Fast Updating of Structured Linear Systems of Equations with Applications in Adaptive Filtering*

S. CHANDRASEKARAN,† M. GU,‡ AND ALI H. SAYED§

Abstract

Linear systems of equations with structured coefficient matrices arise in several applications in signal processing and communications. In this paper we develop fast algorithms for updating the solutions of such systems when the coefficient matrices undergo rank-one updates that preserve the matrix structure. An application in adaptive filtering is noted.

1 Introduction

In this paper we study the problem of updating the solutions of symmetric positive-definite linear systems of equations with structured coefficient matrices. We focus on the case of real-valued data, although the extension to complex-valued data is immediate. Likewise, the restriction to symmetric coefficient matrices can be removed easily.

Let x_0 denote the solution of $T_0 x_0 = b_0$, where x_0 and b_0 are real n-dimensional vectors, and T_0 is an $n \times n$ symmetric positive-definite matrix. Assume further that T_0 is a structured matrix in the sense that the difference $T_0 - FT_0F^T$ has low rank (say $\alpha \ll n$) for some $n \times n$ lower triangular matrix F. This is equivalent to saying that there exists an $n \times \alpha$ matrix G_0, and an $\alpha \times \alpha$ signature matrix J, such that

$$T_0 - FT_0F^T = G_0 J G_0^T .$$

Matrices T_0 that satisfy conditions of this type are said to have displacement structure [1]. Actually, it also holds that there should exist an $n \times \alpha$ matrix H_0 such that the inverse of T_0 satisfies a similar displacement equation, viz.,

$$T_0^{-1} - F^T T_0^{-1} F = H_0 J H_0^T ,$$

with the same signature matrix J but with the roles of $\{F, F^T\}$ reversed (see, e.g., [1, 2]).

Now assume that for successive time instants $k \geq 0$ the values of $\{T_k, b_k\}$ are obtained recursively as follows:

$$T_{k+1} = T_k + a_k a_k^T , \qquad b_{k+1} = b_k + \eta(k) a_k ,$$

for some known column vectors $\{a_k\}$ and scalars $\{\eta(k)\}$. In other words, T_{k+1} is a rank-one update of T_k and b_{k+1} is obtained from b_k by adding a scaled version of a_k to it. Assume further that the successive $\{T_k\}$ all have the same displacement rank with respect to the same matrix F, i.e., they satisfy displacement equations of the form

$$T_k - FT_kF^T = G_k J G_k^T ,$$

for some $n \times \alpha$ matrices $\{G_k\}$. It then follows that their inverses also have displacement ranks α, say

$$T_k^{-1} - F^T T_k^{-1} F = H_k J H_k^T ,$$

for some $n \times \alpha$ matrices $\{H_k\}$. We shall not need to know explicitly the matrices $\{G_k, H_k\}$. Instead, we shall develop a recursive procedure for evaluating the successive $\{H_k\}$ starting from H_0. This procedure will be enough for our purposes.

Let x_k be the solution of the system of equations $T_k x_k = b_k$. The first problem we consider is the derivation of a fast algorithm for computing x_{k+1} recursively from x_k. By fast we mean an algorithm that is an order of magnitude faster than $O(n^2)$ flops per iteration. We shall present an algorithm that can compute x_{k+1} recursively from x_k in $O(m + l_1 + n\alpha^2)$ flops, where m is the number of flops required to multiply a matrix with same displacement structure as T_k with an n-dimensional vector, α is the displacement rank of T_k, and l_1 is the number of flops required to multiply an n-dimensional vector by F.

2 Structured Updates

We start by introducing the vector $r_k = T_k^{-1} a_k$. From the matrix inversion lemma we obtain

$$T_{k+1}^{-1} = (T_k + a_k a_k^T)^{-1} = T_k^{-1} - \beta(k) r_k r_k^T,$$

*This material was based on work supported in part by NSF awards CCR-9732376 and CCR-9734290, and by ARO DAAH04-96-1-0176-P00005.

†Dept. Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106.

‡Dept. of Mathematics, University of California, Los Angeles, CA 90095.

§Electrical Engineering Dept., University of California, Los Angeles, CA 90095.

where we defined the scalar

$$\beta(k) \triangleq \frac{1}{1 + a_k^T r_k} = \frac{1}{1 + a_k^T T_k^{-1} a_k}. \qquad (1)$$

Using these quantities we obtain, after some algebra, the following update for the solution x_{k+1}:

$$\begin{aligned} x_{k+1} &= (T_k^{-1} - \beta(k) r_k r_k^T)(b_k + \eta(k) a_k) \\ &= x_k + \beta(k) r_k [\eta(k) - a_k^T x_k] . \end{aligned}$$

To compute r_k we can use H_k and form $T_k^{-1} a_k$ fast in $O(m)$ flops by using fast matrix-vector multiplication procedures for matrices with displacement structure (see, e.g., [1]). When F is the lower triangular shift matrix Z, this step can be done in $O(n \log n)$ flops. For diagonal matrices F, it can be achieved in $O(n)$ flops.

We still need to show how to propagate the successive generator matrices $\{H_k\}$. For this purpose, note that

$$T_{k+1}^{-1} - F^T T_{k+1}^{-1} F = H_k J H_k^T -$$

$$-\beta(k) r_k r_k^T + \beta(k) F^T r_k r_k^T F .$$

This suggests that we first compute the following reduced QR factorization:

$$\left[\begin{array}{ccc} H_k & \sqrt{\beta(k)} r_k & \sqrt{\beta(k)} F^T r_k \end{array} \right] = Q_k R_k , \qquad (2)$$

where Q_k is an $n \times (\alpha + 2)$ matrix, and R_k is an $(\alpha + 2) \times (\alpha + 2)$ upper triangular matrix. This takes $O(n\alpha^2 + l_1)$ flops using standard algorithms for (skinny) QR factorizations (see, e.g., [3]). We then compute the symmetric eigendecomposition:

$$R_k (J \oplus -1 \oplus 1) R_k^T = W_k \Lambda_k W_k^T, \qquad (3)$$

where W_k is orthogonal and Λ_k is diagonal. This computation can be done in $O(\alpha^3)$ operations using standard algorithms [3]. But since, by assumption, the displacement rank of T_{k+1} should be α, it follows that Λ_k should have two zero entries on its diagonal. Now define H_{k+1} as follows:

$$H_{k+1} = Q_k W_k \sqrt{|\Lambda_k|} ,$$

and discard the two zero columns of H_{k+1}. Forming H_{k+1} from Q_k, W_k, and Λ_k in this way takes $O(n\alpha^2)$ operations to do the multiplications. This gives the recurrence for H_{k+1}.

The cost per iteration of the above algorithm is $O(\alpha^3 + n\alpha^2 + m + l_1)$. We summarize the main steps below.

Fast Updating of Structured Linear Equations (FUS)

1. Initialization (overhead costs). Given $\{G_0, H_0\}$, compute x_0 and r_0 using G_0 and the so-called generalized Schur algorithm with back-substitution, or by using H_0 and fast matrix-vector multiplication procedures for matrices with displacement structure (see, e.g., [1, 2, 4, 5]). Compute also $\beta(0)$ as an inner product.
2. Assume that we have $\{H_k, r_k, x_k, \beta(k)\}$ and iterate:
   - $x_{k+1} = x_k + \beta(k) r_k [\eta(k) - a_k^T x_k]$ [$O(n)$ flops].
   - Compute the QR factorization (2) [$O(n\alpha^2 + l_1)$ flops].
   - Compute the eigendecomposition (3) [$O(\alpha^3)$ flops].
   - Let $H_{k+1} = Q_k W_k \sqrt{|\Lambda_k|}$ and discard the two zero columns of H_{k+1} [$O(n\alpha^2)$ flops].
   - Compute $r_{k+1} = T_{k+1}^{-1} a_{k+1}$ fast by using H_{k+1} and fast matrix vector-multiplication procedures [$O(m)$ flops].
   - Compute $\beta(k+1) = a_{k+1}^T r_{k+1}$ [$O(n)$ flops].

3 Doubly Structured Updates

The second problem we consider is a special case of the first one. We now assume that the vectors a_k are further related among themselves as follows:

$$a_{k+1} = F a_k + \delta(k) g ,$$

where F is the *same* matrix that appears in the displacement equation for T_k, $\delta(k)$ is a scalar, and g is an n-dimensional vector. By using this additional relationship among the $\{a_k\}$ we can get a faster algorithm. More specifically, we shall now derive an $O(n\alpha^2 + l_1 + l_2)$ procedure for updating the solutions $\{x_k\}$, where l_2 is the number of flops required to multiply F^{-1} with an n-dimensional vector. The assumption that F be invertible is not necessary (see next section).

We introduce the following three additional vectors:

$$q_k \triangleq T_k^{-1} F a_k , \quad p_k \triangleq T_k^{-1} g , \quad v_k \triangleq T_k^{-1} F g,$$

and proceed to determine updates for them. First note that

$$\begin{aligned} p_{k+1} &= T_{k+1}^{-1} g = T_k^{-1} g - \beta(k) r_k r_k^T g \\ &= p_k - \beta(k)(r_k^T g) r_k \\ &= p_k - \beta(k)(a_k^T p_k) r_k . \end{aligned}$$

Either of the last two expressions can be evaluated in $O(n)$ flops. For special F and/or special g, it might be possible to efficiently evaluate p_{k+1} directly from H_{k+1}. In such cases there is no need to propagate p_k.

Second, observe that

$$\begin{aligned} v_{k+1} &= T_{k+1}^{-1}Fg = T_k^{-1}Fg - \beta(k)r_k r_k^T Fg \\ &= v_k - \beta(k)(r_k^T Fg)r_k \\ &= v_k - \beta(k)(a_k^T v_k)r_k. \end{aligned}$$

The last expression can be evaluated in $O(n)$ flops, and the one preceding it in $O(n+l_1)$ flops. For some special F and g we may again be able to efficiently evaluate v_{k+1} directly from H_{k+1}. In such cases there is no need to propagate v_k.

We can now derive at least two recurrences for r_{k+1}. Thus note that

$$\begin{aligned} r_{k+1} &= T_{k+1}^{-1}a_{k+1} \\ &= (T_k^{-1} - \beta(k)r_k r_k^T)a_{k+1} \\ &= T_k^{-1}(Fa_k + \delta(k)g) - \beta(k)(r_k^T a_{k+1})r_k \\ &= q_k + \delta(k)p_k - \beta(k)(r_k^T a_{k+1})r_k. \end{aligned}$$

This expression can be evaluated in $O(n)$ flops. We can get another recurrence as follows:

$$\begin{aligned} r_{k+1} &= T_{k+1}^{-1}a_{k+1} \\ &= T_{k+1}^{-1}(Fa_k + \delta(k)g) \\ &= (T_k^{-1} - \beta(k)r_k r_k^T)Fa_k + \delta(k)p_{k+1} \\ &= q_k - \beta(k)(r_k^T Fa_k)r_k + \delta(k)p_{k+1} \\ &= q_k - \beta(k)(a_k^T q_k)r_k + \delta(k)p_{k+1}. \end{aligned}$$

This expression can also be evaluated in $O(n)$ flops. More such expressions can be derived. Numerical considerations might lead to a suitable choice.

We now turn our attention to q_{k+1}. This is the technically hardest part of the algorithm. We first assume that F is nonsingular and that F^{-1} can be applied to an n-dimensional vector in $O(l_2)$ flops. Again several recursions can be derived. We satisfy ourselves with two of them. First observe that

$$\begin{aligned} q_{k+1} &= T_{k+1}^{-1}Fa_{k+1} \\ &= (T_k^{-1} - \beta(k)r_k r_k^T)Fa_{k+1} \\ &= T_k^{-1}Fa_{k+1} - \beta(k)(r_k^T Fa_{k+1})r_k \\ &= T_k^{-1}F^2a_k + \delta(k)T_k^{-1}Fg - \beta(k)(r_k^T Fa_{k+1})r_k \\ &= T_k^{-1}F^2a_k + \delta(k)v_k - \beta(k)(r_k^T Fa_{k+1})r_k. \end{aligned}$$

Now multiplying by F^T we have

$$\begin{aligned} F^T(q_{k+1} - \delta(k)v_k + \beta(k)(r_k^T Fa_{k+1})r_k) &= \\ &= F^T T_k^{-1}FFa_k \\ &= (T_k^{-1} - H_k J H_k^T)Fa_k \\ &= q_k - H_k J H_k^T Fa_k. \end{aligned}$$

The right-hand side of this expression can be evaluated in $O(n\alpha + l_1)$ flops, and then q_{k+1} can be obtained in $O(n + l_1 + l_2)$ flops, provided F is invertible. We now derive another expression for q_{k+1}:

$$\begin{aligned} q_{k+1} &= T_{k+1}^{-1}Fa_{k+1} \\ &= T_{k+1}^{-1}F^2a_k + \delta(k)T_{k+1}^{-1}Fg. \end{aligned}$$

Multiplying by F^T as before, we have

$$\begin{aligned} F^T(q_{k+1} - \delta(k)v_{k+1}) &= \\ &= F^T T_{k+1}^{-1}FT_k T_k^{-1}Fa_k \\ &= (T_{k+1}^{-1} - H_{k+1}JH_{k+1}^T)T_k q_k \\ &= T_{k+1}^{-1}T_k q_k - H_{k+1}JH_{k+1}^T Fa_k \\ &= (T_k^{-1} - \beta(k)r_k r_k^T)T_k q_k - H_{k+1}JH_{k+1}^T Fa_k \\ &= q_k - \beta(k)(r_k^T Fa_k)r_k - H_{k+1}JH_{k+1}^T Fa_k \\ &= q_k - \beta(k)(a_k^T q_k)r_k - H_{k+1}JH_{k+1}^T Fa_k. \end{aligned}$$

Either of the last two right-hand sides can be evaluated in $O(n\alpha + l_1)$ flops. Then q_{k+1} can be obtained in $O(n + l_2)$ flops. For diagonal and bidiagonal F, both l_1 and l_2 are $O(n)$. The above recursions for q_{k+1} require F to be nonsingular.

The cost per iteration of the above algorithm is $O(n\alpha^2 + l_1 + l_2)$. We summarize its steps below.

Fast Updating of Structured Linear Equations with Affine Transformed Updates (FUSA)

1. Initialization (overhead costs). Given $\{G_0, H_0\}$, compute x_0 and $\{r_0, v_0, q_0, p_0\}$ using G_0 and the so-called generalized Schur algorithm with back-substitution, or by using H_0 and fast matrix-vector multiplication procedures for matrices with displacement structure (see, e.g., [1, 2, 4, 5]). Compute also $\beta(0)$ as an inner product.

2. Assume that we have $\{H_k, r_k, p_k, v_k, q_k, x_k, \beta(k)\}$ and iterate:

   - $x_{k+1} = x_k + \beta(k)r_k[\eta(k) - a_k^T x_k]$ [$O(n)$ flops].
   - Compute the QR factorization (2) [$O(n\alpha^2 + l_1)$ flops].
   - Compute the eigendecomposition (3) [$O(\alpha^3)$ flops].
   - Let $H_{k+1} = Q_k W_k \sqrt{|\Lambda_k|}$ and discard the two zero columns of H_{k+1} [$O(n\alpha^2)$ flops].
   - Compute $p_{k+1} = p_k - \beta(k)[a_k^T p_k]r_k$ [$O(n)$ flops].
   - Compute $v_{k+1} = v_k - \beta(k)[a_k^T v_k]r_k$ [$O(n)$ flops].
   - Compute $r_{k+1} = q_k + \delta(k)p_k - \beta(k)[r_k^T a_{k+1}]r_k$ [$O(n)$ flops].

- Compute $q_{k+1} = \delta(k)v_k - \beta(k)[r_k^T F a_{k+1}]r_k + F^{-T}[q_k - H_k J H_k^T F a_k]$ [$O(l_1 + l_2 + n\alpha)$ flops].
- Compute $\beta(k+1) = a_{k+1}^T r_{k+1}$ [$O(n)$ flops].

4 Shift Structured Updates

Let us now consider the case in which F is singular, e.g., $F = Z$, the lower triangular shift matrix (i.e., a Jordan block with ones on the first sub-diagonal), or $F = Z \oplus Z$, etc. We focus on the case $F = Z$ since the argument can be generalized to other singular matrices F.

The only difficulty that we need to resolve in this situation is how to compute q_{k+1} recursively in $O(n\alpha^2 + l_1 + l_2)$ flops. Recall from the first recursion for q_{k+1} in the previous section that

$$F^T(q_{k+1} - \delta(k)v_k + \beta(k)(r_k^T F a_{k+1})r_k) = q_k - H_k J H_k^T F a_k. \tag{4}$$

Since we have assumed that $F = Z$, this equation can be solved for all the components of q_{k+1} *except* the first one. So all we need to do is to recover the first component of q_{k+1} efficiently.

To do this, we examine the following equation:

$$q_{k+1} - \delta(k)v_k + \beta(k)(r_k^T F a_{k+1})r_k = T_k^{-1} F^2 a_k.$$

We see that all we need is the first row of T_k^{-1}. Therefore let $y_k = T_k^{-1} e_1$, where e_1 is the first basis vector. Then we see that

$$e_1^T q_{k+1} = y_k^T F^2 a_k + \delta(k) e_1^T v_k - \beta(k)(r_k^T F a_{k+1}) e_1^T r_k, \tag{5}$$

which can be computed in $O(n)$ flops if y_k is available. It is convenient to combine equations (4) and (5) into one equation as follows (exploiting the fact that $F = Z$):

$$q_{k+1} = \delta(k)v_k - \beta(k)(r_k^T F a_{k+1})r_k + y_k^T F^2 a_k e_1 + F(q_k - H_k J H_k^T F a_k).$$

The cost of evaluating q_{k+1} from this formula is $O(n\alpha)$ flops.

Clearly $y_0 = T_0^{-1} e_1$ can be computed in $O(n^2)$ flops using G_0. This can be regarded as an overhead cost. We now derive a rapid recurrence for y_{k+1} as follows:

$$\begin{aligned} y_{k+1} &= T_{k+1}^{-1} e_1 = T_k^{-1} e_1 - \beta(k) r_k r_k^T e_1 \\ &= y_k - \beta(k)(r_k^T e_1) r_k \\ &= y_k - \beta(k)(a_k^T y_k) r_k. \end{aligned}$$

Either of the last two expressions can be evaluated in $O(n)$ flops to yield y_{k+1} recursively.

The cost per iteration of the above algorithm is $O(n\alpha^2)$.

Fast Updating of Structured Linear Equations with Affine Transformed Updates with $F = Z$ (FUSAZ)

1. Initialization (overhead costs). Given $\{G_0, H_0\}$, compute x_0 and $\{r_0, v_0, q_0, p_0, y_0\}$ using G_0 and the so-called generalized Schur algorithm with back-substitution, or by using H_0 and fast matrix-vector multiplication procedures for matrices with displacement structure (see, e.g., [1, 2, 4, 5]). Compute also $\beta(0)$ as an inner product.

2. Given $\{H_k, r_k, p_k, v_k, q_k, , y_k, x_k, \beta(k)\}$, iterate:
   - $x_{k+1} = x_k + \beta(k) r_k [\eta(k) - a_k^T x_k]$ [$O(n)$ flops].
   - Compute the QR factorization (2) [$O(n\alpha^2 + l_1)$ flops].
   - Compute the eigendecomposition (3) [$O(\alpha^3)$ flops].
   - Let $H_{k+1} = Q_k W_k \sqrt{|\Lambda_k|}$ and discard the two zero columns of H_{k+1} [$O(n\alpha^2)$ flops].
   - Compute $p_{k+1} = p_k - \beta(k)[a_k^T p_k] r_k$ [$O(n)$ flops].
   - Compute $v_{k+1} = v_k - \beta(k)[a_k^T v_k] r_k$ [$O(n)$ flops].
   - Compute $y_{k+1} = y_k - \beta(k)[a_k^T y_k] r_k$ [$O(n)$ flops].
   - Compute $r_{k+1} = q_k + \delta(k) p_k - \beta(k)[r_k^T a_{k+1}] r_k$ [$O(n)$ flops].
   - Compute $q_{k+1} = \delta(k) v_k - \beta(k)[r_k^T F a_{k+1}] r_k + y_k^T F^2 a_k e_1 + F[q_k - H_k J H_k^T F a_k]$ [$O(n\alpha)$ flops].
   - Compute $\beta(k+1) = a_{k+1}^T r_{k+1}$ [$O(n)$ flops].

5 Adaptive Filtering

We now comment briefly on an application in the context of adaptive filtering [6, 7]. More details along with connections with, and alternatives to, existing fast RLS schemes [8, 9, 10] and fast state-space estimation algorithms [11, 12] will be pursued elsewhere.

Thus consider a sequence of k scalar data points, $\{d(j)\}_{j=1}^k$, also known as reference or desired signals, and a sequence of k row vectors $\{u_j^T\}_{j=1}^k$, also known as input signals, with the entries of each u_j^T denoted by

$$u_j^T = \begin{bmatrix} u(j) & u(j-1) & \ldots & u(j-M+1) \end{bmatrix}. \tag{6}$$

Consider also a positive-definite weighting matrix Π_0. The objective is to minimize the following cost function over w:

$$\min_w \left[w^T \Pi_0^{-1} w + \sum_{j=1}^k |d(j) - u_j^T w|^2 \right]. \tag{7}$$

The optimal solution w_k of (7) is well-known to be the solution of the linear system of equations $\Phi_k w_k = s_k$, where Φ_k and s_k satisfy the time-update relations

$$\begin{aligned} \Phi_{k+1} &= \Phi_k + u_{k+1}u_{k+1}^T , & (8) \\ s_{k+1} &= s_k + d(k+1)u_{k+1} , & (9) \end{aligned}$$

with initial conditions $\Phi_0 = \Pi_0^{-1}$ and $s_0 = 0$.

It can be verified that, in general, the difference $\Phi_{k+1} - Z\Phi_k Z^T$ has rank 3 and inertia $(1, 1, -1)$, so that generally Φ_k itself will have displacement rank $\alpha = 4$, viz., it will satisfy a displacement equation of the form

$$\Phi_k - Z\Phi_k Z^T = G_k J G_k^T ,$$

where G_k has four columns and $J = \text{diag}(1, 1, -1, -1)$; see also [13] (the effect of Π_0 is not considered in [13]). The values of $\{G_0, H_0\}$ are determined by the choice of Π_0. For example, if we choose Π_0 as $\Pi_0 = \mu I$ for some $\mu > 0$, then

$$\Pi_0 - Z\Pi_0 Z^T = \mu e_1 e_1^T ,$$

so that we can take

$$H_0 = \begin{bmatrix} \sqrt{\mu}e_1 & 0 & 0 & 0 \end{bmatrix} , \quad J = \text{diag}(1, 1, -1, -1) .$$

Moreover, the famed recursive least-squares (RLS) algorithm is a recursive procedure for evaluating the successive w_k. If we define

$$P_k = \Phi_k^{-1} , \quad g_k = \Phi_k^{-1} u_k , \qquad (10)$$

then the RLS algorithm is given by

$$\begin{aligned} w_k &= w_{k-1} + g_k \left[d(k) - u_k^T w_{k-1}\right] , & (11) \\ g_k &= \frac{P_{k-1}u_k}{1 + u_k^T P_{k-1} u_k} , & (12) \\ P_k &= P_{k-1} - g_k u_k^T P_{k-1} . & (13) \end{aligned}$$

In addition, with the RLS problem we associate two residuals at each time instant k: the *a priori* estimation error $e_a(k)$, defined by $e_a(k) = d(k) - u_k^T w_{k-1}$, and the *a posteriori* estimation error $e_p(k)$, defined by $e_p(k) = d(k) - u_k^T w_k$. If we replace w_k in the definition for $e_p(k)$ by its update expression (11), some straightforward algebra will show that $e_p(k) = \gamma(k) e_a(k)$, where the so-called conversion factor $\gamma(k)$ is given by

$$\gamma(k) = \frac{1}{1 + u_k^T P_{k-1} u_k} .$$

Comparing all these results with the definitions employed in the earlier sections for the solution of equations of the form $T_k x_k = b_k$, we see that we can make the identifications shown in the table between the variables of the RLS problem and the variables of the earlier sections.

General problem	RLS problem
T_k	Φ_k
b_k	s_k
x_k	w_k
a_k	u_{k+1}
$\eta(k)$	$d(k+1)$
F	Z
$\delta(k)$	$u(k+2)$
g	e_1
$\beta(k)$	$\gamma(k+1)$
r_k	$g_{k+1}/\gamma(k+1)$

References

[1] T. Kailath and A. H. Sayed, Displacement structure: Theory and applications, *SIAM Review*, vol. 37, no. 3, pp. 297–386, Sep. 1995.

[2] T. Kailath and A. H. Sayed, eds., *Fast Reliable Algorithms for Matrices with Structure*, SIAM, PA, 1999.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd edition, The Johns Hopkins University Press, Baltimore, 1996.

[4] S. Chandrasekaran and A. H. Sayed, Stabilizing the generalized Schur algorithm, *SIAM J. Matrix Anal. Appl.* vol. 17, no. 4, pp. 950–983, 1996.

[5] S. Chandrasekaran and A. H. Sayed, A fast stable solver for nonsymmetric Toeplitz and quasi-Toeplitz systems of linear equations, *SIAM J. Mat. Anal. and Appl.*, vol. 19, no. 1, pp. 107–139, Jan. 1998.

[6] S. Haykin, *Adaptive Filter Theory*, 3rd edition, Prentice Hall, NJ, 1996.

[7] A. H. Sayed and T. Kailath, A state-space approach to adaptive RLS filtering, *IEEE Signal Processing Magazine*, 11, pp. 18–60, July 1994.

[8] G. Carayannis, D. Manolakis, and N. Kalouptsidis, A fast sequential algorithm for least-squares filtering and prediction, *IEEE Trans. Acoust. Speech Signal Process.*, vol. 31, pp. 1394–1402, Dec. 1983.

[9] J. Cioffi and T. Kailath, Fast recursive-least-squares transversal filters for adaptive filtering, *IEEE Trans. Acoust. Speech Signal Process.*, vol. 32, pp. 304–337, April 1984.

[10] R. Merched and A. H. Sayed, Fast RLS Laguerre adaptive filtering, *Proc. Allerton Conference on Communication, Control, and Computing*, Allerton, IL, Sep. 1999.

[11] A. H. Sayed and T. Kailath, Extended Chandrasekhar recursions, *IEEE Trans. Automat. Contr.*, 39, pp. 619–623, Mar. 1994.

[12] A. P. Mullhaupt and K. S. Riedel, Fast adaptive identification of stable innovation filters, *IEEE Transactions on Signal Processing*, vol.45, no.10, p. 2616-2619, Oct. 1997.

[13] R. A. Regalia and F. Desbouvries, Displacement structures of covariance matrices, lossless systems, and numerical algorithm design, *SIAM J. Matrix Anal. Appl.*, vol. 16, no. 2, pp. 536–564, April 1995.