1 Introduction

The classical Wiener filter is used to produce a minium mean squared error estimate of a process. That is, suppose that one has a process, $y[n]$, that can be represented as follows:

$$y[n] = x[n] + e[n],$$

where $x[n]$ represents the noiseless signal and $e[n]$ is the noise. The goal is to estimate $\hat{x}[n]$ in order to minimize:

$$E\{\mid x[n] - \hat{x}[n] \mid^2\}.$$

Suppose that instead of filtering the noisy signal to estimate the true value of $x[n]$, one wishes to predict a future value of the signal. The filtering problem can be cast into a linear prediction problem by setting $x[n] = y[n+1]$. Finding the weights for this filter requires solving the Wiener-Hopf system of equations:

$$\begin{bmatrix} r_x(0) & r_x^*(1) & r_x^*(2) & \dots & r_x^*(p-1) \\ r_x(1) & r_x(0) & r_x^*(1) & \dots & r_x^*(p-2) \\ r_x(2) & r_x(1) & r_x(0) & \dots & r_x^*(p-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_x(p-1) & r_x(p-2) & r_x(p-3) & \dots & r_x(0) \end{bmatrix} \begin{bmatrix} w(0) \\ w(1) \\ w(2) \\ \vdots \\ w(p-1) \end{bmatrix} = \begin{bmatrix} r_x(1) \\ r_x(2) \\ r_x(3) \\ \vdots \\ r_x(p) \end{bmatrix}. \tag{1}$$

Derivation of the Wiener filter for filtering and linear prediction of univariate data can be found in Hayes, 1996 (section 7.2).

Now, suppose one has multivariate/multivariate data—data consisting of multiple signals—and wants to predict the future values for each of these signals. This can be done using the Levinson-Wiggins-Robinson algorithm. For derivation of this algorithm, see Benesty et al., 2008. I will simply pose the problem and summarize the algorithm.

2 Solving Forward Linear Prediction Problem using Multivariate Wiener Filter

Suppose that one has an M-channel time series

$$\chi(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_M(t) \end{bmatrix}$$

where t represents the time index. The goal is to minimize the foward prediction error, given by:

$$\mathbf{E}_{f,L} = \chi(k) - \hat{\chi}(k) \tag{2}$$

Since $\hat{\chi}$ is a linear combination of previous time series values and filter coefficients, it can be represented as

$$\hat{\chi} = \sum_{l=1}^{L} \mathbf{A}_{L,l}\chi(k-l) = \mathbf{A}^T\mathbf{x}(k-1),$$

where l represents the time lag index and L is the number of previous values taken into account;

$$\mathbf{A}_L = \begin{bmatrix} \mathbf{A}_{L,1} \\ \mathbf{A}_{L,2} \\ \vdots \\ \mathbf{A}_{L,L} \end{bmatrix}$$

is an $ML \times M$ forward prediction matrix, with each sub-block $\mathbf{A}_{L,l}$ being an $M \times M$ matrix of prediction coefficients; and

$$\mathbf{x}(k-1) = \begin{bmatrix} \chi(k-1) \\ \chi(k-2) \\ \vdots \\ \chi(k-L) \end{bmatrix}$$

is an ML length vector. Thus, we can write

$$\mathbf{E}_{f,L}(k) = \chi(k) - \mathbf{A}_L^T \mathbf{x}(k-1) \tag{3}$$

To find the optimal forward Wiener filter coefficient matrix, $\mathbf{A}_L^*$, it is necessary to minimize the mean squared error (MSE),

$$J_f(\mathbf{A}_L) = E\{\mathbf{e}_{f,L}^T(k)\mathbf{e}_{f,L}(k)\}. \tag{4}$$

This can be done by solving the multivariate Wiener-Hopf equations,

$$\mathbf{R}_L \mathbf{A}_L^* = \mathbf{R}_f(1/L); \tag{5}$$

where

$$\begin{aligned} \mathbf{R}_L &= E\{\mathbf{x}(k-1)\mathbf{x}^T(k-1)\} \\ &= E\{\mathbf{x}(k)\mathbf{x}^T(k)\} \\ &= \begin{bmatrix} \mathbf{R}(0) & \mathbf{R}(1) & \mathbf{R}(2) & \dots & \mathbf{R}(L-1) \\ \mathbf{R}^T(1) & \mathbf{R}(0) & \mathbf{R}(1) & \dots & \mathbf{R}(L-2) \\ \mathbf{R}^T(2) & \mathbf{R}^T(1) & \mathbf{R}(0) & \dots & \mathbf{R}(L-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{R}^T(L-1) & \mathbf{R}^T(L-2) & \mathbf{R}^T(L-3) & \dots & \mathbf{R}(0) \end{bmatrix} \end{aligned} \tag{6}$$

is composed of $M \times M$ autocorrelation matrix block elements of the form

$$\mathbf{R}(l) = E\{\chi(k)\chi^T(k-l)\} \quad \text{and}$$

$$\mathbf{R}(-l) = E\{\chi(k-l)\chi^T(k)\} = \mathbf{R}^T(l), \quad \text{for } l = 0, 1, \dots, L-1,$$

and

$$\begin{aligned} \mathbf{R}_f(1/L) &= E\{\mathbf{x}(k-1)\chi^T(k)\} \\ &= \begin{bmatrix} \mathbf{R}(1) \\ \mathbf{R}(2) \\ \vdots \\ \mathbf{R}(L) \end{bmatrix}. \end{aligned}$$

3 Levinson-Wiggins-Robinson Algorithm Summary

To summarize the algorithm, I will need to introduce a few more matrices involved in the recursion. For a detailed understanding of their derivation, see Benesty et al., 2008.

So far, I have introduced $\mathbf{R}(l)$, $\mathbf{R}_f(l)$, $\mathbf{E}_{f,l}$, and $\mathbf{A}_l^*$ in relation to the forward prediction. However, this algorithm also heavily relies on backward predictors. Below, I have listed all the matrices related the backward prediction relevant in the implementation of this algorithm:

1. $$\mathbf{B}_L = \begin{bmatrix} \mathbf{B}_{L,1} \\ \mathbf{B}_{L,2} \\ \vdots \\ \mathbf{B}_{L,L} \end{bmatrix}$$
   is an $MLxM$ backward prediction matrix, with each sub-block $\mathbf{B}_{L,l}$ being an $M \times M$ matrix of prediction filter coefficients.
2. $\mathbf{E}_{b,L} = E\{\mathbf{e}_{b,L}^*(k)\mathbf{e}_{b,L}^{*T}(k)\} = \mathbf{R}(0) - \mathbf{R}_b^T(1/L)\mathbf{B}_L^*$ is the backward error matrix of size $M \times M$. (Note: $\mathbf{R}_b(1/L)$ is the same as $\mathbf{R}_f(1/L)$, but with it's elements transposed).
3. $\mathbf{K}_{b,L} = \mathbf{R}(L) - \mathbf{R}_f^T(1/L-1)\mathbf{B}_{L-1}^*$ (Note: $\mathbf{K}_{b,L} = \mathbf{K}_{f,L}^T$).

With this information in mind, the recursion can be implemented as follows:

3.1 Initialization

$$\mathbf{E}_{f,0} = \mathbf{E}_{b,0} = \mathbf{R}(0)$$

3.2 First loop of Recursion: $l = 1$

$$\begin{aligned} \mathbf{K}_{b,l} &= R(l) \\ \mathbf{A}_l^* &= \mathbf{0}_{M\times M} + \mathbf{I}_{M\times M}\mathbf{E}_{b,l-1}^{-1}\mathbf{K}_{b,l}^T \\ \mathbf{B}_l^* &= \mathbf{0}_{M\times M} + \mathbf{I}_{M\times M}\mathbf{E}_{f,l-1}^{-1}\mathbf{K}_{b,l} \\ \mathbf{E}_{f,l} &= \mathbf{E}_{f,l} - \mathbf{K}_{b,l}\mathbf{E}_{b,l-1}^{-1}\mathbf{K}_{b,l}^T \\ \mathbf{E}_{b,l} &= \mathbf{E}_{b,l} - \mathbf{K}_{b,l}^T\mathbf{E}_{f,l-1}^{-1}\mathbf{K}_{b,l} \end{aligned}$$

3.3 Remainder of Recursion: $l = 2, 3, ..., L$

$$\begin{aligned} \mathbf{K}_{b,l} &= R(l) - \mathbf{R}_f^T(1:l-1)\mathbf{B}_{l-1}^* \\ \mathbf{A}_l^* &= \begin{bmatrix} \mathbf{A}_{l-1}^* \\ \mathbf{0}_{M\times M} \end{bmatrix} - \begin{bmatrix} \mathbf{B}_{l-1}^* \\ -\mathbf{I}_{M\times M} \end{bmatrix} \mathbf{E}_{b,l-1}^{-1}\mathbf{K}_{b,l}^T \\ \mathbf{B}_l^* &= \begin{bmatrix} \mathbf{0}_{M\times M} \\ \mathbf{B}_{l-1}^* \end{bmatrix} - \begin{bmatrix} -\mathbf{I}_{M\times M} \\ \mathbf{A}_{l-1}^* \end{bmatrix} \mathbf{E}_{f,l-1}^{-1}\mathbf{K}_{b,l} \\ \mathbf{E}_{f,l} &= \mathbf{E}_{f,l} - \mathbf{K}_{b,l}\mathbf{E}_{b,l-1}^{-1}\mathbf{K}_{b,l}^T \\ \mathbf{E}_{b,l} &= \mathbf{E}_{b,l} - \mathbf{K}_{b,l}^T\mathbf{E}_{f,l-1}^{-1}\mathbf{K}_{b,l} \end{aligned}$$

4 Levinson-Wiggins-Robinson Algorithm Applied to Neuron Brain Data

The Levinson-Wiggins-Robinson algorithm was applied to a dataset of neuron positions. The data consists of different neurons' (x, y)-positions tracked across time. The bivariate form of the algorithm can be applied to any neuron's time series. I took one series and tested the prediction accuracy of this algorithm on it

by comparing the predicted and actual neuron location at various points in time, varying the number of previous frames taken into account to make the prediction. Figures 1 and 2, below, illustrate these results.

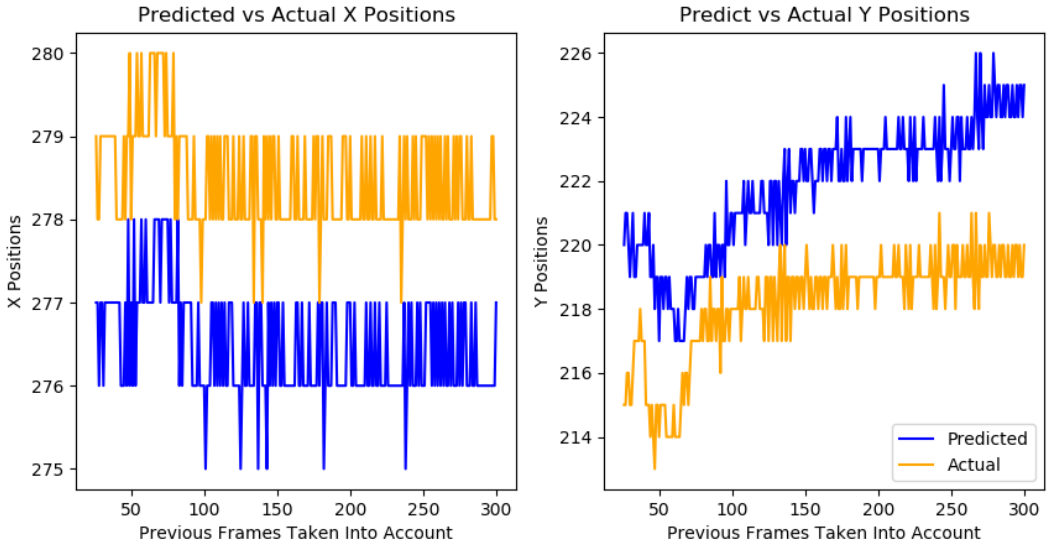

Figure 1: Comparison between actual and predicted nueron positions, varying the number of frames taken into account

Figure 1 illustrates the actual and predicted x and y coordinates of the neurons on different frames. As I move from one frame to the next in these calculations, I continue to update the sample; so as we move through future frames, I increase the number of previous frames used to predict future neuron locations. The shapes of the graphs cooresponding with the actual positions clearly match those of the predicted neuron positions. However, there is displacement between the actual and predicted neuron location graphs. The x-coordinate predictions are too low, while the y-coordinate predictions are too high. This leads me to believe that there may be a bug in the code. I will review the code for this bug and make the necessary adjustments.

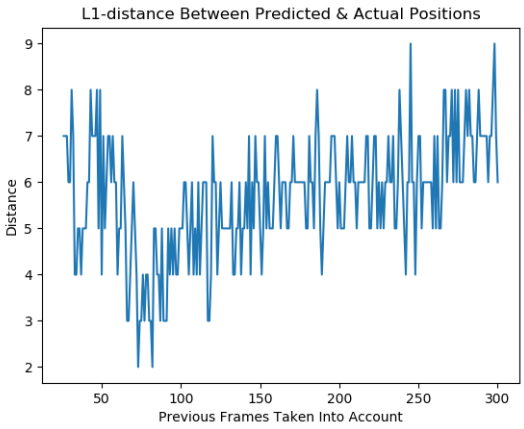

Figure 2: The L1-norm taken the actual and predicted neuron positions, varying the number of frames taken into account

When looking at Figure 2, one can see that the error is lowest when taking around 75 frames into account in making the prediction. However, I am skeptical that this is actually the case. That is, I would expect the results to become more accurate as the number of frame taken into account increases. This is likely related to the same bug that plagues Figure 1.

5 References

[1] M.H. Hayes, Statistical Digital Signal Processing and Modeling, Wiley, 1996.

[2] J. Benesty, M.M. Sondhi, Y.A. Huang, Springer Handbook of Speech Processing, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.