, Npoints);

% initialize
mu_new = zeros(n,M);
sigma_new = zeros(n,n,M);
weights_new = zeros(M,1);

% Gaussian EM - M-step
for m=1:M
    for i=1:Npoints
        % update mean of distribution m
        mu_new(:,m) = mu_new(:,m) + r(m,i)*X(:,i);

        % update covariance matrix of distribution m
        sigma_new(:,:,m) = sigma_new(:,:,m) + r(m,i) * ...
            (X(:,i) - mu(:,m))*(X(:,i) - mu(:,m))';

        % update weight of distribution m
        weights_new(m,1) = weights_new(m,1) + r(m,i);
    end

```

```
    % sanity check - do these match?
%     fprintf('Nm(%d) = %f\n', m, weights_new(m,1))
%     fprintf('Nm(%d) = sum(r(m,:)) =  %f\n\n', m, sum(r(m,:)))

    % current number of points assigned to cluster m
    Nm = sum(r(m,:));

    % normlize by total number of data points
    mu_new(:,m)      = mu_new(:,m)/Nm;
    sigma_new(:,:,m) = sigma_new(:,:,m)/Nm;
    weights_new(m,1) = weights_new(m,1)/Npoints;

end

    % resize parameters
    mu = reshape(mu_new,n,1,M);
    sigma = reshape(sigma_new,n,n,1,M);
    weights = weights_new;

end
```

Mann Extended-Logarithm Functions Library

```
function [ out ] = eexp( x )
% EEXP - Extended Exponential function

    % if x == 'LOGZERO'
    if isnan(x)
        out = 0;
    else
        out = exp(x);
    end

end

function [ out ] = eln( x )
% ELN - Extended Natural Logarathm function
%   Computes the extended natural logarithm as defined by Mann, 2006

    if x == 0
        % out = 'LOGZERO';
        out = NaN;
    elseif x > 0
        out = log(x);
    else
        error('eln() negative input error')
    end

end

function [ prod ] = elnprod( eln_x, eln_y )
% ELNSUM - Extended Logartithm Product function
%   Computes the extended logarithm of the product of x and y given as
%   given as inputs the extended logarithm of x and y, as defined by Mann, 2006

    % if strcmp(eln(x),'LOGZERO') || strcmp(eln(y),'LOGZERO')
    if isnan(eln_x) || isnan(eln_y)
        % prod = 'LOGZERO';
        prod = NaN;
    else
        prod = eln_x + eln_y;
    end

end

function [ sum ] = elnsum( eln_x, eln_y )
% ELNSUM - Extended Logartithm Sum function
```

```
%   Computes the extended logarithm of the sum of x and y given as inputs
%   the extended logarithm of x and y, as defined by Mann, 2006

    % if strcmp(eln(x),'LOGZERO') || strcmp(eln(y),'LOGZERO')
    if isnan(eln_x) || isnan(eln_y)
        % if strcmp(eln(x),'LOGZERO')
        if isnan(eln_x)
            sum = eln_y;
        else
            sum = eln_x;
        end
    else
        if eln_x > eln_y
            sum = eln_x + eln(1 + exp(eln_y-eln_x));
        else
            sum = eln_y + eln(1 + exp(eln_x-eln_y));
        end
    end

end
```