function [rate\_func] = HMM\_v2(x)

% [rate\_func] = HMM(x)

%

% Function `HMM' returns the firing rate selected as an alternative hidden state.

% Original paper:

% Mochizuki and Shinomoto, Analog and digital codes in the brain

% https://arxiv.org/abs/1311.4035

%

% Example usage:

% rate\_func = HMM(x);

%

% Input argument

% x: Sample data vector.

%

% Output argument

% rate\_func: ratefunction.

% 2D array stores

% 1: begining time of each bins in second

% 2: rate of each bin

% made by Yasuhiro Mochizuki

% revised by Kazuki Nakamura

% Contact: Shigeru Shinomoto: shinomoto@scphys.kyoto-u.ac.jp

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

onset = x(1) - 0.001 \* (x(length(x)) - x(1));

offset = x(length(x)) + 0.001 \* (x(length(x)) - x(1));

optw = (offset-onset)/(length(x)) \* 5;

rate\_func = get\_hmm\_ratefunc(x, optw);

drawHMM(rate\_func)

end

%sub function

function vec\_Xi = get\_vec\_Xi(vec\_spkt, bin\_width)

%%%%vec\_Xi = zeros(length(vec\_spkt),1);

bin\_num=ceil(vec\_spkt(length(vec\_spkt))/bin\_width);

vec\_Xi = zeros(bin\_num, 1);

for i=1:length(vec\_spkt)

bin\_id=fix(vec\_spkt(i)/bin\_width)+1;

if bin\_id<bin\_num + 1

vec\_Xi(bin\_id) = vec\_Xi(bin\_id)+1;

end

end

end

% func to get emisson probs.

function mat\_emission = get\_mat\_emission(vec\_Xi, vec\_lambda)

mat\_emission=zeros(length(vec\_Xi),length(vec\_lambda));

for n=1:length(vec\_Xi)

for i=1:length(vec\_lambda)

mat\_emission(n,i)=vec\_lambda(i)^vec\_Xi(n)\*exp(-1.0\*vec\_lambda(i))/factorial(vec\_Xi(n));

end

end

end

% func to get alpha and C

function [vec\_C, mat\_alpha] = get\_alpha\_C(mat\_A, vec\_pi, mat\_emission)

num\_of\_states=length(vec\_pi);

num\_of\_obs=length(mat\_emission(:, 1));

vec\_C = zeros(num\_of\_obs,1);

%n=1

for i=1:num\_of\_states

mat\_alpha(1,i) = mat\_emission(1,i) \* vec\_pi(i);

end

vec\_C(1) = sum(mat\_alpha(1,:));

for i=1:num\_of\_states

mat\_alpha(1,i) = mat\_alpha(1,i)/vec\_C(1);

end

%n>1

for n=2:num\_of\_obs

for i=1:num\_of\_states

sum\_j=0.0;

for j=1:num\_of\_states

sum\_j = sum\_j + mat\_alpha(n-1,j)\*mat\_A(j,i);

end

mat\_alpha(n,i) = mat\_emission(n,i)\*sum\_j;

end

vec\_C(n)=sum(mat\_alpha(n,:));

mat\_alpha(n,:) = mat\_alpha(n,:)./vec\_C(n);

end

end

function mat\_beta = get\_beta(mat\_A, vec\_pi, mat\_emission, vec\_C)

num\_of\_states=length(vec\_pi);

num\_of\_obs=length(mat\_emission(:, 1));

% initialize

mat\_beta = zeros(num\_of\_obs, num\_of\_states);

% n=N

mat\_beta(num\_of\_obs,:)=1.0;

% n<N

for m=2:num\_of\_obs

n=(num\_of\_obs+1-m);

for i=1:num\_of\_states

sum\_j=0.0;

for j=1:num\_of\_states

sum\_j= sum\_j+mat\_beta(n+1,j)\*mat\_emission(n+1,j)\*mat\_A(i,j);

end

mat\_beta(n,i)=(sum\_j/vec\_C(n+1));

end

end

end

function [mat\_Gamma, mat\_Xi] = get\_Gamma\_Xi(mat\_A, mat\_emission, mat\_alpha, mat\_beta, vec\_C)

[num\_of\_obs, num\_of\_states] = size(mat\_emission);

% gamma matrix

mat\_Gamma=zeros(num\_of\_obs, num\_of\_states);

for n=1:num\_of\_obs

for i=1:num\_of\_states

mat\_Gamma(n,i)=mat\_alpha(n,i)\*mat\_beta(n,i);

end

end

% Xi matrix

mat\_Xi=zeros(num\_of\_obs-1, num\_of\_states, num\_of\_states);

for m=1:num\_of\_obs-1

for i=1:num\_of\_states

for j=1:num\_of\_states

mat\_Xi(m,i,j)=(mat\_alpha(m,i)\*mat\_emission(m+1,j)\*mat\_A(i,j)\*mat\_beta(m+1,j))/vec\_C(m+1);

end

end

end

end

% HMM\_E\_step

function [mat\_Gamma, mat\_Xi] = HMM\_E\_step(vec\_Xi, mat\_A, vec\_lambda, vec\_pi)

mat\_emission=get\_mat\_emission(vec\_Xi,vec\_lambda);

[vec\_C, mat\_alpha]=get\_alpha\_C(mat\_A, vec\_pi, mat\_emission);

mat\_beta=get\_beta(mat\_A, vec\_pi, mat\_emission, vec\_C);

[mat\_Gamma, mat\_Xi]=get\_Gamma\_Xi(mat\_A, mat\_emission, mat\_alpha, mat\_beta, vec\_C);

end

% HMM\_M\_step

function [vec\_pi\_new, vec\_lambda\_new, mat\_A\_new] = HMM\_M\_step(vec\_Xi, mat\_A, vec\_lambda, vec\_pi, mat\_Gamma, mat\_Xi)

num\_of\_states=length(mat\_A);

num\_of\_obs=length(vec\_Xi);

% maximize wrt pi vector

pi\_denom=sum(mat\_Gamma(1,:));

vec\_pi\_new=mat\_Gamma(1,:)./pi\_denom;

% maximize wrt lambda vector

vec\_lambda\_new=zeros(num\_of\_states,1);

for k=1:num\_of\_states

lambda\_denom=sum(mat\_Gamma(:,k));

lambda\_nume=0.0;

for n=1:num\_of\_obs

lambda\_nume=lambda\_nume+(mat\_Gamma(n,k)\*vec\_Xi(n));

end

if lambda\_nume==0.0

vec\_lambda\_new(k)=0.0;

else

vec\_lambda\_new(k)=lambda\_nume/lambda\_denom;

end

end

% maximize wrt A matirx

mat\_A\_new=zeros(num\_of\_states, num\_of\_states);

for j=1:num\_of\_states

A\_denome=0.0;

for n=1:num\_of\_obs-1

for l=1:num\_of\_states

A\_denome=A\_denome+mat\_Xi(n,j,l);

end

end

for k=1:num\_of\_states

A\_nume=0.0;

for n=1:num\_of\_obs-1

A\_nume = A\_nume+mat\_Xi(n,j,k);

if A\_nume==0.0

mat\_A\_new(j,k)=0.0;

else

mat\_A\_new(j,k)=A\_nume/A\_denome;

end

end

end

end

end

% HMM\_Viterbi

function vec\_hs\_seq=HMM\_Viterbi(vec\_Xi, mat\_A, vec\_lambda, vec\_pi)

mat\_emission=get\_mat\_emission(vec\_Xi, vec\_lambda);

num\_of\_states=length(mat\_A);

num\_of\_obs=length(vec\_Xi);

mat\_hs\_seq=zeros(num\_of\_states, num\_of\_obs);

vec\_logp\_seq=zeros(num\_of\_states,1);

% n=1

for j=1:num\_of\_states

mat\_hs\_seq(j,1)=j;

vec\_logp\_seq(j)=log(vec\_pi(j)\*mat\_emission(1,j))/log(10);

end

% n>1

for n=2:num\_of\_obs

% copy the seq. up to n-1

mat\_hs\_seq\_buf=mat\_hs\_seq;

vec\_logp\_seq\_buf=vec\_logp\_seq;

% nth node->j

for j=1:num\_of\_states

% n-1th node->j

% compute logp for i->j trans

for i=1:num\_of\_states

vec\_h\_logprob\_i(i)=vec\_logp\_seq(i)+log(mat\_emission(n,j)\*mat\_A(i,j))/log(10);

end

% get max logp

[max\_element,max\_pos]=max(vec\_h\_logprob\_i);

vec\_logp\_seq\_buf(j)=max\_element;

mat\_hs\_seq\_buf(j,:)=mat\_hs\_seq(max\_pos,:);

mat\_hs\_seq\_buf(j,n)=j;

end

% updata the seq.

mat\_hs\_seq=mat\_hs\_seq\_buf;

vec\_logp\_seq=vec\_logp\_seq\_buf;

end

[max\_element, max\_pos]=max(vec\_logp\_seq);

vec\_hs\_seq=mat\_hs\_seq(max\_pos,:);

end

% get\_hmm\_ratefunc

function rate\_func= get\_hmm\_ratefunc(spike\_time, bin\_width)

EMloop\_num=5000; % number of EM itteration

mat\_A=[0.999 0.001; 0.001 0.999];

vec\_pi=[0.5 0.5];

mean\_rate=length(spike\_time)/(spike\_time(length(spike\_time))-spike\_time(1));

vec\_lambda=[(mean\_rate\*0.75)\*bin\_width (mean\_rate\*1.25)\*bin\_width];

% 2D array stores

% 1: begining time of each bins in second

% 2: rate of each bin

% get hmm rate func

vec\_spkt=zeros(length(spike\_time),1);

for i=1:length(spike\_time)

vec\_spkt(i)=spike\_time(i)-spike\_time(1);

end

vec\_Xi=get\_vec\_Xi(vec\_spkt, bin\_width);

[mat\_Gamma, mat\_Xi]=HMM\_E\_step(vec\_Xi, mat\_A, vec\_lambda, vec\_pi);

mat\_A\_old = mat\_A;

vec\_pi\_old=vec\_pi;

vec\_lambda\_old=vec\_lambda;

loop=0;

flag=0;

while (loop<=EMloop\_num && flag==0)

[vec\_pi\_new, vec\_lambda\_new, mat\_A\_new]=HMM\_M\_step(vec\_Xi, mat\_A, vec\_lambda, vec\_pi, mat\_Gamma, mat\_Xi);

vec\_pi=vec\_pi\_new;

vec\_lambda=vec\_lambda\_new;

mat\_A=mat\_A\_new;

sum\_check=0.0;

num\_state=length(vec\_pi);

for i=1:num\_state

for j=1:num\_state

sum\_check=sum\_check+abs(mat\_A\_old(i,j)-mat\_A(i,j));

end

sum\_check=sum\_check+abs(vec\_pi\_old(i)-vec\_pi(i));

sum\_check=sum\_check+abs(vec\_lambda\_old(i)-vec\_lambda(i));

end

if sum\_check/(1.0\*num\_state\*(num\_state+2))<10^(-7)

flag=flag+1;

end

mat\_A\_old=mat\_A;

vec\_pi\_old=vec\_pi;

vec\_lambda\_old=vec\_lambda;

[mat\_Gamma, mat\_Xi]=HMM\_E\_step(vec\_Xi, mat\_A, vec\_lambda, vec\_pi);

loop=loop+1;

end

vec\_hidden=HMM\_Viterbi(vec\_Xi, mat\_A, vec\_lambda, vec\_pi);

rate\_func=zeros(length(vec\_Xi),2);

%%%%c\_time=0.0;

onset = spike\_time(1) - 0.001 \* (spike\_time(length(spike\_time)) - spike\_time(1));

c\_time = onset;

for n=1:length(vec\_Xi)

state\_id=vec\_hidden(n);

rate\_func(n,1)=round(c\_time\*100)/100.0;

rate\_func(n,2)=round(vec\_lambda(state\_id)\*100)/(bin\_width\*100.0);

c\_time=c\_time+bin\_width;

end

end

function drawHMM(rate\_func)

x = rate\_func(:, 1);

y = rate\_func(:, 2);

ind = 1;

x\_new(ind) = x(1);

y\_new(ind) = min(y);

ind = ind + 1;

x\_new(ind) = x(1);

y\_new(ind) = y(1);

ind = ind + 1;

for i = 2 : length(y)

if y(i - 1) ~= y(i)

t = (x(i - 1) + x(i)) / 2;

x\_new(ind) = t;

y\_new(ind) = y(i - 1);

ind = ind + 1;

x\_new(ind) = t;

y\_new(ind) = y(i);

ind = ind + 1;

end

end

x\_new(ind) = x(length(x));

y\_new(ind) = y(length(y));

plot(x\_new, y\_new);

axis([min(x) max(x) 0 max(y) \* 1.1]);

end