Homework 6

86-595 Neural Data Analysis
Name: Shashank Singh
Email: sss1@andrew.cmu.edu
Due: Tuesday, December 4, 2012

Problem 1

Using the previous estimate as the prior will bias the next estimated velocity toward the previous estimates, because the prior for each estimate will include the product of all previous priors. As a result, earlier estimates will play a significant role in later estimate, even after long periods of time. Changes in velocity will be estimated more slowly and to a lesser degree than they occur. As time progresses, this effect will compound, until the estimator is no longer sensitive to changes in velocity, because the prior is so strong, and the estimate will converge on a steady state velocity.

Problem 2

---

a. See Figures 1, 2 and 3. Blue plots are x-coordinates, and red plots are y-coordinates. Solid lines are actual, velocities and dotted lines are predicted velocities.

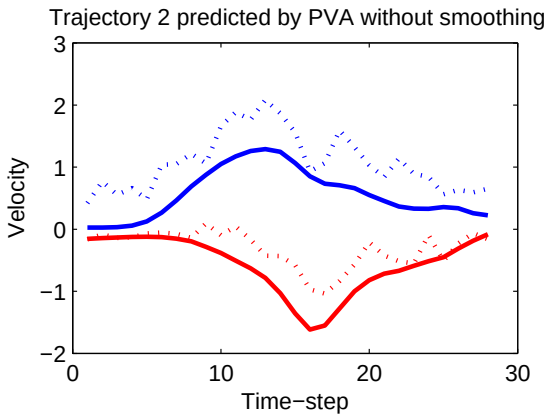

Figure 1: No smoothing.

b. See Figures 4, 5 and 6.

c. File sent in email.

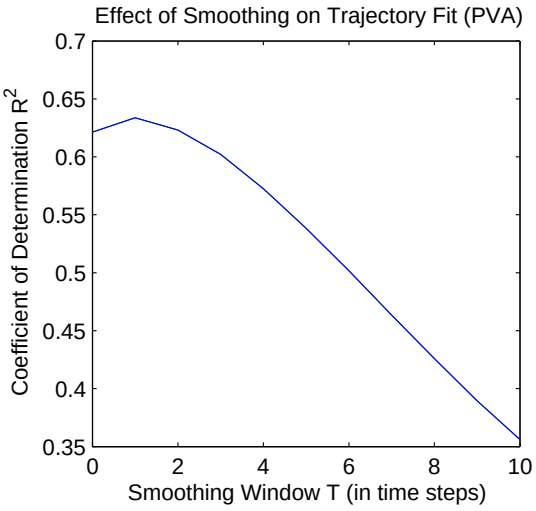

Figure 2: R^2 value for each smoothing window size

Problem 3

Question 1 (concerning Kalman filters) seemed pretty apparent after thinking for a few minutes about how the estimator would work. Of the programming section, part 2a. took most of the time, since PVA has a few steps. Part 2b, on the other hand, was just one line of code on top of a few lines for smoothing, so that, after implementing smoothing for part 2a., part 2b., took only a couple of minutes.

Since I wasn't able to spend much time on part 2c., I didn't end up implementing a Kalman filter or a Poisson MLE estimator and just used my OLE implementation from part 2b. But I imagine that, if I had more time, trying to implement a better estimator would have been informative. I might have liked a bit more theory to help understand the details of OLE and Kalman filters, or perhaps to derive the MLE hinted at in part 2c.

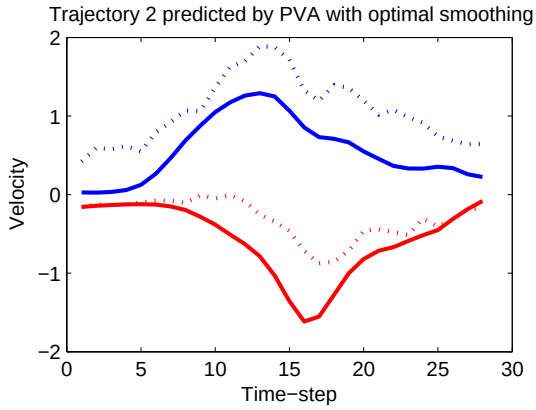

Figure 3: Optimal smoothing

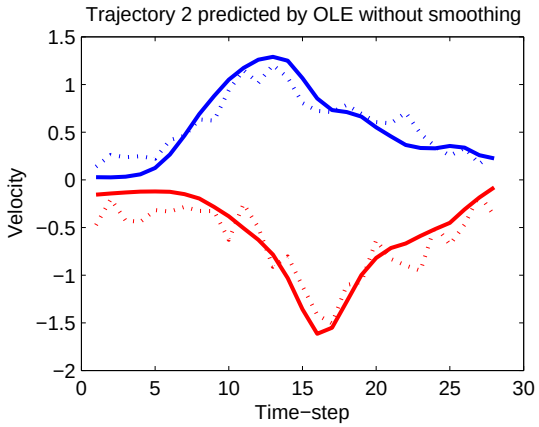

Figure 4: No smoothing.

Code

PVA implementation:

```
function pred_traj = PVA(spikes, cosine_fit, dt, T)

  [neurons times] = size(spikes);
  b0 = cosine_fit.b0s;
  B = cosine_fit.Bs;

  % use Euclidean norm of each b as modulation depth
  mod = sqrt(diag(B * B'));
```

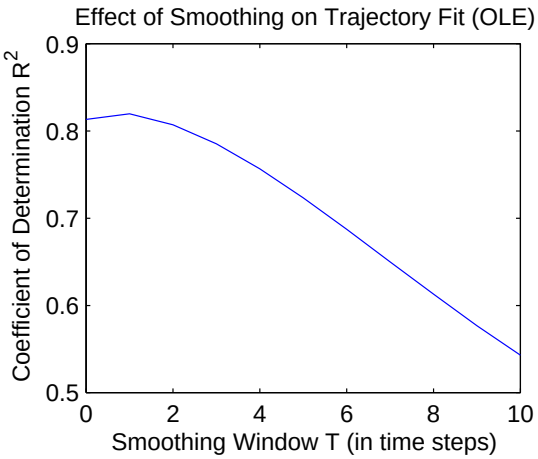

Figure 5: R^2 value for each smoothing window size

```
  % normalize firing rates at each time step
  normed = zeros(size(spikes));
  for time = 1:times
    normed(:,time) = (spikes(:,time) - b0) ./ mod;
  end

  % smooth firing rates; traverse normed backward
  for time = times:-1:1
    normed(:,time) = mean(normed(:,max(1,time - T):time), 2);
  end

  % pool firing rates at each time step
  pred_traj = zeros(times, 2);
  for time = 1:times
    pred_traj(time, :) = (normed(:,time)') * B / neurons;
  end
end
```

OLE implementation:

```
function pred_traj = OLE(spikes, cosine_fit, dt, T)
  [~, times] = size(spikes);
  B = cosine_fit.Bs;
  sigma = cosine_fit.sigma;

  % smooth firing rates; traverse spikes backward
  for time = times:-1:1
    spikes(:,time) = mean(spikes(:,max(1,time - T):time), 2);
  end
```

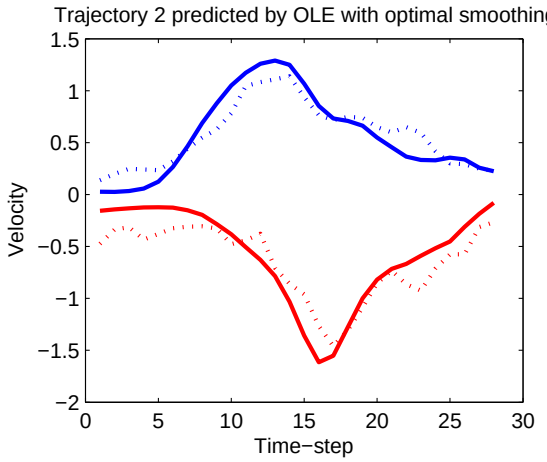

Figure 6: Optimal smoothing

```
 % decoding matrix
 G = inv\left( B') * inv(sigma) * B) * (B') * inv(sigma);

 pred_traj = (G * spikes)';
\right)
```