Recursive Least Squares For Multivariate Arbitrary Signals

Franco Marchesoni-Acland[a,c],
[a] Laboratorio de Energía Solar, Universidad de la República (Udelar), Uruguay
[c] Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República, Uruguay

***Abstract*—We found that the Recursive Least Squares filter has an easier more general formulation than that of the books. The deduction is presented hereafter. This results in an linear adaptive filter that predicts Y from X and evolves with time. Implementation is also provided.**

***Index Terms*—Adaptive filters, Least Squares, Recursive Least Squares, Kalman Filter.**

I. Introduction

We will be treating discrete time signals, namely the input $X(t)$ and the target $Y(t)$. These quantities can be vectors, thus we will represent matrices in bold font. We want to find a linear function $f(X(t)) = \boldsymbol{W^T} X(t)$ that approximates well $Y(t)$ in an exponentially weighted squared error sense. What use do we make of this formulation if $X(t)$ and $Y(t)$ are measured, discovered or available at the same time t? The caveat here is that t is just the mathematical index, so many uses arises from wisely choosing the signals X and Y. The filter heavily uses the forgetting factor λ, that is introduced in Eq. (2). When $\lambda = 1$ least squares without weighting is recovered.

A. Examples

1) One step ahead predictor: For instance, the traditional formulation of the RLS filter is given as a one step predictor. If we have an univariate signal $s(n)$ and we want to forecast $s(n+1)$ then we can put $X(t) = s(n-1)$ and $Y(t) = s(n)$ for every n, run the filter, and when desired we can make the forecast using the last computed coefficients and the last measured signal as $\hat{s}(n_{\text{base}} + 1) = \boldsymbol{W}^{\boldsymbol{T}}_{\text{last}} s(n_{\text{base}})$.

2) Multiple steps ahead predictor: Predicting multiple steps ahead is as easy as generalizing the last approach to an arbitrary lead time h, this is, $X(t) = s(n-h), Y(t) = s(n)$.

3) AR filter: Many times one want to use the well known AutoRegressive (AR) model to make forecasts and now vectors come handy. Define $X(t) = [s(n-1), s(n-2), s(n-3)]^T$ and $Y(t) = s(n)$ and run the filter. This can be generalized to ARMAX processes, provided that the innovations or errors and the exogenous variables are known.

4) 2D inputs: This is not restricted to one dimensional problems, just to one dimensional vectors. Consider the linear transformation of an image of a sequence into some image of another sequence. Now, flatten those images and you can use this filter. It will not necessarily work well however, images are complicated!

B. Kalman Filter

This could be seen as an special case of the Kalman filter. We make the analogy for scalar outputs for simplicity.

Remember the traditional state space model formulation:

$$\boldsymbol{x_{k+1}} = \boldsymbol{A_k x_k} + \boldsymbol{w_k}$$

$$\boldsymbol{y_k} = \boldsymbol{B_k x_k} + \boldsymbol{v_k}$$

where $\boldsymbol{w_k}, \boldsymbol{v_k}$ represent gaussian noise. Now rewrite $\boldsymbol{x_k} = \boldsymbol{W_k}, \boldsymbol{A_k} = \boldsymbol{I}, \boldsymbol{\Sigma_w} = \boldsymbol{0}, \boldsymbol{B_k} = X(t)^T, y_k = Y(t)^T$ and $\sigma_v = \lambda$. The Kalman algorithm equations now give us virtually the same equations. Careful deduction follows.

Algorithm 1 Recursive Least Squares Algorithm

Initialization
$\boldsymbol{W_0} \leftarrow \boldsymbol{0}$
$P(0) \leftarrow b\boldsymbol{I}$ (b could be large)

When new data arrives
$X(n), Y(n) \leftarrow$ new data
$K(n) \leftarrow \frac{\boldsymbol{P(n-1)} X(n)}{\lambda + X(n)^T \boldsymbol{P(n-1)} X(n)}$
$\boldsymbol{P(n)} \leftarrow \frac{1}{\lambda}\left(\boldsymbol{P(n-1)} - K(n) X(i)^T \boldsymbol{P(n-1)}\right)$
$\boldsymbol{W_n} \leftarrow \boldsymbol{W_{n-1}} + K(n)\left(Y(n)^T - X(n)^T \boldsymbol{W_{n-1}}\right)$

Prediction
$\hat{Y}(u) \leftarrow \boldsymbol{W_n^T} X(u)$

II. Deduction

Let's minimize the following error

$$C(\boldsymbol{W_n}) = \sum_{i=0}^{n} \lambda^{n-i} e^2(i), \tag{2}$$

being $e(i)$ the prediction error of observation i (we shall use "i" as the index now). Define the error $e(i) = \boldsymbol{W_n^T} X(i) - Y(i)$, being $X(i)$ a vector including all input variables.

Differentiating and equaling to zero:

$$\frac{\mathrm{d}C(\boldsymbol{W_n})}{\mathrm{d}\boldsymbol{W_n}} = \sum_{i=0}^{n} 2\lambda^{n-i} X(i)\left[X(i)^T \boldsymbol{W_n} - Y(i)^T\right] = 0$$

$$\Rightarrow \sum_{i=0}^{n} \lambda^{n-i} X(i) X(i)^T \boldsymbol{W_n} = \sum_{i=0}^{n} \lambda^{n-i} X(i) Y(i)^T$$

that can be shown equivalent to:

$$\left(\underbrace{\sum_{i=0}^{n}\lambda^{n-i}X(i)X(i)^T}_{\boldsymbol{\Theta(n)}}\right)\boldsymbol{W_n}=\underbrace{\sum_{i=0}^{n}\lambda^{n-i}X(i)Y(i)^T}_{\boldsymbol{r(n)}},$$

identifying $\boldsymbol{\Theta(n)}$ the weighted sample covariance matrix for X and being $\boldsymbol{r(n)}$ the estimate of the cross-covariance between Y and X. So we have: $\boldsymbol{\Theta(n)W_n}=\boldsymbol{r(n)}$. Now note that we could write Θ as a recursion:

$$\boldsymbol{\Theta(n)}=X(n)X(n)^T+\lambda\underbrace{\sum_{i=0}^{n-1}\lambda^{n-1-i}X(i)X(i)^T}_{\boldsymbol{\Theta(n)}}. \quad (1)$$

Remembering the Woodbury Matrix Identity:

$$(A+UCV)^{-1}=A^{-1}-A^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1},$$

and identifying right hand side of Eq. (1) as $A+UCV$

$$A^{-1}=\frac{1}{\lambda}\boldsymbol{\Theta^{-1}(n-1)},\quad U=X(n),\quad C=\boldsymbol{I},\quad V=X(n)^T,$$

we can invert $\boldsymbol{\Theta(n)}$, obtaining,

$$\begin{aligned}\boldsymbol{\Theta^{-1}(n)}&=\left(\lambda\boldsymbol{\Theta(n)}+X(n)X(n)^T\right)^{-1}=(A+UCV)^{-1}\\&=A^{-1}-A^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1}\\&=\frac{1}{\lambda}\left(\boldsymbol{P(n-1)}-\underbrace{\frac{\boldsymbol{P(n-1)}X(n)}{\lambda+X(n)^T\boldsymbol{P(n-1)}X(n)}}_{K(n)}X(n)^T\boldsymbol{P(n-1)}\right),\end{aligned}$$

where $\boldsymbol{P(n-1)}=\boldsymbol{\Theta^{-1}(n-1)}$ and $\boldsymbol{P(n)}=\boldsymbol{\Theta^{-1}(n)}$.

So we have:

$$\boldsymbol{P(n)}=\frac{1}{\lambda}\left(\boldsymbol{P(n-1)}-K(n)X(n)^T\boldsymbol{P(n-1)}\right).$$

Playing with $K(n)$:

$$\begin{aligned}K(n)&=\frac{\lambda^{-1}\boldsymbol{P(n-1)}X(i)}{1+\lambda^{-1}X(i)^T\boldsymbol{P(n-1)}X(i)}\\\Rightarrow K(n)\left(1+\lambda^{-1}X(i)^T\boldsymbol{P(n-1)}X(i)\right)&=\lambda^{-1}\boldsymbol{P(n-1)}X(i)\\\Rightarrow K(n)=\lambda^{-1}\left(\boldsymbol{P(n-1)}-K(n)X(i)^T\boldsymbol{P(n-1)}\right)X(i)&\\=\boldsymbol{P(n)}X(i)&\end{aligned}$$

From the definition of $\boldsymbol{r(n)}$:

$$\begin{aligned}\boldsymbol{r(n)}=\sum_{i=0}^{n}\lambda^{n-i}X(i)Y(i)^T&=X(n)Y(n)^T+\lambda\sum_{i=0}^{n-1}\lambda^{n-1-i}X(i)Y(i)^T\\&=\lambda\boldsymbol{r(n-1)}+X(n)Y(n)^T\end{aligned}$$

Recalling $\boldsymbol{\Theta(n)W_n}=\boldsymbol{r(n)}$ and replacing $\boldsymbol{r(n)}$:

$$\begin{aligned}\boldsymbol{W_n}&=\boldsymbol{P(n)r(n)}\\\Rightarrow\boldsymbol{W_n}&=\lambda\boldsymbol{P(n)r(n-1)}+\boldsymbol{P(n)}X(n)Y(i)^T\\&=\boldsymbol{W_{n-1}}-K(n)X(i)^T\boldsymbol{W_{n-1}}+\boldsymbol{P(n)}X(n)Y(n)^T\\&=\boldsymbol{W_{n-1}}-K(n)X(i)^T\boldsymbol{W_{n-1}}+K(n)Y(n)^T\\&=\boldsymbol{W_{n-1}}+K(n)\left(Y(n)^T-X(n)^T\boldsymbol{W_{n-1}}\right)\end{aligned}$$

That is the coefficient update summarized in Algorithm 1. Summing up, the equations are

$$K(n)=\frac{\lambda^{-1}\boldsymbol{P(n-1)}X(n)}{1+\lambda^{-1}X(n)^T\boldsymbol{P(n-1)}X(n)}$$

$$\boldsymbol{P(n)}=\frac{1}{\lambda}\left(\boldsymbol{P(n-1)}-K(n)X(n)^T\boldsymbol{P(n-1)}\right)$$

$$\boldsymbol{W_n}=\boldsymbol{W_{n-1}}+K(n)\left(Y(n)^T-X(n)^T\boldsymbol{W_{n-1}}\right)$$