

A New Approach to Adaptive Signal Processing

Muhammad Ali Raza Anjum

Department of Electrical Engineering, Army Public College of Management and Sciences,
Rawalpindi, Pakistan
e-mail: ali.raza.anjum@apcoms.edu.pk

Abstract

A unified linear algebraic approach to adaptive signal processing (ASP) is presented. Starting from just $Ax=b$, key ASP algorithms are derived in a simple, systematic, and integrated manner without requiring any background knowledge to the field. Algorithms covered are Steepest Descent, LMS, Normalized LMS, Kaczmarz, Affine Projection, RLS, Kalman filter, and MMSE/Least Square Wiener filters. By following this approach, readers will discover a synthesis; they will learn that one and only one equation is involved in all these algorithms. They will also learn that this one equation forms the basis of more advanced algorithms like reduced rank adaptive filters, extended Kalman filter, particle filters, multigrid methods, preconditioning methods, Krylov subspace methods and conjugate gradients. This will enable them to enter many sophisticated realms of modern research and development. Eventually, this one equation will not only become their passport to ASP but also to many highly specialized areas of computational science and engineering.

Keywords: adaptive signal processing, adaptive filters, adaptive algorithms, Kalman filter, RLS, Wiener filter

1. Introduction

In last few decades, Adaptive Signal Processing (ASP) has emerged into one of the most prolific areas in Electrical and Computer Engineering (ECE). It is based on the concept of intelligent systems that can automatically adapt to changes in their environment without the need for manual intervention [1]. ASP is gaining popularity with each passing day and is forming the basis of many key future technologies: including robots, gyroscopes, power systems, e-health systems, communication networks, audio and video technologies, etc [2-14]. As a result, ASP has become very important from both practical and pedagogical viewpoints. Therefore, its importance can hardly be over-emphasized.

But before one can enter into the realm of ASP, there are a lot of problems to face. To begin with, ASP lacks a unified framework. Numerous approaches to ASP can be found in literature and each of them tackles ASP in its own particular way. Secondly, the notation is mostly author specific. This makes it very difficult for the reader to understand even a single concept from two different manuscripts. Thirdly, almost every author leaves something out. For example, most authors drop the Kalman filter which has been recognized as one of most crucial topics in postgraduate research. Fourthly, authors make no attempt at making the concepts portable. Finally, the subject has a heavy probabilistic/statistical outlook which is overemphasized most of the time.

This work will provide a unified linear algebraic approach to ASP. Starting from just one equation in one unknown and one observation, all the key ASP algorithms - from Steepest Descent to Kalman filter - will be derived. During the derivation process, notation of algorithms will be kept uniform to make them consistent. Transitions from one algorithm to other will be kept systematic to make them portable. Probability and statistics shall not be invoked to make the algorithms accessible as well. The treatment of ASP will be entirely linear algebraic. Moreover, connection of ASP to other highly specialized domains like extended Kalman filter, reduced-rank adaptive filters, particle filters, multigrid methods, pre-conditioning methods, and Krylov subspace methods will also be established.

2. System Model

Let a system of linear equations be described as:

$$\mathbf{A}_m \mathbf{x} = \mathbf{b}_m \quad (1)$$

With,

$$\mathbf{A}_m = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_m^T \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{b}_m = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$\mathbf{x}$ represents the unknown system parameters. $\mathbf{A}_m$ contains the rows of the input data $\mathbf{a}_i$'s that are applied to this system. $\mathbf{b}_m$ comprises of the observations b_i 's at the output of the system. $\mathbf{x}$ is an $(n \times 1)$ vector. $\mathbf{A}_m$ is an $(m \times n)$ matrix. $\mathbf{b}_m$ is an $(m \times 1)$ vector. According to this nomenclature, m inputs have been applied to the system with n unknown parameters and so far m outputs have been observed.

3. Least Mean Squares (LMS) Algorithm

Beginning with $m = 1$, there is one equation and one observation b_1 .

$$\mathbf{a}_1^T \mathbf{x} = b_1 \quad (2)$$

Equation (2) seeks a vector $\mathbf{x}$ that has a projection of magnitude b_1 over the vector $\mathbf{a}_1$. We begin with an arbitrary guess, say $\mathbf{x}[\mathbf{k}]$ at time-step k . Since our guess is purely arbitrary, the projection of $\mathbf{x}[\mathbf{k}]$ over $\mathbf{a}_1$ may not be equal to b_1 , i.e., $\mathbf{a}_1^T \mathbf{x}[\mathbf{k}] \neq b_1$. There will be a difference. This difference can be denoted by an error $e[\mathbf{k}]$.

$$e[\mathbf{k}] = b_1 - \mathbf{a}_1^T \mathbf{x}[\mathbf{k}] \quad (3)$$

$e[\mathbf{k}]$ is the error in projection of $\mathbf{x}[\mathbf{k}]$ over $\mathbf{a}_1$. It lies in the direction of $\mathbf{a}_1$. The complete error vector $\mathbf{e}[\mathbf{k}]$ will be $e[\mathbf{k}]\mathbf{a}_1$. This error can be added to the initial guess $\mathbf{x}[\mathbf{k}]$ to rectify it. But the question is how much of it to add? Let there be a scalar μ in the range of $[0,1]$ such that the error vector is multiplied with μ before adding it to $\mathbf{x}[\mathbf{k}]$. A value of $\mu = 0$ indicates that no correction is required and the initial guess is accurate whereas a value of $\mu = 1$ indicates that a full correction is required.

$$\mathbf{x}[\mathbf{k} + 1] = \mathbf{x}[\mathbf{k}] + \mu e[\mathbf{k}]\mathbf{a}_1 \quad (4)$$

$\mathbf{x}[\mathbf{k} + 1]$ is the updated vector. Substituting Equation (3) in Equation (4).

$$\mathbf{x}[\mathbf{k} + 1] = \mathbf{x}[\mathbf{k}] + \mu(b_1 - \mathbf{a}_1^T \mathbf{x}[\mathbf{k}])\mathbf{a}_1 \quad (5)$$

If k observations have been received up till the time-step,

$$\mathbf{x}[\mathbf{k} + 1] = \mathbf{x}[\mathbf{k}] + \mu(b_k - \mathbf{a}_k^T \mathbf{x}[\mathbf{k}])\mathbf{a}_k \quad (6)$$

Equation (6) represents the famous LMS algorithm. The parameter μ in Eq. (6) is known as the step-size. It plays a crucial role in the convergence of the LMS algorithm. A very small step-size makes the algorithm crawl towards the true solution and hence terribly slows it down. This phenomenon is known as *lagging*. A large step-size makes the algorithm leap towards the true solution. Though it will make the algorithm much faster, movement in large steps never allows the algorithm to approach the true solution within a close margin. The algorithm keeps jumping around the true value but never converges to it and a further decrease in error becomes impossible. This phenomenon is known as *misadjustment* [1]. Effect of these phenomena on the convergence properties of LMS algorithm are illustrated in Figure 1. These effects arise due to the fact that the step-size of LMS algorithm has to be adjusted manually. This is its major drawback. But inherent simplicity of LMS algorithm still keeps it much popular.

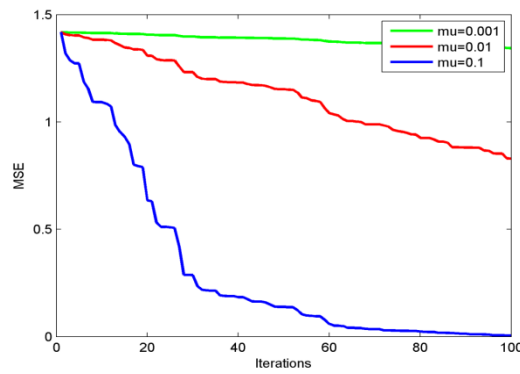


Figure 1. Comparison of the convergence properties of LMS algorithm with different step-sizes. LMS was employed to identify a system with an impulse response of length $n = 5$

4. Normalized Least Mean Squares (NLMS) Algorithm

NLMS provides an automatic adjustment in step-size. It is based on the criteria of selecting the best step-size for a given iteration. The term best is explained as follows. If error during the iteration is large, step-size is kept large so that the algorithm can quickly catch up with true solution. If the error decreases, step-size is lowered to allow the algorithm to zoom into the true solution. Hence, NLMS tries to select a step-size that minimizes the error in each iteration. In order to show how NLMS achieves it, we have to re-consider Equation (2). It represents the projection of vector x over the vector a_1 such that this projection has magnitude equal to b_1 . But this is not the usual definition of dot product that represents orthogonal projections [15]. Actual definition requires the normalization of a_1 in Equation (2).

$$\frac{a_1^T x}{a_1^T a_1} = \frac{b_1}{a_1^T a_1} \quad (7)$$

Orthogonal projections make sure that error is orthogonal and hence, minimum. This in turn appears as a constraint on the step-size as we show now. Continuing in the similar fashion by choosing an arbitrary vector $x[k]$, the projection may not necessarily equal the right hand side and there will again be a difference.

$$\frac{b_1}{a_1^T a_1} - \frac{a_1^T x[k]}{a_1^T a_1} = \frac{1}{a_1^T a_1} (b_1 - a_1^T x[k]) = \frac{1}{a_1^T a_1} e[k]$$

Where,

$$e[k] = (b_1 - a_1^T x[k]) \quad (8)$$

The error vector will be,

$$e[k] = \frac{1}{a_1^T a_1} e[k] a_1 \quad (9)$$

Adding this correction to the original guess in order to improve it for the next iteration $x[k+1]$,

$$x[k+1] = x[k] + \frac{1}{a_1^T a_1} e[k] a_1 \quad (10)$$

Substituting Equation (8) in Equation (10),

$$x[k+1] = x[k] + \frac{1}{a_1^T a_1} (b_1 - a_1^T x[k]) a_1 \quad (11)$$

Or in general if k observations have been received up till the time-step,

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \frac{1}{\mathbf{a}_k^T \mathbf{a}_k} (b_k - \mathbf{a}_k^T \mathbf{x}[k]) \mathbf{a}_k \quad (12)$$

Equation (12) represents the NLMS algorithm. Comparing Equation (12) with Equation (6), it can be observed that,

$$\mu = \frac{1}{\mathbf{a}_k^T \mathbf{a}_k} \quad (13)$$

In contrast to the LMS algorithm, step size of the NLMS algorithm is a variable which is automatically adjusted for each input row of data $\mathbf{a}_k$ according to its norm $\mathbf{a}_k^T \mathbf{a}_k$. By this automatic adjustment of step size, NLMS is able to avoid the problems of lagging and misadjustment that plague the LMS algorithm. Therefore, it has better convergence properties than LMS algorithm. Since NLMS achieves this advantage by the normalization step performed in Eq. (7), hence follows the name normalized LMS [1].

5. Kaczmarz Algorithm

Equation (11) for NLMS algorithm can be re-written as:

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \frac{1}{\mathbf{a}_k^T \mathbf{a}_k} (b_k - \mathbf{a}_k^T \mathbf{x}[k]) \mathbf{a}_k \quad (\text{Repeat})$$

This is the Kaczmarz equation. Kaczmarz algorithm was originally proposed for solving the under-determined systems ($m < n$) [16]. Due to a limited number of rows, the Kaczmarz equation keeps jumping back to the first row after it has reached the last row until the solution converges. Hence, Kaczmarz algorithm is recurrent in terms of rows. Whereas in NLMS, new rows are added continuously due to the arrival of new data and the system, therefore, becomes over-determined, i.e., more rows than the columns. In this case, previous rows are never used. Therefore, NLMS algorithm is not recurrent. Otherwise both algorithms are identical.

6. Affine Projection (AP) Algorithm

Re-writing Equation (11) for NLMS algorithm,

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mathbf{a}_1 (\mathbf{a}_1^T \mathbf{a}_1)^{-1} (b_1 - \mathbf{a}_1^T \mathbf{x}[k]) \quad (14)$$

NLMS algorithm tries to reduce the error in Equation (3) with respect to a single row of data $\mathbf{a}_1$. The idea behind the AP algorithm is to choose a step-size that minimizes the error with respect to all the k rows of data that have been received up to time-step k to improve its convergence properties [17]. Hence, the vector $\mathbf{a}_1$ in Equation (14) is replaced by a matrix $\mathbf{A}_k$ which contains all the k data rows.

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mathbf{A}_k^T (\mathbf{A}_k \mathbf{A}_k^T)^{-1} (b_k - \mathbf{A}_k \mathbf{x}[k]) \quad (15)$$

As long as the number of these data rows remains less than dimension of the system ($k < n$), the system can be solved by Equation (15). Such a system is called an under-determined system [18]. It means that there are more unknowns than the number of equations which in turn implies that much less data is available. Equation (15) is known as AP algorithm and $\mathbf{A}_k^T (\mathbf{A}_k \mathbf{A}_k^T)^{-1}$ is defined as the pseudoinverse for an under-determined system [18]. It is important to note that the pseudoinverse $\mathbf{A}_k^T (\mathbf{A}_k \mathbf{A}_k^T)^{-1}$ is dependent on time-step k . Thus, the solution to Equation (15) is obtained by computing the term $\mathbf{A}_k^T (\mathbf{A}_k \mathbf{A}_k^T)^{-1}$ at each time-step. In this way, AP algorithm may appear much more complex than NLMS and Kaczmarz algorithms. Also there is no indirect way of computing the pseudoinverse. However, these drawbacks are offset by its much faster convergence as compared to NLMS and Kaczmarz algorithms.

7. Recursive Least Squares (RLS) Algorithm

AP algorithm implies that the amount of data available is much less for the system to be full-determined ($k < n$). However, this case seldom occurs in practice. On the contrary, more data keeps arriving with each time-step and the number of data rows exceeds the dimensions of the system ($k > n$). As a result, the system becomes over-determined. There are more equations than the unknowns. In this case, the pseudoinverse for an under-determined systems $A_k^T(A_k A_k^T)^{-1}$ in Equation (15) is replaced by the Least Squares pseudoinverse defined for an over-determined case $(A_k^T A_k)^{-1} A_k^T$ [18].

$$x[k+1] = x[k] + (A_k^T A_k)^{-1} A_k^T (b_k - A_k x[k]) \quad (16)$$

But before we can proceed with the iterative solution, we must consider $(A_k^T A_k)^{-1}$ term in Equation (16). This term will make Equation (16) converge in one iteration.

$$x[k+1] = x[k] + (A_k^T A_k)^{-1} A_k^T b_k - (A_k^T A_k)^{-1} A_k^T A_k x[k] = x[k] + x - x[k] \quad (17)$$

Hence,

$$x[k+1] = x \quad (18)$$

But if we can compute this term beforehand, then the whole point of iterative solution becomes useless and the system can be directly solved in one step using Least Squares. Luckily, a lemma is available that can avoid the direct computation of $(A_k^T A_k)^{-1}$. This lemma is known as the *matrix inversion lemma* [18]. Here, we explain the great advantage achieved by this lemma over the AP algorithm where no such flexibility is available. We begin by examining the $A_k^T A_k$ term. This term can be decomposed as a sum of k rank-1 matrices.

$$A_k^T A_k = [a_1 \ a_2 \ \dots \ a_k] \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_k^T \end{bmatrix} = a_1 a_1^T + a_2 a_2^T + \dots + a_{k-1} a_{k-1}^T + a_k a_k^T$$

$$A_k^T A_k = A_{k-1}^T A_{k-1} + a_k a_k^T \quad (19)$$

The term $a_k a_k^T$ is known as the rank-1 *update-term* because it updates the $A_{k-1}^T A_{k-1}$ when a single new row of data a_k is added to the system at time-step k . The matrix inversion lemma computes the inverse $(A_k^T A_k)^{-1}$ by incorporating the rank-1 update $a_k a_k^T$ into the inverse $(A_{k-1}^T A_{k-1})^{-1}$. An identity matrix can be chosen as a starting candidate for $(A_{k-1}^T A_{k-1})^{-1}$ term.

$$(A_{k-1}^T A_{k-1} + a_k a_k^T)^{-1} = (A_{k-1}^T A_{k-1})^{-1} - \frac{(A_{k-1}^T A_{k-1})^{-1} a_k a_k^T (A_{k-1}^T A_{k-1})^{-1}}{(1 + a_k^T (A_{k-1}^T A_{k-1})^{-1} a_k)} \quad (20)$$

In this way, the system does not have to wait for all the rows of data to form its estimate x . Instead, the estimate is continually updated with the arrival of every new row of data. Also, the lemma seamlessly updates the estimate with every new observation and saves the toil of solving the system all over when new data arrives. Substituting Equation (19) in Equation (16) and modifying Equation (16) for one row of data at a time,

$$x[k+1] = x[k] + (A_{k-1}^T A_{k-1} + a_k a_k^T)^{-1} a_k (b_k - a_k^T x[k]) \quad (21)$$

Equation (21) is known as the RLS algorithm. Due to its dependency on matrix inversion lemma, RLS is much more complex than the rest of the algorithms discussed so far.

But despite that, it has been the most popular algorithm to date. Reasons for this are its ease of implementation, its excellent convergence properties, and its ability to update the estimate after every new observation without the need of solving the entire system of linear equations all over again. Figure 2 depicts the superior convergence properties of RLS algorithm as compared to the NLMS and LMS algorithms. Table 1 compares the computational complexity of the algorithms discussed so far [17].

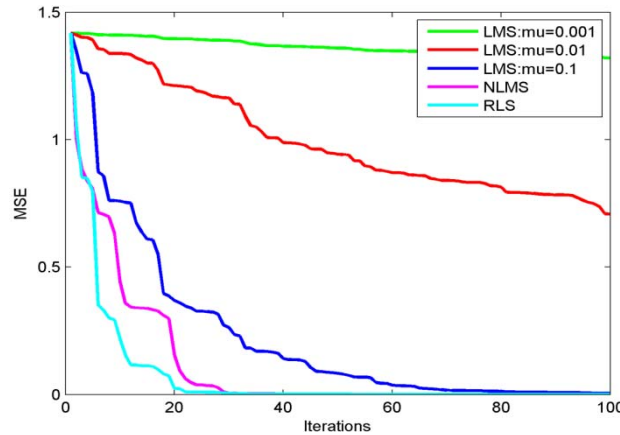


Figure 2. Comparison of the convergence properties of RLS, NLMS, and LMS algorithms

These algorithms were employed to identify a time-invariant system with an impulse response of length $n = 5$. System was over-determined

Table 1. Comparison of the computational complexity of various ASP algorithms

Algorithm	Complexity
LMS	$2N + 1$
NLMS	$3M + 2$
Kaczmarz	$3M + 2$
AF	N^2
RLS	N^2

8. Kalman Filter

All the algorithms that have been discussed up till now have a major underlying assumption. They provide an iterative solution to a time-invariant system. By time-invariant we mean that the vector x depicting the system parameters in Equation (1) remains unchanged during the iterative process. This assumption is relatively weak as it often happens in physical situations that the parameters of a system change during the convergence process, say for example in wireless communications [19]. Therefore, an algorithm must track the system during the transition process in addition to assuring the convergence within the intervals between the transitions. These are precisely the objectives of Kalman filter [20]. In order to show that how Kalman filter accomplishes it, we modify the system model in Equation (1) to incorporate the changes in nomenclature that arise due to the time varying nature of x .

$$A_{mn}x_n = b_m \quad (22)$$

A_{mn} is an $(m \times n)$ matrix. x is an $(n \times 1)$ vector. b_m is an $(m \times 1)$ vector. The system is over-determined ($m > n$). m data rows and m observations have been received and the solution has been obtained using RLS algorithm in Equation (21). Let us assume that the system in Equation (22) changes its state from x_n to $\hat{x}_n$. This change of state will create a new set of unknowns $\hat{x}_n$ with the following linear relationship to previous unknowns x_n .

$$\hat{x}_n = F_{nn}x_n + c_n \quad (23)$$

$\hat{x}_n$ is an $(n \times 1)$ vector with n new system unknowns. F is an $(n \times n)$ matrix responsible for the change of state. It is also known as the *state-transition matrix*. c_n is an $(n \times 1)$ vector of constants. Re-writing Equation (23),

$$-F_{nn}x_n + \hat{x}_n = c_n \quad (24)$$

Combining Equation (22) and (24) yields,

$$\begin{bmatrix} A_{mn} & 0_{nn} \\ -F_{nn} & I_{nn} \end{bmatrix} \begin{bmatrix} x_n \\ \hat{x}_n \end{bmatrix} = \begin{bmatrix} b_m \\ c_n \end{bmatrix} \quad (25)$$

0_{nn} is an $(n \times n)$ matrix with all zero entries. I_{nn} is $(n \times n)$ identity matrix. Least Squares solution to Equation (25) is,

$$\begin{bmatrix} A_{mn}^T & -F_{nn}^T \\ 0_{nn} & I_{nn} \end{bmatrix} \begin{bmatrix} A_{mn} & 0_{nn} \\ -F_{nn} & I_{nn} \end{bmatrix} \begin{bmatrix} x_n \\ \hat{x}_n \end{bmatrix} = \begin{bmatrix} A_{mn}^T & -F_{nn}^T \\ 0_{nn} & I_{nn} \end{bmatrix} \begin{bmatrix} b_m \\ c_n \end{bmatrix} \quad (26)$$

Adopting column-wise multiplication in Equation (26).

$$\left[\begin{bmatrix} A_{mn}^T \\ 0_{nn} \end{bmatrix} [A_{mn} \quad 0_{nn}] + \begin{bmatrix} -F_{nn}^T \\ I_{nn} \end{bmatrix} [-F_{nn} \quad I_{nn}] \right] \begin{bmatrix} x_n \\ \hat{x}_n \end{bmatrix} = \begin{bmatrix} A_{mn}^T \\ 0_{nn} \end{bmatrix} b_m + \begin{bmatrix} -F_{nn}^T \\ I_{nn} \end{bmatrix} c_n \quad (27)$$

In order to find the value of new unknowns, we only need the inverse of the first term in the square brackets on the left hand side of Equation (27).

$$\begin{bmatrix} A_{mn}^T \\ 0_{nn} \end{bmatrix} [A_{mn} \quad 0_{nn}] + \begin{bmatrix} -F_{nn}^T \\ I_{nn} \end{bmatrix} [-F_{nn} \quad I_{nn}] \quad (28)$$

But we do not have to compute this inverse from right from start. From Equation (25), we observe that n new rows and n new columns have been added to the system in Equation (22) by F_{nn} and I_{nn} matrices. These extra rows and columns can be incorporated as rank-1 updates in the original matrix A_{mn} . This is because the zeros matrix 0_{nn} to the right of A_{mn} in Eq. (25) has no impact on the rows of A_{mn} other than increasing the length of individual rows from n to $2n$ by appendage n zeros at the end. This also leaves the product $A_{mn}^T A_{mn}$ unchanged. Only its dimensions have increased from $n \times n$ to $2n \times 2n$ with all the entries zeros except the first $n \times n$ ones which are equal to $A_{mn}^T A_{mn}$.

$$\begin{bmatrix} A_{mn}^T \\ 0_{nn} \end{bmatrix} [A_{mn} \quad 0_{nn}] = \begin{bmatrix} A_{mn}^T A_{mn} & 0_{nn} \\ 0_{nn} & 0_{nn} \end{bmatrix} \quad (29)$$

Therefore, only the change in nomenclature is required to be precise. Otherwise the rest stays the same with this term. Therefore, we term the vector $[A_{mn} \quad 0]$ to $A_{m(2n)}$, meaning that $A_{m(2n)}$ has m rows each of length $2n$ with last n entries all zero. Similarly, we term $[-F_{nn} \quad I_{nn}]$ as $\hat{F}_{n(2n)}$ because each of its rows is subjoined by the corresponding row of the identity matrix I_{nn} . Both the new and old unknowns in Equation (25) are jointly represented by x_{2n} . Equation (25) becomes,

$$\begin{bmatrix} A_{m(2n)} \\ \hat{F}_{n(2n)} \end{bmatrix} x_{2n} = \begin{bmatrix} b_m \\ c_n \end{bmatrix} \quad (30)$$

Subsequently, Equation (28) becomes:

$$\begin{bmatrix} A_{mn}^T \\ 0_{nn} \end{bmatrix} [A_{mn} \quad 0_{nn}] + \begin{bmatrix} -F_{nn}^T \\ I_{nn} \end{bmatrix} [-F_{nn} \quad I_{nn}] = A_{m(2n)}^T A_{m(2n)} + \hat{F}_{n(2n)}^T \hat{F}_{n(2n)} \quad (31)$$

Matrix $\hat{\mathbf{F}}_{n(2n)}^T \hat{\mathbf{F}}_{n(2n)}$ in Equation (31) can be decomposed into a series of rank-one updates:

$$\mathbf{A}_{m(2n)}^T \mathbf{A}_{m(2n)} + \hat{\mathbf{F}}_{n(2n)}^T \hat{\mathbf{F}}_{n(2n)} = \mathbf{A}_{m(2n)}^T \mathbf{A}_{m(2n)} + \sum_{i=0}^n \hat{\mathbf{f}}_{i(2n)} \hat{\mathbf{f}}_{i(2n)}^T \quad (32)$$

So the inverse required in Equation (26) can be written as:

$$\left[\begin{bmatrix} \mathbf{A}_{mn}^T & \mathbf{0}_{nn} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{mn} & \mathbf{0}_{nn} \end{bmatrix} + \begin{bmatrix} -\mathbf{F}_{nn}^T & \mathbf{I}_{nn} \end{bmatrix} \begin{bmatrix} -\mathbf{F}_{nn} & \mathbf{I}_{nn} \end{bmatrix} \right]^{-1} = \left(\mathbf{A}_{m(2n)}^T \mathbf{A}_{m(2n)} + \sum_{i=0}^n \hat{\mathbf{f}}_{i(2n)} \hat{\mathbf{f}}_{i(2n)}^T \right)^{-1} \quad (33)$$

The rank-1 updates of $\hat{\mathbf{F}}_{n(2n)}$ matrix can be used to form the inverse in Equation (33) using matrix inversion lemma. This inverse can then be used to from the pseudoinverse to compute the value of $\mathbf{x}_{2n}$.

$$\mathbf{x}_{2n} = \left(\mathbf{A}_{m(2n)}^T \mathbf{A}_{m(2n)} + \hat{\mathbf{F}}_{n(2n)}^T \hat{\mathbf{F}}_{n(2n)} \right)^{-1} \left(\mathbf{A}_{m(2n)}^T \mathbf{b}_m + \hat{\mathbf{F}}_{n(2n)}^T \mathbf{c}_n \right) \quad (34)$$

In this way, the values of the new unknowns can be determined without the need of explicitly computing the pseudoinverse. It is important to note that despite the system has changed state, new data measurements have not yet arrived. By solving for $\mathbf{x}_{2n}$ in advance, values of new unknowns can be *predicted* and the values of previous unknowns can be *smoothed*. Hence, Equation (34) is known as *prediction equation*. When the new measurement arrives, it is added as a new row of data to Equation (30).

$$\begin{bmatrix} \mathbf{A}_{m(2n)} \\ \hat{\mathbf{F}}_{n(2n)} \\ \mathbf{a}_{k(2n)}^T \end{bmatrix} \mathbf{x}_{2n} = \begin{bmatrix} \mathbf{b}_m \\ \mathbf{c}_n \\ b_k \end{bmatrix} \quad (35)$$

RLS can now be invoked to update the predicted vector $\mathbf{x}_{2n}[k]$ in the light of new measurements.

$$\mathbf{x}_{2n}[k+1] = \mathbf{x}_{2n}[k] + \left(\mathbf{A}_{m(2n)}^T \mathbf{A}_{m(2n)} + \hat{\mathbf{F}}_{n(2n)}^T \hat{\mathbf{F}}_{n(2n)} + \mathbf{a}_{k(2n)} \mathbf{a}_{k(2n)}^T \right)^{-1} \mathbf{a}_{k(2n)} (b_k - \mathbf{a}_{k(2n)}^T \mathbf{x}_{2n}[k]) \quad (36)$$

Again the matrix inversion lemma can be used to compute the term $(\mathbf{A}_{m(2n)}^T \mathbf{A}_{m(2n)} + \mathbf{F}_{n(2n)}^T \mathbf{F}_{n(2n)} + \mathbf{a}_{k(2n)} \mathbf{a}_{k(2n)}^T)^{-1}$ by employing the rank-1 update $\mathbf{a}_{k(2n)} \mathbf{a}_{k(2n)}^T$ in the term $\mathbf{A}_{m(2n)}^T \mathbf{A}_{m(2n)} + \hat{\mathbf{F}}_{n(2n)}^T \hat{\mathbf{F}}_{n(2n)}$ calculated in Equation (32). Equation (36) is known as *update equation*. Following term in update equation,

$$\left(\mathbf{A}_{m(2n)}^T \mathbf{A}_{m(2n)} + \hat{\mathbf{F}}_{n(2n)}^T \hat{\mathbf{F}}_{n(2n)} + \mathbf{a}_{k(2n)} \mathbf{a}_{k(2n)}^T \right)^{-1} \mathbf{a}_{k(2n)} \quad (37)$$

Is known as *Kalman gain*. As more and more measurements arrive, Equation (36) is run to update the predicated estimate. In case the state of the system changes again, new unknowns can be added to the system in the same manner as described before in Equation (25) and the same procedure can be repeated. Though we have kept the information of old states together with the new states by the process of *smoothing*, it is not always necessary to do so. As soon as the measurements arrive and the estimated is update, old estimates can be thrown away altogether to keep the information about the most recent state only. This change of states which arises due to the time-varying nature of the system is the only feature that distinguishes Kalman filter from RLS. Otherwise, Kalman filter is similar to RLS because they perform identically for a time-invariant system. For this reason, Kalman filter is often called the time varying RLS [21].

9. Steepest Descent (SD) Algorithm

Equation (5) for LMS algorithm can be expanded as:

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mu(\mathbf{a}_k b_k - \mathbf{a}_k \mathbf{a}_k^T \mathbf{x}[k]) \quad (38)$$

Instead of taking a row at a time in Equation (38), SD is interested in the average of all the k rows that have been received so far.

$$\frac{1}{k}(\mathbf{a}_1 \mathbf{a}_1^T + \mathbf{a}_2 \mathbf{a}_2^T + \dots + \mathbf{a}_k \mathbf{a}_k^T) = \frac{1}{k} \mathbf{a}_1 \mathbf{a}_1^T + \frac{1}{k} \mathbf{a}_2 \mathbf{a}_2^T + \dots + \frac{1}{k} \mathbf{a}_k \mathbf{a}_k^T \quad (39)$$

$\frac{1}{k}$ represents the probability of each row as each row is equally likely.

$$P(\mathbf{a}_1) \mathbf{a}_1 \mathbf{a}_1^T + P(\mathbf{a}_2) \mathbf{a}_2 \mathbf{a}_2^T + \dots + P(\mathbf{a}_k) \mathbf{a}_k \mathbf{a}_k^T = \sum_{i=1}^k P(\mathbf{a}_i) \mathbf{a}_i \mathbf{a}_i^T = E\{\mathbf{a}_i \mathbf{a}_i^T\} \quad (40)$$

E is an expectation operator and stands for expected or average value. Similarly for $\mathbf{a}_k b_k$ term in Equation (39),

$$\frac{1}{k} \mathbf{a}_1 b_1 + \frac{1}{k} \mathbf{a}_2 b_2 + \dots + \frac{1}{k} \mathbf{a}_k b_k = P(\mathbf{a}_1) \mathbf{a}_1 b_1 + P(\mathbf{a}_2) \mathbf{a}_2 b_2 + \dots + P(\mathbf{a}_k) \mathbf{a}_k b_k \quad (41)$$

$$E\{\mathbf{a}_i b_i\} = \sum_{i=1}^k P(\mathbf{a}_i) \mathbf{a}_i b_i \quad (42)$$

Substituting Equation (42) and (40) in Equation (38),

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mu(E\{\mathbf{a}_i b_i\} - E\{\mathbf{a}_i \mathbf{a}_i^T\} \mathbf{x}[k]) \quad (43)$$

Whereas,

$$\mathbf{R} = E\{\mathbf{a}_i \mathbf{a}_i^T\} \quad (44)$$

$$\mathbf{P} = E\{\mathbf{a}_i b_i\} \quad (45)$$

$\mathbf{R}$ is known as the *auto-correlation matrix* and $\mathbf{P}$ as the *cross-correlation matrix* [1]. Substituting Equation (44) and (45) in Equation (43),

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mu(\mathbf{P} - \mathbf{R} \mathbf{x}[k]) \quad (46)$$

Equation (46) represents the Steepest Descent algorithm. This equation differs from Equation (6) of LMS algorithm only in terms of $\mathbf{R}$ and $\mathbf{P}$ matrices. They are formed in SD algorithm by taking the average of all the available data rows as depicted in Equation (40) and (42). LMS algorithm, on the other hand, only considers one row of data at a time. It has been demonstrated that if the LMS algorithm is run repeatedly for a given problem, then on the average its performance will be equal to that of LMS algorithm [1]. But $\mathbf{R}$ and $\mathbf{P}$ matrices are usually not available beforehand and computing them at run-time can be expensive. Therefore, LMS algorithm is much more popular than SD algorithm due to its simplicity.

10. MMSE and Least Squares Wiener Filters

As Equation (46) represents the recursion equation for SD algorithm, $(\mathbf{P} - \mathbf{R} \mathbf{x}[k])$ must be the gradient.

$$\nabla = \mathbf{P} - \mathbf{R} \mathbf{x}[k] \quad (47)$$

Setting the gradient in Equation (47) to zero and solving directly for $\mathbf{x}$,

$$\mathbf{R} \mathbf{x} = \mathbf{P} \quad (48)$$

Equation (48) is known as Minimum Mean Square Error (MMSE) Wiener-Hop equation. $\mathbf{x}$ is now called MMSE Wiener Filter [1]. Expanding Equation (46),

$$E\{\mathbf{a}_i \mathbf{a}_i^T\} \mathbf{x} = E\{\mathbf{a}_i b_i\} \quad (49)$$

As can be observed from Equation (39) and (40),

$$E\{\mathbf{a}_i \mathbf{a}_i^T\} = \sum_{i=1}^k P(\mathbf{a}_i) \mathbf{a}_i \mathbf{a}_i^T = \frac{1}{k} (\mathbf{a}_1 \mathbf{a}_1^T + \mathbf{a}_2 \mathbf{a}_2^T + \cdots + \mathbf{a}_k \mathbf{a}_k^T) = \frac{1}{k} \mathbf{A}_k^T \mathbf{A}_k \quad (50)$$

Similarly from Equation (41) and (42),

$$E\{\mathbf{a}_i b_i\} = \frac{1}{k} (\mathbf{a}_1 b_1 + \mathbf{a}_2 b_2 + \cdots + \mathbf{a}_k b_k) = \frac{1}{k} [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_k] \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} = \frac{1}{k} \mathbf{A}_k^T \mathbf{b}_k \quad (51)$$

Substituting Equation (50) and (51) in Equation (48),

$$\mathbf{A}_k^T \mathbf{A}_k \mathbf{x} = \mathbf{A}_k^T \mathbf{b}_k \quad (52)$$

Equation (52) is known as the Least Square Wiener-Hop equation. $\mathbf{x}$ is now called Least Squares Wiener Filter [1].

11. Relationship with other Areas

This section establishes the relationship of the work presented in this article to highly specialized fields that are currently the focus of research and development in ECE and applied mathematics. This will enable the readers to explore further opportunities for research and innovation.

11.1. Reduced Rank Adaptive Filters

We begin our discussion of reduced-rank adaptive filters with Equation (52).

$$\mathbf{x} = (\mathbf{A}_k^T \mathbf{A}_k)^{-1} \mathbf{A}_k^T \mathbf{b}_k \quad (53)$$

Least Squares Wiener filter $\mathbf{x}$ in Equation (52) requires the inversion of $\mathbf{A}_k^T \mathbf{A}_k$ matrix. The matrix $\mathbf{A}_k^T \mathbf{A}_k$ is also known as *covariance matrix*. For it to be invertible, it must have full-rank. But this assumption of full-rank has been strongly questioned by the proponents of reduced rank adaptive filters [22]. They claim that such full-rank is not available often in practice, say for example in sensor array processing. This claim not only puts the invertibility of $\mathbf{A}_k^T \mathbf{A}_k$ matrix in question but also the challenges the entire possibility of finding the LS wiener filter. However, the rank of $\mathbf{A}_k^T \mathbf{A}_k$ can be easily analyzed because it is a symmetric matrix. A symmetric matrix can be decomposed into a set of real eigenvalues and orthogonal eigenvectors [15].

$$\mathbf{A}_k^T \mathbf{A}_k = \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^T \quad (54)$$

$\mathbf{\Lambda}$ is an $(n \times n)$ diagonal matrix of eigenvalues. $\mathbf{Q}$ is an $(n \times n)$ matrix of eigenvectors. For a matrix to be of full-rank, none of its eigenvalues should be zero. Even if one of them is zero, the matrix becomes non-invertible. In this case, a Moore and Penrose's pseudoinverse can be formed using Equation (54) [18]. In order to explain the key idea here, we have to concentrate our attention on $\mathbf{Q}$ matrix for a moment. $\mathbf{Q}$ is an orthogonal matrix, i.e., $\mathbf{Q}^T \mathbf{Q} = \mathbf{I}$. This implies that inverse of $\mathbf{Q}$ is equal to its transpose, i.e., $\mathbf{Q}^{-1} = \mathbf{Q}^T$. Inverting $\mathbf{A}_k^T \mathbf{A}_k$ would mean,

$$(\mathbf{A}_k^T \mathbf{A}_k)^{-1} = (\mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^T)^{-1} = (\mathbf{Q}^T)^{-1} \mathbf{\Lambda}^{-1} \mathbf{Q}^{-1} = \mathbf{Q} \mathbf{\Lambda}^{-1} \mathbf{Q}^T \quad (55)$$

Hence, to form the inverse $(\mathbf{A}_k^T \mathbf{A}_k)^{-1}$ we only need to invert the eigenvalues. But since the matrix $\mathbf{A}_k^T \mathbf{A}_k$ is rank-deficient, at least one of its eigenvalues must be zero. Inverting such

an eigenvalue will create problems. Idea behind the Moore and Penrose's pseudoinverse is to leave such an eigenvalue un-inverted while inverting the rest [18].

$$(A_k^T A_k)^+ = Q \Lambda^{-1} Q^T \quad (56)$$

$(A_k^T A_k)^+$ indicates the Moore and Penrose's pseudoinverse. A reduced rank LS Wiener filter can then be obtained by:

$$x = (A_k^T A_k)^+ A_k^T b_k \quad (57)$$

Similarly, the argument can be extended to RLS algorithm which also requires the inversion of the covariance matrix.

$$x[k+1] = x[k] + (A_{k-1}^T A_{k-1} + a_k a_k^T)^{-1} a_k (b_k - a_k^T x[k]) \quad (\text{Repeat})$$

As we know from Equation (19),

$$A_k^T A_k = A_{k-1}^T A_{k-1} + a_k a_k^T \quad (\text{Repeat})$$

Therefore,

$$x[k+1] = x[k] + (A_k^T A_k)^{-1} a_k (b_k - a_k^T x[k]) \quad (58)$$

Here again, if the rank of covariance matrix is not full, pseudo inverse $(A_k^T A_k)^+$ can be used in place of $(A_k^T A_k)^{-1}$.

$$x[k+1] = x[k] + (A_k^T A_k)^+ a_k (b_k - a_k^T x[k]) \quad (59)$$

However, this is not the only case argued by the theory of reduced rank adaptive filters. Now that we can form the pseudoinverse, rank of the covariance matrix can be deliberately reduced if some of its eigenvalues are very small. In this way, larger eigenvalues of the covariance matrix can be given preference which have a larger impact on the system performance. This idea is known by the name of compressing [22]. It allows the removal of unnecessary data which has no or very less impact on systems performance. As a result, data processing ability of system is greatly improved.

11.2. Extended Kalman Filter for Non-Linear Control/Signal Processing

In non-linear systems, the system model in Eq. (22) and state transition model in Equation (23) become non-linear. In this case, the system matrix A_{nn} and the state transition matrix F_{nn} for Kalman filter are obtained from a first order Taylor series expansion of these models, a technique called *linearization*. Rest stays the same and Equation (36) is now called the extended Kalman filter (EKF) [23]. It is the most popular filter for tackling non-linear filtering problems and forms the core of non-linear signal processing, non-linear control theory, and robotics.

11.3. Particle Filters and Bayesian Signal Processing

If more terms of the Taylor series are retained during the linearization process, higher order EKF is obtained which performs much better than its predecessor. But these extra terms introduce additional complexity. Hence, there is a classic tradeoff between complexity and accuracy. An alternative approach to this problem is to use Monte Carlo techniques to estimate system parameters for better performance. In this case, system and state transition matrices in Equation (22) and (23) are replaced by probability estimates and Equation (36) is now termed as a particle filter. It is major driving concept behind particle filtering theory and Bayesian signal processing [24].

11.4. Multigrid Methods for Boundary Value Problems

Repeating Equation (1) for the system model,

$$A_m x = b_m \quad (\text{Repeat})$$

If we make an arbitrary guess $x[k]$ for the unknown x , then the right hand side may not be equal to the left hand side and there will be a difference. This ensuing difference is known as the residue term r_m in multigrid methods.

$$r_m = b_m - A_m x[k] \quad (60)$$

Multiplying both sides of Equation (60) with the psuedoinverse $(A_m^T A_m)^{-1} A_m^T$,

$$(A_m^T A_m)^{-1} A_m^T r_m = (A_m^T A_m)^{-1} A_m^T b_m - (A_m^T A_m)^{-1} A_m^T A_m x[k] \quad (61)$$

From Equation (61), it follows,

$$(A_m^T A_m)^{-1} A_m^T r_m = x - x[k] = e \quad (62)$$

Or,

$$x = x[k] + e \quad (63)$$

e is the error between the true solution and our guess. It is termed as error in the estimate or simply the error. Re-arranging Equation (62),

$$A_m e = r_m \quad (64)$$

Equation (64) represents the relationship between the error and the residue. It is identical in structure to the system model in Equation (1). Residue r_m takes place of the observations b_m and the error e becomes the unknown. However trivial it may seem, it signifies a very important concept behind multigrid methods. An initial guess $x[k]$ for the estimate x is made. This guess is used to obtain the residue r_m in Equation (60). The residue can then be used to solve for the error e in Equation (64). Once Equation (64) is solved, e can be added back to $x[k]$ to find the true estimate x . Now we move on to explain the second driving concept behind the multigrid methods. Equation (6) for LMS algorithm can be used to solve Equation (1).

$$x[k+1] = x[k] + \mu(b_k - a_k^T x[k])a_k \quad (\text{Repeat})$$

It can also be used to solve Equation (62).

$$e[k+1] = e[k] + \mu(r_k - a_k^T e[k])a_k \quad (65)$$

This is because Equation (64) is similar in structure to Equation (1). Initially Eq. (6) is run to obtain $x[k]$. Then $x[k]$ is plugged into Eq. (60) to obtain r_m . After that Equation (65) is run to directly improve the error in the estimate e . It happens in practice that error constitutes a mixture of high and low frequency oscillations, say for example in the numerical solution of Laplace Equation [18]. High frequency oscillations in the error die out quickly as manifested in fast initial convergence of the algorithm. Low frequency oscillations, on the other hand, tend to linger on. These low frequency errors are responsible for settling the algorithm in the steady state in which even after significant number of iterations, reduction in error is negligible. It is well-known from multirate signal processing that down-sampling a signal increases its frequency [25]. So by down-sampling the error, these low frequency oscillations can be converted to high frequency ones. Afterwards, the algorithm can be re-run to damp them out. When the error ceases to decrease, it can be further down-sampled and the algorithm is once again re-run. In this way, error can be reduced to any desirable tolerance in multiple cycles and at multiple scales. These multiple scales are known as multigrids and, hence, follows the name multigrid

methods. Multigrid methods are the power house of boundary value problems arising in computational electromagnetics, computational fluid dynamics, statistical physics, wavelet theory, applied mathematics, and many other branches of computational science and engineering [18].

11.5. Preconditioning Methods for Large Linear Systems

Consider Equation (6) for LMS algorithm once again,

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mu \mathbf{a}_k (b_k - \mathbf{a}_k^T \mathbf{x}[k]) \quad (\text{Repeat})$$

The term $(b_k - \mathbf{a}_k^T \mathbf{x}[k])$ is the residue for k -th row of $\mathbf{A}_k$ matrix in Equation (1). The residue is generated as soon as the k -th data row becomes available. This is ideal for run-time or on the fly operation. But if all the input and output data is available beforehand, say for example pre-recorded in a laboratory, then the entire $\mathbf{A}_m$ matrix and the $\mathbf{b}_m$ vector can be used at once in Equation (6) instead of the individual data rows and observations.

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mu \mathbf{A}_m^T (\mathbf{b}_m - \mathbf{A}_m \mathbf{x}[k]) \quad (66)$$

We have used $\mathbf{A}_m$ and $\mathbf{b}_m$ instead of $\mathbf{A}_k$ and $\mathbf{b}_k$ because only m input data rows and m output observations are available in the lab. So, we have to find $\mathbf{x}$ that agrees with all the available observations. No new measurements will be available. Now the question arises about the choice of best step-size for Equation (66). If we want to solve Equation (66) in one-step, the ideal step-size would be $\mu = (\mathbf{A}_m^T \mathbf{A}_m)^{-1}$.

$$\mathbf{x}[k+1] = \mathbf{x}[k] + (\mathbf{A}_m^T \mathbf{A}_m)^{-1} \mathbf{A}_m^T (\mathbf{b}_m - \mathbf{A}_m \mathbf{x}[k]) \quad (67)$$

Expanding Equation (67),

$$\mathbf{x}[k+1] = \mathbf{x}[k] + (\mathbf{A}_m^T \mathbf{A}_m)^{-1} \mathbf{A}_m^T \mathbf{b}_m - (\mathbf{A}_m^T \mathbf{A}_m)^{-1} \mathbf{A}_m^T \mathbf{A}_m \mathbf{x}[k] \quad (68)$$

Finally,

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \mathbf{x} - \mathbf{x}[k] = \mathbf{x} \quad (69)$$

So the system will converge in one step. Observe that for this to happen we practically need to multiply the residue term $(\mathbf{b}_m - \mathbf{A}_m \mathbf{x}[k])$ in Equation (67) with the pseudoinverse $(\mathbf{A}_m^T \mathbf{A}_m)^{-1} \mathbf{A}_m^T$. But if the pseudoinverse $(\mathbf{A}_m^T \mathbf{A}_m)^{-1} \mathbf{A}_m^T$ is available beforehand, whole point of iterative solution becomes moot. Instead the system can be solved directly. Here the preconditioning methods come to rescue. These methods suggest that instead of multiplying the pseudoinverse $(\mathbf{A}_m^T \mathbf{A}_m)^{-1} \mathbf{A}_m^T$ with the residue term, we can choose another matrix $\mathbf{P}$ that is much easier to invert.

$$\mathbf{x}[n+1] = \mathbf{x}[n] + \mathbf{P}^{-1} (\mathbf{b}_m - \mathbf{A}_m \mathbf{x}[n]) \quad (70)$$

$\mathbf{P}$ can be the diagonal part of the pseudoinverse or may constitute its lower-triangular portions. Former choice is known as the *Jacobi iteration* and the latter one as the *Gauss-Seidel iteration*. *Successive Over Relaxation (SOR)* is a combination of both [18]. Anyhow, the idea is to find a $\mathbf{P}$ that is much easier to invert. Solving Equation (70),

$$\mathbf{P} \mathbf{x}[n+1] = \mathbf{P} \mathbf{x}[n] + (\mathbf{b}_m - \mathbf{A}_m \mathbf{x}[n]) \quad (71)$$

Or,

$$\mathbf{P} \mathbf{x}[n+1] = (\mathbf{P} - \mathbf{A}_m) \mathbf{x}[n] + \mathbf{b}_m \quad (72)$$

Equation (72) is known as the *preconditioning equation*. $\mathbf{P}$ is known as the *preconditioning matrix*. The process $(\mathbf{P} - \mathbf{A})$ is termed as *splitting*. Preconditioning methods are

popular for solving large linear systems that are too expensive for traditional methods based on matrix factorizations.

11.6. Krylov Subspace Methods and Conjugate Gradients for Optimization and Control

If identity matrix is chosen as the precondition matrix $\mathbf{P}$ and $\mathbf{b}_m$ as the initial guess $\mathbf{x}[\mathbf{n}]$, Equation (72) becomes,

$$\mathbf{x}[\mathbf{n} + 1] = (\mathbf{I} - \mathbf{A}_m)\mathbf{b}_m + \mathbf{b}_m = 2\mathbf{b}_m - \mathbf{A}_m\mathbf{b}_m \quad (73)$$

Iterating Equation (73) for the second time,

$$\mathbf{x}[\mathbf{n} + 2] = (\mathbf{I} - \mathbf{A}_m)\mathbf{x}[\mathbf{n} + 1] + \mathbf{b}_m = 3\mathbf{b}_m - 3\mathbf{A}_m\mathbf{b}_m + \mathbf{A}_m^2\mathbf{b}_m \quad (74)$$

$\mathbf{b}_m, \mathbf{A}_m\mathbf{b}_m, \dots, \mathbf{A}_m^n\mathbf{b}_m$ constitute the basis vectors for the Krylov subspace. According to Equation (74), $\mathbf{x}$ can be obtained by a linear combination of Krylov basis vectors. Hence, the solution to Equation (1) can be found in Krylov subspace. This is the driving concept behind Krylov subspace methods. Advanced Krylov subspace methods include conjugate gradient method and minimum Residual methods: MINRES and GMRES [18]. These methods are central to optimization theory and automatic control systems.

12. Discussion

Various authors have tried to address the problems mentioned in the Introduction section, though in part only. Widrow & Stearn [26], for example, have tried to present a broad conceptual introduction to the basic algorithms. Their treatment of subject is light on mathematics and is definitely not for someone who wishes to implement these algorithms. Boroujeny [1] has attempted to make these algorithms more implementable by providing a concrete mathematical approach. However, the usual clarity and the knack of exposition of the author just disappear as the algorithms become more advanced. For example, author just restates the major results in RLS filtering in terms of his own notation and drops the Kalman filter entirely. Poularikas [27] has made an effort to make the basic algorithms accessible by using the simulation approach. Yet again, Kalman filter is entirely left out and RLS is barely touched. Diniz [28] has provided a detailed treatment of these algorithms by following a statistical/linear algebraic approach to the subject. But when it comes to Kalman filter, the author switches to state space paradigm of control theory for a brief introduction to the topic. Strang [20, 21] has tried to tackle the RLS and Kalman filtering problem and has endeavored to bring it to linear algebraic perspective. But the author entirely leaves out the Wiener filters, Kaczmarz, LMS, and NLMS algorithms. Dwight [29] has tried to categorize the basic algorithms in two different paradigms, the least squares paradigm and MMSE paradigm using linear algebraic approach. But when it comes to the exposition of Kalman filter, the author completely switches to Wiener's system theory and statistical signal processing paradigm resulting in almost no connection between his earlier and later results. To unify adaptive signal processing and statistical signal processing platforms, an effort has been made by Manolakis et al [30]. But the book has rather obscure notation and the topics are almost inaccessible without familiarization with it. Chong [16] provides a treatment of only RLS and Kaczmarz algorithms from optimization theoretic viewpoint. Whereas Hayes [31], Syed [32], Haykin [33], and Candy [24] make no attempt at all at making these algorithms accessible.

As we have been through this journey, we have discovered two things that have shaped our methodology. Firstly, no matter how hidden it is behind all the veils just mentioned, it is all linear algebra. Secondly, from the basic Steepest Descent algorithm to fairly advanced Kalman filter, one and only one equation is involved. This equation assumes a different name and wears a different mantle of symbols in each algorithm. If somehow this link can be discovered, a relationship can be established among all the algorithms. Then the transition from one algorithm to the other will be more logical, more systematic, and much easier to understand. This will provide an opportunity to focus more on the underlying problem, the reason for the progression from one algorithm to the other, and the advantage provided by the new algorithm over the previous one. Also their notation can be made more uniform which will generate certain terseness in their explanation. In this way, one approach can be picked and followed to the end

without the need for switching between different subjects. We have adopted a linear algebraic approach for this purpose. It is the simplest possible approach in a sense that it asks for no technical introduction, making it possible for everyone to benefit. We have started from the simplest possible case, one equation in one unknown and one observation. From this one equation, we have derived all the key algorithms from SD to Kalman filter while leaving nothing out. Notation was kept consistent throughout. Transitions from one algorithm to the other were logical. They were also systematic to make the concepts portable. Calculus, Probability, and Statistics were not invoked in order to make them accessible as well. The treatment was entirely linear algebraic. Each step was explained but the explanations were kept concise in order to prevent the manuscript from becoming too voluminous.

13. Conclusion

Our work provides a uniform approach to students, researchers, and practitioners from different academic backgrounds working in different areas. Anyone can learn these key algorithms without any previous knowledge. The understanding of readers will be enhanced and their efforts will be minimized as they will see just one equation in action. It will enable them to focus more on the concept rather than the symbols, helping them to master the essentials in minimum amount of time. As there is only one equation to deal with, it will also ease their burden of programming. They can program it without the need of special programming skills. They will be able to appreciate how just one equation has been the centre of research and development for last many decades and how it has vexed scientists and engineers in many different fields. In fact, to demonstrate the benefit of this synthesis, an entire area in applied mathematics known as multigirid methods that aim to solve differential equations by operating at multiple resolutions, stems from this one equation. Other specialized domains like non-linear control and particle filter theory also take the lead from this one equation. Finally, a better understanding of these algorithms will ultimately lead to a better understanding of image processing, video processing, wireless communications, pattern recognition, machine learning, optimization theory and many other subjects that utilize these core concepts. For anyone, this one equation will eventually be their passport to the realm of computational science and engineering. For students, this one equation will save them the toil of going through these algorithms again and again in various courses where they just change their name and switch their clothes. For faculty, this one equation will spare their efforts to teach these algorithms repeatedly in various courses. For universities, this one equation will enable them to develop a consolidated course based on these algorithms that are central to so many subjects in the ECE and applied mathematics.

References

- [1] B Farhang-Boroujeny. Adaptive Filters: Theory and Applications. John Wiley & Sons. 2013.
- [2] P Avirajamanjula, P Palanivel. Corroboration of Normalized Least Mean Square Based Adaptive Selective Current Harmonic Elimination in Voltage Source Inverter using DSP Processor. *International Journal of Power Electronics and Drive Systems (IJPEDS)*. 2015; 6.
- [3] Y Zheng-Hua, R Zi-Hui, Z Xian-Hua, L Shi-Chun. Path Planning for Coalmine Rescue Robot Based on Hybrid Adaptive Artificial Fish Swarm Algorithm. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2014; 12: 7223-7232.
- [4] F Wang, Z Zhang. An Adaptive Genetic Algorithm for Mesh Based NoC Application Mapping. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2014; 12: 7869-7875.
- [5] X Yan, J Li, Z Li, Y Yang, C Zhai. Design of Adaptive Filter for Laser Gyro. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2014; 12: 7816-7823.
- [6] J Mohammed. Low Complexity Adaptive Noise Canceller for Mobile Phones Based Remote Health Monitoring. *International Journal of Electrical and Computer Engineering (IJECE)*. 2014; 4: 422-432.
- [7] KF Akingbade, IA Alimi. Separation of Digital Audio Signals using Least Mean Square LMS Adaptive Algorithm. *International Journal of Electrical and Computer Engineering (IJECE)*. 2014; 4: 557-560.
- [8] J Mohammed, M Shafi. An Efficient Adaptive Noise Cancellation Scheme Using ALE and NLMS Filters. *International Journal of Electrical and Computer Engineering (IJECE)*. 2012; 2: 325-332.
- [9] J Mohammed. A Study on the Suitability of Genetic Algorithm for Adaptive Channel Equalization. *International Journal of Electrical and Computer Engineering (IJECE)*. 2012; 2: 285-292.

- [10] J Huang, J Xie, H Li, G Tian, X Chen. Self Adaptive Decomposition Level Denoising Method Based on Wavelet Transform. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10: 1015-1020.
- [11] N Makwana, M SPIT, N Mishra, B Sagar. Hilbert Transform Based Adaptive ECG R Peak Detection Technique. *International Journal of Electrical and Computer Engineering (IJECE)*. 2012; 2: 639-643.
- [12] Z Chen, U Qiqihar, X Dai, L Jiang, C Yang, B Cai. Adaptive Iterated Square Root Cubature Kalman Filter and Its Application to SLAM of a Mobile Robot. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11: 7213-7221.
- [13] Z Ji, Adaptive Cancellation of Light Relative Intensity Noise for Fiber Optic Gyroscope. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11: 7490-7499.
- [14] Q Liu. An Adaptive Blind Watermarking Algorithm for Color Image. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11: 302-309.
- [15] G Strang. Introduction to Linear Algebra. Wellesley-Cambridge Press. 2003.
- [16] EK Chong, SH Zak. An Introduction to Optimization. John Wiley & Sons. 2013; 76.
- [17] KA Lee, WS Gan, SM Kuo. Subband Adaptive Filtering: Theory and Implementation. John Wiley & Sons. 2009.
- [18] G Strang. Computational Science and Engineering. Wellesley-Cambridge Press. 2007.
- [19] TS Rappaport. Wireless Communications: Principles and Practice. Prentice hall PTR New Jersey. 1996; 2.
- [20] G Strang, K Borre. Linear Algebra, Geodesy, and GPS. Wellesley-Cambridge Press. 1997.
- [21] G Strang. Introduction to Applied Mathematics. Wellesley-Cambridge Press. 1986; 16.
- [22] JS Goldstein, IS Reed. Reduced-rank Adaptive Filtering. *IEEE Transactions on Signal Processing*. 1997; 45: 492-496.
- [23] J Mochnác, S Marchevský, P Kocan. *Bayesian Filtering Techniques: Kalman and Extended Kalman Filter Basics*. In Radioelektronika, 2009. RADIOELEKTRONIKA'09. 19th International Conference. 2009: 119-122.
- [24] JV Candy. Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods. John Wiley & Sons. 2011; 54.
- [25] G Strang, T Nguyen. Wavelets and Filter Banks. Wellesley-Cambridge Press. 1996.
- [26] B Widrow, SD Stearns. Adaptive Signal Processing. Prentice-Hall, Inc. 1985; 1.
- [27] AD Poularikas, ZM Ramadan. Adaptive Filtering Primer with MATLAB. CRC Press. 2006.
- [28] PS Diniz. Adaptive Filtering. Springer. 1997.
- [29] DF Mix. Random Signal Processing. Prentice-Hall, Inc. 1995.
- [30] DG Manolakis, VK Ingle, SM Kogon. Statistical and Adaptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering, and Array Processing. Artech House Norwood. 2005; 46.
- [31] MH Hayes. Statistical Digital Signal Processing and Modeling. John Wiley & Sons. 1996.
- [32] AH Sayed. Fundamentals of Adaptive Filtering. John Wiley & Sons. 2003.
- [33] SS Haykin. Adaptive Filter Theory. Pearson Education India. 2008.