

Probabilistic Algorithms for Aerospace Autonomy
ASEN 6519 - Homework 1
Inference on Hidden Markov Models

Carl Stahoviak

February 26, 2019

Contents

Problem 1 - Forward-Backward Algorithm, Short Sequence	2
Problem 2 - Likelihood-Weighted Approximate Inference, Short Sequence	3
Problem 3 - Mann Extended-Logarithm Forward-Backward Algorithm, Long Sequence	4
Appendix - MATLAB Code	6

Problem 1 - Forward-Backward Algorithm

Implement the forward-backward algorithm for the `nominal_hmm_short_log.txt` sequence. Report the posterior probabilities $P(x_k|y_{1:T})$ for each timestep k .

Table 1: Posterior Probabilities, $P(x_k|y_{1:T})$

timestep, k	$P(x_k = x_1 y_{1:T})$	$P(x_k = x_2 y_{1:T})$	$P(x_k = x_3 y_{1:T})$	$P(x_k = x_4 y_{1:T})$
1	0	0.9512	0.0487	0
2	0	0.0123	0.9877	0
3	0	0.1538	0.8462	0
4	0	0.1057	0.8943	0
5	0	0.8982	0.1018	0
6	0	0.0022	0.9978	0
7	0	0.9418	0.0582	0
8	0.1833	0	0	0.8167
9	0.0100	0	0	0.9900
10	0.9768	0	0	0.0232
11	0.0012	0	0	0.9988
12	0.9310	0	0	0.0690
13	0.0023	0	0	0.9977
14	0.9509	0	0	0.0491
15	0.0549	0	0	0.9451

Report the data log-likelihood $\log P(y_{1:T})$ for all T observations. Starting with the definition of $\alpha(x_k)$, we have $\alpha(x_k) = P(x_k, y_{1:k})$. At the final timestep T , we have $\alpha(x_T) = P(x_T, y_{1:T})$. Marginalizing over x_T gives the data likelihood $P(y_{1:T})$

$$P(y_{1:T}) = \sum_{x_T} P(x_T, y_{1:T}) = \sum_{x_T} \alpha(x_T) = \sum_{i=1}^n \alpha_i(x_T) \quad (1)$$

where n is the number of discrete states. And thus the data *log*-likelihood, can be written as:

$$\log P(y_{1:T}) = \log \sum_{x_T} \alpha(x_T) = -28.138526 \quad (2)$$

Use the resulting posterior $P(x_k|y_{1:T})$ to classify the most likely state x_k for each timestep $k = 1 : T$ (plot these as a time trace).

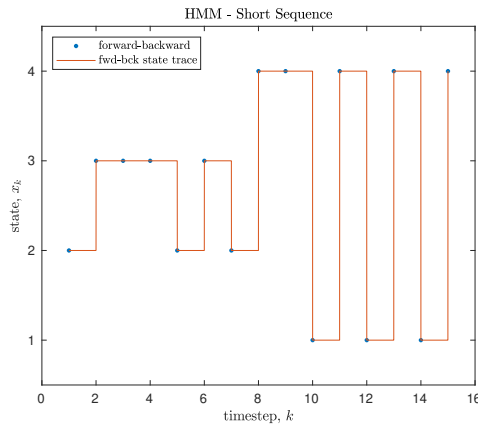


Figure 1: Forward-Backward Algorithm state trace - Short Sequence

Problem 2 - Likelihood-Weighted Approximate Inference

The posterior $P(x_k|y_{1:T})$ can be constructed from the Monte Carlo sample sequences as follows

$$P(x_k = x_i|y_{1:T}) = \frac{\sum_{s=1}^{N_s} w_s \cdot \text{ind}(x_k = x_i)}{\sum_{s=1}^{N_s} w_s} \quad (3)$$

where w_s is the Monte Carlo sequence weight generated according to Algorithm 2.5 in Kochenderfer, and $\text{ind}()$ is the indicator function, and simply returns one when $x_k = x_i$.

For Monte Carlo samples sizes N_s of 100, 1000 and 10,000 the following results are achieved. Each MC sample sequence is a sequence of discrete states chosen randomly according to the state transition probability table, $P(x_k|x_{k-1})$. For sample sizes of less than 10,000 samples, the results (the predicted discrete states) of the likelihood-weighted approximate inference method are unreliable. Given a sample size of 10,000 or greater, the likelihood-weighted approximate inference agrees with the forward-backward algorithm.

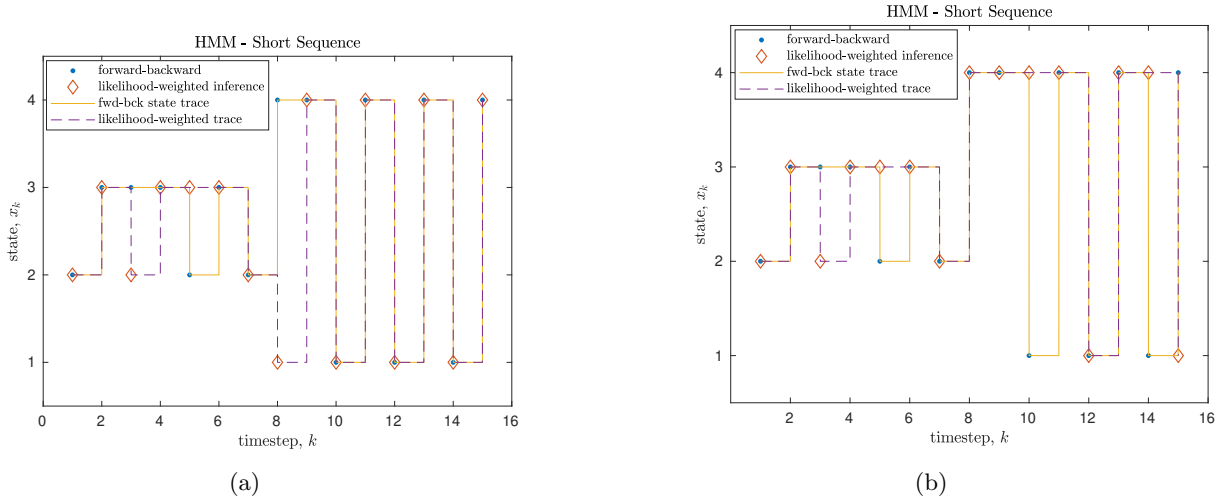


Figure 2: Monte Carlo sample size, $N_s = 100$

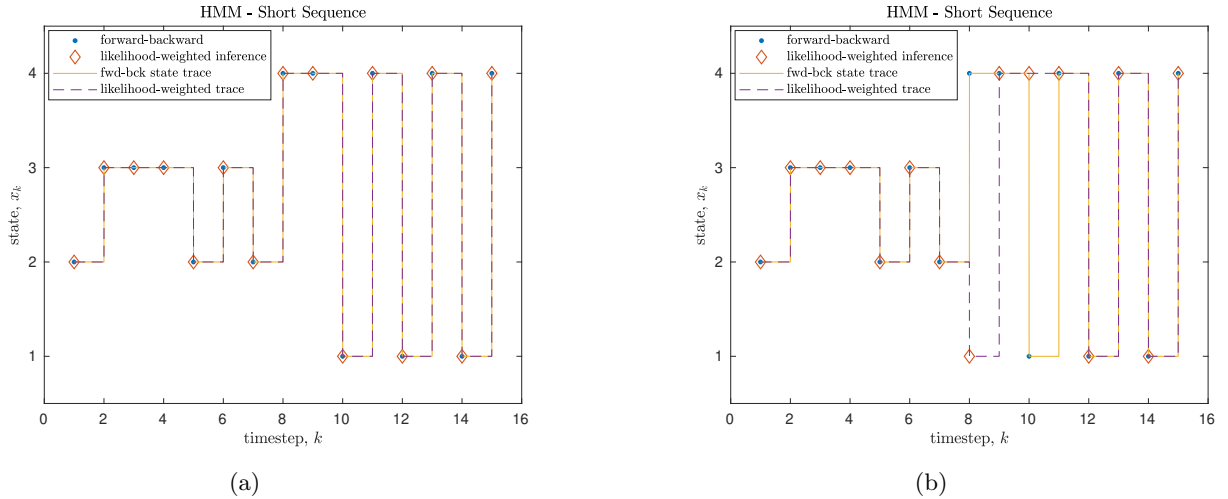
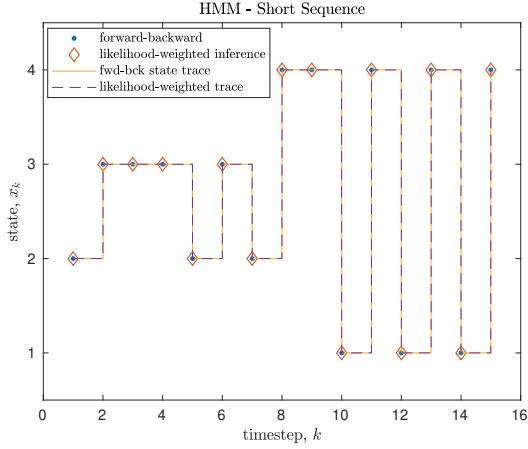
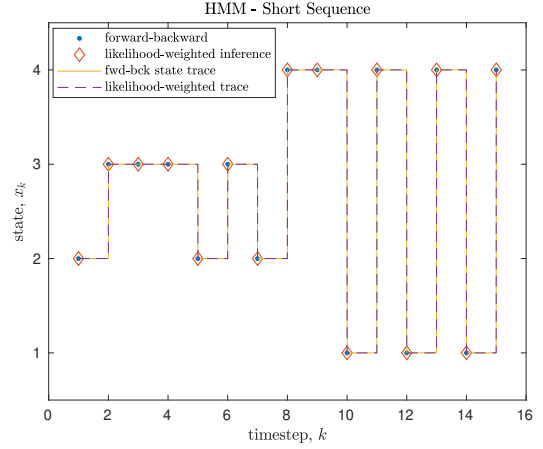


Figure 3: Monte Carlo sample size, $N_s = 1000$



(a)



(b)

Figure 4: Monte Carlo sample size, $N_s = 10,000$

Problem 3 - Mann Extended-Logarithm Forward-Backward Algorithm

Implement the forward-backward algorithm for the `nominal_hmm_long_log.txt` sequence. Use the resulting posterior $P(x_k|y_{1:T})$ to classify the most likely state x_k for each timestep $k = 1 : T$ (plot these as a time trace). In the state trace shown below, the states predicted by the standard Forward-Backward algorithm are compared against the states predicted by the Mann Extended-Logarithm Forward Backward algorithm.

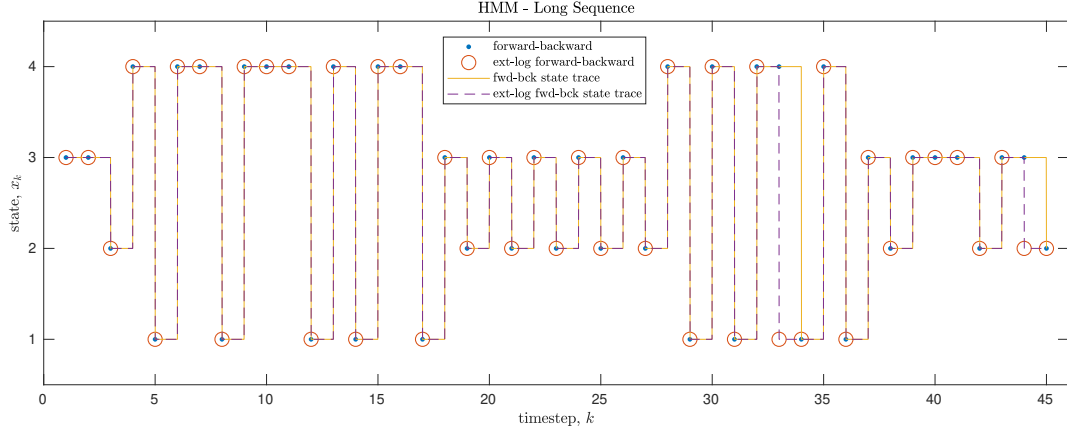


Figure 5: Mann Extended-Logarithm Forward-Backward Algorithm state trace - Long Sequence

Additionally, the data log-likelihood $\log P(y_{1:T})$, can be reported as follows:

$$\log P(y_{1:T}) = \log \sum_{x_T} \alpha(x_T) = -84.743390 \quad (4)$$

Report the posterior probabilities for only the first five and last five steps in the sequence.

Table 2: Forward-Backward Posterior Probabilities, $P(x_k|y_{1:T})$

timestep, k	$P(x_k = x_1 y_{1:T})$	$P(x_k = x_2 y_{1:T})$	$P(x_k = x_3 y_{1:T})$	$P(x_k = x_4 y_{1:T})$
1	0	0.1723	0.8277	0
2	0	0.0058	0.9942	0
3	0	0.9951	0.0049	0
4	0.0062	0	0	0.9938
5	0.7125	0	0	0.2875
\vdots				
41	0	0.0132	0.9868	0
42	0	0.9435	0.0565	0
43	0	0.0134	0.9866	0
44	0	0.1575	0.8425	0
45	0	0.8313	0.1687	0

The Tobias Mann Extended-Logarithm posterior probabilities are shown below for comparison.

Table 3: Mann Ext-Log Forward-Backward Posterior Probabilities, $P(x_k|y_{1:T})$

timestep, k	$P(x_k = x_1 y_{1:T})$	$P(x_k = x_2 y_{1:T})$	$P(x_k = x_3 y_{1:T})$	$P(x_k = x_4 y_{1:T})$
1	0	0.1404	0.8569	0
2	0	0.0838	0.9162	0
3	0	0.9553	0.0447	0
4	0.0087	0	0	0.9913
5	0.9279	0	0	0.0721
\vdots				
41	0	0.0699	0.9301	0
42	0	0.9317	0.0683	0
43	0	0.0097	0.9903	0
44	0	0.5034	0.4966	0
45	0	0.8313	0.1687	0

Appendix - MATLAB Code

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

hw1_hmm.m - main script

```
1 %% Header
2
3 % Author:      Carl Stahoviak
4 % Date Created: 2/19/2019
5
6 clc;
7 clear;
8 close ALL;
9
10 %% Load data
11
12 load('nominal_hmm_params.mat')
13
14 trans_prob = pxk.xkml;
15 obs_prob = pyk.xk;
16
17 %% The Forward-Backward Algorithm (+ Ext-Log FB Alg.)
18
19 load('nominal_hmm_short_log.mat')
20 % load('nominal_hmm_long_log.mat')
21
22 %% standard forward-backward algorithm
23 [alpha, alpha2] = forward( px0, trans_prob, obs_prob, y_obs ); % Txn
24 [beta, beta2] = backward( trans_prob, obs_prob, y_obs ); % Txn
25 % get the posterior distribution
26 posterior = fb_posterior( alpha(2:end,:), beta ); % Txn
27 [i,idx] = max(posterior');
28
29 % calculate data log-likelihood for forward-backward alg.
30 data_ll_fb = log(sum(alpha(end,:)));
31 fprintf('\nFB data log-likelihood = %f\n\n', data_ll_fb);
32
33 %% Mann log-weighted (numerically-stable) forward-backward alg.
34 eln_alpha = forward_eln( px0, trans_prob, obs_prob, y_obs );
35 eln_beta = backward_eln( trans_prob, obs_prob, y_obs );
36 % get the posterior distribution
37 eln_posterior = elnfb_posterior( eln_alpha(2:end,:), eln_beta );
38 [i,eln_idx] = max(eln_posterior');
39
40 % calculate data log-likelihood for ext-log forward-backward alg.
41 data_ll_elnfb = nansum(nansum(eln_alpha,2));
42 fprintf('\nExt-Log FB data log-likelihood = %f\n\n', data_ll_elnfb);
43
44 %% Likelihood-Weighted Sampling
45
46 n = size(px0,1); % number of states
47 Ns = 10000; % number of Monte Carlo sample sequences
48
49 % get Ns Monte Carlo sample sequences of length T, and
50 % corresponding sequence weights
51 T = size(y_obs,1);
52 [lw_samples, weights] = lw_sampling( Ns, T, px0, trans_prob, ...
53 obs_prob, y_obs);
54
55 % get likelihood-weighted (approximate?) inference posterior
56 lw_posterior = lw_inference( n, lw_samples, weights );
57 [i,lw_idx] = max(lw_posterior');
58
59 %% Plot Data
60
```

```

61 % create timeseries
62 t = linspace(1,size(y_obs,1),5000)';
63
64 % create empty continuous state vectors
65 state = zeros(size(t,1),1);
66 eln_state = zeros(size(t,1),1);
67 lw_state = zeros(size(t,1),1);
68
69 % create plot-friendly continuous states
70 for i=1:length(t)
71     state(i,1) = idx(floor(t(i)));
72     eln_state(i,1) = eln_idx(floor(t(i)));
73     lw_state(i,1) = lw_idx(floor(t(i)));
74 end
75
76 figure(1)
77 plot(idx, '.', 'MarkerSize',10); hold on;
78 plot(eln_idx, 'o', 'MarkerSize',10);
79 plot(lw_idx, 'diamond', 'MarkerSize',7);
80 plot(t, state);
81 plot(t, eln_state, '--');
82 % plot(t, lw_state, '--');
83 xlim([0, size(y_obs,1)+1])
84 ylim([0.5, 4.5]); yticks([1 2 3 4 5])
85 title('HMM - Long Sequence', 'Interpreter', 'latex');
86 xlabel('timestep, $k$', 'Interpreter', 'latex');
87 ylabel('state, $x_k$', 'Interpreter', 'latex');
88 hdl = legend('forward-backward', 'ext-log forward-backward', ...
89             'likelihood-weighted inference', 'fwd-bck state trace', ...
90             'ext-log fwd-bck state trace');
91 set(hdl, 'Interpreter', 'latex', 'Location', 'Northwest')

```

forward.m - forward pass of the forward-backward algorithm

```

1 function [ alpha, alpha2 ] = forward( px0, trans_prob, obs_prob, y_obs )
2
3 n = size(px0,1);
4 T = size(y_obs,1);
5
6 % initialization - alpha(x0) = prior
7 alpha = zeros(n,T+1);
8 alpha(:,1) = px0;
9
10 alpha2 = zeros(n,T+1);
11 alpha2(:,1) = px0;
12
13 % forward pass
14 for k=1:T
15     % matrix math version... works!
16     alpha2(:,k+1) = alpha2(:,k)'*trans_prob'*diag(obs_prob(y_obs(k),:));
17     for i=1:n
18         for j=1:n
19             alpha(i,k+1) = alpha(i,k+1) + ( alpha(j,k) * ...
20                 trans_prob(i,j) * obs_prob(y_obs(k),i) );
21         end
22     end
23     % normalize (do NOT normalize alpha values!)
24     % alpha(:,k+1) = alpha(:,k+1)./sum(alpha(:,k+1));
25 end
26
27 % return alpha with dimensions Txn
28 alpha = alpha';
29 alpha2 = alpha2';
30
31 end

```

backward.m - backward pass of the forward-backward algorithm

```
1 function [ beta, beta2 ] = backward( trans_prob, obs_prob, y_obs )
2
3 n = size(trans_prob,1);
4 T = size(y_obs,1);
5
6 % initialization
7 beta = zeros(n,T);
8 beta(:,end) = ones(n,1);
9
10 beta2 = zeros(n,T);
11 beta2(:,end) = ones(n,1);
12
13 % backward pass
14 for k=(T-1):-1:1
15     % matrix math version... not working
16     % beta2(:,k) = beta2(:,k+1)'*trans_prob*diag(obs_prob(y_obs(k+1),:));
17     beta2(:,k) = obs_prob(y_obs(k+1,:))*trans_prob*diag(beta2(:,k+1));
18
19     for i=1:n
20         for j=1:n
21             beta(i,k) = beta(i,k) + ( beta(j,k+1) * ...
22                 trans_prob(j,i) * obs_prob(y_obs(k+1),j) );
23         end
24     end
25     % normalize (do NOT normalize beta values!)
26     % beta(:,k) = beta(:,k)./sum(beta(:,k));
27 end
28
29 % return beta with dimensions Txn
30 beta = beta';
31 beta2 = beta2';
32
33 end
```

fb_posterior.m - forward-backward algorithm posterior calculation

```
1 function [ posterior ] = fb_posterior( alpha, beta )
2
3 if size(alpha,2) ≠ size(beta,2)
4     error('alpha, beta size mismatch')
5
6 else
7     n = size(alpha,1);
8     T = size(alpha,2);
9     posterior = zeros(n,T);
10
11     for k=1:T
12         % sanity check
13         fprintf('sum(alpha(:,%d).*beta(:,%d)) = %e\n', k, ...
14             k, sum(alpha(:,k).*beta(:,k)))
15
16         posterior(:,k) = (alpha(:,k).*beta(:,k)) / ...
17             sum( alpha(:,k).*beta(:,k) );
18
19         % normalize
20         posterior(:,k) = posterior(:,k)./sum(posterior(:,k));
21     end
22
23     % return a Txn matrix
24     posterior = posterior';
25 end
```


forward_eln.m - extended-logarithm forward pass

```
1 function [ eln_alpha ] = forwardeln( px0, trans_prob, obs_prob, y_obs )
2
3 % use Mann notation
4 trans_prob = trans_prob';
5
6 n = size(px0,1);
7 T = size(y_obs,1);
8 eln_alpha = zeros(n,T+1);
9
10 % initialization
11 for i=1:n
12     % eln_alpha(i,1) = elnprod( eln(px0(i,1)), ...
13     %     eln(obs_prob(y_obs(1),i)) );
14     eln_alpha(i,1) = eln(px0(i,1));
15 end
16
17 for k=2:T+1
18     for j=1:n
19         logalpha = NaN;
20         for i=1:n
21             logalpha = elnsum( logalpha, ...
22                 elnprod( eln_alpha(i,k-1), eln(trans_prob(i,j)) ));
23         end
24         eln_alpha(j,k) = elnprod( logalpha, ...
25             eln(obs_prob(y_obs(k-1),j)) );
26     end
27 end
28
29 % return a Txn matrix
30 eln_alpha = eln_alpha';
31
32 end
```

backward_eln.m - extended-logarithm backward pass

```
1 function [ eln_beta ] = backwardeln( trans_prob, obs_prob, y_obs )
2
3 % use Mann notation
4 trans_prob = trans_prob';
5
6 n = size(trans_prob,1);
7 T = size(y_obs,1);
8
9 % initialization
10 eln_beta = zeros(n,T);
11
12 for k=(T-1):-1:1
13     for i=1:n
14         logbeta = NaN;
15         for j=1:n
16             logbeta = elnsum( logbeta, ...
17                 elnprod( eln(trans_prob(i,j)), ...
18                     elnprod( obs_prob(y_obs(k+1),j), eln_beta(j,k+1) )));
19         end
20         eln_beta(i,k) = logbeta;
21     end
22 end
23
24 % return a Txn matrix
25 eln_beta = eln_beta';
```

elnfb_posterior.m - extended-logarithm posterior calculation

```
1 function [ eln_post ] = elnfb_posterior( eln_alpha, eln_beta )
2
3 if size(eln_alpha,2) ≠ size(eln_beta,2)
4     error('eln_alpha, eln_beta size mismatch')
5
6 else
7     n = size(eln_alpha,1);
8     T = size(eln_alpha,2);
9
10    eln_gamma = zeros(n,T);    % log-posterior
11    gamma = zeros(n,T);        % true posterior
12
13    for k=1:T
14        normalizer = NaN;
15        for i=1:n
16            eln_gamma(i,k) = elnprod(eln_alpha(i,k),eln_beta(i,k));
17            normalizer = elnsum(normalizer,eln_gamma(i,k));
18        end
19
20        for i=1:n
21            eln_gamma(i,k) = elnprod(eln_gamma(i,k),-normalizer);
22            gamma(i,k) = exp(eln_gamma(i,k));
23        end
24        % sanity check
25        fprintf('1 - sum(gamma(:,%d)) = %e\n', k, 1-sum(gamma(:,k)))
26    end
27 end
28
29 % return a Txn matrix
30 eln_post = gamma';;
```

lw_sampling.m - likelihood-weighted sampling

```
1 function [ lw_samples, weights ] = lw_sampling( Ns, T, px0, trans_prob, obs_prob, y_obs )
2 %LW_SAMPLING Likelihood Weighted Sampling
3
4 lw_samples = zeros(Ns,T);    % V.E - likelihood weighted sample
5 weights = ones(Ns,1);        % W.E - likelihood weights
6
7 % get initial state from prior distribution, px0
8 [~,idx] = max(px0);
9
10 for s=1:Ns
11     for k=1:T
12         if k==1
13             % draw random sample according to initial (known) state, x0
14             lw_samples(s,k) = randsample(4,1,true, trans_prob(:,idx));
15         else
16             % draw random sample according to state transtion probabilities
17             lw_samples(s,k) = randsample(4,1,true, trans_prob(:,lw_samples(s,k-1)));
18         end
19
20         % update sequence weight:
21         % weight = weight * P( y_k | Pa(y_k) = x_k )
22         % NOTE: parent of each observation y_k is the state x_k
23         weights(s,1) = weights(s,1) * obs_prob(y_obs(k),lw_samples(s,k));
24     end
25 end
26
27 end
```

lw_inference.m - likelihood-weighted approximate inference

```
1 function [ posterior ] = lw_inference( n, lw_samples, weights )
2 %LW-INFERENCE Likelihood-weighted approximate inference
3
4 T = size(lw_samples,2);
5 posterior = zeros(n,T);
6
7 for k=1:T
8     for i=1:n
9         % get index location of where x.k = x.i across all sequences
10        idx = (lw_samples(:,k) == i);
11
12        % the posterior for each state i at timestep k is the weighted sum
13        % of the relaizations (indicator function) of state x.i at
14        % timestep k across all MC sample sequences
15        posterior(i,k) = sum(weights(idx))/sum(weights);
16    end
17 end
18
19 posterior = posterior';
20
21 end
```

Mann Extended-Logarithm Functions Library

```
1 function [ out ] = eexp( x )
2 % EEXP - Extended Exponential function
3
4     % if x == 'LOGZERO'
5     if isnan(x)
6         out = 0;
7     else
8         out = exp(x);
9     end
10
11 end
12
13 function [ out ] = eln( x )
14 % ELN - Extended Natural Logarithm function
15 % Computes the extended natural logarithm as defined by Mann, 2006
16
17 if x == 0
18     % out = 'LOGZERO';
19     out = NaN;
20 elseif x > 0
21     out = log(x);
22 else
23     error('eln() negative input error')
24 end
25
26 end
27
28 function [ prod ] = elnprod( eln_x, eln_y )
29 % ELNSUM - Extended Logarithm Product function
30 % Computes the extended logarithm of the product of x and y given as
31 % given as inputs the extended logarithm of x and y, as defined by Mann, 2006
32
33 % if strcmp(eln(x),'LOGZERO') || strcmp(eln(y),'LOGZERO')
34 if isnan(eln_x) || isnan(eln_y)
35     % prod = 'LOGZERO';
36     prod = NaN;
37 else
38     prod = eln_x + eln_y;
39 end
40
```

```

41 end
42
43 function [ sum ] = elnsum( eln_x, eln_y )
44 % ELNSUM - Extended Logarithm Sum function
45 %   Computes the extended logarithm of the sum of x and y given as inputs
46 %   the extended logarithm of x and y, as defined by Mann, 2006
47
48 % if strcmp(eln(x),'LOGZERO') || strcmp(eln(y),'LOGZERO')
49 if isnan(eln_x) || isnan(eln_y)
50     % if strcmp(eln(x),'LOGZERO')
51     if isnan(eln_x)
52         sum = eln_y;
53     else
54         sum = eln_x;
55     end
56 else
57     if eln_x > eln_y
58         sum = eln_x + eln(1 + exp(eln_y - eln_x));
59     else
60         sum = eln_y + eln(1 + exp(eln_x - eln_y));
61     end
62 end
63
64 end

```