MotorHMM1

A simple demonstration of the HMM analysis of molecular motor time-courses

This program implements the algorithms in the manuscript by S. Sayed, F. Müllner, P. Selvin and F. Sigworth. It is distributed as a Matlab script `MotorHMM1.m` along with a collection of Matlab functions (`.m` files) and machine-specific Matlab-extension files (`.mex`). The latter are not needed if you use the default `SimpleFB` option in the script. You will need Matlab version 7 or higher to run the program (we use versions 7.5 and 7.6). Copy all the files into a directory, set Matlab to that directory, and type `MotorHMM1`.

If you run the script as-is it will generate a 500-point simulated trace, and analyze it according to the "1-state" HMM. It takes about 30 s to run on my Mac Pro computer, and it generates three figures. The default simulation is alternating steps with sizes of 8 and 28 nm. There is a small random variation in each step size (Gaussian with S.D. = 0.3 nm). The mean dwell time between steps is 10 time points, Poisson distributed. The simulation uses a continuous time variable, so that multiple steps sometimes occur within a single sample interval. The noise is Gaussian with S.D. = 6 nm.

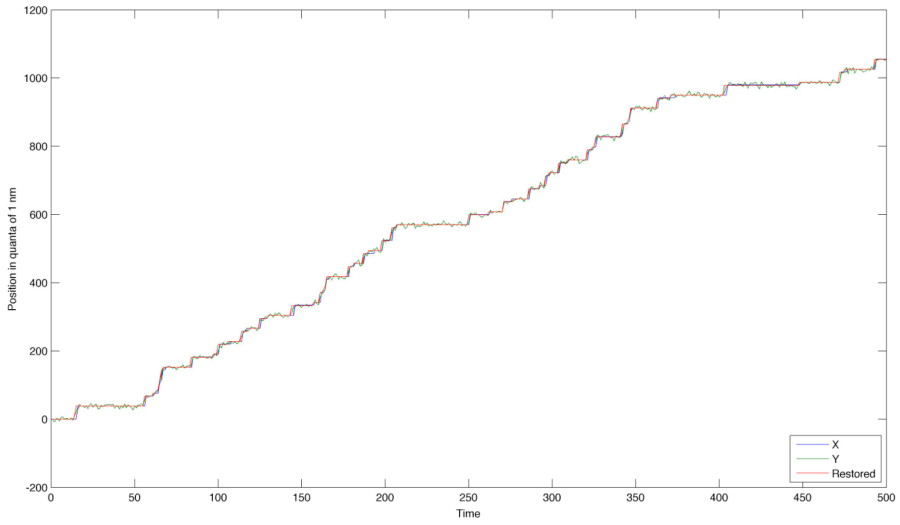


Figure 1 (above) shows the "noiseless" time course X created by the simulator, along with the noisy timecourse which also includes the effect of a detector's integration time.

During the Forward-Backward and Baum-Welsh iterations Figure 2 (below) is updated to show the estimated probabilities of transitions $C_{11}(w)$ with step sizes w. You will see it start out as broad peaks and become narrow as the iterations progress. Fig. 2 also shows running values of various other parameters, including the estimated noise standard deviation and the mean dwell time.

nu = 112 quant= 1 diff = 3.877e-06

Iterations: 86

L = -1689.3888

sigma = 4.9795

a_{11} = 0.8897 (dwell = 9.0665)

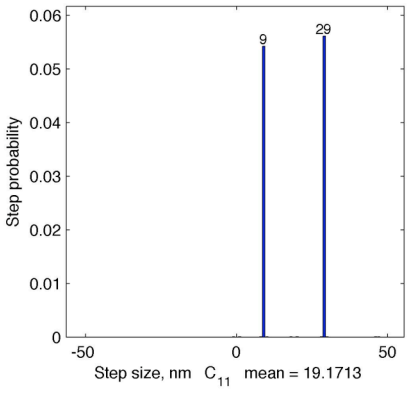
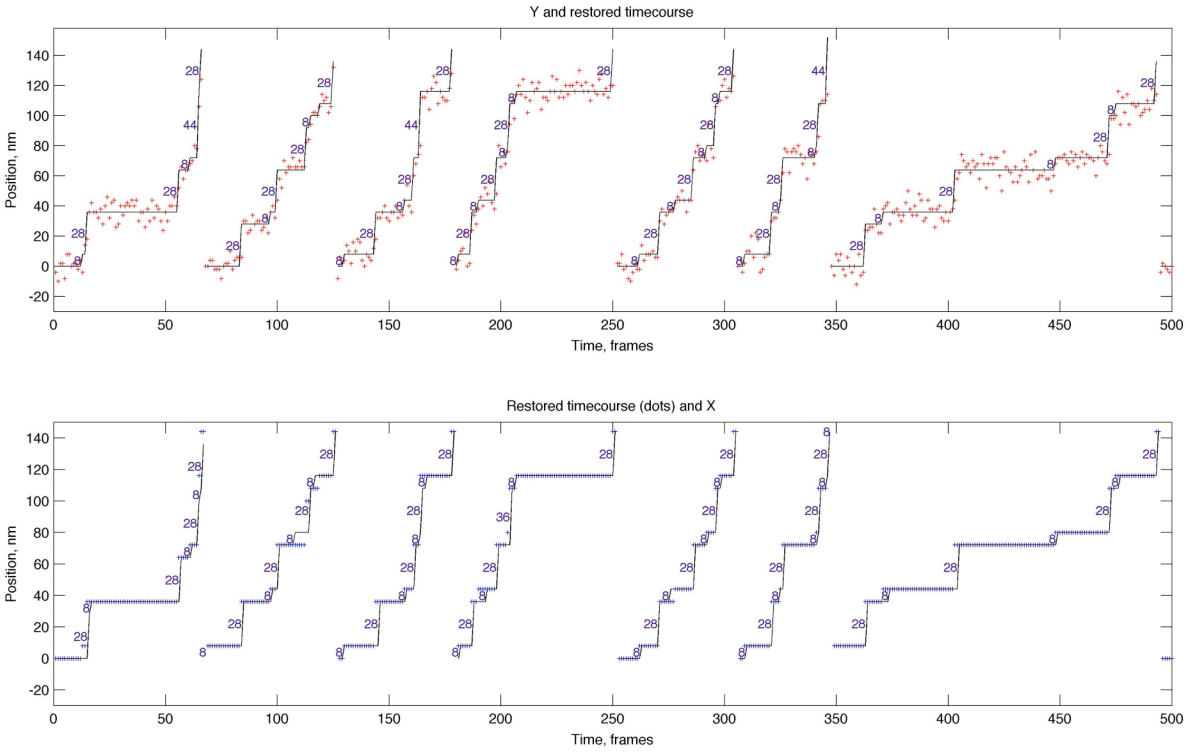


Figure 3 is generated at the end of the run, and makes a comparison between the restored motor time course (lines) and the raw data (red dots) in the upper panel. When the noiseless trace X is available, it also draws a comparison between the simulated steps (drawn as lines in the lower panel) with the restored trace (blue dots). At this level of noise (6 nm) the step sizes are



estimated quite well, but the exact timing of steps obtained in the Viterbi restoration is not always correct.

You can try changing parameters in the script. For example, you can increase the `simnoise` to say 8 (8 nm rms) in line 9. With this level of noise the estimation of step sizes becomes unreliable. If you reduce the `simnoise` to 4 or lower the HMM will start to show spurious step sizes e.g. at 18 nm due to intermediate points from the simulated integration time of the detector. This is because the script by default sets `SimpleFB`=1 in line 22, calling the simple, fast, FFT-based forward-backward implementation `ForBackF` which assumes that there is no memory in the detection system. If you set `SimpleFB=0` the slower but more rigorous function `ForwardBackward_3` is called. This function uses considerably more CPU time and memory, and it also makes use of the `.mex` files, which implement functions originally coded in C. We have set the model so that the detector duty cycle parameter $\eta = 1$, as it is in the simulated data. The rigorous forward-backward routine (which also includes the MAP estimation) does not show this problem.

You can also try analyzing your own data. Comment out the lines 8-13 that perform the simulation, put your own data into a vector `Y`, and let it run.

Some technical details. All the parameters of the HMM are stored in a "model" data structure. The step simulator and the forward-backward routines make use of the same data structure. Thus we create one model `M0` for simulation and use a different model `M` for analysis. We initialize the data structures with the function MakeMonotonicModel which describes a uni-directional, cyclic model of molecular states. For each state change, you can specify no step, a step of Gaussian-distributed size, or two different step sizes selected randomly. We use MakeMonotonicModel to make our simulated scheme `M0` with two alternating steps. We also use it to make our one-state starting model `M` having a uniform step-size distribution (specified as a Gaussian-distributed step size with infinite S.D.).

The forward-backward routines can handle gaps in the data. Occasionally some position measurements may be missing due to the loss of the fluorescence signal. For `ForBackF`, you can mark the points where the position is unknown by setting the value in `Y` to `NaN` (not a number). In `ForwardBackward_3` this is handled by passing a relative intensity vector `I0`. This gives the relative light intensity giving rise to each position value. Its values are nominally equal to 1 but approach zero at times when the number of photons collected during the frame is small. Type `help ForBackF` for example to see the built-in documentation for that function.

Fred Sigworth
19 July 2008

Below is a listing of the script

```
% MotorHMM1.m
% Analyze noisy motor protein data, using the "one state" HMM and automatic
% assignment of parameters.  fs 19 jul 08

% The user needs to provide a data trace in the vector Y, or else use the
% following 6 lines of code to make simulated data.
% --alternating steps of 8 and 28 nm (+/- .3 nm), dwell time = 10
nt=500;  % no. of time points
simnoise=6;  % noise, in nm rms
M0=MakeMonotonicModel(100, 1, simnoise, [0.1 0.1], [8 28], 0.3);
rand('state',0); randn('state',0);  % Reset the random number generators (if
desired)
M0.DutyCycle=1;  % Simulate the full CCD integration time
[X Y]=StepSimulatorC(M0,nt);  % Continuous-time simulator


% *** Analyze the data vector Y, and if given, the "noiseless" version X
% for comparison. ***

% Analysis parameters
SimpleFB = 1;  % 1: Use the simple but fast FFT-based calculation. No prior.
               % 0: use the full HMM.
quantFrac=0.5;  % Maximum size of the quantum, as fraction of noise sigma.
tol=1e-5;       % Step-to-step tolerance
maxiters=200;

% If an X vector isn't provided, just copy Y.
try ntx=numel(X); catch err; ntx=0; end;
if ntx ~= nt
    X=Y;
end;

% Pick values for the quantum and the number of position states nu.
[s n]=EstSigmaAndNu(Y);
yquantum=Step125(quantFrac*s/2.5); % yquantum is always smaller than the
estimated sigma times quantFrac.
nu=NextNiceNumber(n/yquantum,7);  % FFT-friendly value.

% quantize the data and display it.
Y1=reshape(round(Y/yquantum),nt,1);  % force it to be a column vector.
X1=reshape(round(X/yquantum),nt,1);
figure(1); clf;
plot([X1,Y1]);
xlabel('Time');
ylabel(['Position in quanta of ' num2str(yquantum) ' nm']);
drawnow;

% Starting model.  The wild guess is that the dwell time is about sqrt(nt).
M=MakeMonotonicModel(nu,yquantum,0,1/sqrt(nt),0,inf); % flat step-size
distribution
M.DutyCycle=1;       % Only used when SimpleFB=0.
M.Epsi=1/sqrt(nt);   % For MAP prior.  Only used when SimpleFB=0.

Mold=M;
d=inf;
iter=1;
figure(2);  % This figure shows the progress as we iterate.

```

```
% Main loop
while (d>tol) && (iter<maxiters)
    if SimpleFB
        [L M gamma] = ForBackF(M,Y1);  % Fast, FFT-based
    else
        [L M gamma] = ForwardBackward_3(M,Y1);
    end;
    d=RelModelChange(M,Mold);
    Mold=M;
    str=sprintf('nu = %d  quant=%4.1g   diff = %8.4g',nu,M.YQuantum,d);
    DisplayModel(M, L, iter, str, 0);
    iter=iter+1;
end;

% Done optimizing the model M.  Now get the Viterbi restored timecourse
EstY=ViterbiRestoration(M,Y1);
% Make the expanded plot(s) of the restoration with the data
figure(3); clf;
if ntx>0  % We plot the X trace, detecting steps assuming it is noiseless.
    subplot(2,1,2);
    RestorationPlotM(EstY,X1,nu,M.YQuantum,0,'b+');
    title('Restored timecourse (dots) and X');
    subplot(2,1,1);
end;
RestorationPlotM(Y1,EstY,nu,M.YQuantum);
title('Y and restored timecourse');

% Re-plot the original data with the restoration
figure(1);
plot([X1 Y1 EstY]);
legend('X', 'Y', 'Restored', 'location', 'southeast');
xlabel('Time');
ylabel(['Position in quanta of ' num2str(yquantum) ' nm']);
```