Sliding Window Adaptive SVD Algorithms

Roland Badeau, Gaël Richard, Bertrand David

***Abstract*— The singular value decomposition (SVD) is an important tool for subspace estimation. In adaptive signal processing, we are especially interested in tracking the SVD of a recursively updated data matrix. This paper introduces a new tracking technique, designed for rectangular sliding window data matrices. This approach, derived from the classical bi-orthogonal iteration SVD algorithm, shows excellent performance in the context of frequency estimation. It proves to be very robust to abrupt signal changes, due to the use of a sliding window. Finally, an ultra-fast tracking algorithm with comparable performance is proposed.**

***Index Terms*— Subspace tracking, sliding window, SVD.**

I. Introduction

SUBSPACE-based signal analysis consists in splitting the observations into a set of desired and a set of disturbing components, which can be viewed in terms of signal and noise subspaces. This approach has been widely studied in the fields of adaptive filtering, source localization, or parameter estimation [1]. The eigenvalue decomposition (EVD) and the singular value decomposition (SVD) are commonly used in subspace estimation. However, they usually lead to computationally demanding algorithms. Therefore, in an adaptive signal processing context, there is a real need for fast tracking techniques.

A reference method in subspace tracking is I. Karasalo's algorithm [2], which involves the full SVD of a small matrix. More recently, the FST algorithm presented in [3] replaces this SVD by Givens rotations, resulting in a faster tracking. Another approach consists in interlacing a recursive update of the estimated covariance matrix or the data matrix with one or a few steps of a standard SVD or power iteration algorithm. This is the case of the Jacobi SVD method [4], the transposed QR-iteration [5], the orthogonal / bi-orthogonal iteration [6], [7], and the power method [8]. Some tracking techniques are based on other matrix decompositions, such as the rank-revealing QR factorization [9], the rank-revealing URV decomposition [10], and the Lankzos (bi)-diagonalization [11]. A conceptually different approach considers the principal subspace estimation as a constrained or unconstrained optimization problem [12]–[17]. In particular, it is established in [13], [18] that the classical Oja method [12] can be viewed as an approximated gradient descent of a mean square error function. A number of faster subspace tracking methods have been developed based on the combination of the gradient descent approach with a projection approximation hypothesis [18]–[21]. Other techniques rely on the noise and signal subspace averaging method [22], the maximum likelihood principle [23], the operator restriction analysis [24], or the perturbation theory [25]. A review of former literature can be found in [1].

Most of these adaptive techniques are designed for exponential forgetting windows. Indeed, this choice tends to smooth the signal variations and thus allows a low-complexity update at each time step. However, it is only suitable for slowly varying signals. Conversely, a few subspace trackers are based on sliding windows, which generally require more computations, but offer a faster tracking response to sudden signal changes [18], [26]. The tracking of the full SVD in the sliding window case was investigated in [27] and [28].

In this paper, we will focus on the bi-orthogonal iteration SVD method [29], [30]. This technique has been widely investigated by P. Strobach, who proposed various subspace tracking algorithms designed for exponential forgetting windows [6], [7]. In [27], the sliding window case was addressed, but the approach was limited to real square Hankel data matrices. The adaptive SVD technique presented in this paper overcomes this limitation. Our work mainly differs from that presented in [7] by the way the basic sequential bi-iteration SVD algorithm is simplified.

Compared to the above mentioned subspace tracking methods, our fastest algorithm has the advantage of

- computing an orthonormal subspace basis *at each time step*, which is required for some subspace-based estimation methods, such as MUSIC [31],
- relying on a sliding window, which offers a faster tracking response to abrupt signal variations,
- tracking the full SVD, which may be useful for rank estimation and tracking, as in [7] and [28],
- relying on an approximation of the data matrix which is less restrictive than the classical *projection approximation* [18], leading to better tracking results.

The paper is organized as follows. In section II, we recall the principles of the bi-orthogonal iteration approach, from which our new Sliding Window Adaptive SVD (SWASVD) algorithm is derived. A fast implementation of SWASVD is then presented in section III. In section IV, the capacity of these new tracking algorithms to cope with transients is illustrated in the context of frequency estimation. Their performance is compared to that of some of the most robust and efficient methods found in the literature. Finally, the main conclusions of this paper are summarized in section V.

II. Sliding window adaptive SVD

The bi-orthogonal iteration SVD algorithm is a straightforward extension of the classical orthogonal iteration, which computes the EVD of a square matrix [32, section 8.2.4]. In this section, it will be shown how this algorithm can be made adaptive, and how its computational complexity can be reduced with a low-rank approximation of the data matrix.

Roland Badeau, Gaël Richard and Bertrand David are with the Department of Signal and Image Processing, Ecole Nationale Supérieure des Télécommunications (ENST), Paris, France. E-mail: [roland.badeau, gael.richard, bertrand.david]@enst.fr.

TABLE I
BI-ORTHOGONAL ITERATION SVD ALGORITHM

```
Initialize : $\boldsymbol{Q}_A(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{0} \end{bmatrix}$
FOR $n = 1, 2 \ldots$ UNTIL CONVERGENCE DO :
    First Iteration :
        $\boldsymbol{B}(n) = \boldsymbol{X}\,\boldsymbol{Q}_A(n-1)$            matrix product
        $\boldsymbol{B}(n) = \boldsymbol{Q}_B(n)\,\boldsymbol{R}_B(n)$        skinny QR factorization
    Second Iteration :
        $\boldsymbol{A}(n) = \boldsymbol{X}^H\,\boldsymbol{Q}_B(n)$          matrix product
        $\boldsymbol{A}(n) = \boldsymbol{Q}_A(n)\,\boldsymbol{R}_A(n)$        skinny QR factorization
```

A. The bi-orthogonal iteration SVD algorithm

The bi-orthogonal iteration algorithm computes the r dominant singular values and vectors of a data matrix $\boldsymbol{X} \in \mathbb{C}^{L\times N}$ (with $r \leq r_{\max} \triangleq \min(L, N)$). The SVD of $\boldsymbol{X}$ is the factorization $\boldsymbol{X} = \boldsymbol{U}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^H$, where $\boldsymbol{U} \in \mathbb{C}^{L\times r_{\max}}$ and $\boldsymbol{V} \in \mathbb{C}^{N\times r_{\max}}$ are orthonormal matrices and $\boldsymbol{\Sigma} \in \mathbb{R}^{r_{\max}\times r_{\max}}$ is a non negative diagonal matrix: $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{r_{\max}})$ where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{r_{\max}} \geq 0$. Thus the r dominant singular values are $\{\sigma_1, \sigma_2, \ldots, \sigma_r\}$, the r dominant left singular vectors are the r first columns of the matrix $\boldsymbol{U}$, and the r dominant right singular vectors are the r first columns of the matrix $\boldsymbol{V}$. In many signal processing applications, r is much lower than $r_{\max}$.

The quasi-code of the bi-orthogonal iteration SVD algorithm is given in Table I. This algorithm generates two auxiliary matrices $\boldsymbol{B}(n) \in \mathbb{C}^{L\times r}$ and $\boldsymbol{A}(n) \in \mathbb{C}^{N\times r}$. It can be shown [29], [30] that the columns of $\boldsymbol{Q}_B(n)$ converge to the r dominant left singular vectors, the columns of $\boldsymbol{Q}_A(n)$ converge to the r dominant right singular vectors, and $\boldsymbol{R}_B(n)$ and $\boldsymbol{R}_A(n)$ both converge to $\boldsymbol{\Sigma}$.

B. The sequential bi-iteration SVD algorithm

The bi-orthogonal iteration algorithm can simply be adapted in a tracking context. Suppose the data matrix is updated according to the following scheme:

$$\boldsymbol{X}(t) = \begin{bmatrix} \boldsymbol{x}(t)^H \\ \boldsymbol{x}(t-1)^H \\ \vdots \\ \boldsymbol{x}(t-L+1)^H \end{bmatrix}$$

where $\boldsymbol{x}(t)$ is the N dimensional data vector at time t[1]. The SVD of $\boldsymbol{X}(t)$ can be approximated and updated just by replacing the iteration index n in Table I by the discrete time index t.

The sequential bi-iteration algorithm is summarized in Table II. In the right column, the computational complexities are quantified with a multiplicative factor related to the *flop* (real FLoating point OPeration) count, as obtained with the Matlab *flops* command [32, section 1.2.4]. For example, a dot product of N dimensional complex vectors involves $8N$ flops.

[1]In the context of frequency estimation, the coefficients of $\boldsymbol{x}(t)$ are the successive samples of the signal: $\boldsymbol{x}(t) = [x(t), x(t-1), \ldots, x(t-N+1)]^T$. In the context of Direction Of Arrival (DOA) estimation, $\boldsymbol{x}(t)$ is the snapshot vector received from the N captors.

In spite of its robustness, the main drawback of this SVD tracking algorithm is its high computational complexity (since in practice $r << \max(N, L)$, its dominant cost is $16NLr$). However, some simplifications will be brought below, that will result in lower-complexity algorithms.

TABLE II
SEQUENTIAL BI-ITERATION SVD ALGORITHM

Initialize : $\boldsymbol{Q}_A(0) = \begin{bmatrix} \boldsymbol{I}_r \\ \boldsymbol{0} \end{bmatrix}$	
FOR EACH TIME STEP DO :	
First Iteration :	Complexity :
$\boldsymbol{B}(t) = \boldsymbol{X}(t)\,\boldsymbol{Q}_A(t-1)$	$8NLr$
$\boldsymbol{B}(t) = \boldsymbol{Q}_B(t)\,\boldsymbol{R}_B(t)$	$19Lr^2$
Second Iteration :	
$\boldsymbol{A}(t) = \boldsymbol{X}(t)^H\,\boldsymbol{Q}_B(t)$	$8NLr$
$\boldsymbol{A}(t) = \boldsymbol{Q}_A(t)\,\boldsymbol{R}_A(t)$	$19Nr^2$

C. Low-rank approximation of the updated data matrix

In this section, a low-rank approximation of the data matrix $\boldsymbol{X}(t)$ will be introduced. In array processing, it is well known that rank reductions have a noise-cleaning effect. Here, this approximation will result in a faster tracking algorithm.

First, the time-updating structure of the data matrix can advantageously be taken into account. Indeed, it can be noticed that

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

Now consider the compressed data vector $\boldsymbol{h}(t) = \boldsymbol{Q}_A(t-1)^H\,\boldsymbol{x}(t)$. According to the definition of $\boldsymbol{B}(t)$ (see Table II), equation (1) becomes

$$\begin{bmatrix} \boldsymbol{B}(t) \\ \times \ldots \times \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}(t)^H \\ \boldsymbol{X}(t-1)\,\boldsymbol{Q}_A(t-1) \end{bmatrix} \quad (2)$$

where the symbol $\times$ denotes uninteresting quantities.

To go further, P. Strobach [7] introduces the low-rank approximation $\widetilde{\boldsymbol{X}}(t) = \boldsymbol{X}(t)\left(\boldsymbol{Q}_A(t-1)\,\boldsymbol{Q}_A(t-1)^H\right) = \boldsymbol{Q}_B(t)\,\boldsymbol{R}_B(t)\,\boldsymbol{Q}_A(t-1)^H$ of $\boldsymbol{X}(t)$, which corresponds to the projection of the rows of $\boldsymbol{X}(t)$ onto the subspace spanned by $\boldsymbol{Q}_A(t-1)$. Consequently,

$$\widetilde{\boldsymbol{X}}(t-1)\,\boldsymbol{Q}_A(t-1) = \boldsymbol{Q}_B(t-1)\,\boldsymbol{R}_B(t-1)\,\boldsymbol{\Theta}_A(t-1)$$

where $\boldsymbol{\Theta}_A(t-1) = \boldsymbol{Q}_A(t-2)^H\,\boldsymbol{Q}_A(t-1)$. It can be seen that this approximation is less restrictive than the classical *projection approximation* [18], which implicitely assumes that $\boldsymbol{\Theta}_A(t-1) = \boldsymbol{I}_r$.

However, we prefer use the low-rank approximation $\hat{\boldsymbol{X}}(t) = \left(\boldsymbol{Q}_B(t)\,\boldsymbol{Q}_B(t)^H\right)\boldsymbol{X}(t) = \boldsymbol{Q}_B(t)\,\boldsymbol{R}_A(t)^H\,\boldsymbol{Q}_A(t)^H$. It corresponds to the projection of the columns of $\boldsymbol{X}(t)$ onto the subspace spanned by $\boldsymbol{Q}_B(t-1)$. Consequently,

$$\hat{\boldsymbol{X}}(t-1)\,\boldsymbol{Q}_A(t-1) = \boldsymbol{Q}_B(t-1)\,\boldsymbol{R}_A(t-1)^H$$

This choice has the advantage of involving more up to date matrix factors than $\widetilde{\boldsymbol{X}}(t)$. Moreover, the explicit computation of the matrix $\boldsymbol{\Theta}_A(t-1)$ is avoided.

The substitution of $\hat{\boldsymbol{X}}(t-1)$ to $\boldsymbol{X}(t-1)$ in equation (2) yields

$$\left[\begin{array}{c}\boldsymbol{B}(t)\\ \hline \times\ldots\times\end{array}\right] \simeq \left[\begin{array}{c}\boldsymbol{h}(t)^H\\ \hline \boldsymbol{Q}_B(t-1)\,\boldsymbol{R}_A(t-1)^H\end{array}\right]. \tag{3}$$

In the same way, it can be noticed that

$$\begin{aligned}&\left[\begin{array}{c:c}\boldsymbol{X}(t)^H & \boldsymbol{x}(t-L)\end{array}\right]\left[\begin{array}{c}\boldsymbol{Q}_B(t)\\ \hline 0\ \ldots\ 0\end{array}\right]\\ =\ &\left[\begin{array}{c:c}\boldsymbol{x}(t) & \boldsymbol{X}(t-1)^H\end{array}\right]\left[\begin{array}{c}\boldsymbol{Q}_B(t)\\ \hline 0\ \ldots\ 0\end{array}\right].\end{aligned} \tag{4}$$

According to the definition of $\boldsymbol{A}(t)$ (see Table II), equation (4) becomes

$$\boldsymbol{A}(t) = \left[\begin{array}{c:c}\boldsymbol{x}(t) & \boldsymbol{X}(t-1)^H\end{array}\right]\left[\begin{array}{c}\boldsymbol{Q}_B(t)\\ \hline 0\ \ldots\ 0\end{array}\right]. \tag{5}$$

Taking into account that the sequential bi-iteration SVD algorithm satisfies the equation $\boldsymbol{Q}_A(t-1)^H\boldsymbol{A}(t) = \boldsymbol{B}(t)^H\boldsymbol{Q}_B(t) = \boldsymbol{R}_B(t)^H$, a pre-multiplication of both sides of (5) by $\boldsymbol{Q}_A(t-1)^H$ yields

$$\begin{aligned}\boldsymbol{R}_B(t)^H \ =\ &\left[\begin{array}{c:c}\boldsymbol{h}(t) & \boldsymbol{Q}_A(t-1)^H\,\boldsymbol{X}(t-1)^H\end{array}\right]\\ &\left[\begin{array}{c}\boldsymbol{Q}_B(t)\\ \hline 0\ \ldots\ 0\end{array}\right].\end{aligned} \tag{6}$$

Then let $\boldsymbol{x}_\perp(t) = \boldsymbol{x}(t) - \boldsymbol{Q}_A(t-1)\,\boldsymbol{h}(t)$. This vector is orthogonal to $\mathrm{span}(\boldsymbol{Q}_A(t-1))$, so that $\boldsymbol{x}(t)$ can be written as a sum of two orthogonal vectors

$$\boldsymbol{x}(t) = \boldsymbol{Q}_A(t-1)\,\boldsymbol{h}(t) + \boldsymbol{x}_\perp(t). \tag{7}$$

The substitution of $\hat{\boldsymbol{X}}(t-1)$ to $\boldsymbol{X}(t-1)$ in equations (5) and (6) respectively yields

$$\begin{aligned}&\boldsymbol{A}(t) \simeq \left[\begin{array}{c:c}\boldsymbol{Q}_A(t-1) & \boldsymbol{x}_\perp(t)\end{array}\right]\\ &\left[\begin{array}{c|c}\boldsymbol{h}(t) & \boldsymbol{R}_A(t-1)\boldsymbol{Q}_B(t-1)^H\\ \hline 1 & 0\ \ldots\ 0\end{array}\right]\left[\begin{array}{c}\boldsymbol{Q}_B(t)\\ \hline 0\ \ldots\ 0\end{array}\right]\end{aligned} \tag{8}$$

and

$$\begin{aligned}\boldsymbol{R}_B(t)^H \ \simeq\ &\left[\begin{array}{c:c}\boldsymbol{h}(t) & \boldsymbol{R}_A(t-1)\,\boldsymbol{Q}_B(t-1)^H\end{array}\right]\\ &\left[\begin{array}{c}\boldsymbol{Q}_B(t)\\ \hline 0\ \ldots\ 0\end{array}\right].\end{aligned} \tag{9}$$

Let $\boldsymbol{q}_{B_1}(t)$ be the column vector obtained by transposing the first row of $\boldsymbol{Q}_B(t)$. Equations (8) and (9) finally yield

$$\boldsymbol{A}(t) \simeq \boldsymbol{Q}_A(t-1)\,\boldsymbol{R}_B(t)^H + \boldsymbol{x}_\perp(t)\,\boldsymbol{q}_{B_1}(t)^H. \tag{10}$$

Note that the exact computation of $\boldsymbol{B}(t)$ and $\boldsymbol{A}(t)$ requires $16NLr$ operations whereas the approximated matrices (3) and (10) can be computed in $4Lr^2$ and $4Nr^2$ operations. Therefore, introducing these approximations in the sequential bi-iteration SVD algorithm leads to the lower complexity algorithm herein called SWASVD, summarized in Table III. Its dominant cost is only $23(L+N)r^2$. Moreover, it can be seen that for all $r \leq r_{\max}$, SWASVD requires less computations than the sequential bi-iteration algorithm. From now on, $\boldsymbol{B}(t)$ and $\boldsymbol{A}(t)$ will denote the approximated auxiliary matrices.

TABLE III
SLIDING WINDOW ADAPTIVE SVD ALGORITHM (SWASVD)

Initialize : $\boldsymbol{Q}_A(0) = \left[\begin{array}{c}\boldsymbol{I}_r\\ \hline \boldsymbol{0}\end{array}\right]$; $\boldsymbol{Q}_B(0) = \left[\begin{array}{c}\boldsymbol{I}_r\\ \hline \boldsymbol{0}\end{array}\right]$; $\boldsymbol{R}_A(0) = \boldsymbol{I}_r$;	
FOR EACH TIME STEP DO :	
Input : $\boldsymbol{x}(t)$	
First Iteration :	Complexity :
$\boldsymbol{h}(t) = \boldsymbol{Q}_A(t-1)^H\,\boldsymbol{x}(t)$	$8Nr$
$\left[\begin{array}{c}\boldsymbol{B}(t)\\ \hline \times\ldots\times\end{array}\right] = \left[\begin{array}{c}\boldsymbol{h}(t)^H\\ \hline \boldsymbol{Q}_B(t-1)\,\boldsymbol{R}_A(t-1)^H\end{array}\right]$	$4Lr^2$
$\boldsymbol{B}(t) = \boldsymbol{Q}_B(t)\,\boldsymbol{R}_B(t)$	$19Lr^2$
Second Iteration :	
$\boldsymbol{x}_\perp(t) = \boldsymbol{x}(t) - \boldsymbol{Q}_A(t-1)\,\boldsymbol{h}(t)$	$8Nr$
$\boldsymbol{A}(t) = \boldsymbol{Q}_A(t-1)\,\boldsymbol{R}_B(t)^H + \boldsymbol{x}_\perp(t)\,\boldsymbol{q}_{B_1}(t)^H$	$4Nr^2$
$\boldsymbol{A}(t) = \boldsymbol{Q}_A(t)\,\boldsymbol{R}_A(t)$	$19Nr^2$

III. FAST IMPLEMENTATION OF THE SLIDING WINDOW ADAPTIVE SVD ALGORITHM

A major drawback in the SWASVD algorithm is the explicit computation and QR factorization of the approximated matrices $\boldsymbol{B}(t)$ and $\boldsymbol{A}(t)$. However, these operations can be avoided by directly updating the QR factorizations.

Since this update is simpler in the case of $\boldsymbol{A}(t)$, the optimization of the second iteration will be presented first.

A. Fast implementation of the second iteration

In the second member of equation (10), the vector $\boldsymbol{x}_\perp(t)$ is orthogonal to $\mathrm{span}(\boldsymbol{Q}_A(t-1))$. It can be normalized as

$$\bar{\boldsymbol{x}}_\perp(t) = \frac{\boldsymbol{x}_\perp(t)}{\|\boldsymbol{x}_\perp(t)\|} \tag{11}$$

(in the special case $\boldsymbol{x}_\perp(t) = \boldsymbol{0}$, $\bar{\boldsymbol{x}}_\perp(t)$ is forced to be $\boldsymbol{0}$). Then $\boldsymbol{A}(t)$ can be written as the product

$$\boldsymbol{A}(t) = \left[\begin{array}{c:c}\boldsymbol{Q}_A(t-1) & \bar{\boldsymbol{x}}_\perp(t)\end{array}\right]\boldsymbol{T}_A(t) \tag{12}$$

of a $N\times(r+1)$ orthonormal matrix by the $(r+1)\times r$ matrix

$$\boldsymbol{T}_A(t) = \left[\begin{array}{c}\boldsymbol{R}_B(t)^H\\ \hline \|\boldsymbol{x}_\perp(t)\|\,\boldsymbol{q}_{B_1}(t)^H\end{array}\right]. \tag{13}$$

Now consider the QR factorization of $\boldsymbol{T}_A(t)$:

$$\boldsymbol{T}_A(t) = \boldsymbol{G}_A(t)\left[\begin{array}{c}\boldsymbol{R}_A(t)\\ \hline 0\ldots0\end{array}\right] \tag{14}$$

where $\boldsymbol{G}_A(t)$ is a square $(r+1)\times(r+1)$ orthonormal matrix and $\boldsymbol{R}_A(t)$ is a square $r\times r$ upper-triangular matrix (it will be shown below that $\boldsymbol{R}_A(t)$ is also the triangular factor in the QR factorization of $\boldsymbol{A}(t)$, as defined in section II). Equations (12) and (14) yield

$$\boldsymbol{A}(t) = \left(\left[\begin{array}{c:c}\boldsymbol{Q}_A(t-1) & \bar{\boldsymbol{x}}_\perp(t)\end{array}\right]\boldsymbol{G}_A(t)\right)\left[\begin{array}{c}\boldsymbol{R}_A(t)\\ \hline 0\ldots0\end{array}\right]. \tag{15}$$

This last equation shows an explicit QR factorization of $\boldsymbol{A}(t)$. From (15), $\boldsymbol{Q}_A(t)$ can be directly extracted:

$$\left[\begin{array}{c|c} \boldsymbol{Q}_A(t) & \begin{matrix}\times\\ \vdots \\ \times\end{matrix}\end{array}\right] = \left[\begin{array}{c:c} \boldsymbol{Q}_A(t-1) & \bar{\boldsymbol{x}}_\perp(t)\end{array}\right]\boldsymbol{G}_A(t). \quad (16)$$

Therefore, the QR factorization of $\boldsymbol{A}(t)$ can be updated with the smaller factorization (14) and the product (16).

B. Fast implementation of the first iteration

The QR factorization of $\boldsymbol{B}(t)$ is more difficult to update because of the row-shifting in the updating scheme of the data matrix. An elegant but complex way of achieving this update can be found in [27]. A simpler solution, inspired from the considerations of section III-A, is proposed below.

Let $\boldsymbol{q}_{B_L}(t-1)$ be the column vector obtained by transposing the last row of $\boldsymbol{Q}_B(t-1)$. Consider the orthonormal matrix $\tilde{\boldsymbol{Q}}_B(t-1)$ obtained by a circular permutation of the rows of $\boldsymbol{Q}_B(t-1)$:

$$\tilde{\boldsymbol{Q}}_B(t-1) = \left[\begin{array}{c|c} 0\dots0 & 1 \\ \hline \boldsymbol{I}_{L-1} & \begin{matrix}0\\ \vdots\\ 0\end{matrix}\end{array}\right]\boldsymbol{Q}_B(t-1).$$

Finally, consider the L dimensional vector $\boldsymbol{z} = [1, 0\dots0]^T$. Equation (3) yields

$$\boldsymbol{B}(t) = \tilde{\boldsymbol{Q}}_B(t-1)\,\boldsymbol{R}_A(t-1)^H + \boldsymbol{z}\,\tilde{\boldsymbol{h}}(t)^H$$

where $\tilde{\boldsymbol{h}}(t) = \boldsymbol{h}(t) - \boldsymbol{R}_A(t-1)\,\boldsymbol{q}_{B_L}(t-1)$.

Now, the orthogonal decomposition of $\boldsymbol{x}(t)$ given in equation (7) will be transposed to $\boldsymbol{z}$. Thus, let $\boldsymbol{z}_\perp(t) = \boldsymbol{z} - \tilde{\boldsymbol{Q}}_B(t-1)\,\boldsymbol{q}_{B_L}(t-1)$. It can be noticed that $\boldsymbol{q}_{B_L}(t-1) = \tilde{\boldsymbol{Q}}_B(t-1)^H\,\boldsymbol{z}$, so that the vector $\boldsymbol{z}_\perp(t)$ is orthogonal to $\operatorname{span}(\tilde{\boldsymbol{Q}}_B(t-1))$. Then $\boldsymbol{z}$ can be written as a sum of two orthogonal vectors:

$$\boldsymbol{z} = \tilde{\boldsymbol{Q}}_B(t-1)\,\boldsymbol{q}_{B_L}(t-1) + \boldsymbol{z}_\perp(t). \quad (17)$$

As for $\boldsymbol{x}(t)$, let

$$\bar{\boldsymbol{z}}_\perp(t) = \frac{\boldsymbol{z}_\perp(t)}{\|\boldsymbol{z}_\perp(t)\|}$$

(in the special case $\boldsymbol{z}_\perp(t) = \boldsymbol{0}$, $\bar{\boldsymbol{z}}_\perp(t)$ is forced to be $\boldsymbol{0}$). Finally, $\boldsymbol{B}(t)$ can be written as the product

$$\boldsymbol{B}(t) = \left[\begin{array}{c:c} \tilde{\boldsymbol{Q}}_B(t-1) & \bar{\boldsymbol{z}}_\perp(t)\end{array}\right]\boldsymbol{T}_B(t) \quad (18)$$

of a $L\times(r+1)$ orthonormal matrix by the $(r+1)\times r$ matrix

$$\boldsymbol{T}_B(t) = \left[\begin{array}{c} \boldsymbol{R}_A(t-1)^H \\ \hdashline 0\dots0\end{array}\right] + \left[\begin{array}{c} \boldsymbol{q}_{B_L}(t-1) \\ \hdashline \|\boldsymbol{z}_\perp(t)\|\end{array}\right]\tilde{\boldsymbol{h}}(t)^H. \quad (19)$$

Now consider the QR factorization of $\boldsymbol{T}_B(t)$:

$$\boldsymbol{T}_B(t) = \boldsymbol{G}_B(t)\left[\begin{array}{c} \boldsymbol{R}_B(t) \\ \hdashline 0\dots0\end{array}\right] \quad (20)$$

where $\boldsymbol{G}_B(t)$ is a square $(r+1)\times(r+1)$ orthonormal matrix and $\boldsymbol{R}_B(t)$ is a square $r\times r$ upper-triangular matrix (it will be shown below that $\boldsymbol{R}_B(t)$ is also the triangular factor in the QR factorization of $\boldsymbol{B}(t)$, as defined in section II). Equations (18) and (20) yield

$$\boldsymbol{B}(t) = \left(\left[\begin{array}{c:c} \tilde{\boldsymbol{Q}}_B(t-1) & \bar{\boldsymbol{z}}_\perp(t)\end{array}\right]\boldsymbol{G}_B(t)\right)\left[\begin{array}{c} \boldsymbol{R}_B(t) \\ \hdashline 0\dots0\end{array}\right]. \quad (21)$$

This last equation shows an explicit QR factorization of the matrix $\boldsymbol{B}(t)$. As for $\boldsymbol{Q}_A(t)$, $\boldsymbol{Q}_B(t)$ can be directly extracted from this factorization:

$$\left[\begin{array}{c|c} \boldsymbol{Q}_B(t) & \begin{matrix}\times\\ \vdots \\ \times\end{matrix}\end{array}\right] = \left[\begin{array}{c:c} \tilde{\boldsymbol{Q}}_B(t-1) & \bar{\boldsymbol{z}}_\perp(t)\end{array}\right]\boldsymbol{G}_B(t). \quad (22)$$

Therefore, the time-consuming direct QR factorization of $\boldsymbol{B}(t)$ can be split into the smaller QR factorization (20) and the product (22). Finally, equations (14), (16), (20) and (22) lead to the fast implementation of the SWASVD algorithm given in Table IV [2], herein called SWASVD2. Its dominant cost is only $8(N+L)r^2$. Therefore, SWASVD2 is approximately three times faster than SWASVD. As a comparison, the dominant cost of the exponential forgetting window Bi-SVD1 algorithm presented in [7] is $8Nr^2$ at each time step. It can be seen that SWASVD2 requires a number of additional operations proportional to the sliding window length. However, this increased computational cost is compensated by better performance, as shown in the next section.

C. A step towards linear complexity

In spite of the various optimizations that were introduced above, the SWASVD2 algorithm is not the fastest subspace tracker which can be found in the literature (for instance, the algorithms presented in [18]–[21] require only $O(Nr)$ operations).

To reach this minimal complexity, P. Strobach [7] assumes that the matrix $\boldsymbol{\Theta}_A(t) = \boldsymbol{Q}_A(t-1)^H\boldsymbol{Q}_A(t)$ is close to the $r\times r$ identity matrix (which is the same as the classical projection approximation [18], as mentioned in section II-C). Such an approximation is not required here, since the use of $\hat{\boldsymbol{X}}(t-1)$ instead of $\widetilde{\boldsymbol{X}}(t-1)$ avoids the explicit computation of $\boldsymbol{\Theta}_A(t)$ in SWASVD2.

Table IV shows that the $8(N+L)r^2$ dominant cost of SWASVD2 is due to the use of the full rotation matrices $\boldsymbol{G}_B(t)$ and $\boldsymbol{G}_A(t)$. These matrices are computed so as to make $\boldsymbol{R}_B(t)$ and $\boldsymbol{R}_A(t)$ upper triangular.

In fact, it can be shown that this triangular constraint does not affect the signal subspace estimation. If $\boldsymbol{R}_B(t)$ and $\boldsymbol{R}_A(t)$ were not triangular, the algorithm would also converge to an orthonormal matrix spanning the signal subspace (this approach is known as the power method [8]). The triangular

[2]The computation of $\bar{\boldsymbol{x}}_\perp(t)$ is subject to rounding errors that might affect the algorithm stability, due to a loss of orthogonality among the columns of $\boldsymbol{Q}_A$. Note that the orthogonality can be maintained by repeating one or a few times the following operations:

- projection of $\bar{\boldsymbol{x}}_\perp(t)$ onto $\operatorname{span}(\boldsymbol{Q}_A)^\perp$;
- renormalization of $\bar{\boldsymbol{x}}_\perp(t)$.

The same method can be applied to $\bar{\boldsymbol{z}}_\perp(t)$, in order to maintain the orthogonality among the columns of $\tilde{\boldsymbol{Q}}_B$.

TABLE IV
FAST IMPLEMENTATION OF THE SLIDING WINDOW ADAPTIVE SVD ALGORITHM (SWASVD2)

FOR EACH TIME STEP DO :

First Iteration :	Complexity :
$\boldsymbol{h}(t) = \boldsymbol{Q}_A(t-1)^H\,\boldsymbol{x}(t)$	$8Nr$
$\boldsymbol{z}_\perp(t) = [1, 0\dots 0]^T - \tilde{\boldsymbol{Q}}_B(t-1)\,\boldsymbol{q}_{B_L}(t-1)$	$8Lr$
$\bar{\boldsymbol{z}}_\perp(t) = \frac{\boldsymbol{z}_\perp(t)}{\lVert \boldsymbol{z}_\perp(t)\rVert}$	$10L$
$\tilde{\boldsymbol{h}}(t) = \boldsymbol{h}(t) - \boldsymbol{R}_A(t-1)\boldsymbol{q}_{B_L}(t-1)$	$4r^2$
$\left[\begin{array}{c}\boldsymbol{R}_A(t-1)^H \\ \hline 0\dots 0\end{array}\right] + \left[\begin{array}{c}\boldsymbol{q}_{B_L}(t-1) \\ \hline \lVert \boldsymbol{z}_\perp(t)\rVert\end{array}\right]\tilde{\boldsymbol{h}}(t)^H$	
$= \boldsymbol{G}_B(t)\left[\begin{array}{c}\boldsymbol{R}_B(t) \\ \hline 0\dots 0\end{array}\right]$	$12r^3$
$\begin{bmatrix}\boldsymbol{Q}_B(t) & \times\end{bmatrix} = \begin{bmatrix}\tilde{\boldsymbol{Q}}_B(t-1) & \bar{\boldsymbol{z}}_\perp(t)\end{bmatrix}\boldsymbol{G}_B(t)$	$8Lr^2$
Second Iteration :	
$\boldsymbol{x}_\perp(t) = \boldsymbol{x}(t) - \boldsymbol{Q}_A(t-1)\,\boldsymbol{h}(t)$	$8Nr$
$\bar{\boldsymbol{x}}_\perp(t) = \frac{\boldsymbol{x}_\perp(t)}{\lVert \boldsymbol{x}_\perp(t)\rVert}$	$10N$
$\left[\begin{array}{c}\boldsymbol{R}_B(t)^H \\ \hline \lVert \boldsymbol{x}_\perp(t)\rVert\,\boldsymbol{q}_{B_1}(t)^H\end{array}\right] = \boldsymbol{G}_A(t)\left[\begin{array}{c}\boldsymbol{R}_A(t) \\ \hline 0\dots 0\end{array}\right]$	$12r^3$
$\begin{bmatrix}\boldsymbol{Q}_A(t) & \times\end{bmatrix} = \begin{bmatrix}\boldsymbol{Q}_A(t-1) & \bar{\boldsymbol{x}}_\perp(t)\end{bmatrix}\boldsymbol{G}_A(t)$	$8Nr^2$

constraint is only required to guarantee the convergence to the r dominant singular vectors.

Therefore, linear complexity can be reached by simply relaxing this constraint. The exact QR factorization can be replaced by an "approximated QR factorization", which involves a "nearly triangular" right factor. This method, herein called SWASVD3, is presented in the appendix and requires $O(Nr)$ operations. Its subspace tracking performance is exactly the same as that of SWASVD2. Although the convergence to the singular vectors and values is no longer theoretically guaranteed, the algorithm proves to robustly track their variations.

IV. SIMULATION RESULTS

In this section, the performance of the new tracking algorithms is illustrated in the context of frequency estimation. A discrete signal $x(t)$ can be described using a Hankel data matrix $\boldsymbol{X}(t)$. In the Exponentially Damped Sinusoidal (EDS) model case, it can be shown that $\mathrm{span}(\boldsymbol{X}(t))$ is a r dimensional subspace, where r is the number of complex sinusoids. The ESPRIT high resolution method can be used to estimate the model parameters, among which the frequencies of the sinusoids [33], [34].

Here, this high resolution method has been tested in conjunction with several subspace trackers on a synthetic signal (an application to real audio signals was proposed in [33], involving the sequential iteration SVD algorithm).

The test signal of Figure 1-a is a sum of $r = 4$ complex sinusoidal sources plus a complex white gaussian noise. The frequencies of the sinusoids vary according to a "jump scenario" (proposed by P. Strobach in the context of DOA estimation [35]): their values abruptly change at different time instants, between which they remain constant. Their variations are represented on Figure 1-b.

The SWASVD2 algorithm was applied to this signal with matrix dimensions $N = 80$ and $L = 120$. As in [35], the signal-to-noise (SNR) ratio was fixed to 5.7dB.

Figure 2-a shows the frequency tracking result. The dotted line indicates the true frequency parameters, while the solid line indicates the estimated frequencies. It can be noticed that SWASVD2 robustly tracks abrupt frequency variations.

The performance of the subspace estimation is also analyzed in terms of the maximum principal angle between the true dominant subspace of the data matrix (obtained via an exact singular value decomposition), and the estimated dominant subspace of the same data matrix (obtained with the tracker). This error criterion was originally proposed by P. Comon and G.H. Golub as a measure of the distance between equidimensional subspaces [1]. Figure 2-b shows that the subspace estimation fails on transient regions, but gives excellent results everywhere else. This is not surprising since the subspace modeling does not make sense in transient regions.

Figure 3 shows the result obtained with the ultra-fast SWASVD3 algorithm on the same test signal. It can be noticed that this algorithm reaches the same performance as SWASVD2.

These results have been compared to those obtained with some of the most robust subspace trackers found in the literature [2], [5]–[8], [18]–[20], [27], [28]. Three of them are illustrated in figures 4, 5 and 6:

- the exponential forgetting window Bi-SVD1 algorithm by Strobach [7],
- the FAST algorithm by Real *et al.* [28], which is a recent contribution to sliding window SVD subspace tracking,
- our sliding window version of the NIC subspace tracker by Miao and Hua [19].

Despite the good performance of the Bi-SVD1 algorithm, its convergence is slower than that of SWASVD3 after abrupt signal variations[3]. This may be explained by the use of an exponential forgetting window. Note that the Bi-SVD3 subspace tracker, also presented in [7], has a lower complexity (its dominant cost is $80Nr$), but it proved to be unstable on this test signal.

Concurrently, the FAST subspace tracker is better than Bi-SVD1 in terms of the maximum principal angle error (figure 5-b). However, its dominant cost is $8NLr$, and the frequency tracking response (figure 5-a) remains slower than that of SWASVD3 . Note that the dominant cost of the approximated FAST2 algorithm [28] is also $8NLr$.

The Novel Information Criterion (NIC) subspace tracker was introduced in [19] as a robust generalization of the PAST algorithm [18]. Figure 6-a shows the frequency tracking obtained with our sliding window version of NIC[4], whose dominant cost is $30Nr$. It can be noticed that this fast subspace tracker is very stable and converges much faster than Bi-SVD1 and FAST. However, this algorithm only converges to

[3]The forgetting factor $\alpha \simeq 0.99$ was chosen to get an effective window length equal to L.

[4]The learning step size η was equal to 0.5.

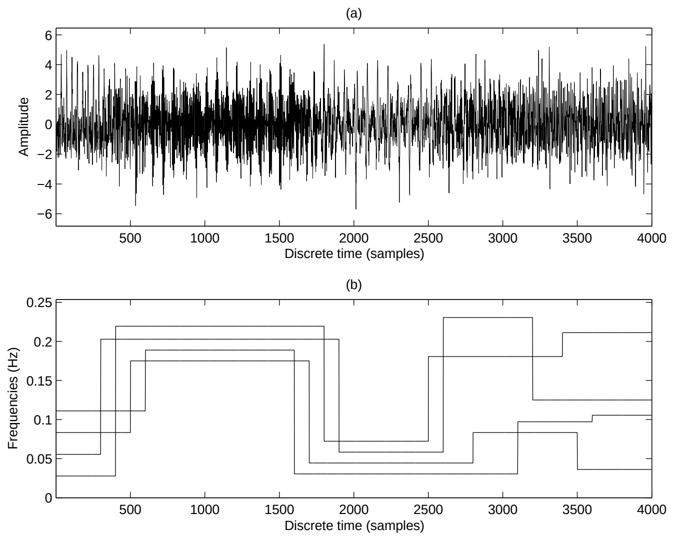

Fig. 1. (a): Test signal; (b): Normalized frequencies of the sinusoids.

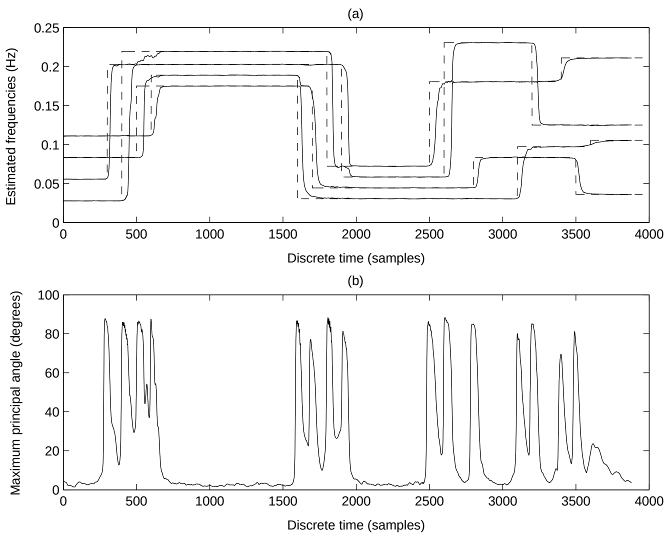

Fig. 3. $O((N+L)r)$ SWASVD3 algorithm: (a): Frequency tracking; (b): Maximum principal angle trajectory.

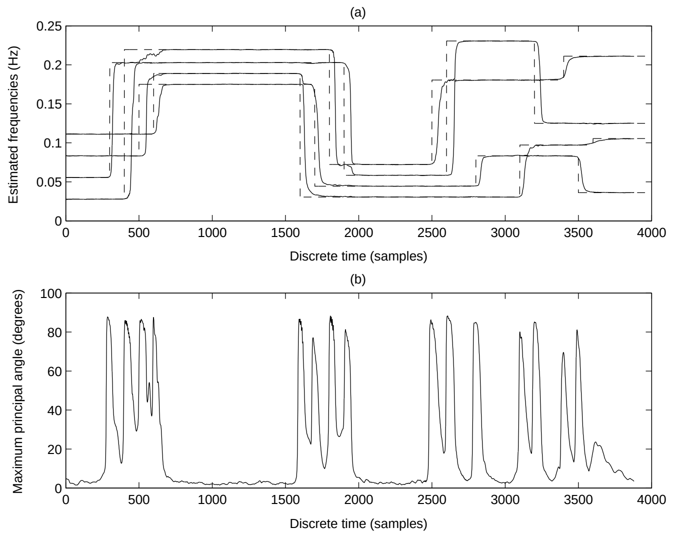

Fig. 2. $O((N+L)r^2)$ SWASVD2 algorithm: (a): Frequency tracking; (b): Maximum principal angle trajectory.

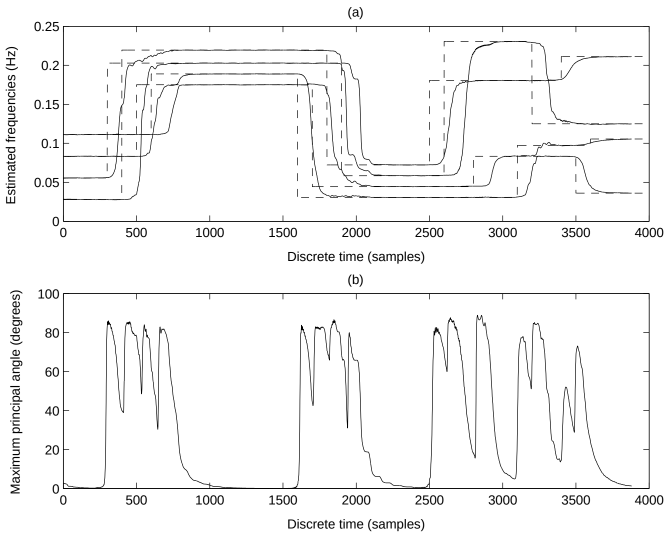

Fig. 4. $O(Nr^2)$ Bi-SVD1 algorithm: (a): Frequency tracking; (b): Maximum principal angle trajectory.

an orthonormal matrix spanning the principal subspace. It does not compute the singular vectors and values of the data matrix (which might be important for rank estimation and tracking), and does not guarantee the orthonormality of the subspace basis *at each time step* (which is required for some subspace-based estimation methods, such as MUSIC [31]).

Finally, SWASVD outperformed all the other subspace trackers that we have tested on the same test signal (Karasalo's algorithm [2], TQR-SVD [5], Loraf [6], Bi-SVD3 [7], NP3 [8], PAST [18], OPAST [20], SHSVD [27] and FAST2 [28]). These results were not presented here to keep the presentation as concise as possible.

V. Conclusions

This paper introduced new SVD tracking algorithms, derived from the classical bi-orthogonal iteration method. These algorithms have been designed for a sliding window data matrix, a characteristic that distinguishes them from most of existing subspace tracking techniques. The results obtained on synthetic signals in the frequency estimation context showed their robustness to abrupt signal variations.

We successfully obtained an ultra-fast tracking algorithm with linear complexity, without degrading the excellent performance of our $O((N+L)r^2)$ subspace tracker. This could be achieved by means of an approximated fast QR factorization.

Finally, these subspace tracking algorithms may be considered as the starting point of a real-time frequency tracker, whose full implementation would additionally require an adaptive version of the ESPRIT algorithm.

Appendix

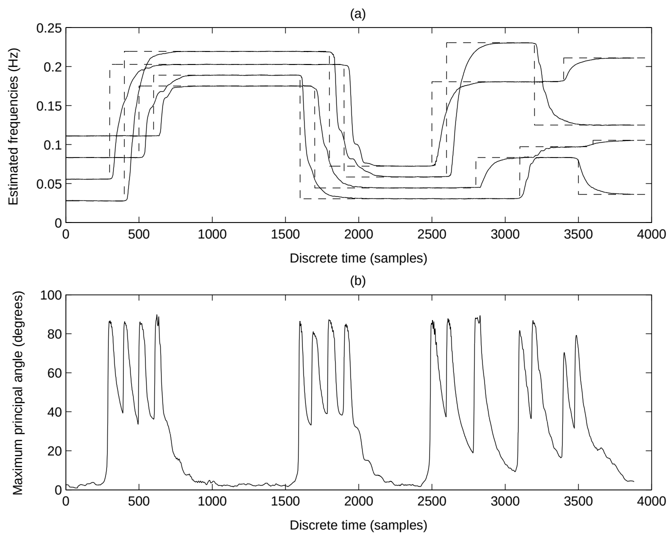

Fig. 5. $O(NLr)$ FAST subspace tracker: (a): Frequency tracking; (b): Maximum principal angle trajectory.

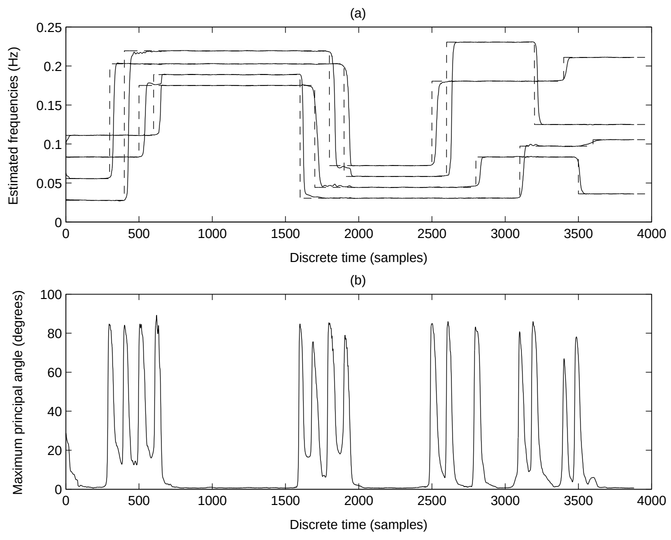

Fig. 6. $O(Nr)$ sliding window NIC subspace tracker: (a): Frequency tracking; (b): Maximum principal angle trajectory.

Ultra-fast SWASVD3 algorithm

This appendix introduces the ultra-fast SWASVD3 tracking algorithm. Since there is no room here for a complete description, only the main steps will be highlighted, and some details required for a full implementation will be skipped.

A. Fast approximated QR factorization

Remember that the first iteration in SWASVD2 relies on the low-dimensional QR factorization (20). Generally, this factorization requires $12r^3$ operations. Now suppose that $\boldsymbol{R}_A(t-1)$ is not only upper triangular, but also diagonal (in practice, this is nearly the case, since $\boldsymbol{R}_A$ converges to the diagonal matrix $\boldsymbol{\Sigma}$ in the original bi-orthogonal iteration SVD algorithm of Table I). In this case, $\boldsymbol{R}_A(t-1)^H$ is also diagonal, and therefore upper triangular, so that $\boldsymbol{T}_B(t)$ defined in equation (19) is an upper triangular plus rank one matrix. In particular, it is well known that the QR factorization of such a matrix can be achieved in $O(r^2)$ computations, using only $2r$ Givens rotations [32, section 12.5]. Therefore, equation (20) can be written

$$\boldsymbol{T}_B(t) = \widetilde{\boldsymbol{G}}_B(t)\widetilde{\boldsymbol{R}}_B(t) \qquad (23)$$

where $\widetilde{\boldsymbol{G}}_B(t)$ is a product of $2r$ Givens rotations and $\widetilde{\boldsymbol{R}}_B(t)$ is a $(r+1)\times r$ upper-triangular matrix (whose last row is equal to $\boldsymbol{0}$ in this particular case).

In practice, $\boldsymbol{R}_A(t-1)$ is not diagonal, and this fast QR-factorization can not be achieved. However, since $\boldsymbol{R}_A(t-1)$ is nearly diagonal, applying the fast QR-factorization technique as it is with this non diagonal matrix gives a *nearly* upper triangular matrix $\widetilde{\boldsymbol{R}}_B(t)$.

This fast approximated QR-factorization is the key step of our ultra-fast tracking algorithm. Note that equation (23) is not an approximation but a strict equality.

B. Modification of the first iteration

TABLE V
Ultra-fast Sliding Window Adaptive SVD algorithm (SWASVD3)

FOR EACH TIME STEP DO :	
First Iteration :	Complexity :
$\boldsymbol{h}(t) = \boldsymbol{Q}_A(t-1)^H\,\boldsymbol{x}(t)$	$8Nr$
$\boldsymbol{z}_\perp(t) = [1,0\dots0]^T - \tilde{\boldsymbol{Q}}_B(t-1)\,\boldsymbol{q}_{B_L}(t-1)$	$8Lr$
$\bar{\boldsymbol{z}}_\perp(t) = \frac{\boldsymbol{z}_\perp(t)}{\\|\boldsymbol{z}_\perp(t)\\|}$	$10L$
$\tilde{\boldsymbol{h}}(t) = \boldsymbol{h}(t) - \boldsymbol{R}_A(t-1)\boldsymbol{q}_{B_L}(t-1)$	$4r^2$
$\boldsymbol{T}_B(t) = \widetilde{\boldsymbol{G}}_B(t)\,\widetilde{\boldsymbol{R}}_B(t)$	$64r^2$
$\boldsymbol{v}(t) = \boldsymbol{R}_A(t-1)^{-1}\boldsymbol{h}(t) - \boldsymbol{q}_{B_L}(t-1)$	$4r^2$
$\mu(t) = \frac{1+\boldsymbol{q}_{B_L}(t-1)^H\boldsymbol{v}(t)}{\\|\boldsymbol{z}_\perp(t)\\|}$	$8r$
$\boldsymbol{w}(t) = \widetilde{\boldsymbol{G}}_B(t)^H\begin{bmatrix} -\boldsymbol{v}(t) \\ \hdashline \mu(t)\end{bmatrix}$	$64r$
$\bar{\boldsymbol{w}}(t) = \frac{1}{\\|\boldsymbol{w}(t)\\|\exp\left(i\,\mathrm{phase}(w_{r+1}(t))\right)}\,\boldsymbol{w}(t)$	$10r$
$\gamma_r(t) = 1$	
for $n = r$ downto 1	$8r$
$\quad s_n(t) = \frac{\bar{w}_n(t)^*}{\gamma_n(t)}$	
$\quad c_n(t) = \sqrt{1-\lvert s_n(t)\rvert^2}$	
$\quad \gamma_{n-1}(t) = \gamma_n(t)\,c_n(t)$	
end	
$\left[\ \boldsymbol{Q}_B(t)\ \vdots\ \times\ \right] = \left[\ \tilde{\boldsymbol{Q}}_B(t-1)\ \vdots\ \bar{\boldsymbol{z}}_\perp(t)\ \right]\boldsymbol{G}_B(t)$	$96Lr$
Second Iteration :	Complexity :
$\boldsymbol{x}_\perp(t) = \boldsymbol{x}(t) - \boldsymbol{Q}_A(t-1)\,\boldsymbol{h}(t)$	$8Nr$
$\bar{\boldsymbol{x}}_\perp(t) = \frac{\boldsymbol{x}_\perp(t)}{\\|\boldsymbol{x}_\perp(t)\\|}$	$10N$
$\widetilde{\boldsymbol{T}}_A(t) = \widetilde{\boldsymbol{G}}_A(t)\,\widetilde{\boldsymbol{R}}_A(t)$	$32r^2$
$\widetilde{\boldsymbol{R}}_A(t)\,\boldsymbol{G}_B(t) = \boldsymbol{G}_{R_A}(t)\left[\begin{array}{c\|c} & \times \\ \boldsymbol{R}_A(t) & \vdots \\ & \times \\ \hline 0\dots0 & \times \end{array}\right]$	$64r^2$
$\left[\ \boldsymbol{Q}_A(t)\ \vdots\ \times\ \right] = \left[\ \boldsymbol{Q}_A(t-1)\ \vdots\ \bar{\boldsymbol{x}}_\perp(t)\ \right]\boldsymbol{G}_A(t)$	$192Nr$

Equation (21) now becomes:

$$\boldsymbol{B}(t) = \left(\left[\; \tilde{\boldsymbol{Q}}_B(t-1) \; \vdots \; \bar{\boldsymbol{z}}_\perp(t) \; \right] \widetilde{\boldsymbol{G}}_B(t) \right) \widetilde{\boldsymbol{R}}_B(t). \tag{24}$$

A new difficulty arises: $\boldsymbol{Q}_B(t)$ can no longer be directly extracted from this factorization (as in equation (22)), since the last row of the nearly upper triangular matrix $\widetilde{\boldsymbol{R}}_B(t)$ is generally not equal to $\boldsymbol{0}$. Consequently, the dimensions of the second member matrices in equation (24) can not be reduced.

Therefore, it will be necessary to explicitly force this last row to be zero. Suppose that there exists a rotation matrix $\boldsymbol{G}_{R_B}(t)^H$ such that the last row of $\boldsymbol{G}_{R_B}(t)^H \widetilde{\boldsymbol{R}}_B(t)$ is equal to $\boldsymbol{0}$. Then let

$$\begin{bmatrix} \boldsymbol{R}_B(t) \\ \hdashline 0 \dots 0 \end{bmatrix} \triangleq \boldsymbol{G}_{R_B}(t)^H \widetilde{\boldsymbol{R}}_B(t).$$

Now equation (22) stands with

$$\boldsymbol{G}_B(t) \triangleq \widetilde{\boldsymbol{G}}_B(t) \boldsymbol{G}_{R_B}(t). \tag{25}$$

Such a matrix $\boldsymbol{G}_{R_B}(t)^H$ will be given in the next section.

C. Choice of an appropriate rotation matrix

First, note that if $\boldsymbol{z}_\perp(t) = \boldsymbol{0}$, the last row of $\widetilde{\boldsymbol{R}}_B(t)$ is $\boldsymbol{0}$. From now on, suppose that $\boldsymbol{z}_\perp(t) \neq \boldsymbol{0}$. A first step towards the obtention of the rotation matrix $\boldsymbol{G}_{R_B}(t)^H$ will be the computation of a unitary vector $\bar{\boldsymbol{w}}(t)$ such that $\bar{\boldsymbol{w}}(t)^H \widetilde{\boldsymbol{R}}_B(t) = [0 \dots 0]$. Consider the r dimensional vector

$$\boldsymbol{v}(t) = \boldsymbol{R}_A(t-1)^{-1} \boldsymbol{h}(t) - \boldsymbol{q}_{B_L}(t-1)$$

and the scalar

$$\mu(t) = \frac{1 + \boldsymbol{q}_{B_L}(t-1)^H \boldsymbol{v}(t)}{\|\boldsymbol{z}_\perp(t)\|}.$$

Then a direct calculation shows that the vector

$$\boldsymbol{w}(t) = \widetilde{\boldsymbol{G}}_B(t)^H \begin{bmatrix} -\boldsymbol{v}(t) \\ \hdashline \mu(t) \end{bmatrix}$$

satisfies the homogenous equation $\boldsymbol{w}(t)^H \widetilde{\boldsymbol{R}}_B(t) = [0 \dots 0]$, and so does the normalized vector[5]

$$\bar{\boldsymbol{w}}(t) \triangleq \frac{1}{\|\boldsymbol{w}(t)\| \exp\left(i \, \text{phase}(w_{r+1}(t))\right)} \boldsymbol{w}(t)$$

The phase shift is chosen so that $\bar{w}_{r+1}(t) \geq 0$ (this choice will be explained below).

Now, we are looking for a rotation matrix $\boldsymbol{G}_{R_B}(t)^H$ whose last row is $\bar{\boldsymbol{w}}(t)^H$ (so that the last row of $\boldsymbol{G}_{R_B}(t)^H \widetilde{\boldsymbol{R}}_B(t)$ is $\boldsymbol{0}$). An appropriate choice for $\boldsymbol{G}_{R_B}(t)^H$ is a product of r Givens rotations[6] as defined in equation (26) on page 9 (because it would be the classical way of zeroing the last row of $\widetilde{\boldsymbol{R}}_B(t)$ if its r first rows had an exact upper-triangular structure).

Then, it can be easily shown that the last row of $\boldsymbol{G}_{R_B}(t)^H$ is equal to

$$[\gamma_1(t)\, s_1(t), \gamma_2(t)\, s_2(t), \dots, \gamma_r(t) s_r(t) \,|\, \gamma_0(t)]$$

where $\gamma_n(t) = \prod_{i=n+1}^{r} c_i(t)$ for $n \in \{1, \dots, r-1\}$ and $\gamma_r(t) = 1$. To make this row equal to $\bar{\boldsymbol{w}}(t)^H$, the coefficients $c_n(t)$ et $s_n(t)$ can be computed recursively:

```
$\gamma_r(t) = 1$
for $n = r$ downto 1
    $s_n(t) = \frac{\bar{w}_n(t)^*}{\gamma_n(t)}$
    $c_n(t) = \sqrt{1 - |s_n(t)|^2}$
    $\gamma_{n-1}(t) = \gamma_n(t)\, c_n(t)$
end
```

Note that all the $c_n(t)$ are non negative numbers, so that $\gamma_0(t) \geq 0$. Therefore, it can be noticed that $\bar{w}_{r+1}(t) \geq 0$ was a necessary condition to guarantee the equality between $\bar{\boldsymbol{w}}(t)^H$ and the last row of $\boldsymbol{G}_{R_B}(t)^H$ (this condition was sufficient because of the orthonormality of both row vectors).

Finally, the matrix $\boldsymbol{G}_B(t)$ defined in equation (25) is expressed as a product of only $2r + r$ Givens rotations. Therefore, $\boldsymbol{Q}_B(t)$ can be computed using equation (22) in only $96Lr$ operations (by recursively applying the Givens rotations). Consequently, the whole first iteration is reduced to linear complexity (see Table V)[7,8].

D. Modification of the second iteration

Contrary to $\boldsymbol{R}_B(t)$, it will now be shown that $\boldsymbol{R}_A(t)$ can be made exactly upper triangular in $O(r^2)$ operations. Indeed, substituting equations (19),(20) into equations (13),(14) shows that $\boldsymbol{R}_A(t)$ satisfies the recurrence

$$\boldsymbol{G}_A(t) \left[\begin{array}{c|c} \boldsymbol{R}_A(t) & \begin{matrix} \times \\ \vdots \\ \times \end{matrix} \\ \hline 0 \dots 0 & \times \end{array} \right] = \widetilde{\boldsymbol{T}}_A(t)\, \boldsymbol{G}_B(t) \tag{27}$$

where

$$\widetilde{\boldsymbol{T}}_A(t) = \left[\begin{array}{c|c} \boldsymbol{R}_A(t-1) & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \;\; \dots \;\; 0 & 0 \end{array} \right] + \begin{bmatrix} \tilde{\boldsymbol{h}}(t) \\ \hdashline \|\boldsymbol{x}_\perp(t)\| \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_{B_L}(t-1) \\ \hdashline \|\boldsymbol{z}_\perp(t)\| \end{bmatrix}^H .$$

It can be noticed that the first member of equation (27) is an exact QR factorization of the second one. Therefore, $\boldsymbol{G}_A(t)$ and $\boldsymbol{R}_A(t)$ can be obtained by computing this QR factorization instead of using equation (14).

Moreover, $\widetilde{\boldsymbol{T}}_A(t)$ is an upper triangular plus rank-one matrix. It is well-known that the QR factorization of such a matrix can be achieved using only $2r$ Givens rotations.

[5]Note that $\boldsymbol{w}(t)$ can not be equal to $\boldsymbol{0}$, since either $\boldsymbol{v}(t) \neq \boldsymbol{0}$, or $\boldsymbol{v}(t) = \boldsymbol{0}$, which yields $\mu(t) \neq 0$.

[6]Note that these Givens rotations are not real but complex transformations. Their orthonormality is guaranteed if

- $c_n(t) \geq 0$,
- $|c_n(t)|^2 + |s_n(t)|^2 = 1$.

[7]Note that the vector $\boldsymbol{R}_A(t-1)^{-1}\boldsymbol{h}(t)$ can be computed in $4r^2$ operations using simple back substitution, since $\boldsymbol{R}_A(t-1)$ is triangular.

[8]The pseudo-code for SWASVD3 in Table V should not be implemented as it is. An efficient implementation should recursively apply all the Givens rotations without storing them in memory.

$$\boldsymbol{G}_{R_B}(t)^H = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & c_r(t) & -s_r(t)^* \\ & & & s_r(t) & c_r(t) \end{bmatrix} \cdots \begin{bmatrix} 1 & & & & & \\ & c_2(t) & & & & -s_2(t)^* \\ & & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & \\ & s_2(t) & & & & c_2(t) \end{bmatrix} \begin{bmatrix} c_1(t) & & & & -s_1(t)^* \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ s_1(t) & & & & c_1(t) \end{bmatrix} \quad (26)$$

Now consider this fast QR factorization:

$$\widetilde{\boldsymbol{T}}_A(t) = \widetilde{\boldsymbol{G}}_A(t)\, \widetilde{\boldsymbol{R}}_A(t).$$

Finally, the QR factorization of $\widetilde{\boldsymbol{R}}_A(t)\, \boldsymbol{G}_B(t)$ gives

$$\widetilde{\boldsymbol{R}}_A(t)\, \boldsymbol{G}_B(t) = \boldsymbol{G}_{R_A}(t) \left[\begin{array}{c|c} & \times \\ \boldsymbol{R}_A(t) & \vdots \\ & \times \\ \hline 0 \dots 0 & \times \end{array} \right] \quad (28)$$

and equations (27) and (16) now stand with

$$\boldsymbol{G}_A(t) \triangleq \widetilde{\boldsymbol{G}}_A(t)\, \boldsymbol{G}_{R_A}(t). \quad (29)$$

Since $\boldsymbol{G}_B(t)$ is a product of $3r$ Givens rotations, it can be shown that the QR factorization in equation (28) can be achieved using only $4r$ Givens rotations[9]. Therefore, the whole QR factorization in equation (27) requires only $2r+4r$ Givens rotations, *i.e.* $O(r^2)$ operations. Then the matrix $\boldsymbol{Q}_A(t)$ can be computed using equation (16) in $O(Nr)$ operations (by recursively applying the Givens rotations). Finally, the whole second iteration is reduced to linear complexity (see Table V).

It can be seen that the dominant cost of SWASVD3 is $104(2N + L)r$. Although this complexity is linear in r, the multiplicative factor is quite high. Therefore, this algorithm is less computationally demanding than SWASVD2 only for high values of r (for instance, if N is much smaller than L, SWASVD3 is faster than SWASVD2 for all $r \geq 12$; in the general case, $r \geq 24$ is a sufficient condition).

ACKNOWLEDGEMENT

The authors would like to thank their colleague P. Weyer-Brown for his helpful comments on the English grammar. They also acknowledge the anonymous reviewers for their constructive comments and corrections.

[9]In particular, $\boldsymbol{G}_B(t)$ is the product of $\widetilde{\boldsymbol{G}}_B(t)$ (which contains $2r$ Givens rotations) and $\boldsymbol{G}_{R_B}(t)$ (which contains r Givens rotations). Consequently, the QR factorization in equation (28) can be achieved in two steps:

- QR factorization of the product of an upper triangular matrix and $\widetilde{\boldsymbol{G}}_B(t)$. It can be readily verified that the upper triangular structure can be recursively maintained (each Givens rotation in $\widetilde{\boldsymbol{G}}_B(t)$ can be compensated by a Givens rotation in $\boldsymbol{G}_{R_A}(t)$).
- QR factorization of the product of an upper triangular matrix and $\boldsymbol{G}_{R_B}(t)$. It must be noticed that such a product is an upper triangular plus rank one matrix. Therefore, the fast QR factorization method presented in [32, section 12.5] can be applied. It involves only $2r$ Givens rotations.

REFERENCES

[1] P. Comon and G. H. Golub, "Tracking a few extreme singular values and vectors in signal processing," in *Proc. IEEE*, vol. 78, Aug. 1990, pp. 1327–1343.

[2] I. Karasalo, "Estimating the covariance matrix by signal subspace averaging," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 34, pp. 8–12, Feb. 1986.

[3] D. J. Rabideau, "Fast, rank adaptive subspace tracking and applications," *IEEE Trans. Signal Processing*, vol. 44, no. 9, pp. 2229–2244, Sept. 1996.

[4] M. Moonen, P. V. Dooren, and J. Vandewalle, "An SVD updating algorithm for subspace tracking," *SIAM J. Matrix Ana. Appl.*, vol. 13, no. 4, pp. 1015–1038, 1992.

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

[6] P. Strobach, "Low-rank adaptive filters," *IEEE Trans. Signal Processing*, vol. 44, no. 12, pp. 2932–2947, Dec. 1996.

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

[8] 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 Processing*, Oct. 1999.

[9] C. H. Bischof and G. M. Shroff, "On updating signal subspaces," *IEEE Trans. Signal Processing*, vol. 40, pp. 96–105, 1992.

[10] G. W. Stewart, "An updating algorithm for subspace tracking," *IEEE Trans. Signal Processing*, vol. 40, pp. 1535–1541, June 1992.

[11] G. Xu, H. Zha, G. H. Golub, and T. Kailath, "Fast algorithms for updating signal subspaces," *IEEE Trans. Circuits Syst.*, vol. 41, no. 8, pp. 537–549, Aug. 1994.

[12] E. Oja, "Neural networks, principal components and subspaces," *Int. journal of neural systems*, vol. 1, no. 1, pp. 61–68, 1989.

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

[14] T. Chen and S. Amari, "Unified stabilization approach to principal and minor components extraction algorithms," *Neural Networks*, vol. 14, no. 10, pp. 1377–1387, 2001.

[15] S. Y. Kung, K. I. Diamantaras, and J. S. Taur, "Adaptive principal component extraction (apex) and applications," *IEEE Trans. Signal Processing*, vol. 43, no. 1, pp. 1202–1217, Jan. 1995.

[16] G. Mathew and V. U. Reddy, "Adaptive estimation of eigensubspace," *IEEE Trans. Signal Processing*, vol. 43, no. 2, pp. 401–411, Feb. 1995.

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

[18] B. Yang, "Projection Approximation Subspace Tracking," *IEEE Trans. Signal Processing*, vol. 44, no. 1, pp. 95–107, Jan. 1995.

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

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

[21] S. C. Douglas, "Numerically-robust adaptive subspace tracking using Householder transformations," in *Proc. of IEEE Sensor Array and Multichannel Signal Proc. Workshop*, 2000, pp. 499 –503.

[22] R. D. DeGroat, "Noniterative subspace tracking," *IEEE Trans. Signal Processing*, vol. 40, no. 3, pp. 571–577, Mar. 1992.

[23] C. Riou and T. Chonavel, "Fast adaptive eigenvalue decomposition: a maximum likelihood approach," in *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Proc.*, 1997, pp. 3565–3568.

[24] C. S. MacInnes, "Fast, accurate subspace tracking using operator restriction analysis," in *Proc. of IEEE Int. Conf. on Acoustic, Speech and Signal Processing*, 1998, pp. 1357–1360.

[25] B. Champagne, “SVD-updating via constrained perturbations with application to subspace tracking,” *Signals, Systems and Computers*, vol. 2, pp. 1379–1385, 1996.
[26] R. Badeau, K. Abed-Meraim, G. Richard, and B. David, “Sliding Window Orthonormal PAST Algorithm,” Apr. 2003, to be published.
[27] P. Strobach, “Square hankel SVD subspace tracking algorithms,” *Signal Processing*, vol. 57, no. 1, pp. 1–18, Feb. 1997.
[28] E. C. Real, D. W. Tufts, and J. W. Cooley, “Two algorithms for fast approximate subspace tracking,” *IEEE Trans. Signal Processing*, vol. 47, no. 7, pp. 1936–1945, July 1999.
[29] M. Clint and A. Jennings, “A simultaneous iteration method for the unsymmetric eigenvalue problem,” *J. Inst. Math. Appl.*, vol. 8, pp. 111–121, 1971.
[30] G. W. Stewart, *Topics in numerical analysis*, 2nd ed. New York: J.J.H. Miller, 1975, pp. 169–185.
[31] R. O. Schmidt, “A signal subspace approach to multiple emitter location and spectral estimation,” Ph.D. dissertation, Stanford University, Nov. 1981.
[32] G. H. Golub and C. F. V. Loan, *Matrix computations*, 3rd ed. Baltimore and London: The Johns Hopkins University Press, 1996.
[33] R. Badeau, R. Boyer, and B. David, “EDS parametric modeling and tracking of audio signals,” in *Proc. Int. Conf. on Digital Audio Effects DAFx-02*, Sept. 2002.
[34] R. Roy and T. Kailath, “ESPRIT-Estimation of Signal Parameters via Rotational Invariance Techniques,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, no. 7, pp. 984–995, July 1989.
[35] P. Strobach, “Fast recursive subspace adaptive ESPRIT algorithms,” *IEEE Trans. Signal Processing*, vol. 46, no. 9, pp. 2413–2430, Sept. 1998.