Supplemental Material for "Online EM Algorithm for Hidden Markov Models" Published in the Journal of Computational and Graphical Statistics: MATLAB/OCTAVE Function Used for Implementing the Online EM Algorithm in Section 4

Olivier Cappé

LTCI, Telecom ParisTech & CNRS

1 Main Function: ion_online

```
function [q, mu, v] = ion_online(Y, q_0, mu_0, v_0, gam)

% ion_online     On-line EM algorithm for the ion channel model.
% Use: [q, mu, v] = ion_online(Y, q_0, mu_0, v_0, gam) where
%       Y:      observations (n x 1)
%       gam:    stepsizes (n x 1)
%       q_0:    initial estimate of transition matrix (m x m)
%       mu_0:   initial estimates of means (m x 1)
%       v_0:    initial estimate of variance (scalar)
% and
%       q:      estimated transition matrices (m x m x n+1)
%       mu:     estimated means (m x 1 x n+1)
%       v:      estimated variances (1 x 1 x n+1)
%
% m is the number of states and n the number of observations.

% Parameters
INHIBIT_LAG = 20;
% In inhibition stage, let gam(i) = 1/i and do not perfom any update
gam(1:INHIBIT_LAG) = 1./(1:INHIBIT_LAG);

% Dimensions
n = length(Y);
m = length(mu_0);

% Initialization
q = zeros(m,m,n+1);
mu = zeros(m,1,n+1);
v = zeros(1,1,n+1);
q(:,:,1) = q_0;
mu(:,:,1) = mu_0;
v(:,:,1) = v_0;

for t = 1:n
```

```
    % The SA E-step
    if (t == 1)
      % Initial observation
      [filt, T, S_0, S_1, S_2] = ion_e0(Y(1), mu(:,:,1), v(:,:,1));
    else
      [filt, T, S_0, S_1, S_2] = ...
        ion_e(Y(t), filt, T, S_0, S_1, S_2, q(:,:,t), mu(:,:,t), v(:,:,t), gam(t));
    end
    % The M-step
    if (t >= INHIBIT_LAG)
      [T_a, S_0_a, S_1_a, S_2_a] = ion_e_accum(filt, T, S_0, S_1, S_2);
      [q(:,:,t+1), mu(:,:,t+1), v(:,:,t+1)] = ion_m(T_a, S_0_a, S_1_a, S_2_a);
    else
      % Inhibition phase (no update)
      q(:,:,t+1) = q(:,:,t);
      mu(:,:,t+1) = mu(:,:,t);
      v(:,:,t+1) = v(:,:,t);
    end
end
```

2 Subroutines

2.1 ion_e0

```
function [filt, T, S_0, S_1, S_2] = ion_e0(Y, mu, v)

% ion_e0        Compute the EM auxiliary statistics for the initial data point.
% Use: [filt, T, S_0, S_1, S_2, logl] = ion_e0(Y, mu, v)
% where
%       Y:      initial observation (scalar)
%       mu:     means (m x 1)
%       v:      variance (scalar)
% and
%       filt:   filtering prob. (1 x m)
%       T:      Auxiliary EM transition statistics (m x m x m)
%       S_0:    Auxiliary EM conditional statistics (m x m)
%       S_1:        "                               (m x m)
%       S_2:        "                               (m x m)

% Dimensions
m = length(mu);

% Likelihood factor
r = exp(-0.5*(Y(ones(m,1))-mu).^2/v)/sqrt(v);

% Initial filtering (assuming uniform initial distribution)
c = sum(r);
filt = r/c;

% Initial statistics
T = zeros(m,m,m);
S_0 = eye(m);
S_1 = eye(m)*Y;
S_2 = eye(m)*Y^2;
```

2.2 ion_e

```
function [filt, T, S_0, S_1, S_2] = ...
  ion_e(Y, filt, T, S_0, S_1, S_2, q, mu, v, gam)

% ion_e          Update the auxiliary EM statistics for one data point.
% Use: [filt, T, S_0, S_1, S_2] = ...
%   ion_e(Y, filt, T, S_0, S_1, S_2, q, mu, v, gam)
% where
%       Y:      observation (scalar)
%       q:      transition matrix (m x m)
%       mu:     means (m x 1)
%       v:      variance (scalar)
%       gam:    stepsize (scalar)
% and
%       filt:   filtering prob. (1 x m)
%       T:      Auxiliary EM transition statistics (m x m x m)
%       S_0:    Auxiliary EM conditional statistics (m x m)
%       S_1:        "                               (m x m)
%       S_2:        "                               (m x m)

% Dimensions
m = length(mu);

% New likelihood factor
r = (filt*ones(1, m)).*q;
joint = r.*(ones(m,1)*(exp(-0.5*(Y(ones(m,1))-mu).^2/v)/sqrt(v))');
% Filtering update
filt = (sum(joint,1))';
c = sum(filt);
% Normalize
joint = joint/c;
filt = filt/c;
% Retrospective kernel
r = r./(ones(m,1)*sum(r));

% Statistics update
T1 = zeros(m,m,m);
for k = 1:m
  T1(:,k,k) = r(:,k);
end
T = gam*T1+(1-gam)*reshape((reshape(T,m*m,m)*r),m,m,m);
S_0 = gam*eye(m)+(1-gam)*(S_0*r);
S_1 = gam*eye(m)*Y+(1-gam)*(S_1*r);
S_2 = gam*eye(m)*Y^2+(1-gam)*(S_2*r);
```

2.3 ion_e_accum

```
function [T_a, S_0_a, S_1_a, S_2_a] = ion_e_accum(filt, T, S_0, S_1, S_2)

% ion_e_accum    Obtain EM statistics from auxiliary statistics.
% Use: [T_a, S_0_a, S_1_a, S_2_a] = ion_e_accum(filt, T, S_0, S_1, S_2)
%       filt:   filtering prob. (1 x m)
%       T:      Auxiliary EM transition statistics (m x m x m)
```

```
%       S_0:    Auxiliary EM conditional statistics (m x m)
%       S_1:        "                               (m x m)
%       S_2:        "                               (m x m)
% and
%       T_a:    EM transition statistics (m x m x 1)
%       S_0_a:  EM conditional statistics (m x 1)
%       S_1_a:      "                     (m x 1)
%       S_2_a:      "                     (m x 1)

% Dimensions
m = length(S_0);

% Weight with filter values
T_a = reshape((reshape(T,m*m,m)*filt),m,m);
S_0_a = S_0*filt;
S_1_a = S_1*filt;
S_2_a = S_2*filt;
```

2.4 ion_m

```
function [q, mu, v] = ion_m(T, S_0, S_1, S_2);

% ion_m         The M-step in the ion channel model.
% Use:  [q, mu, v] = ion_m(T, S_0, S_1, S_2)
% where
%       T:      EM transition statistics (m x m)
%       S_0:    EM conditional statistics (m x 1)
%       S_1:        "                     (m x 1)
%       S_2:        "                     (m x 1)
% and
%       q:      transition matrix (m x m)
%       mu:     means (m x 1)
%       v:      variance (scalar)

% Dimensions
m = size(T,1);

% Transition matrix
q = T./(sum(T,2)*ones(1,m));

% Means
mu = S_1./S_0;

% Variance
v = sum(S_2-(S_0.*mu.^2))/sum(S_0);
```