Bi-Iterative Least-Square Method for Subspace Tracking

Shan Ouyang, *Member, IEEE,* and Yingbo Hua, *Fellow, IEEE*

***Abstract*—Subspace tracking is an adaptive signal processing technique useful for a variety of applications. In this paper, we introduce a simple bi-iterative least-square (Bi-LS) method, which is in contrast to the bi-iterative singular value decomposition (Bi-SVD) method. We show that for subspace tracking, the Bi-LS method is easier to simplify than the Bi-SVD method. The linear complexity algorithms based on Bi-LS are computationally more efficient than the existing linear complexity algorithms based on Bi-SVD, although both have the same performance for subspace tracking. A number of other existing subspace tracking algorithms of similar complexity are also compared with the Bi-LS algorithms.**

***Index Terms*—Adaptive signal processing, bi-iteration, low-rank approximation, projection approximation, QR decomposition, singular value decomposition, subspace tracking.**

I. Introduction

A. Need for Subspace Tracking

THE subspace of a vector sequence is well known for its importance in a wide range of signal processing applications, such as frequency estimation, target localization, channel estimation, multiuser detection, and image feature extraction, just to name a few. Let $\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots, \boldsymbol{x}(L)$ be a sequence of L vectors, each of which is N-dimensional. The span of this vector sequence can be divided into a principal subspace and a minor subspace, and these two subspaces are orthogonal complement of each other. The minor subspace can be computed uniquely from the corresponding principal subspace, and vice versa. By subspace, we will refer to the principal subspace of a vector sequence unless mentioned otherwise.

The computation of the principal subspace can be done by computing the singular value decomposition (SVD) of the matrix that consists of the above L vectors as columns (or rows). However, the computational cost is high. Assuming $N = L$, the number of required flops is in the order of N^3, i.e., $O(N^3)$. In this paper, a flop is a complex multiplication and a complex addition. We use flop counts as a key measure of complexity of an algorithm, although many other parameters, such as parallelism and memory requirement, are also important in practice. If the dimension r of the subspace is much smaller than N, faster algorithms are available that require $O(N^2r)$ flops. For a review of such algorithms, see the work by Comon and Golub [9].

In adaptive applications, a vector sequence may change frequently with time, and furthermore, a new vector sequence may overlap significantly with an old vector sequence. Spending $O(N^2r)$ flops for each vector sequence can still be too costly. To obtain more efficient algorithms, one idea is that the subspace of an old vector sequence should be used to compute the subspace of a new vector sequence. Many researchers have attempted this idea. Unfortunately, it seems true that if the exact subspace of each new vector sequence is desired, then the computational order cannot be made less than $O(N^2r)$, even if the exact subspace of the old vector sequence is known and the new vector sequence differs from the old only by one vector.

An alternative is that some error is tolerated, and the problem of subspace computation is reformulated as subspace estimation. This is the approach on which we will focus in this paper. Indeed, when a new vector sequence differs from its previous vector sequence by only one vector, the error in the estimated subspace of the new vector sequence can be made small with algorithms of the complexity $O(Nr^2)$, or even $O(Nr)$. Efficient algorithms that *estimate* the subspace of each vector sequence in an adaptive and efficient fashion are collectively called subspace tracking algorithms. The complexity $O(Nr^2)$ or $O(Nr)$ is a linear complexity as it is called a linear function of N.

B. Power Family

Linear complexity subspace tracking algorithms have been an active research topic for many years. Recent reviews of linear complexity algorithms are available in the work by DeGroat *et al.* [10] and Hua *et al.* [18]. It is an interesting observation that most (if not all) of the high accuracy linear complexity algorithms belong to a family of power-based algorithms or, simply, the power family [18]. A key feature of the power family is that the primary new information in the updated subspace comes from multiplying the old subspace matrix by the underlying new data. The power family includes the Oja algorithm [23], the PAST algorithm [33], the NIC algorithm [22], the Bi-SVD algorithm [30], and many other variations [4], [5], [11], [12], [24], [28].

Oja's subspace algorithm is also a gradient-based neural network algorithm [32]. The convergence property of a generalized Oja subspace algorithm with an arbitrarily small step size

Manuscript received November 9, 2003; revised October 11, 2004. This work was supported in part by the U.S. National Science Foundation under Award ECS-0219377 and the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011, the National Nature Science Foundation (Project 60172011) of P.R. China, and the Foundation of Authors of the National Excellent Doctoral Dissertation (Project 200239) of P.R. China. Part of this work was presented at IEEE ICASSP 2004. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Carlos H. Muravchik.

S. Ouyang is with the Department of Communication and Information Engineering, Guilin University of Electronic Technology, Guilin 541004, China (e-mail: hmoysh@gliet.edu.cn).

Y. Hua is with the Department of Electrical Engineering, University of California, Riverside, CA 92521 USA (e-mail: yhua@ee.ucr.edu).

Digital Object Identifier 10.1109/TSP.2005.851102

is established in [7]. The Oja algorithm is perhaps the simplest in computation among all linear complexity algorithms, but the accuracy of the Oja algorithm is highly sensitive to the chosen step size, and it is difficult to choose a proper step size to guarantee both small misadjustment and fast convergence [6].

The PAST algorithm, which was developed by Yang [33], has the computational complexity $3Nr + O(r^2)$ and has a guaranteed stability due to its power-based nature [18]. The PAST algorithm does not guarantee the orthonormality of the estimated subspace matrix [18], although the orthonormality may be desired in some applications [21].

The NIC algorithm introduced in [22] is a generalization of the PAST algorithm. The leakage factor inherent in the NIC algorithm ensures the orthonormality of the subspace matrix after a number of iterations [14]. Alternatively, an explicit orthonormalization at each iteration can be carried out for the PAST algorithm, which results in the OPAST algorithm [1].

An efficient method to guarantee orthonormality is to apply QR decomposition whenever necessary in an algorithm. The QR decomposition is a tool very commonly used in matrix computations [13]. A QR-based subspace tracking algorithm is Karasalo's algorithm [20], which has the complexity $O(Nr^2)$. Replacing a small-dimensional SVD in Karasalo's algorithm by a transposed QR iteration, a TQR-SVD subspace tracking algorithm is developed in [11], which has the same complexity order as Karasalo's algorithm. Based on the URV decomposition due to Stewart [28], a fast subspace tracking (FST) algorithm is developed by Rabideau in [24], which has the complexity $O(Nr)$. Based on the bi-iterative QR decomposition for SVD computation [27], the Bi-SVD subspace tracking algorithm proposed by Strobach [30] has the complexity order $O(Nr^2)$. It is implied in [30] that the Bi-SVD algorithm outperforms all its related predecessors. Note that the Bi-SVD algorithm uses an alternating power iteration and, hence, belongs to the power family.

C. Choice of Windows

The output of a subspace tracking algorithm directly depends on the data window that is either explicitly or implicitly exploited by the algorithm. The data window defines the actual (or effective) vector sequence under consideration. Two types of windows have been used for subspace tracking, which are known as exponential window and sliding rectangular window. Most of the existing subspace tracking algorithms are designed for exponentially windowed data. For an exponentially windowed data matrix, a rank-one update of the underlying covariance matrix is required for each new data vector, which leads to simple subspace tracking algorithms. On the other hand, for a sliding-windowed data matrix, a rank-two update of the underlying covariance matrix is required for each new data vector, which involves more computations than for the exponentially windowed data. A sliding window also requires more memory than an exponential window. Recently, a number of sliding-rectangular-window-based subspace tracking algorithms have been investigated, e.g., see Badeau *et al.* [2], [3], Real *et al.* [25], and Strobach [29]. A motivation for using a sliding window is that the resulting algorithms have a faster convergence speed, as recently shown by Badeau *et al.* We will confirm that the convergence speed is governed by the effective window length and the shape of the window affects the sharpness of the converging edge. In particular, a sliding rectangular window causes a sharper converging edge than an exponential window.

D. Key Contribution of This Paper

We revisit a bi-iterative least-square (Bi-LS) method that computes the optimal low-rank matrix approximation. Since the optimal low-rank matrix approximation carries all the information of the (principal) subspace of the underlying vector sequence, the Bi-LS method is naturally useful for subspace tracking. The Bi-LS method is different from the Bi-SVD method [8], the latter of which has served as the fundamental basis of several other subspace tracking algorithms. We show that for developing efficient subspace tracking algorithms, the Bi-LS method provides a more convenient framework than the Bi-SVD method. As a result, the linear complexity Bi-LS algorithms are computationally more efficient than the linear complexity Bi-SVD algorithms, although both have the same accuracy for subspace tracking.

The rest of this paper is organized as follows. In Section II, we review the Bi-SVD method. In Section III, we present the Bi-LS method based on the QR decomposition. In Section IV, we derive several linear-complexity subspace tracking algorithms based on the Bi-LS method. We also consider a hybrid window, i.e., a sliding exponential window. In Section V, the performance of the Bi-LS subspace tracking algorithms for tracking abrupt changes is demonstrated and compared with those of other algorithms with comparable complexity. Section VI concludes this paper.

II. Review of Bi-Iterative Singular Value Decomposition

Before introducing the Bi-LS method, we first review the bi-iterative SVD (Bi-SVD) method [8], [27], which led to the subspace tracking algorithms shown in [3], [29], [30].

Consider a data matrix $\boldsymbol{X} \in \mathbb{C}^{L\times N}$. The r dominant singular values and the r dominant singular vectors of $\boldsymbol{X}$ can be computed by the bi-iterative method listed in Table I. It is known [8], [27] that the columns of $\boldsymbol{Q}_A(k) \in \mathbb{C}^{L\times r}$ converge to the r dominant left singular vectors, the columns of $\boldsymbol{Q}_B(k) \in \mathbb{C}^{N\times r}$ converge to the r dominant right singular vectors, and each of $\boldsymbol{R}_A(k)$ and $\boldsymbol{R}_B(k)$ converges (in absolute value) to the $r \times r$ diagonal matrix of the dominant singular values of $\boldsymbol{X}$.

For subspace tracking, the data matrix $\boldsymbol{X}(t)$ at time t may be updated with an exponential window as follows:

$$\boldsymbol{X}(t) = \begin{bmatrix} (1-\alpha)^{1/2}\boldsymbol{x}^H(t) \\ \alpha^{1/2}\boldsymbol{X}(t-1) \end{bmatrix} \tag{1}$$

where $0 < \alpha < 1$ is an exponential forgetting factor, and $\boldsymbol{x}(t)$ is the new data vector. Alternatively, the data matrix $\boldsymbol{X}(t)$ may be updated with a sliding rectangular window as follows:

$$\begin{bmatrix} \boldsymbol{X}(t) \\ \boldsymbol{x}^H(t-L) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}^H(t) \\ \boldsymbol{X}(t-1) \end{bmatrix}. \tag{2}$$

To reduce the computational burden, we can use only one iteration for each new data vector, or equivalently, we replace the

TABLE I
BI-SVD METHOD FOR SVD COMPUTATION

Initialization: $\boldsymbol{Q}_B(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}$	
For $k = 1, 2, \cdots$ until convergence Do:	
First Step:	
$\boldsymbol{A}(k) = \boldsymbol{X}\boldsymbol{Q}_B(k-1)$	
$\boldsymbol{A}(k) = \boldsymbol{Q}_A(k)\boldsymbol{R}_A(k)$	skinny QR decomposition
Second Step:	
$\boldsymbol{B}(k) = \boldsymbol{X}^H\boldsymbol{Q}_A(k)$	
$\boldsymbol{B}(k) = \boldsymbol{Q}_B(k)\boldsymbol{R}_B(k)$	skinny QR decomposition

TABLE II
BASIC BI-SVD ALGORITHM FOR SUBSPACE TRACKING

Initialization: $\boldsymbol{Q}_B(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}$	
For $t = 1, 2, \cdots$, Do:	
First Step:	Complexity:
$\boldsymbol{A}(t) = \boldsymbol{X}(t)\boldsymbol{Q}_B(t-1)$	$O(NLr)$
$\boldsymbol{A}(t) = \boldsymbol{Q}_A(t)\boldsymbol{R}_A(t)$	$O(Lr^2)$
Second Step:	
$\boldsymbol{B}(t) = \boldsymbol{X}^H(t)\boldsymbol{Q}_A(t)$	$O(NLr)$
$\boldsymbol{B}(t) = \boldsymbol{Q}_B(t)\boldsymbol{R}_B(t)$	$O(Nr^2)$

iteration index k in Table I by the discrete time index t, which leads to the basic Bi-SVD subspace tracking algorithm given in Table II. The basic Bi-SVD subspace tracking algorithm has the computational complexity $O(NLr)$ for each iteration. Note that because only one iteration is allowed for each new data vector, only an estimate of the principal singular vectors and the principal singular values is obtained at any time t.

As mentioned earlier, in order to develop linear complexity algorithms, some approach of approximations is necessary. Such a approach of approximations is as follows. At each iteration, we partition a power-based estimate of the desired subspace matrix into two components: "innovation" and "propagation". The innovation depends on a new data vector, and the propagation does not. Then, a proper approximation is applied to the propagation. The choice of such an approximation has a major effect on the performance of the resulting algorithm. One such approximation is called low-rank approximation, as described next.

From the first step in Table II, we can express a low-rank approximation to $\boldsymbol{X}(t)$ as

$$\boldsymbol{X}_r(t) \simeq \boldsymbol{A}(t)\boldsymbol{Q}_B^H(t-1) = \boldsymbol{Q}_A(t)\boldsymbol{R}_A(t)\boldsymbol{Q}_B^H(t-1) \tag{3}$$

where $\boldsymbol{X}_r(t)$ denotes the optimal rank-r approximation of $\boldsymbol{X}(t)$, and the right-hand-side terms are suboptimal rank-r approximations. With the above suboptimal rank-r approximation, Strobach [30] developed a fast exponential-window-based Bi-SVD subspace tracking algorithm (Bi-SVD1) with the computational complexity $O(Nr^2)$ per iteration.

From the second step in Table II, we can express another low-rank approximation to $\boldsymbol{X}(t)$ as

$$\boldsymbol{X}_r(t) \simeq \boldsymbol{Q}_A(t)\boldsymbol{B}(t)^H = \boldsymbol{Q}_A(t)\boldsymbol{R}_B^H(t)\boldsymbol{Q}_B^H(t). \tag{4}$$

Using this approximation, a fast sliding-window-based Bi-SVD subspace tracking algorithm is recently shown by Badeau *et al.* [3], which has the computational complexity $O((N+L)r)$.

In fact, the above two low-rank approximations are actually equivalent. From the second step in Table II, we have

$$\boldsymbol{R}_B^H(t)\boldsymbol{Q}_B^H(t) = \boldsymbol{Q}_A^H(t)\boldsymbol{X}(t). \tag{5}$$

Postmultiplying both sides of (5) by $\boldsymbol{Q}_B(t-1)$

$$\begin{aligned}\boldsymbol{R}_B^H(t)\boldsymbol{Q}_B^H(t)\boldsymbol{Q}_B(t-1) &= \boldsymbol{Q}_A^H(t)\boldsymbol{X}(t)\boldsymbol{Q}_B(t-1) \\ &= \boldsymbol{R}_A(t).\end{aligned} \tag{6}$$

Therefore, the above two low-rank approximations yield the same result. Note also from (6) that since $\boldsymbol{R}_B^H(t)$ is lower triangular and $\boldsymbol{R}_A(t)$ is upper triangular, $\boldsymbol{Q}_B^H(t)\boldsymbol{Q}_B(t-1) = \boldsymbol{R}_B^{-H}(t)\boldsymbol{R}_A(t) \neq \boldsymbol{I}_r$, i.e., $\boldsymbol{Q}_B(t) \neq \boldsymbol{Q}_B(t-1)$. This is a drawback for the Bi-SVD method.

III. Bi-ITERATIVE LEAST-SQUARE METHOD

The optimal low-rank (rank-r) approximation to $\boldsymbol{X}$ can be expressed as the solution to the following minimization:

$$\boldsymbol{J}(\boldsymbol{A}, \boldsymbol{B}) = \|\boldsymbol{X} - \boldsymbol{A}B^H\|_F^2 \tag{7}$$

where $\boldsymbol{A} \in \mathbb{C}^{L\times r}$, and $\boldsymbol{B} \in \mathbb{C}^{N\times r}$. Some of the (nonunique) optimal minimizers $\boldsymbol{A}$ and $\boldsymbol{B}$ can be obtained by the iterative-quadratic-minimum-distance (IQMD) [15] method or the alternating power (AP) [16] method:

$$\begin{cases} \boldsymbol{A}(k) = \boldsymbol{X}B(k-1)\boldsymbol{G}(k-1) \\ \boldsymbol{B}(k) = \boldsymbol{X}^H\boldsymbol{A}(k)(\boldsymbol{A}^H(k)\boldsymbol{A}(k))^{-1} \end{cases} \tag{8}$$

where $\boldsymbol{G}(k-1) = (\boldsymbol{B}^H(k-1)\boldsymbol{B}(k-1))^{-1}$. It is known [15], [16] that with a weak condition on the initial matrix $\boldsymbol{B}(0)$ and the data matrix $\boldsymbol{X}$, the product of $\boldsymbol{A}(k)\boldsymbol{B}^H(k)$ from (8) globally and exponentially converges to the optimal rank-r approximation $\boldsymbol{X}_r$ of $\boldsymbol{X}$. Moreover, the global convergence of the AP method is generally not affected, even if $\boldsymbol{G}(k-1)$ is chosen as an arbitrary nonsingular matrix [16].

Similar to the Bi-SVD method, let us write the QR decomposition of $\boldsymbol{A}(k)$ and $\boldsymbol{B}(k)$ as

$$\begin{cases} \boldsymbol{A}(k) = \boldsymbol{Q}_A(k)\boldsymbol{R}_A(k) \\ \boldsymbol{B}(k) = \boldsymbol{Q}_B(k)\boldsymbol{R}_B(k). \end{cases} \tag{9}$$

Substituting $\boldsymbol{B}(k-1) = \boldsymbol{Q}_B(k-1)\boldsymbol{R}_B(k-1)$ and $\boldsymbol{A}(k) = \boldsymbol{Q}_A(k)\boldsymbol{R}_A(k)$ into the right-hand side of (8) yields

$$\begin{cases} \boldsymbol{A}(k) = \boldsymbol{X}\boldsymbol{Q}_B(k-1) \\ \boldsymbol{B}(k) = \boldsymbol{X}^H\boldsymbol{Q}_A(k)\boldsymbol{R}_A^{-H}(k) \end{cases} \tag{10}$$

where, for simplicity, the choice $\boldsymbol{G}(k-1) = \boldsymbol{R}_B^{-1}(k-1)$ is used. This QR-decomposition-based iterative method for optimal low-rank matrix approximation to $\boldsymbol{X}$ is summarized in Table III, which we now call the bi-iterative least-square (Bi-LS) method to highlight a contrast against the Bi-SVD

TABLE III
METHOD FOR OPTIMAL LOW-RANK MATRIX APPROXIMATION

Initialization: $\boldsymbol{Q}_B(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}$

For $k = 1, 2, \cdots$ until convergence Do:

First Step:

$\boldsymbol{A}(k) = \boldsymbol{X}\boldsymbol{Q}_B(k-1)$

$\boldsymbol{A}(k) = \boldsymbol{Q}_A(k)\boldsymbol{R}_A(k)$ skinny QR decomposition

Second Step:

$\boldsymbol{B}(k) = \boldsymbol{X}^H\boldsymbol{Q}_A(k)\boldsymbol{R}_A^{-H}(k)$

$\boldsymbol{B}(k) = \boldsymbol{Q}_B(k)\boldsymbol{R}_B(k)$ skinny QR decomposition

TABLE IV
BASIC BI-LS ALGORITHM FOR SUBSPACE TRACKING

Initialization: $\boldsymbol{Q}_B(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}$

For $t = 1, 2, \cdots$, Do:

First Step:	Complexity:
$\boldsymbol{A}(t) = \boldsymbol{X}(t)\boldsymbol{Q}_B(t-1)$	$O(NLr)$
$\boldsymbol{A}(t) = \boldsymbol{Q}_A(t)\boldsymbol{R}_A(t)$	$O(Lr^2)$
Second Step:	
$\boldsymbol{B}(t) = \boldsymbol{X}^H(t)\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t)$	$O(NLr + Lr^2)$
$\boldsymbol{B}(t) = \boldsymbol{Q}_B(t)\boldsymbol{R}_B(t)$	$O(Nr^2)$

method shown earlier. It is obvious that the principal column span of $\boldsymbol{X}$ is given by the column span of $\boldsymbol{Q}_A(k)$ at convergence and the principal column span of $\boldsymbol{X}^H$ is given by the column span of $\boldsymbol{Q}_B(k)$ at convergence. If only one iteration is allowed for each new data vector, we have the basic Bi-LS algorithm for subspace tracking, as shown in Table IV. The basic Bi-LS algorithm has the complexity $O(NLr)$.

On one hand, we observe that the Bi-LS method is seemingly more complicated than the Bi-SVD method since in the second step of the iteration, the Bi-LS method needs to compute $\boldsymbol{R}_A^{-H}(k)$ additional matrix multiplication. On the other hand, we expect the Bi-LS method to have a higher degree of flexibility for subspace tracking than the Bi-SVD method. This is because the latter is meant to yield the principal singular vectors and the principal singular values, which are more than the optimal low-rank matrix approximation.

A unique property of the Bi-LS method is that from (10)

$$\begin{aligned}\boldsymbol{Q}_B^H(k-1)\boldsymbol{B}(k) &= \boldsymbol{Q}_B^H(k-1)\boldsymbol{X}^H\boldsymbol{Q}_A(k)\boldsymbol{R}_A^{-H}(k) \\ &= \boldsymbol{A}^H(k)\boldsymbol{Q}_A(k)\boldsymbol{R}_A^{-H}(k) = \boldsymbol{I}_r. \end{aligned} \quad (11)$$

As will be seen, this property is very useful for developing fast subspace tracking algorithms. The above property, together with (9), also implies that

$$\boldsymbol{Q}_B^H(k-1)\boldsymbol{Q}_B(k) = \boldsymbol{R}_B^{-1}(k). \quad (12)$$

It can be shown (see the Appendix) that for the Bi-LS method, the upper triangular matrix $\boldsymbol{R}_B(k)$ becomes the identity matrix at the convergence of subspace. Hence, from $\boldsymbol{B}(k) = \boldsymbol{Q}_B(k)\boldsymbol{R}_B(k)$, we may use the approximation $\boldsymbol{B}(k) \simeq \boldsymbol{Q}_B(k)$ to develop more efficient algorithms. More details on this will be shown later.

It is easy to verify that for the Bi-LS method, $\boldsymbol{Q}_A(k)$ converges to a product of the matrix of the left principal singular vectors and an $r \times r$ unitary rotation matrix $\boldsymbol{T}_A(k)$, and $\boldsymbol{Q}_B(k)$ converges to a product of the matrix of the right principal singular vectors and another $r \times r$ unitary rotation matrix $\boldsymbol{T}_B(k)$. Both $\boldsymbol{T}_A(k)$ and $\boldsymbol{T}_B(k)$ depend on the initialization $\boldsymbol{Q}_B(0)$. Each of $\boldsymbol{T}_A(k)$ and $\boldsymbol{T}_B(k)$, when desired, can be computed from the SVD of the $r \times r$ matrix $\boldsymbol{R}_A(k)$, which obviously does not affect the complexity order of the Bi-LS method, provided that r is smaller than $\min(N, L)$. If the matrix of the principal left (or right) singular vectors is explicitly required, then the multiplication $\boldsymbol{Q}_A(k)\boldsymbol{T}_A(k)$ (or $\boldsymbol{Q}_B(k)\boldsymbol{T}_B(k)$) would cost an additional Lr^2 (or Nr^2) flops, but the largest r singular values of the underlying data matrix $\boldsymbol{X}(k)$ can be approximated by the r singular values of the $r \times r$ upper triangular matrix $\boldsymbol{R}_A(k)$ with a negligible cost.

IV. FAST BI-LS ALGORITHMS FOR SUBSPACE TRACKING

In this section, we derive several linear complexity Bi-LS algorithms for subspace tracking.

A. Bi-LS-1 Algorithm

We first define a hybrid data window as follows:

$$\begin{bmatrix} \boldsymbol{X}(t) \\ \alpha^{L/2}\beta^{1/2}\boldsymbol{x}^H(t-L) \end{bmatrix} = \begin{bmatrix} \beta^{1/2}\boldsymbol{x}^H(t) \\ \alpha^{1/2}\boldsymbol{X}(t-1) \end{bmatrix} \quad (13)$$

where $0 < \beta \leq 1$, and $0 < \alpha \leq 1$. Note that if $\alpha = \beta = 1$, (13) reduces to the sliding window (2). If $\beta = 1$, (13) is a sliding exponential window used in [19]. We may also choose $\beta = 1-\alpha$ for a sliding exponential window.

1) Updating $\boldsymbol{Q}_A(t)$: We now refer to the basic Bi-LS algorithm shown in Table IV. By post-multiplying both sides of (13) by $\boldsymbol{Q}_B(t-1)$, we can show that

$$\begin{bmatrix} \boldsymbol{A}(t) \\ \boldsymbol{h}_L^H(t) \end{bmatrix} = \begin{bmatrix} \beta^{1/2}\boldsymbol{h}^H(t) \\ \alpha^{1/2}\boldsymbol{X}(t-1)\boldsymbol{Q}_B(t-1) \end{bmatrix} \quad (14)$$

where $\boldsymbol{h}_L(t) \triangleq \alpha^{L/2}\beta^{1/2}\boldsymbol{Q}_B^H(t-1)\boldsymbol{x}(t-L)$, and $\boldsymbol{h}(t) \triangleq \boldsymbol{Q}_B^H(t-1)\boldsymbol{x}(t)$.

As mentioned before, a key step in developing linear complexity algorithms is to apply a proper approximation to the propagation, which in the current case is the lower matrix on the right-hand side of (14). By the first step in Table IV, we have

$$\boldsymbol{X}(t)\boldsymbol{Q}_B(t-1) = \boldsymbol{Q}_A(t)\boldsymbol{R}_A(t). \quad (15)$$

Applying the low-rank approximation (3) to $\boldsymbol{X}(t-1)$, we obtain

$$\begin{aligned}&\boldsymbol{X}(t-1)\boldsymbol{Q}_B(t-1) \\ &\quad \simeq \boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1)\boldsymbol{Q}_B^H(t-2)\boldsymbol{Q}_B(t-1) \\ &\quad = \boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1)\boldsymbol{R}_B^{-1}(t-1) \end{aligned} \quad (16)$$

where (12) has been employed. Since both $\boldsymbol{R}_A(t-1)$ and $\boldsymbol{R}_B^{-1}(t-1)$ are upper triangular matrices, the product of

$\boldsymbol{R}_A(t-1)$ and $\boldsymbol{R}_B^{-1}(t-1)$ is still an upper triangular matrix. This implies that $\boldsymbol{R}_B^{-1}(t-1)$ does not affect the matrix $\boldsymbol{Q}_A(t-1)$. Furthermore, according to a previous discussion, $\boldsymbol{R}_B(t)$ may be approximated by an identity matrix. Therefore, (16) can be simplified to

$$\boldsymbol{X}(t-1)\boldsymbol{Q}_B(t-1) \simeq \boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1) \tag{17}$$

or

$$\boldsymbol{X}_r(t-1) \simeq \boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1)\boldsymbol{Q}_B^H(t-1). \tag{18}$$

In the Appendix, we explain the difference between the approximation (17) used here for Bi-LS and some similar approximations previously mentioned for Bi-SVD.

Substituting (17) into (14) yields the following update:

$$\begin{aligned}\begin{bmatrix}\boldsymbol{A}(t)\\ \boldsymbol{h}_L^H(t)\end{bmatrix} &\simeq \begin{bmatrix}\beta^{1/2}\boldsymbol{h}^H(t)\\ \alpha^{1/2}\boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1)\end{bmatrix}\\ &= \begin{bmatrix}\left(\frac{\beta}{\alpha}\right)^{1/2}\boldsymbol{h}^H(t)\boldsymbol{R}_A^{-1}(t-1)\boldsymbol{Q}_A^H(t-1)\\ \boldsymbol{I}_L\end{bmatrix}\\ &\quad\times \alpha^{1/2}\boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1).\end{aligned} \tag{19}$$

Thus, dropping the last row of both sides of (19) yields

$$\begin{aligned}\boldsymbol{A}(t) &\simeq \begin{bmatrix}\left(\frac{\beta}{\alpha}\right)^{1/2}\boldsymbol{h}^H(t)\boldsymbol{R}_A^{-1}(t-1)\boldsymbol{Q}_A^H(t-1) & \\ \boldsymbol{I}_{L-1} & \boldsymbol{0}\end{bmatrix}\\ &\quad\times \alpha^{1/2}\boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1)\\ &= \begin{bmatrix}\beta^{1/2}\boldsymbol{h}^H(t) - \alpha^{1/2}\boldsymbol{q}_{A_L}^H(t-1)\boldsymbol{R}_A(t-1)\\ \boldsymbol{O}\end{bmatrix}\\ &\quad+\begin{bmatrix}\boldsymbol{0}^T & 1\\ \boldsymbol{I}_{L-1} & \boldsymbol{0}\end{bmatrix}\alpha^{1/2}\boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1)\end{aligned} \tag{20}$$

where $\boldsymbol{q}_{A_L}(t-1) \triangleq \boldsymbol{Q}_A^H(t-1)[0\cdots 0,1]^T$. Define the matrix

$$\tilde{\boldsymbol{Q}}_A(t-1) = \begin{bmatrix}\boldsymbol{0}^T & 1\\ \boldsymbol{I}_{L-1} & \boldsymbol{0}\end{bmatrix}\boldsymbol{Q}_A(t-1) \tag{21}$$

and define the L-dimensional vector $\boldsymbol{z}_L \triangleq [1,0\cdots 0]^T$. Then, (20) can be expressed compactly as

$$\boldsymbol{A}(t) \simeq \alpha^{1/2}\tilde{\boldsymbol{Q}}_A(t-1)\boldsymbol{R}_A(t-1) + \beta^{1/2}\boldsymbol{z}_L\tilde{\boldsymbol{h}}^H(t) \tag{22}$$

where $\tilde{\boldsymbol{h}}(t) \triangleq \boldsymbol{h}(t) - ((\alpha/\beta))^{1/2}\boldsymbol{R}_A^H(t-1)\boldsymbol{q}_{A_L}(t-1)$.

Notice the alternative expression $\boldsymbol{q}_{A_L}(t-1) = \tilde{\boldsymbol{Q}}_A^H(t-1)\boldsymbol{z}_L$. This enables us to decompose the vector $\boldsymbol{z}_L$ into two components:

$$\boldsymbol{z}_L = \tilde{\boldsymbol{Q}}_A(t-1)\boldsymbol{q}_{A_L}(t-1) + \boldsymbol{z}_\perp(t) \tag{23}$$

where $\boldsymbol{z}_\perp(t)$ is orthogonal to the span of $\tilde{\boldsymbol{Q}}_A(t-1)$. Substituting (23) into (22) yields

$$\begin{aligned}\boldsymbol{A}(t) &\simeq \begin{bmatrix}\tilde{\boldsymbol{Q}}_A(t-1) & \dfrac{\boldsymbol{z}_\perp(t)}{\|\boldsymbol{z}_\perp(t)\|}\end{bmatrix}\\ &\quad\times\begin{bmatrix}\alpha^{1/2}\boldsymbol{R}_A(t-1) + \beta^{1/2}\boldsymbol{q}_{A_L}(t-1)\tilde{\boldsymbol{h}}^H(t)\\ \beta^{1/2}\|\boldsymbol{z}_\perp(t)\|\tilde{\boldsymbol{h}}^H(t)\end{bmatrix}\\ &= \begin{bmatrix}\tilde{\boldsymbol{Q}}_A(t-1) & \dfrac{\boldsymbol{z}_\perp(t)}{\|\boldsymbol{z}_\perp(t)\|}\end{bmatrix}\boldsymbol{G}_A^H(t)\begin{bmatrix}\boldsymbol{R}_A(t)\\ 0\cdots 0\end{bmatrix}\end{aligned} \tag{24}$$

where $\|\boldsymbol{z}_\perp(t)\|$ denotes the norm of the vector $\boldsymbol{z}_\perp(t)$, $\boldsymbol{G}_A(t)$ is a sequence of orthonormal Givens rotations, and $\boldsymbol{R}_A(t)$ is an $r\times r$ upper-triangular matrix that satisfies

$$\begin{aligned}&\begin{bmatrix}\boldsymbol{R}_A(t)\\ 0\cdots 0\end{bmatrix}\\ &= \boldsymbol{G}_A(t)\begin{bmatrix}\alpha^{1/2}\boldsymbol{R}_A(t-1) + \beta^{1/2}\boldsymbol{q}_{A_L}(t-1)\tilde{\boldsymbol{h}}^H(t)\\ \beta^{1/2}\|\boldsymbol{z}_\perp(t)\|\tilde{\boldsymbol{h}}^H(t)\end{bmatrix}.\end{aligned} \tag{25}$$

Clearly, (24) suggests a QR decomposition of $\boldsymbol{A}(t)$. The desired orthonormal matrix $\boldsymbol{Q}_A(t)$ can be obtained from the following recursion:

$$[\boldsymbol{Q}_A(t) \quad \star] = \begin{bmatrix}\tilde{\boldsymbol{Q}}_A(t-1) & \dfrac{\boldsymbol{z}_\perp(t)}{\|\boldsymbol{z}_\perp(t)\|}\end{bmatrix}\boldsymbol{G}_A^H(t) \tag{26}$$

where the symbol $\star$ denotes a column vector of no interest.

Note that (25) reveals a special updating problem of the form "upper triangular plus rank one". This special updating problem can be solved simply by $2r$ Givens plane rotations. Alternatively, (25) may be rewritten as

$$\begin{aligned}\begin{bmatrix}\boldsymbol{R}_A(t)\\ 0\cdots 0\end{bmatrix} = \boldsymbol{G}_A(t)\Bigg\{&\begin{bmatrix}\alpha^{1/2}\boldsymbol{R}_A(t-1)\\ 0\cdots 0\end{bmatrix}\\ &+\beta^{1/2}\begin{bmatrix}\boldsymbol{q}_{A_L}(t-1)\\ \|\boldsymbol{z}_\perp(t)\|\end{bmatrix}\tilde{\boldsymbol{h}}^H(t)\Bigg\}.\end{aligned} \tag{27}$$

A two-step strategy [13] can be employed to triangularize the form of "upper triangular plus rank one" on the right side of (27). Let us set

$$\boldsymbol{G}_A(t) \triangleq \boldsymbol{G}_A^{II}(t)\boldsymbol{G}_A^I(t);\quad \boldsymbol{R}_A \triangleq \begin{bmatrix}\alpha^{1/2}\boldsymbol{R}_A(t-1)\\ 0\cdots 0\end{bmatrix}$$

$$\boldsymbol{q}_A \triangleq \beta^{1/2}\begin{bmatrix}\boldsymbol{q}_{A_L}(t-1)\\ \|\boldsymbol{z}_\perp(t)\|\end{bmatrix}.$$

The first step constructs $\boldsymbol{G}_A^I(t)$ such that

$$\boldsymbol{G}_A^I(t)\boldsymbol{q}_A = \pm\|\boldsymbol{q}_A\|\boldsymbol{z}_{r+1}.$$

Hence, $\boldsymbol{G}_A^I(t)\boldsymbol{R}_A$ is upper Hessenberg. For example, if $r=4$, we have

$$\boldsymbol{G}_A^I(t)\boldsymbol{q}_A = \begin{bmatrix}\times\\ 0\\ 0\\ 0\\ 0\end{bmatrix};\quad \boldsymbol{G}_A^I(t)\boldsymbol{R}_A = \begin{bmatrix}\times & \times & \times & \times\\ \times & \times & \times & \times\\ 0 & \times & \times & \times\\ 0 & 0 & \times & \times\\ 0 & 0 & 0 & \times\end{bmatrix}.$$

This step involves r Givens rotations, as embodied in $\boldsymbol{G}_A^I(t)$.

Since $\boldsymbol{R}_A^I \triangleq \boldsymbol{G}_A^I(t)\boldsymbol{R}_A + \boldsymbol{G}_A^I(t)\boldsymbol{q}_A\tilde{\boldsymbol{h}}^H(t)$ is now upper Hessenberg, the second step is to find r rotations in $\boldsymbol{G}_A^{II}(t)$ such that

$$\begin{aligned}\boldsymbol{G}_A^{II}(t)\boldsymbol{R}_A^I &= \boldsymbol{G}_A^{II}(t)\begin{bmatrix}\times & \times & \times & \times\\ \times & \times & \times & \times\\ 0 & \times & \times & \times\\ 0 & 0 & \times & \times\\ 0 & 0 & 0 & \times\end{bmatrix}\\ &= \begin{bmatrix}\times & \times & \times & \times\\ 0 & \times & \times & \times\\ 0 & 0 & \times & \times\\ 0 & 0 & 0 & \times\\ 0 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}\boldsymbol{R}_A(t)\\ 0\cdots 0\end{bmatrix}.\end{aligned}$$

Therefore, $\boldsymbol{Q}_A(t)$ in (26) can be computed recursively and requires only $8Lr$ flops.

2) Updating $\boldsymbol{Q}_B(t)$: We now consider the following identity:

$$[\boldsymbol{X}^H(t) \quad \alpha^{L/2}\beta^{1/2}\boldsymbol{x}(t-L)]\begin{bmatrix}\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t)\\ 0\cdots 0\end{bmatrix} = [\beta^{1/2}\boldsymbol{x}(t) \quad \alpha^{1/2}\boldsymbol{X}^H(t-1)]\begin{bmatrix}\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t)\\ 0\cdots 0\end{bmatrix}. \quad (28)$$

According to the second step in Table IV, we get

$$\boldsymbol{B}(t) = [\beta^{1/2}\boldsymbol{x}(t) \quad \alpha^{1/2}\boldsymbol{X}^H(t-1)]\begin{bmatrix}\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t)\\ 0\cdots 0\end{bmatrix}. \quad (29)$$

Premultiplying both sides of (29) by $\boldsymbol{Q}_B^H(t-1)$ and employing (14) yields

$$\begin{aligned}\boldsymbol{Q}_B^H(t-1)\boldsymbol{B}(t) &= \left[\beta^{1/2}\boldsymbol{h}(t) \quad \alpha^{1/2}\boldsymbol{Q}_B^H(t-1)\boldsymbol{X}^H(t-1)\right]\\ &\quad\times\begin{bmatrix}\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t)\\ 0\cdots 0\end{bmatrix}\\ &= \begin{bmatrix}\boldsymbol{A}(t)\\ \boldsymbol{h}_L^H(t)\end{bmatrix}^H\begin{bmatrix}\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t)\\ 0\cdots 0\end{bmatrix}\\ &= \boldsymbol{A}^H(t)\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t)\\ &= \boldsymbol{I}_r \end{aligned} \quad (30)$$

which confirms (11). However, the inner structure shown in (30) will be useful next.

We decompose the vector $\boldsymbol{x}(t)$ into two components:

$$\boldsymbol{x}(t) = \boldsymbol{Q}_B(t-1)\boldsymbol{h}(t) + \boldsymbol{x}_\perp(t) \quad (31)$$

where $\boldsymbol{x}_\perp(t)$ is orthogonal to the span of $\boldsymbol{Q}_B(t-1)$. Substituting the representation $\boldsymbol{x}(t)$ and the low-rank approximation (18) into (29) yields

$$\begin{aligned}\boldsymbol{B}(t) &\simeq \left[\boldsymbol{Q}_B(t-1) \quad \frac{\boldsymbol{x}_\perp(t)}{\|\boldsymbol{x}_\perp(t)\|}\right]\\ &\quad\times\begin{bmatrix}\beta^{1/2}\boldsymbol{h}(t) & \alpha^{1/2}\boldsymbol{R}_A^H(t-1)\boldsymbol{Q}_A^H(t-1)\\ \beta^{1/2}\|\boldsymbol{x}_\perp(t)\| & 0\cdots 0\end{bmatrix}\\ &\quad\times\begin{bmatrix}\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t)\\ 0\cdots 0\end{bmatrix}. \end{aligned} \quad (32)$$

Let $\boldsymbol{q}_{A_1}(t) \triangleq \boldsymbol{Q}_A^H(t)\boldsymbol{z}_L$. By (30), we can simplify (32) to yield

$$\begin{aligned}\boldsymbol{B}(t) &\simeq \left[\boldsymbol{Q}_B(t-1) \quad \frac{\boldsymbol{x}_\perp(t)}{\|\boldsymbol{x}_\perp(t)\|}\right]\\ &\quad\times\begin{bmatrix}\boldsymbol{I}_r\\ \beta^{1/2}\|\boldsymbol{x}_\perp(t)\|\boldsymbol{q}_{A_1}^H(t)\boldsymbol{R}_A^{-H}(t)\end{bmatrix}\\ &= \left[\boldsymbol{Q}_B(t-1) \quad \frac{\boldsymbol{x}_\perp(t)}{\|\boldsymbol{x}_\perp(t)\|}\right]\boldsymbol{G}_B^H(t)\begin{bmatrix}\boldsymbol{R}_B(t)\\ 0\cdots 0\end{bmatrix} \end{aligned} \quad (33)$$

where $\boldsymbol{G}_B(t)$ consists of r Givens rotations such that

$$\begin{bmatrix}\boldsymbol{R}_B(t)\\ 0\cdots 0\end{bmatrix} = \boldsymbol{G}_B(t)\begin{bmatrix}\boldsymbol{I}_r\\ \beta^{1/2}\|\boldsymbol{x}_\perp(t)\|\boldsymbol{q}_{A_1}^H(t)\boldsymbol{R}_A^{-H}(t)\end{bmatrix}. \quad (34)$$

The desired orthonormal matrix $\boldsymbol{Q}_B(t)$ can be updated recursively from the following:

$$[\boldsymbol{Q}_B(t) \quad \star] = \left[\boldsymbol{Q}_B(t-1) \quad \frac{\boldsymbol{x}_\perp(t)}{\|\boldsymbol{x}_\perp(t)\|}\right]\boldsymbol{G}_B^H(t). \quad (35)$$

This updating operation requires $4Nr$ flops due to r rotations. In addition, let

$$\boldsymbol{q}_{A_1}(t) = \boldsymbol{R}_A(t)\tilde{\boldsymbol{q}}_{A_1}(t). \quad (36)$$

Since $\boldsymbol{R}_A(t)$ is the upper triangular matrix, the vector $\tilde{\boldsymbol{q}}_{A_1}(t)$ is easy to be solved by only $0.5(r^2+r)$ back substitution operations.

3) Summary of the Bi-LS-1 Algorithm: Finally, a complete quasicode of the Bi-LS-1 algorithm is summarized in Table V. Note that a minor computational cost of α and β is not counted in the Table. This algorithm has a principal computational complexity of $6Nr+9Lr+6r^2+O(r)$ iterations. Basically, updating the left subspace requires $9Lr$ flops, and updating the right subspace requires $6Nr$ vectors.

B. Bi-LS-2 Algorithm

This algorithm is a slight variation of the Bi-LS-1 algorithm. We observe that (33) can be rewritten as

$$\boldsymbol{B}(t) \simeq \boldsymbol{Q}_B(t-1) + \beta^{1/2}\boldsymbol{x}_\perp(t)\tilde{\boldsymbol{q}}_{A_1}^H(t) \quad (37)$$

where the two matrices on the right are orthogonal to each other. In other words, the column span of $\boldsymbol{B}(t)$ is decomposed into the old subspace span $(\boldsymbol{Q}_B(t-1))$ and the rank-one innovation span $(\boldsymbol{x}_\perp(t)\tilde{\boldsymbol{q}}_{A_1}^H(t))$. This suggests the following short-cut:

$$\boldsymbol{Q}_B(t) \simeq \boldsymbol{Q}_B(t-1) + \beta^{1/2}\boldsymbol{x}_\perp(t)\tilde{\boldsymbol{q}}_{A_1}^H(t) \quad (38)$$

where, however, the columns of $\boldsymbol{Q}_B(t)$ are no longer orthonormal. Since the above shortcut yields a major reduction of computations (i.e., a reduction of $3Nr$ flops), we list the resulting algorithm as the Bi-LS-2 algorithm in Table VI. The Bi-LS-2 algorithm requires a principal computational complexity of $3Nr+9Lr+5r^2+O(r)$ flops.

C. Bi-LS-3 Algorithm

We first observe that a major complexity component of the previous Bi-LS algorithms comes from updating the left subspace matrix $\boldsymbol{Q}_A(t)$ through (25) and (26). For applications where one is only interested in the right subspace (or the row span) of $\boldsymbol{X}(t)$, updating the left subspace is simply an extra burden. We show next that this extra burden can be removed if an exponential window is used.

With an exponential window, we now have the following update scheme of the data matrix $\boldsymbol{X}(t)$:

$$\boldsymbol{X}(t) = \begin{bmatrix}(1-\alpha)^{1/2}\boldsymbol{x}^H(t)\\ \alpha^{1/2}\boldsymbol{X}(t-1)\end{bmatrix}. \quad (39)$$

Similar to (14), we now have

$$\boldsymbol{A}(t) = \begin{bmatrix}(1-\alpha)^{1/2}\boldsymbol{h}^H(t)\\ \alpha^{1/2}\boldsymbol{X}(t-1)\boldsymbol{Q}_B(t-1)\end{bmatrix}. \quad (40)$$

TABLE V
BI-LS-1 ALGORITHM: USING HYBRID WINDOW. THE COMPLEXITY IS $6Nr + 9Lr + 6r^2 + O(r)$. $\boldsymbol{G}_A(t)$ IS A SEQUENCE OF $2r$ GIVENS ROTATIONS, AND $\boldsymbol{G}_B(t)$ IS A SEQUENCE OF r GIVENS ROTATIONS

Initialization: $\boldsymbol{z}_L = [1, 0 \dots 0]^T$; $\boldsymbol{Q}_B(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}_{N\times r}$; $\boldsymbol{Q}_A(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}_{L\times r}$; $\boldsymbol{R}_A(0) = \boldsymbol{I}_r$	
For each t, Do:	
Input: $\boldsymbol{x}(t)$	
First Step:	Complexity:
$\widetilde{\boldsymbol{Q}}_A(t-1) = \begin{bmatrix} \boldsymbol{0}^T & 1 \\ \boldsymbol{I}_{L-1} & \boldsymbol{0} \end{bmatrix} \boldsymbol{Q}_A(t-1)$	
$\boldsymbol{q}_{A_L}(t-1) = \widetilde{\boldsymbol{Q}}_A^H(t-1)\boldsymbol{z}_L$	
$\boldsymbol{h}(t) = \boldsymbol{Q}_B^H(t-1)\boldsymbol{x}(t)$	Nr
$\widetilde{\boldsymbol{h}}(t) = \boldsymbol{h}(t) - (\frac{\alpha}{\beta})^{1/2}\boldsymbol{R}_A^H(t-1)\boldsymbol{q}_{A_L}(t-1)$	$0.5r^2 + O(r)$
$\boldsymbol{z}_\perp(t) = \boldsymbol{z}_L - \widetilde{\boldsymbol{Q}}_A(t-1)\boldsymbol{q}_{A_L}(t-1)$	Lr
$\begin{bmatrix} \boldsymbol{R}_A(t) \\ 0 \dots 0 \end{bmatrix} = \boldsymbol{G}_A(t) \begin{bmatrix} \alpha^{1/2}\boldsymbol{R}_A(t-1) + \beta^{1/2}\boldsymbol{q}_{A_L}(t-1)\widetilde{\boldsymbol{h}}^H(t) \\ \beta^{1/2}\|\boldsymbol{z}_\perp(t)\|\widetilde{\boldsymbol{h}}^H(t) \end{bmatrix}$	$4r^2 + O(r)$
$[\boldsymbol{Q}_A(t) \quad \star] = \left[\widetilde{\boldsymbol{Q}}_A(t-1) \quad \frac{\boldsymbol{z}_\perp(t)}{\|\boldsymbol{z}_\perp(t)\|}\right] \boldsymbol{G}_A^H(t)$	$8Lr$
Second Step:	
$\boldsymbol{q}_{A_1}(t) = \boldsymbol{Q}_A^H(t)\boldsymbol{z}_L$	
$\boldsymbol{x}_\perp(t) = \boldsymbol{x}(t) - \boldsymbol{Q}_B(t-1)\boldsymbol{h}(t)$	Nr
$\boldsymbol{R}_A(t)\widetilde{\boldsymbol{q}}_{A_1}(t) = \boldsymbol{q}_{A_1}(t) \xrightarrow{\text{back substitution}} \widetilde{\boldsymbol{q}}_{A_1}(t)$	$0.5r^2 + O(r)$
$\begin{bmatrix} \boldsymbol{R}_B(t) \\ 0 \cdots 0 \end{bmatrix} = \boldsymbol{G}_B(t) \begin{bmatrix} \boldsymbol{I}_r \\ \beta^{1/2}\|\boldsymbol{x}_\perp(t)\|\widetilde{\boldsymbol{q}}_{A_1}^H(t) \end{bmatrix}$	$r^2 + O(r)$
$[\boldsymbol{Q}_B(t) \quad \star] = \left[\boldsymbol{Q}_B(t-1) \quad \frac{\boldsymbol{x}_\perp(t)}{\|\boldsymbol{x}_\perp(t)\|}\right] \boldsymbol{G}_B^H(t)$	$4Nr$

TABLE VI
BI-LS-2 ALGORITHM: USING HYBRID WINDOW. THE RIGHT SUBSPACE BASIS VECTORS ARE NOT ORTHONORMAL. THE COMPLEXITY IS $3Nr + 9Lr + 5r^2 + O(r)$. $\boldsymbol{G}_A(t)$ IS A SEQUENCE OF $2r$ GIVENS ROTATIONS

Initialization: $\boldsymbol{z}_L = [1, 0 \dots 0]^T$; $\boldsymbol{Q}_B(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}_{N\times r}$; $\boldsymbol{Q}_A(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}_{L\times r}$; $\boldsymbol{R}_A(0) = \boldsymbol{I}_r$	
For each t, Do:	
Input: $\boldsymbol{x}(t)$	
First Step:	Complexity:
$\widetilde{\boldsymbol{Q}}_A(t-1) = \begin{bmatrix} \boldsymbol{0}^T & 1 \\ \boldsymbol{I}_{L-1} & \boldsymbol{0} \end{bmatrix} \boldsymbol{Q}_A(t-1)$	
$\boldsymbol{q}_{A_L}(t-1) = \widetilde{\boldsymbol{Q}}_A^H(t-1)\boldsymbol{z}_L$	
$\boldsymbol{h}(t) = \boldsymbol{Q}_B^H(t-1)\boldsymbol{x}(t)$	Nr
$\widetilde{\boldsymbol{h}}(t) = \boldsymbol{h}(t) - (\frac{\alpha}{\beta})^{1/2}\boldsymbol{R}_A^H(t-1)\boldsymbol{q}_{A_L}(t-1)$	$0.5r^2 + O(r)$
$\boldsymbol{z}_\perp(t) = \boldsymbol{z}_L - \widetilde{\boldsymbol{Q}}_A(t-1)\boldsymbol{q}_{A_L}(t-1)$	Lr
$\begin{bmatrix} \boldsymbol{R}_A(t) \\ 0 \dots 0 \end{bmatrix} = \boldsymbol{G}_A(t) \begin{bmatrix} \alpha^{1/2}\boldsymbol{R}_A(t-1) + \beta^{1/2}\boldsymbol{q}_{A_L}(t-1)\widetilde{\boldsymbol{h}}^H(t) \\ \beta^{1/2}\|\boldsymbol{z}_\perp(t)\|\widetilde{\boldsymbol{h}}^H(t) \end{bmatrix}$	$4r^2 + O(r)$
$[\boldsymbol{Q}_A(t) \quad \star] = \left[\widetilde{\boldsymbol{Q}}_A(t-1) \quad \frac{\boldsymbol{z}_\perp(t)}{\|\boldsymbol{z}_\perp(t)\|}\right] \boldsymbol{G}_A^H(t)$	$8Lr$
Second Step:	
$\boldsymbol{q}_{A_1}(t) = \boldsymbol{Q}_A^H(t)\boldsymbol{z}_L$	
$\boldsymbol{x}_\perp(t) = \boldsymbol{x}(t) - \boldsymbol{Q}_B(t-1)\boldsymbol{h}(t)$	Nr
$\boldsymbol{R}_A(t)\widetilde{\boldsymbol{q}}_{A_1}(t) = \boldsymbol{q}_{A_1}(t) \xrightarrow{\text{back substitution}} \widetilde{\boldsymbol{q}}_{A_1}(t)$	$0.5r^2 + O(r)$
$\boldsymbol{Q}_B(t) = \boldsymbol{Q}_B(t-1) + \beta^{1/2}\boldsymbol{x}_\perp(t)\widetilde{\boldsymbol{q}}_{A_1}^H(t)$	Nr

TABLE VII
BI-LS-3 ALGORITHM: USING EXPONENTIAL WINDOW. TRACKING THE RIGHT SUBSPACE. THE COMPLEXITY IS $6Nr + 3.5r^2 + O(r)$. $\boldsymbol{G}_A(t)$ IS A SEQUENCE OF r GIVENS ROTATIONS

Initialization: $\boldsymbol{Q}_B(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}_{N\times r}$; $\boldsymbol{R}_A(0) = \boldsymbol{I}_r$	
For each t,Do:	
Input: $\boldsymbol{x}(t)$	
First Step:	Complexity:
$\boldsymbol{h}(t) = \boldsymbol{Q}_B^H(t-1)\boldsymbol{x}(t)$	Nr
$\begin{bmatrix} \boldsymbol{R}_A(t) \\ 0\ldots 0 \end{bmatrix} = \boldsymbol{G}_A(t) \begin{bmatrix} \alpha^{1/2}\boldsymbol{R}_A(t-1) \\ (1-\alpha)^{1/2}\boldsymbol{h}^H(t) \end{bmatrix}$	$2r^2 + O(r)$
$[\boldsymbol{q}_{A_1}^H(t) \quad \star] = [0\cdots 0 \quad 1]\,\boldsymbol{G}_A^H(t)$	
Second Step:	
$\boldsymbol{x}_\perp(t) = \boldsymbol{x}(t) - \boldsymbol{Q}_B(t-1)\boldsymbol{h}(t)$	Nr
$\boldsymbol{R}_A(t)\widetilde{\boldsymbol{q}}_{A_1}(t) = \boldsymbol{q}_{A_1}(t)$ $\xrightarrow{\text{back substitution}}$ $\widetilde{\boldsymbol{q}}_{A_1}(t)$	$0.5r^2 + O(r)$
$\begin{bmatrix} \boldsymbol{R}_B(t) \\ 0\cdots 0 \end{bmatrix} = \boldsymbol{G}_B(t) \begin{bmatrix} \boldsymbol{I}_r \\ (1-\alpha)^{1/2}\|\boldsymbol{x}_\perp(t)\|\widetilde{\boldsymbol{q}}_{A_1}^H(t) \end{bmatrix}$	$r^2 + O(r)$
$[\boldsymbol{Q}_B(t) \quad \star] = \left[\boldsymbol{Q}_B(t-1) \quad \frac{\boldsymbol{x}_\perp(t)}{\|\boldsymbol{x}_\perp(t)\|}\right] \boldsymbol{G}_B^H(t)$	$4Nr$

Substituting (17) into (40) yields

$$\boldsymbol{A}(t) \simeq \begin{bmatrix} \boldsymbol{0}^T & 1 \\ \boldsymbol{Q}_A(t-1) & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \alpha^{1/2}\boldsymbol{R}_A(t-1) \\ (1-\alpha)^{1/2}\boldsymbol{h}^H(t) \end{bmatrix}. \tag{41}$$

Thus, we have

$$\boldsymbol{Q}_A(t) = \begin{bmatrix} \boldsymbol{0}^T & 1 \\ \boldsymbol{Q}_A(t-1) & \boldsymbol{0} \end{bmatrix} \boldsymbol{G}_A^H(t) \tag{42}$$

where $\boldsymbol{G}_A(t)$ is a sequence of r orthonormal Givens rotations such that

$$\begin{bmatrix} \boldsymbol{R}_A(t) \\ 0\cdots 0 \end{bmatrix} = \boldsymbol{G}_A(t) \begin{bmatrix} \alpha^{1/2}\boldsymbol{R}_A(t-1) \\ (1-\alpha)^{1/2}\boldsymbol{h}^H(t) \end{bmatrix}. \tag{43}$$

According to the second step in Table IV and similar to the derivation of $\boldsymbol{Q}_B(t)$ in Section IV-A, we have

$$\begin{aligned} \boldsymbol{B}(t) &= \boldsymbol{X}^H(t)\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t) \\ &= [(1-\alpha)^{1/2}\boldsymbol{x}(t) \quad \alpha^{1/2}\boldsymbol{X}^H(t-1)]\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t) \\ &\simeq \begin{bmatrix} \boldsymbol{Q}_B(t-1) & \dfrac{\boldsymbol{x}_\perp(t)}{\|\boldsymbol{x}_\perp(t)\|} \end{bmatrix} \\ &\quad \times \begin{bmatrix} \boldsymbol{I}_r \\ (1-\alpha)^{1/2}\|\boldsymbol{x}_\perp(t)\|\boldsymbol{q}_{A_1}^H(t)\boldsymbol{R}_A^{-H}(t) \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{Q}_B(t-1) & \dfrac{\boldsymbol{x}_\perp(t)}{\|\boldsymbol{x}_\perp(t)\|} \end{bmatrix} \boldsymbol{G}_B^H(t) \begin{bmatrix} \boldsymbol{R}_B(t) \\ 0\cdots 0 \end{bmatrix} \end{aligned} \tag{44}$$

where

$$\begin{aligned} \boldsymbol{q}_{A_1}^H(t) &= [1 \quad 0\cdots 0] \begin{bmatrix} \boldsymbol{0}^T & 1 \\ \boldsymbol{Q}_A(t-1) & \boldsymbol{0} \end{bmatrix} \boldsymbol{G}_A^H(t) \\ &= [0\cdots 0 \quad 1]\boldsymbol{G}_A^H(t) \end{aligned} \tag{45}$$

which replaces the original form $\boldsymbol{q}_{A_1}(t) = \boldsymbol{Q}_A^H(t)z_L$. We have now completely avoided the computation of the left subspace matrix $\boldsymbol{Q}_A(t)$.

To ensure the orthonormalization of $\boldsymbol{Q}_B(t)$, we should use (35) instead of (38) to update $\boldsymbol{Q}_B(t)$.

Table VII summarizes the above algorithm as the Bi-LS-3 algorithm, which tracks the right subspace using the exponential window. The principal complexity of this algorithm is $6Nr + 3.5r^2 + O(r)$.

D. Bi-LS-4 Algorithm

This is a variation of the Bi-LS-3 algorithm. To further reduce the complexity, we can use (38) instead of (35) to update $\boldsymbol{Q}_B(t)$. Of course, the columns of $\boldsymbol{Q}_B(t)$ are no longer orthonormal. The principal complexity is now $3Nr + 2.5r^2 + O(r)$. Table VIII summarizes this algorithm as the Bi-LS-4 algorithm.

E. Comparison of the Bi-LS Algorithms and Others

Table IX compares the complexity of all the Bi-LS algorithms and other fast algorithms. We see that most of the other algorithms only track a single side (right) subspace due to the exponential window used. Our new algorithms Bi-LS-1 and Bi-LS-2 are capable of obtaining both subspaces at a relatively low cost. The Bi-LS-3 and Bi-LS-4 algorithms can be employed to extract a single side subspace at an even lower cost.

The complexity of the Bi-LS-4 algorithm is comparable with that of the PAST algorithm [33] and the NIC algorithm [22]. The PAST algorithm appears to be the fastest algorithm with a guaranteed stability. With a singular value decomposition of the upper triangular matrix $\boldsymbol{R}_A(t)$, the Bi-LS-4 algorithm allows us to track (estimate) the principal right singular vectors and the corresponding singular values of $\boldsymbol{X}(t)$ at an additional cost $Nr^2 + O(r^3)$. For the PAST and NIC algorithms, the principal

TABLE VIII
BI-LS-4 ALGORITHM: USING EXPONENTIAL WINDOW. TRACKING THE RIGHT SUBSPACE WITHOUT ORTHONORMALIZATION. THE COMPLEXITY IS $3Nr + 2.5r^2 + O(r)$. $\boldsymbol{G}_A(t)$ IS A SEQUENCE OF r GIVENS ROTATIONS

Initialization: $\boldsymbol{Q}_B(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{O} \end{bmatrix}_{N\times r}$; $\boldsymbol{R}_A(0) = \boldsymbol{I}_r$

For each t, Do:

Input: $\boldsymbol{x}(t)$

First Step:	Complexity:
$\boldsymbol{h}(t) = \boldsymbol{Q}_B^H(t-1)\boldsymbol{x}(t)$	Nr
$\begin{bmatrix} \boldsymbol{R}_A(t) \\ 0\ldots 0 \end{bmatrix} = \boldsymbol{G}_A(t) \begin{bmatrix} \alpha^{1/2}\boldsymbol{R}_A(t-1) \\ (1-\alpha)^{1/2}\boldsymbol{h}^H(t) \end{bmatrix}$	$2r^2 + O(r)$
$[\boldsymbol{q}_{A_1}^H(t) \quad \star] = [0\cdots 0 \quad 1]\,\boldsymbol{G}_A^H(t)$	
Second Step:	
$\boldsymbol{x}_\perp(t) = \boldsymbol{x}(t) - \boldsymbol{Q}_B(t-1)\boldsymbol{h}(t)$	Nr
$\boldsymbol{R}_A(t)\widetilde{\boldsymbol{q}}_{A_1}(t) = \boldsymbol{q}_{A_1}(t)$ $\xrightarrow{\text{back substitution}}$ $\widetilde{\boldsymbol{q}}_{A_1}(t)$	$0.5r^2 + O(r)$
$\boldsymbol{Q}_B(t) = \boldsymbol{Q}_B(t-1) + (1-\alpha)^{1/2}\boldsymbol{x}_\perp(t)\widetilde{\boldsymbol{q}}_{A_1}^H(t)$	Nr

TABLE IX
COMPARISON OF LINEAR COMPLEXITY SUBSPACE TRACKING ALGORITHMS. NOTE THAT N IS THE DIMENSION OF THE RIGHT SUBSPACE, L IS THE DIMENSION OF THE LEFT SUBSPACE, AND r IS THE RANK OF THE DESIRED SUBSPACE

Algorithm	Principal complexity (flops)	window	subspace(s)	Orthon.[a]
Bi-SVD1 [30]	$Nr^2 + 3Nr + 3r^3 + O(r^2)$	exponential	right	Yes
SWASVD3[3]	$27Nr + 13Lr + O(r^2)$	sliding	left & right	Yes
FST [24]	$10Nr + O(r^2)$	exponential	right	Yes
PAST [33]	$3Nr + O(r^2)$	exponential	right	No
NIC [22]	$5Nr + O(r^2)$	exponential	right	No
OPAST[1]	$4Nr + O(r^2)$	exponential	right	Yes
SW-PAST [2]	$5Nr + O(r^2)$	sliding	right	No
SW-OPAST [b] [2]	$7Nr + 2prN + O(r^2)$	sliding	right	Yes
Bi-LS-1	$6Nr + 9Lr + O(r^2)$	hybrid	left & right	Yes
Bi-LS-2	$3Nr + 9Lr + O(r^2)$	hybrid	left & right	No
Bi-LS-3	$6Nr + O(r^2)$	exponential	right	Yes
Bi-LS-4	$3Nr + O(r^2)$	exponential	right	No

[a] Orthon. stands for orthonormality of subspace basis vectors after each iteration.
[b] $0 < p \leq 4$ denotes the rank of a constructed matrix.

right singular vectors and the corresponding singular values of $\boldsymbol{X}(t)$ can be done similarly.

V. PERFORMANCE ILLUSTRATION

A. Effect of Windows on Subspace Tracking

We measure the accuracy of an estimated subspace at time t by the maximum principal angle [13] between the estimated subspace and the exact subspace at the time t. The exact subspace is computed by performing a full SVD on $\boldsymbol{X}(t)$ at each t.

There are three windows considered in this paper: the hybrid window, the sliding window, and the exponential window. To show the window effects, we consider the estimated right subspaces by the Bi-LS-1 algorithm and the Bi-LS-3 algorithm. The Bi-LS-1 algorithm applies to both the hybrid window and the sliding window ($\alpha = \beta = 1$), and the Bi-LS-3 algorithm is meant for the exponential window.

We construct the input vector $\boldsymbol{x}(t) = [x(t), x(t+1), \ldots, x(t+N-1)]^T$ (with $N = 80$) from the time series $x(t) = e^{j\varphi t}$, where

$$\varphi = \begin{cases} 0, & t \leq 100 \\ \pi/2, & t > 100. \end{cases}$$

The estimated subspace at time t under consideration is given by the one-dimensional subspace ($r = 1$) spanned by $\boldsymbol{Q}_B(t)$.

Fig. 1 compares the transient patterns of the maximum principal angle caused by three different windows. For the sliding window, we have $L = 99$. For the hybrid window, we have $L = 99$ and $\alpha = 1 - \beta = 0.98$. For the exponential window,

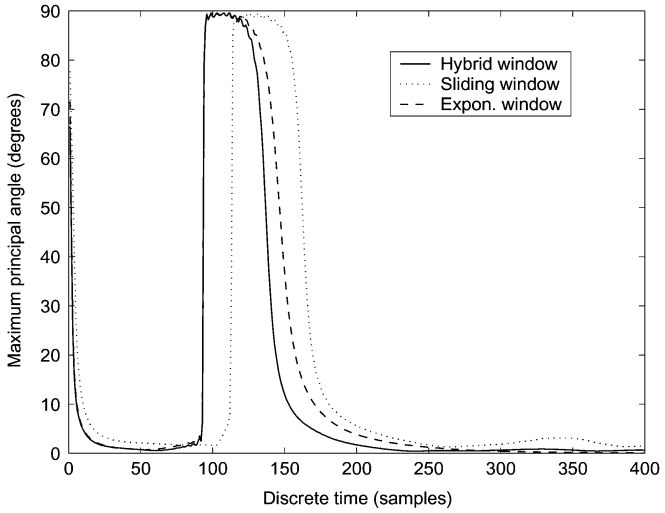

Fig. 1. Effect of windows on the subspace tracking performance.

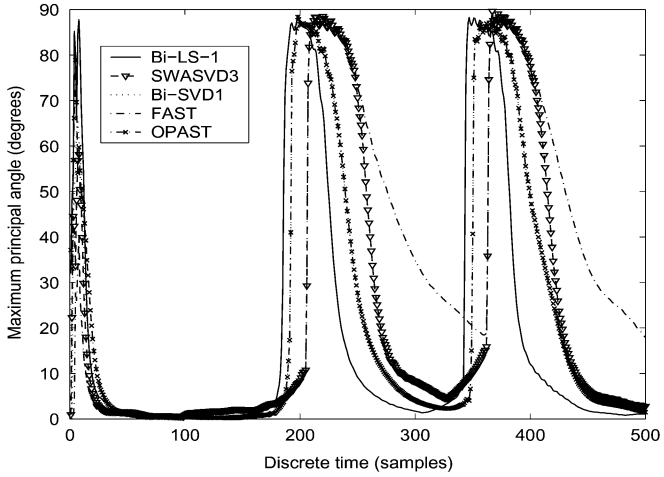

Fig. 2. Maximum principal angles of several orthonormal subspace algorithms. Note that the Bi-SVD1 and the OPAST are overlapping.

we also have $\alpha = 1 - \beta = 0.98$. With the same L, the hybrid window (with an exponentially decaying factor) always has an effective window length that is less than that of the sliding window. This is consistent with the transient widths as observed from the figure, i.e., the sliding window causes a longer transient width than the hybrid window (with the same L). Note that a transient width is the interval between a rising (diverging) edge and a corresponding falling (converging) edge in the figure. The chosen value of L is actually given by an approximate effective window length of exponentially decaying window. Therefore, as we can observe from the figure, the overall transient width caused by the exponential window is about the same as that by the sliding window, but the sliding window causes a sharper converging edge than the exponential window, which is also consistent with intuition. The converging edge caused by the hybrid window is actually (slightly) less sharp than that by the sliding window, which is, again, as expected. The diverging edge caused by the sliding window has a larger delay than the other two windows. The explanation is that the other two windows put more weights on the new sample vectors, and hence, their estimated subspace is more likely to be disturbed by the new samples that deviate significantly from the original subspace.

B. Performance Illustration of the Bi-LS Algorithms and Other Algorithms

The test data are now chosen to be the same as in [31]:

$$x(t) = \sum_{k=1}^{r} e^{j2\pi f_k t} + \omega(t)$$

which is a sum of complex exponentials plus white noise (with SNR $= 5.7$ dB). Here, we consider $r = 2$. The two frequencies change abruptly at two different time instants. The first frequency varies from $f = 0.0556$ Hz to $f = 0.2028$ Hz at $t = 200$ and the second from $f = 0.0278$ Hz to $f = 0.2194$ Hz at $t = 350$. As in the previous case, the maximum principal angle [13] is used to measure the performance of subspace tracking, but in addition, the two estimated frequencies are obtained from the estimated subspace at each value of t. The frequency estimation method is the ESPRIT/MatrixPencil method [17], [26].

We first consider five subspace tracking algorithms: the Bi-LS-1 algorithm, the SWASVD3 algorithm [3], the Bi-SVD-1 algorithm [30], the FAST algorithm [25], and the OPAST algorithm [1]. Among them, the SWASVD3 and the FAST are based on sliding rectangular window, whereas the Bi-SVD-1 and the OPAST are based on the exponential window. All these algorithms produce orthonormal subspace basis vectors. For all algorithms, we choose $N = 80, \alpha = 0.98$, and $L = 99$.

Fig. 2 shows the subspace tracking performance of the five algorithms. Bi-LS-1 has the shortest transient width largely because the sliding exponential window used has the shortest effective duration. The performances of Bi-SVD-1 and OPAST are almost identical, both of which use the same exponential window. Since the effective window length for Bi-SVD-1 and OPAST is longer than that for Bi-LS-1, the transient width of the former two is longer than that of the latter. SWASVD3 uses a sliding rectangular window of length equal to the effective window length of the exponential window used by Bi-SVD-1 and OPAST. The transient width of SWASVD3 is about the same as that of Bi-SVD-1 and OPAST, although SWASVD3 has a sharper converging edge. The FAST algorithm does not perform as well as the other four. Note that unlike the other four, the FAST algorithm does not belong to the power family.

Fig. 3 shows the frequency tracking performance of the five algorithms. What is interesting here is that the estimated frequencies by each algorithm all have a smooth transition between the old and the new. Otherwise, the patterns we see from Fig. 3 are basically consistent with the patterns we see from Fig. 2. We stress here that for each algorithm, the transient width can be varied by varying α and/or L.

We next consider Bi-LS-2, Bi-LS-4, PAST [33], and SW-PAST [2]. All of these four algorithms yield the nonorthonormal subspace matrix. Bi-LS-4 and PAST use exponential window, Bi-LS-2 uses sliding exponential window, and SW-PAST uses sliding window. Once again, we choose $N = 80, \alpha = 0.98$, and $L = 99$ for all algorithms.

Fig. 4 shows the subspace tracking performance of the above four algorithms, and Fig. 5 shows the frequency tracking

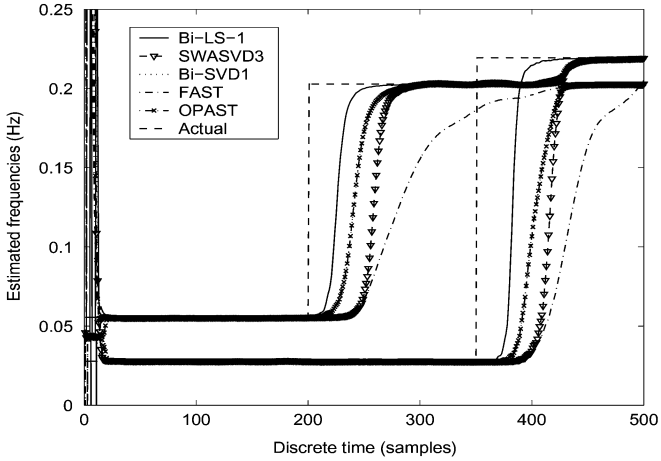

Fig. 3. Frequency tracking performance of several orthonormal subspace algorithms. Note that the Bi-SVD1 and the OPAST are overlapping.

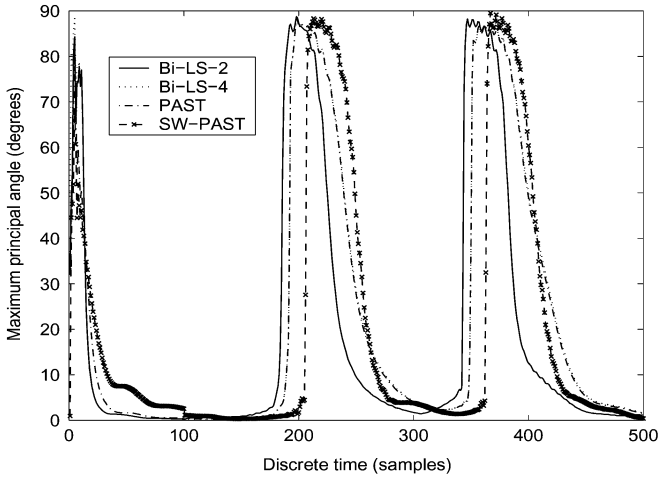

Fig. 4. Maximum principal angles of several nonorthonormal subspace algorithms. Note that the Bi-LS-4 and the PAST are overlapping.

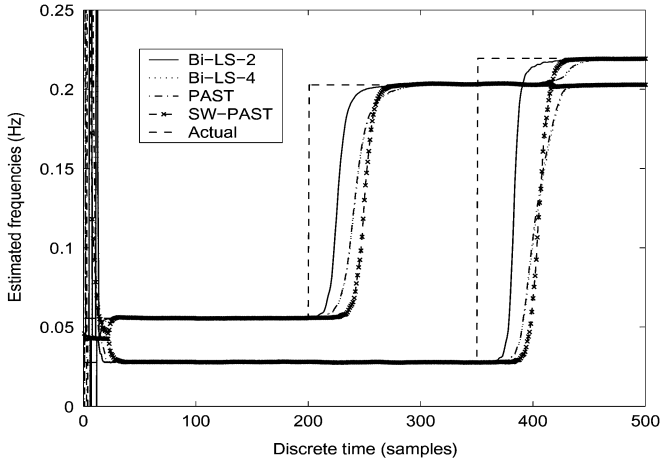

Fig. 5. Frequency tracking performance of several nonorthonormal subspace algorithms. Note that the Bi-LS-4 and the PAST are overlapping.

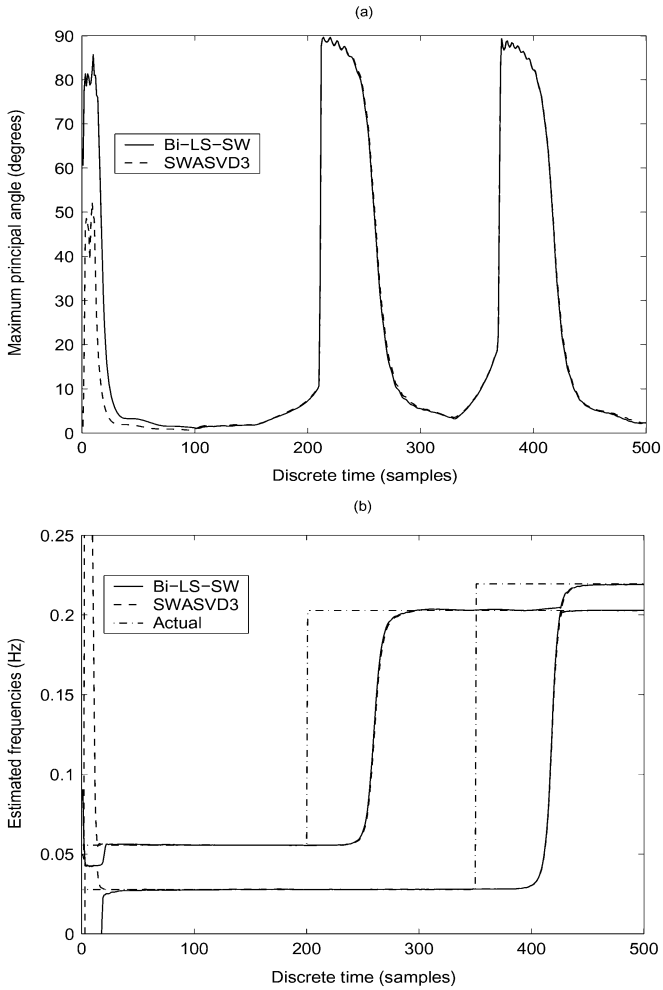

Fig. 6. Comparison of the tracking performances of the Bi-LS-1 algorithm with a sliding rectangular window (referred to as Bi-LS-SW) and the sliding rectangular window adaptive SVD (SWASVD3) algorithm shown in [3]. (a) Maximum principal angles. (b) Estimated frequencies. The two algorithms have identical performances, although the complexities are significantly different.

performance of the same algorithms. As expected, the transient widths shown in the figures are governed by the effective window length. We also see that Bi-LS-4 and PAST have almost identical performances. Furthermore, Bi-LS-2, as shown in Figs. 4 and 5, has the same performance as Bi-LS-1, as shown in Figs. 2 and 3 (in terms of the accuracy of both estimated subspace and estimated frequencies).

In Fig. 6, we compare the tracking performance of two bi-iterative sliding window subspace algorithms. One comes from the Bi-LS-1 algorithm by setting $\alpha = \beta = 1$ (referred to as Bi-LS-SW), and the other is the sliding window adaptive SVD algorithm (SWASVD3) [3]. As expected, the two sliding window subspace algorithms have the same tracking performances. However, our algorithm (Bi-LS-SW) is computationally more efficient than SWASVD3, i.e., Bi-LS-SW has the complexity of $6Nr + 9Lr + O(r^2)$, but SWASVD3 has the complexity of $27Nr + 13Lr + O(r^2)$.

VI. Conclusion

We have introduced the bi-iterative least-square (Bi-LS) method based on the QR decomposition and compared this method with the bi-iterative singular value decomposition (Bi-SVD) method. The Bi-LS method is designed to construct the optimal low-rank approximation of a matrix, but the Bi-SVD method is designed to compute more than that. Both methods can be adopted for subspace tracking. We have

shown that for subspace tracking, more efficient algorithms can be derived from the Bi-LS method than from the Bi-SVD method, although both methods have the same accuracy of subspace tracking. We have derived several Bi-LS subspace tracking algorithms of varied complexities. These algorithms are new additions to a family of power-based subspace tracking algorithms, many of which have excellent performances.

APPENDIX
ON THE VALIDITY OF (17)

We first explain that while (17) is a good approximation for Bi-LS, it is not for Bi-SVD. Note that the approximation in (17) is the same as the following so-called "projection approximation" by Yang:

$$\begin{aligned}\boldsymbol{X}(t-1)\boldsymbol{Q}_B(t-1) &\simeq \boldsymbol{X}(t-1)\boldsymbol{Q}_B(t-2)\\ &= \boldsymbol{A}(t-1) = \boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1).\end{aligned} \tag{46}$$

The corresponding low-rank approximation used by Strobach [30] is

$$\boldsymbol{X}(t-1)\boldsymbol{Q}_B(t-1) \simeq \boldsymbol{Q}_A(t-1)\boldsymbol{R}_A(t-1)\boldsymbol{\Theta}_B(t-1) \tag{47}$$

where $\boldsymbol{\Theta}_B(t-1) = \boldsymbol{Q}_B^H(t-2)\boldsymbol{Q}_B(t-1)$ describes a so-called "gap" between $\boldsymbol{Q}_B(t-2)$ and $\boldsymbol{Q}_B(t-1)$. With the approximation (47), a linear complexity Bi-SVD algorithm, of the complexity $O(Nr^2)$, is developed in [30]. By further assuming that $\boldsymbol{\Theta}_B(t-1) = \boldsymbol{I}_r$, Strobach [30] proposed a ultra fast Bi-SVD algorithm with the complexity $O(Nr)$. Researchers (e.g., [3]) have observed that this ultra fast Bi-SVD algorithm has a poor performance. In Section II, we have shown that $\boldsymbol{Q}_B(t) \neq \boldsymbol{Q}_B(t-1)$ or, equivalently, $\boldsymbol{\Theta}_B(t-1) \neq \boldsymbol{I}_r$ for the Bi-SVD method even at the convergence of subspace. On the other hand, (17) holds well for the Bi-LS method, i.e., $\boldsymbol{Q}_B(t) = \boldsymbol{Q}_B(t-1)$ holds for the Bi-LS method at the convergence of subspace. To explain this further, let us assume that $\boldsymbol{X}(t)$ has a constant row span and a constant column span. Then, we can write the SVD of $\boldsymbol{X}(t)$ as $\boldsymbol{X}(t) = \boldsymbol{U}\boldsymbol{S}(t)\boldsymbol{V}^H = \boldsymbol{U}_1\boldsymbol{S}_1(t)\boldsymbol{V}_1^H + \boldsymbol{U}_2\boldsymbol{S}_2(t)\boldsymbol{V}_2^H$, where the first term is principal, and the second term is minor. Now, let us assume a convergence of subspace, i.e., $\boldsymbol{Q}_B(t-1) = \boldsymbol{V}_1\boldsymbol{T}(t)$, where $\boldsymbol{T}(t)$ is an orthonormal matrix.

Following the basic Bi-LS method in Table IV, we have $\boldsymbol{A}(t) = \boldsymbol{X}(t)\boldsymbol{Q}_B(t-1) = \boldsymbol{U}_1\boldsymbol{S}_1(t)\boldsymbol{T}(t) = \boldsymbol{Q}_A(t)\boldsymbol{R}_A(t)$, where the last expression is the QR decomposition determined by $\boldsymbol{Q}_A(t) = \boldsymbol{U}_1\boldsymbol{T}'(t)$ and $\boldsymbol{R}_A(t) = \boldsymbol{T}'^H(t)\boldsymbol{S}_1(t)\boldsymbol{T}(t)$. Here, $\boldsymbol{T}'(t)$ is another orthonormal matrix. Then, we have $\boldsymbol{B}(t) = \boldsymbol{X}^H(t)\boldsymbol{Q}_A(t)\boldsymbol{R}_A^{-H}(t) = \boldsymbol{V}\boldsymbol{S}(t)\boldsymbol{U}^H\boldsymbol{U}_1\boldsymbol{T}'(t)\boldsymbol{T}'^H(t)\boldsymbol{S}_1^{-1}(t)\boldsymbol{T}(t) = \boldsymbol{V}_1\boldsymbol{T}(t) = \boldsymbol{Q}_B(t)\boldsymbol{R}_B(t)$, where $\boldsymbol{Q}_B(t) = \boldsymbol{V}_1\boldsymbol{T}(t)$, and $\boldsymbol{R}_B(t) = \boldsymbol{I}_r$. Therefore, $\boldsymbol{Q}_B(t) = \boldsymbol{Q}_B(t-1)$.

We now follow the basic Bi-SVD method in Table II. The first step leads to the same result as for the Bi-LS method, but at the second step, we have $\boldsymbol{B}(t) = \boldsymbol{X}^H(t)\boldsymbol{Q}_A(t) = \boldsymbol{V}\boldsymbol{S}(t)\boldsymbol{U}^H\boldsymbol{U}_1\boldsymbol{T}'(t) = \boldsymbol{V}_1\boldsymbol{S}_1(t)\boldsymbol{T}'(t) = \boldsymbol{Q}_B(t)\boldsymbol{R}_B(t)$, where the last term is the QR decomposition defined by $\boldsymbol{Q}_B(t) = \boldsymbol{V}_1\boldsymbol{T}''(t)$ and $\boldsymbol{R}_B(t) = \boldsymbol{T}''^H(t)\boldsymbol{S}_1(t)\boldsymbol{T}'(t)$. Here, $\boldsymbol{T}''(t)$ denotes another orthonormal matrix. It is easy to verify that $\boldsymbol{T}''^H(t)\boldsymbol{S}_1^2(t)\boldsymbol{T}(t) = \boldsymbol{R}_B(t)\boldsymbol{R}_A(t)$. This is an SVD of the asymmetric upper triangular matrix $\boldsymbol{R}_B(t)\boldsymbol{R}_A(t)$. Therefore, in general, $\boldsymbol{T}''(t) \neq \boldsymbol{T}(t)$, and hence, $\boldsymbol{Q}_B(t) \neq \boldsymbol{Q}_B(t-1)$.

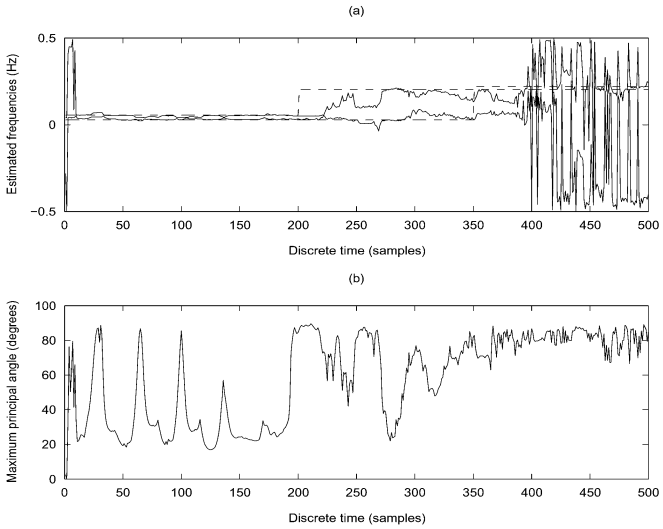

Fig. 7. Tracking performance of the Bi-SVD3 algorithm [30] for subspace tracking. (a) Estimated frequencies. (b) Maximum principal angle.

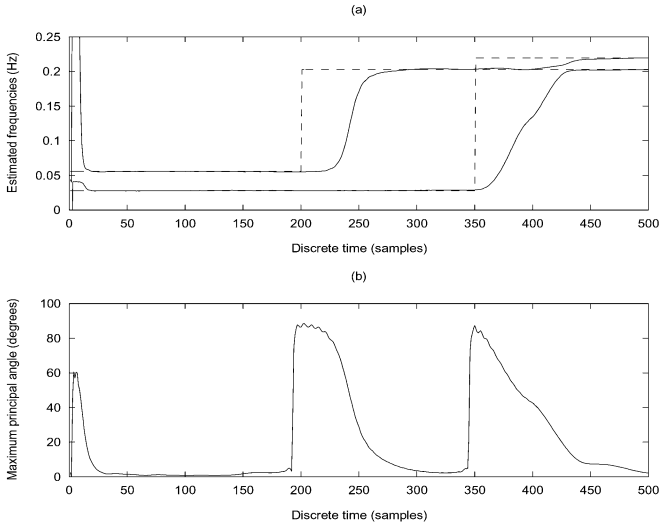

Fig. 8. Tracking performance of the modified Bi-SVD3 algorithm. (a) Estimated frequencies. (b) Maximum principal angle.

The negative impact of $\boldsymbol{\Theta}_B(t-1) = \boldsymbol{I}_r$ for the Bi-SVD3 algorithm [30] is confirmed by the simulation results shown in Fig. 7. The configuration of the simulation is the same as that of Section V-B. It can be seen that the Bi-SVD3 algorithm is very unstable.

However, we can make a simple modification to the Bi-SVD3 algorithm to improve its performance. Referring to [30], we can replace (24) of the Bi-SVD1 in Table II with (21b). Keep in mind that for the Bi-SVD1 algorithm, $\boldsymbol{H}_R(t)$ in Table II should not be removed from (34) in [30]. Although $\boldsymbol{R}_A(t-1)\boldsymbol{H}_R(t)$ in (34) is not an upper triangular matrix, we may take its upper triangular part with little performance penalty. The modified Bi-SVD3 algorithm has a principal complexity $10Nr + 3r^3 + O(r^2)$. Fig. 8 shows the simulation results of the modified Bi-SVD3 algorithm, which performs much better than the original.

REFERENCES

[1] K. Abed-Meraim, A. Chkeif, and Y. Hua, "Fast orthonormal PAST algorithm," *IEEE Signal Process. Lett.*, vol. 7, no. 3, pp. 60–62, Mar. 2000.

[2] R. Badeau, K. Abed-Meraim, G. Richard, and B. David, "Sliding window orthonormal PAST algorithms," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, v-261–v264, Hong Kong, Apr. 2003.

[3] R. Badeau, G. Richard, and B. David, "Sliding window adaptive SVD algorithms," *IEEE Trans. Signal Process.*, vol. 52, no. 1, pp. 1–10, Jan. 2004.

[4] C. H. Bischof and G. M. Shroff, "On updating signal subspaces," *IEEE Trans. Signal Process.*, vol. 40, no. 1, pp. 96–105, Jan. 1992.

[5] B. Champagne and Q. Liu, "Plane rotation-based EVD updating schemes for efficient subspace tracking," *IEEE Trans. Signal Process.*, vol. 46, no. 7, pp. 1886–1900, Jul. 1998.

[6] C. Chatterjee, V. Roychowdhury, and E. Chong, "On relative convergence properties of principal component analysis algorithms," *IEEE Trans. Neural Netw.*, vol. 9, no. 2, pp. 319–329, Mar. 1998.

[7] T. Chen, Y. Hua, and W. Yan, "Global convergence of Oja's subspace algorithm for principal component extraction," *IEEE Trans. Neural Netw.*, vol. 9, no. 1, pp. 58–67, Jan. 1998.

[8] M. Clint and A. Jennings, "A simultaneous iteration method for the unsymmetric eigenvalue problem," *J. Inst. Math. Appl.*, vol. 8, pp. 111–121, 1971.

[9] P. Comon and G. H. Golub, "Tracking a few extreme singular values and vectors in signal processing," *Proc. IEEE*, vol. 78, no. 8, pp. 1327–1343, Aug. 1990.

[10] *Digital Signal Processing Handbook*, V. K. Madisetti and D. B. Williams, Eds., CRC, Boca Raton, FL, 1999. R. D. DeGroat, E. M. Dowling, and A. D. Linebarger, "Subspace tracking".

[11] E. M. Dowling, L. P. Ammann, and R. D. DeGroat, "A TQR-iteration based adaptive SVD for real angle and frequency tracking," *IEEE Trans. Signal Process.*, vol. 42, no. 4, pp. 914–926, Apr. 1994.

[12] Z. Fu and E. M. Dowling, "Conjugate gradient eigenstructure tracking for adaptive spectral estimation," *IEEE Trans. Signal Process.*, vol. 43, no. 5, pp. 1151–1160, May 1995.

[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Third ed. Baltimore, MD: The Johns Hopkins Univ. Press, 1996.

[14] Y. Hua and T. Chen, "On convergence of the NIC algorithm for subspace computation," *IEEE Trans. Signal Process.*, vol. 52, no. 4, pp. 1112–1115, Apr. 2004.

[15] Y. Hua and M. Nikpour, "Computing the reduced rank Wiener filter by IQMD," *IEEE Signal Process. Lett.*, vol. 6, no. 9, pp. 240–242, Sep. 1999.

[16] Y. Hua, M. Nikpour, and P. Stoica, "Optimal reduced-rank estimation and filtering," *IEEE Trans. Signal Process.*, vol. 49, no. 3, pp. 457–469, Mar. 2001.

[17] Y. Hua and T. K. Sarkar, "On SVD for estimating generalized eigenvalues of singular matrix pencils in noise," *IEEE Trans. Signal Process.*, vol. 39, no. 4, pp. 892–900, Apr. 1991.

[18] Y. Hua, Y. Xiang, T. Chen, K. Abed-Meraim, and Y. Miao, "A new look at the power method for fast subspace tracking," *Digital Signal Process.*, vol. 9, no. 4, pp. 297–314, Oct. 1999.

[19] J. K. Hwang and S. T. Wei, "Adaptive DOA tracking by rank-revealing QR updating and exponential sliding windows techniques," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 3, 1994, pp. 19–22.

[20] I. Karasalo, "Estimating the covariance matrix by signal subspace averaging," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 1, pp. 8–12, Feb. 1986.

[21] S. Marcos, A. Marsal, and M. Benidir, "The Propagator method for source bearing estimation," *Signal Process.*, vol. 42, pp. 121–138, Apr. 1995.

[22] Y. Miao and Y. Hua, "Fast subspace tracking and neural network learning by a novel information criterion," *IEEE Trans. Signal Process.*, vol. 46, no. 7, pp. 1967–1979, Jul. 1998.

[23] E. Oja, "Neural networks, principal components, and subspaces," *Int. J. Neural Syst.*, vol. 1, no. 1, pp. 61–68, 1989.

[24] D. Rabideau, "Fast, rank adaptive subspace tracking and applications," *IEEE Trans. Signal Process.*, vol. 44, no. 9, pp. 2229–2244, Sep. 1996.

[25] E. C. Real, D. W. Tufts, and J. W. Cooley, "Two algorithms for fast approximate subspace tracking," *IEEE Trans. Signal Process.*, vol. 47, no. 7, pp. 1936–1945, Jul. 1999.

[26] R. Roy and T. Kailath, "ESPRIT-estimation of signal parameters via rotational invariance techniques," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 7, pp. 984–995, Jul. 1989.

[27] G. W. Stewart, *Topics in Numerical Analysis*, Second ed, J. J. H. Miller, Ed. New York: Academic, 1975.

[28] ——, "An updating algorithm for subspace tracking," *IEEE Trans. Signal Process.*, vol. 40, no. 6, pp. 1535–1541, Jun. 1992.

[29] P. Strobach, "Square hankel SVD subspace tracking algorithms," *Signal Process.*, vol. 57, no. 1, pp. 1–18, Feb. 1997.

[30] ——, "Bi-iteration SVD subspace tracking algorithms," *IEEE Trans. Signal Process.*, vol. 45, no. 5, pp. 1222–1240, May 1997.

[31] ——, "Fast recursive subspace adaptive ESPRIT algorithms," *IEEE Trans. Signal Process.*, vol. 46, no. 9, pp. 2413–2430, Sep. 1998.

[32] L. Xu, "Least mean square error reconstruction principle for self-organizing neural-nets," *Neural Netw.*, vol. 6, pp. 627–648, 1993.

[33] B. Yang, "Projection approximation subspace tracking," *IEEE Trans. Signal Process.*, vol. 43, no. 1, pp. 95–107, Jan. 1995.

Shan Ouyang (M'02) received the B.S. degree in electronic engineering from Guilin University of Electronic Technology, Guilin, China, in 1986 and the M.S. and Ph.D. degrees in electronic engineering from Xidian University, Xi'an, China, in 1992 and 2000, respectively.

In 1986, he jointed Guilin University of Electronic Technology, where he is presently a Professor and the Chair in the Department of Communication and Information Engineering. From May 2001 to May 2002, he was a Research Associate with the Department of Electronic Engineering, The Chinese University of Hong Kong. From December 2002 to January 2004, he was a Research Fellow with the Department of Electrical Engineering, University of California, Riverside. His research interests are mainly in the areas of signal processing for communications and radar, adaptive filtering, and neural network learning theory and applications.

Dr. Ouyang received the Outstanding Youth Award of the Ministry of Electronic Industry and Guanxi Province Outstanding Teacher Award, China, in 1995 and 1997, respectively. He received the National Excellent Doctoral Dissertation of China in 2002.

Yingbo Hua (S'86–M'88–SM'92–F'02) received B.S. degree from Nanjing Institute of Technology, Nanjing, China, in 1982 and the M.S. and Ph.D. degrees from Syracuse University, Syracuse, NY, in 1983 and 1988, respectively.

He was a research fellow at Syracuse University, consulting for Syracuse Research and Aeritalia in Italy, from 1988 to 1989. He was Lecturer from 1990 to 1992, Senior Lecturer from 1993 to 1995, and Reader from 1996 to 2001 with the University of Melbourne, Parkville, Australia. He served as a visiting professor with Hong Kong University of Science and Technology, and consulted for Microsoft Research, Redmond, WA, from 1999 to 2000. Since February 2001, he has been Professor of electrical engineering with the University of California, Riverside.

Dr. Hua won a Chinese Government Scholarship for Overseas Graduate Study during 1983–1984 and a Syracuse University Graduate Fellowship during 1985–1986. He is an author/coauthor of more than 240 articles in journals, conference proceedings, and books. He is a co-editor of *Signal Processing Advances in Wireless and Mobile Communications* (Englewood Cliffs, NJ: Prentice-Hall, 2001) and *High-Resolution and Robust Signal Processing*, (New York: Marcel Dekker, 2003). He served as Associate Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING and the IEEE SIGNAL PROCESSING LETTERS. He has been a member on several IEEE Signal Processing Society's Technical Committees.