# GREEDY RLS FOR SPARSE FILTERS

*Bogdan Dumitrescu, Ioan Tăbuş*

Department of Signal Processing
Tampere University of Technology
PO BOX 553, 33101 Tampere, Finland
e-mail: bogdan.dumitrescu@tut.fi, ioan.tabus@tut.fi

**ABSTRACT**

We present an adaptive version of the greedy least squares method for finding a sparse approximate solution, with fixed support size, to an overdetermined linear system. The information updated at each time moment consists of a partial orthogonal triangularization of the system matrix and of partial scalar products of its columns, among them and with the right hand side. Since allowing arbitrary changes of the solution support at each update leads to high computation costs, we have adopted a neighbor permutation strategy that changes at most a position of the support with a new one. Hence, the number of operations is lower than that of the standard RLS. Numerical comparisons with standard RLS in an adaptive FIR identification problem show that the proposed greedy RLS has faster convergence and smaller stationary error.

## 1. INTRODUCTION

The interest in adaptive algorithms dedicated to sparse FIR filters started about a decade ago [6], the main application being echo cancellation. There are at least two types of approaches. The first, illustrated by [9, 10, 12] among others, tries to use techniques that traditionally belong to field of adaptive filters in order to decide what filter coefficients are nonzero and hence should be updated. The second line of attack, to which this paper belongs, uses ideas from the developing topic of sparse approximations. Before reviewing the literature, let us first state the problem. The aim is to find recursive least-squares (RLS) solutions to the overdetermined system

$$\mathbf{A}_t \mathbf{x}_t \approx \mathbf{b}_t \tag{1}$$

where $t \in \mathbb{N}$ is the current time; the matrix $\mathbf{A}_t \in \mathbb{R}^{t\times N}$ and the vector $\mathbf{b}_t$ are given. We want sparse solutions, namely vectors $\mathbf{x}_t$ with at most $M$ nonzero elements, where $M \in \mathbb{N}$ is given, such that, ideally, $\|\mathbf{b}_t - \mathbf{A}_t\mathbf{x}_t\|_2$ is minimized. In a time-varying environment, a forgetting factor $\lambda \leq 1$ is used and, at each time $t$, the given part of (1) is built by

$$\mathbf{A}_t = \begin{bmatrix} \sqrt{\lambda}\cdot\mathbf{A}_{t-1} \\ \mathbf{a}_t^T \end{bmatrix}, \quad \mathbf{b}_t = \begin{bmatrix} \sqrt{\lambda}\cdot\mathbf{b}_{t-1} \\ b_t \end{bmatrix}, \tag{2}$$

the vector $\mathbf{a}_t \in \mathbb{R}^N$ and the scalar $b_t$ depending on data available at time $t$.

For illustration, we consider the standard identification problem of the FIR system (channel) defined by

$$d(t) = \sum_{i=0}^{N-1} \tilde{h}_i u(t-i) + \eta(t), \tag{3}$$

where $u(t)$, $d(t)$, $\eta(t)$ are the input, output and noise signals, respectively; the true coefficients $\tilde{h}_i$ are not available. Given the input and output signals, the aim is to find a best fit model, i.e. the filter $H(z) = \sum_{i=0}^{N-1} h_i z^{-i}$, having at most $M$ nonzero coefficients, that minimizes the RLS criterion

$$J(t) = \sum_{\tau=1}^{t} \lambda^{t-\tau} |e(\tau)|^2, \tag{4}$$

where

$$e(t) = d(t) - \sum_{i=0}^{N-1} h_i u(t-i) \tag{5}$$

is the estimation error. In this context, the data appearing in (2) are

$$\mathbf{a}_t = [u(t)\ u(t-1)\ \ldots\ u(t-N+1)]^T,\ b_t = d(t). \tag{6}$$

In this paper, we present an adaptive version of the greedy LS algorithm [3] (named also forward regression for subset selection) for solving (1). In signal processing literature, the algorithm is known as optimized orthogonal matching pursuit [11]. This algorithm (reviewed in section 2) selects the nonzero elements of the solution one by one, at each step choosing the position that mostly decreases the LS residual; it gives typically better results than matching pursuit (MP) and orthogonal MP (OMP), but has a higher complexity. Adaptive versions of MP and OMP are given in [5] and [8], respectively; however, the latter paper does not gives implementation details, simply stating that (1) is solved via OMP. The alternative to greedy algorithms in the field of sparse approximation is the use of a lasso criterion, in which the (weighted) term $\|\mathbf{h}\|_1$ is added to the criterion (4) in order to force a sparse solution. This is the technique used in [1, 2]. (A similar idea is employed with an LMS criterion in [4, 7].)

The challenge of such algorithms is to have better performance (lower stationary error, shorter adaptation time) than the standard RLS algorithm, with lower computation complexity. The difficulty is that not only the coefficients of the filter have to be adapted, but also its support. Our algorithm (presented in section 3) achieves low complexity by letting the support change with *at most* one position at each time $t$. This constraint is not likely to reduce performance, since anyway the RLS algorithm is unable to react instantly to sudden changes in the channel. Numerical evidence presented in section 4 confirms our claims.

## 2. GREEDY LS ALGORITHM

The greedy least squares algorithm finds a sparse approximate minimizer $\mathbf{x} \in \mathbb{R}^N$ to $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2$, with $\mathbf{A} \in \mathbb{R}^{T\times N}$,

Work supported by Tekes FiDiPro program. B.Dumitrescu is on leave from Department of Automatic Control and Computers, "Politehnica" University of Bucharest.

 ISSN 2076-1465 

by selecting a subset $\mathbf{\Phi}_M$ of $M$ columns of $\mathbf{A}$ (named active columns) and then minimizing $\|\mathbf{b}-\mathbf{\Phi}_M\mathbf{z}\|_2$. The vector $\mathbf{z}$ contains the nonzero elements of $\mathbf{x}$, whose positions are those of the columns from $\mathbf{\Phi}_M$ in $\mathbf{A}$. The algorithm works iteratively, starting with $\mathbf{\Phi}_0=\emptyset$ and, at each stage $k$, adding to $\mathbf{\Phi}_{k-1}$ the column minimizing the residual $\|\mathbf{b}-\mathbf{\Phi}_k\zeta\|_2$, i.e. greedily finds the best LS solution to $\mathbf{\Phi}_k\zeta\approx\mathbf{b}$, given that the first $k-1$ columns of $\mathbf{\Phi}_k$ are those from $\mathbf{\Phi}_{k-1}$. The algorithm can be implemented using basically an orthogonal triangularization with column pivoting, see also [3, sec. 4.3].

*Algorithm Greedy_LS*

Input: $\mathbf{A}\in\mathbb{R}^{T\times N}$, $\mathbf{b}\in\mathbb{R}^T$, $M$

0. $\mathbf{p}=[1\ 2\ \dots\ N]$

1. for $k=1:M$

1.1. $j=\arg\max_{\ell=k:N}|\mathbf{A}(k:T,\ell)^T\mathbf{b}(k:T)|/\|\mathbf{A}(k:T,\ell)\|$

1.2. Swap columns $k$ and $j$ of matrix $\mathbf{A}$. Swap $p_k\leftrightarrow p_j$.

1.3. Find Householder reflector $\mathbf{U}_k$ that zeros the $k$-th column of $\mathbf{A}$ below the diagonal.

1.4. Put $\mathbf{A}\leftarrow\mathbf{U}_k\mathbf{A}$, $\mathbf{b}\leftarrow\mathbf{U}_k\mathbf{b}$.

Output: the solution of the upper triangular system $\mathbf{A}(1:M,1:M)\mathbf{z}=\mathbf{b}(1:M)$. The nonzero elements of the solution are $\mathbf{x}(p(i))=\mathbf{z}(i)$, for $i=1:M$.

At iteration $k$, pivoting in step 1.2 ensures that the active columns are in the first $k$ positions, their initial indices being stored in the first $k$ positions of the permutation vector $\mathbf{p}$; although the algorithm can be implemented without actually permuting the matrix, we use explicit permutations for the sake of simpler presentation. Step 1.1 finds what column should be added to the set of active columns by comparing normalized scalar products between orthogonal projections of the inactive columns of $\mathbf{A}$ on the subspace $(\mathrm{Im}\mathbf{\Phi}_{k-1})^\perp$ (the orthogonal complement of the subspace spanned by the active columns) with the current residual (this step could be implemented more efficiently, but the presented form helps further explanations). Initially, the residual is equal to $\mathbf{b}$. The orthogonal triangularization process ensures that the projections of the inactive columns on $(\mathrm{Im}\mathbf{\Phi}_{k-1})^\perp$ and the current residual are obtained by simply ignoring the first $k-1$ rows of the current $\mathbf{A}$ and $\mathbf{b}$.

## 3. GREEDY RLS

We want now to give a recursive version of the greedy LS algorithm, that, at time $t$, uses in an efficient manner the current information (6) to produce an updated sparse LS solution to system (1). The challenge is that not only the coefficients of the solution may change, but also the support. We stress that, once a support is chosen (in a way that can never be guaranteed to be optimal), we want to compute the exact LS solution with that support.

Let us assume that, before the data available at time $t$ are considered, a greedy LS algorithm has been run on $\mathbf{A}_{t-1}$, $\mathbf{b}_{t-1}$. The algorithm may differ from that presented in section 2 in the selection of the active columns, but is otherwise identical. We denote here $\mathbf{A}$ and $\mathbf{b}$ the output of the algorithm, with

$$\mathbf{A}=\begin{bmatrix}\underbrace{\mathbf{0}}_{M} & \underbrace{\mathbf{R}\\ \mathbf{F}}_{N-M}\end{bmatrix},\quad \mathbf{b}=\begin{bmatrix}\mathbf{c}\\ \mathbf{g}\end{bmatrix}\begin{matrix}\}M\\ \}t-1-M\end{matrix} \tag{7}$$

The (fat) upper triangular matrix $\mathbf{R}\in\mathbb{R}^{M\times N}$ is $\mathbf{A}(1:M,1:N)$ and the vector $\mathbf{c}\in\mathbb{R}^M$ is $\mathbf{b}(1:M)$; they correspond to "present" information. The first $M$ columns of $\mathbf{R}$ are the active ones, while the others are inactive. We assume that a vector $\mathbf{p}$ is available, containing a permutation of $1:N$, the first $M$ positions being the indices of the active columns. The remaining rows of $\mathbf{A}$ and $\mathbf{b}$ represent the "past", which is not stored completely. Instead of $\mathbf{F}$ and $\mathbf{g}$, we have access (and store) only the scalar products of columns (projected on the current $(\mathrm{Im}\mathbf{\Phi}_M)^\perp$) with the residual

$$\mathbf{s}=\mathbf{A}(M+1:t-1,1:N)^T\cdot\mathbf{b}\in\mathbb{R}^N$$

and the scalar products between columns

$$\mathbf{\Psi}=\mathbf{A}(M+1:t-1,1:N)^T\cdot\mathbf{A}(M+1:t-1,1:N).$$

We note that the first $M$ elements of $\mathbf{s}$ are zero (the other being equal to $\mathbf{F}^T\mathbf{g}$) and that $\mathbf{\Psi}$ is a symmetric matrix whose first $M$ rows (and columns) are zero (the lower right $(N-M)\times(N-M)$ block is $\mathbf{F}^T\mathbf{F}$); we will take this into account only when computing updating costs, but not for presentation. So, these scalar products correspond to the past of inactive columns. (The active columns have no past.)

We aim to design a greedy RLS algorithm in which the information that is available and that will be updated at time $t$ consists of the variables $\mathbf{p}$, $\mathbf{R}$, $\mathbf{c}$, $\mathbf{s}$ and $\mathbf{\Psi}$. The nonzero elements of the LS solution at time $t$ are those of the solution of the upper triangular system $\mathbf{R}(1:M,1:M)\mathbf{z}=\mathbf{c}$.

### 3.1 Update without permutation

Let us first discuss the case when the active set is not changed, hence the permutation $\mathbf{p}$ is conserved at time $t$. Similarly to (2), we append a new row to $\mathbf{R}$ and $\mathbf{c}$ by

$$\mathbf{R}\leftarrow\begin{bmatrix}\sqrt{\lambda}\cdot\mathbf{R}\\ \mathbf{a}_t^T(\mathbf{p})\end{bmatrix},\quad \mathbf{c}\leftarrow\begin{bmatrix}\sqrt{\lambda}\cdot\mathbf{c}\\ b_t\end{bmatrix}, \tag{8}$$

where $\mathbf{a}_t(\mathbf{p})$ is the vector $\mathbf{a}_t$ permuted according to $\mathbf{p}$, e.g. the vector from (6) becomes $[\dots\ u(t-p_i+1)\ \dots]_{i=1:N}$. Due to the arrow structure of the matrix, the triangular form can be restored using Givens rotations, applied to both $\mathbf{R}$ and $\mathbf{c}$. Finally, before retaining only the first $M$ rows of $\mathbf{R}$ and $\mathbf{c}$, the last row must be used for updating the inactive scalar products. The complete algorithm is the following.

*Algorithm No_permutation_update*

1. for $k=1:M$

1.1. Compute Givens rotation $\mathbf{G}_k$ that zeros $\mathbf{R}(M+1,k)$.

1.2. Put $\mathbf{R}\leftarrow\mathbf{G}_k\mathbf{R}$, $\mathbf{c}\leftarrow\mathbf{G}_k\mathbf{c}$.

2. Update $\mathbf{s}\leftarrow\lambda\mathbf{s}+\mathbf{R}(M+1,1:N)^T\mathbf{c}(M+1)$

3. $\mathbf{\Psi}\leftarrow\lambda\mathbf{\Psi}+\mathbf{R}(M+1,1:N)^T\mathbf{R}(M+1,1:N)$

4. Delete row $M+1$ of $\mathbf{R}$ and $\mathbf{c}$

The current nonzero coefficients are given by the solution of the upper triangular system $\mathbf{R}(1:M,1:M)\mathbf{z}=\mathbf{c}$. We note that the multiplications in step 1.2 affect only rows $k$ and $M+1$, hence the cost of triangularization is $O(MN)$. The other expensive task is the update in step 3, whose cost is $3/2\cdot(N-M)^2$ operations (an operation is either an addition or a multiplication).

### 3.2 Update with neighbor permutation

If, at time $t$, we allow all columns to compete for the first $M$ positions, the computational costs become very high. The triangular form of $\mathbf{R}$ may be completely destroyed by permutations and its restoration (whose details are not yet clear,

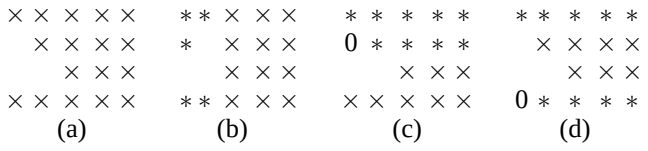

Figure 1: Matrix $\mathbf{R}$ during the first stage of orthogonal triangularization with permutation ($M = 3$, $N = 5$).

since when an inactive column becomes active, the "past" information from the scalar products $\mathbf{s}$ and $\mathbf{\Psi}$ must be taken into account) requires at least $O(M^2N)$ operations. Although not yet discussed, the update of the scalar products $\mathbf{s}$ and $\mathbf{\Psi}$ should also be more complex than in the no permutation case.

However, it is unlikely that the changes in the system structure are very sudden. Even if they are, the RLS algorithm cannot react instantly. So, instead of allowing the permutation $\mathbf{p}$ to change completely at each time $t$, we propose the following strategy:

- for each position $k = 1 : M-1$ we let compete only columns $k$ and $k+1$; so, the permutations that may occur are only between neighbors in the active set;
- for the last active position ($M$), we allow all columns (positions $M$ to $N$) to compete; so, at most a single active column (presumably the one with the least contribution in decreasing the residual) may be replaced by an inactive one.

Moreover, this permutation strategy needs not be applied at each time $t$, but only at multiples of a small integer $\tau_0$; slower changes in the system allow a larger $\tau_0$.

We discuss now the operations required by the neighbor permutations. For illustration, consider the case of the first two columns, detailed in Figure 1. Diagram (a) shows the matrix $\mathbf{R}$ for $M = 3$, $N = 5$, after appending the row $\mu = M+1$ corresponding to the data considered at time $t$; nonzero elements are represented with $\times$. If the first two columns are swapped, the shape (b) is obtained (the nonzero elements affected by the change are denoted by $*$). Triangularization in the first column implies the use of two Givens rotations: the first zeros the element in position $(2,1)$ and produces the matrix (c), while the second zeros element $(\mu,1)$ and produces the desired structure (d). If no permutation is necessary, then one goes directly from (a) to (d) like in algorithm *No_permutation_update*.

The decision to permute two columns of $\mathbf{R}$ is based on on their norms and on their scalar products with $\mathbf{c}$ (see step 1 of algorithm Greedy_LS). This implies only few operations, as only two or three nonzero elements are involved in the computation. So, the cost of the triangularization of the first $M-1$ active columns is at most twice the cost of the triangularization part of algorithm *No_permutation_update*, i.e. still $O(MN)$. The following algorithm results.

*Algorithm Neighbor_permutations*

1. for $k = 1 : M-1$
 1.1. Denote $\rho_1 = [\mathbf{R}(k,k)\ 0\ \mathbf{R}(\mu,k)]^T$
 1.2. $\rho_2 = [\mathbf{R}(k,k+1)\ \mathbf{R}(k+1,k+1)\ \mathbf{R}(\mu,k+1)]^T$
 1.3. $\gamma = [\mathbf{c}(k)\ \mathbf{c}(k+1)\ \mathbf{c}(\mu)]^T$
 1.4. If $|\rho_1^T\gamma|/\|\rho_1\| < |\rho_2^T\gamma|/\|\rho_2\|$
  1.4.1. Swap columns $k$ and $k+1$ of $\mathbf{R}$. Swap $p_k \leftrightarrow p_{k+1}$.
  1.4.2. Compute Givens rot. $\hat{\mathbf{G}}_k$ that zeros $\mathbf{R}(k+1,k)$.
  1.4.3. Put $\mathbf{R} \leftarrow \hat{\mathbf{G}}_k\mathbf{R}$, $\mathbf{c} \leftarrow \hat{\mathbf{G}}_k\mathbf{c}$.
 1.5. Compute Givens rotation $\mathbf{G}_k$ that zeros $\mathbf{R}(\mu,k)$.
 1.6. Put $\mathbf{R} \leftarrow \mathbf{G}_k\mathbf{R}$, $\mathbf{c} \leftarrow \mathbf{G}_k\mathbf{c}$.

### 3.3 Selection of last active column

After the triangularization with neighbor permutation of the first $M-1$ columns, all remaining columns compete for the $M$-th position, the last active one. The best column is that for which the quantity

$$\alpha(\ell) = \frac{|\mathbf{R}(M,\ell)\mathbf{c}(M) + \mathbf{R}(\mu,\ell)\mathbf{c}(\mu) + \mathbf{s}(\ell)|}{\sqrt{\mathbf{R}(M,\ell)\mathbf{R}(M,\ell) + \mathbf{R}(\mu,\ell)\mathbf{R}(\mu,\ell) + \mathbf{\Psi}(\ell,\ell)}} \tag{9}$$

is maximum (with $\ell = M : N$). The numerator accounts for the scalar product of column $\ell$ with the residual. Due to the orthogonalization process in the first $M-1$ columns, the "present" contributes only with rows $M$ and $M+1$ of $\mathbf{R}$, while the "past" part of the product is completely contained in $\mathbf{s}$. Similarly, the denominator of (9) is the squared norm of columns, for which the past is stored on the diagonal of $\mathbf{\Psi}$. The cost of the decision is only $O(N-M)$ operations.

After permuting the best column in position $M$, we have to zero it below the diagonal with orthogonal transformations. Element $\mathbf{R}(M+1,M)$ can be zeroed with a Givens rotation, which affects only the last two rows of $\mathbf{R}$ and $\mathbf{c}$, as in the last iteration of algorithm *No_permutation_update*. However, if an inactive column became active, the whole past must be zeroed. In (7), a column of $\mathbf{F}$ was swapped with column $M$, which was zero. To make it zero, we need a Householder reflector. Let us first review the basic operations associated with a Householder reflector

$$\mathbf{U} = \mathbf{I} - (\mathbf{u}\mathbf{u}^T)/\beta,$$

defined by the vector $\mathbf{u} \in \mathbb{R}^m$ and the scalar $\beta$.

Given a vector $\mathbf{y} \in \mathbb{R}^m$, the reflector that zeros all elements of the vector excepting the first is computed by the operations

1. $\sigma = \text{sgn}(\mathbf{y}(1))\sqrt{\sum_{i=1}^m \mathbf{y}(i)^2} = \text{sgn}(\mathbf{y}(1))\sqrt{\mathbf{y}(1)^2 + \mathbf{y}(2:m)^T\mathbf{y}(2:m)}$
2. $\mathbf{u}(1) = \mathbf{y}(1) + \sigma$, $\mathbf{u}(i) = \mathbf{y}(i)$ for $i = 2 : m$
3. $\beta = \mathbf{u}(1)\sigma$

The first element of $\mathbf{U}\mathbf{y}$ is $-\sigma$ (the norm is conserved by an orthogonal transformation).

If the reflector $\mathbf{U}$ is multiplied with another vector $\mathbf{w}$, then, taking into account that

$$\tilde{\mathbf{w}} = \mathbf{U}\mathbf{w} = \mathbf{w} - (\mathbf{u}^T\mathbf{w}/\beta)\mathbf{u},$$

the computation of $\tilde{\mathbf{w}}$ is performed by

1. $\theta = \mathbf{u}^T\mathbf{w}/\beta = (\mathbf{u}(1)\mathbf{w}(1) + \mathbf{u}(2:m)^T\mathbf{w}(2:m))/\beta$
2. for $i = 1 : m$
 2.1. $\tilde{\mathbf{w}}(i) \leftarrow \mathbf{w}(i) - \theta\mathbf{u}(i)$

Returning to our problem, we work with (virtual) vectors of unknown size, whose first element is that on row $M$ of $\mathbf{R}$ (or $\mathbf{c}$), which is known, and whose other elements are columns of $\mathbf{F}$ (or $\mathbf{g}$), which are not directly available. The vector $\mathbf{y}$ corresponds to column $M$ and $\mathbf{w}$ to any of the inactive columns (or the right hand side of the linear system). We note from both above algorithms that the computation of the reflector and of the value $\tilde{\mathbf{w}}(1)$ depend only on scalar products involving the unknown part (indices $2 : m$) of the vectors $\mathbf{u}$ (equal to $\mathbf{y}$ for these indices) and $\mathbf{w}$. Hence, it is possible to compute the Householder reflector zeroing the past of

the new $M$-th column and the element $\tilde{\mathbf{w}}(1)$ using only the information from the scalar products $\mathbf{\Psi}$ and $\mathbf{s}$.

Finally, we have to update these scalar products. Let $\tilde{\mathbf{w}}_1 = \mathbf{U}\mathbf{w}_1$, $\tilde{\mathbf{w}}_2 = \mathbf{U}\mathbf{w}_2$. Since multiplication with an orthogonal matrix conserves the scalar products, it results that

$$\tilde{\mathbf{w}}_1(2:m)^T\tilde{\mathbf{w}}_2(2:m) = \mathbf{w}_1(2:m)^T\mathbf{w}_2(2:m) + \mathbf{w}_1(1)\mathbf{w}_2(1) - \tilde{\mathbf{w}}_1(1)\tilde{\mathbf{w}}_2(1) \quad (10)$$

and hence the update of the scalar products can be computed. Aggregating all information developed in this section, we obtain the following algorithm.

*Algorithm Last_active_column*

1. Put $\mathbf{s} \leftarrow \lambda\mathbf{s}$, $\mathbf{\Psi} \leftarrow \lambda\mathbf{\Psi}$
2. $j = \arg\max_{\ell=M:N} \alpha(\ell)$, see (9)
3. Swap columns $M$ and $j$ of $\mathbf{R}$. Swap $p_M \leftrightarrow p_j$, $\mathbf{s}(M) \leftrightarrow \mathbf{s}(j)$. Swap rows and columns $M$ and $j$ of $\mathbf{\Psi}$.
4. Compute Givens rotation $\mathbf{G}$ that zeros $\mathbf{R}(\mu, M)$.
5. Put $\mathbf{R} \leftarrow \mathbf{G}\mathbf{R}$, $\mathbf{c} \leftarrow \mathbf{G}\mathbf{c}$.
6. $\sigma = \mathrm{sgn}(\mathbf{R}(M,M))\sqrt{\mathbf{R}(M,M)^2 + \mathbf{\Psi}(M,M)}$
7. $\mathbf{u}(1) = \mathbf{R}(M,M) + \sigma$, $\beta = \mathbf{u}(1)\sigma$
8. Save $\mathbf{v}^T = \mathbf{R}(M, 1:N)$, $\gamma = \mathbf{c}(M)$
9. $\mathbf{R}(M,M) \leftarrow -\sigma$
10. for $k = M+1 : N$
10.1. $\theta = (\mathbf{u}(1)\mathbf{R}(M,k) + \mathbf{\Psi}(M,k))/\beta$
10.2. $\mathbf{R}(M,k) \leftarrow \mathbf{R}(M,k) - \theta\mathbf{u}(1)$
11. $\theta = (\mathbf{u}(1)\mathbf{c}(M) + \mathbf{s}(M))/\beta$
12. $\mathbf{c}(M) \leftarrow \mathbf{c}(M) - \theta\mathbf{u}(1)$
13. $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{v}\gamma - \mathbf{R}(M, 1:N)\mathbf{c}(M) + \mathbf{R}(\mu, 1:N)\mathbf{c}(\mu)$
14. $\mathbf{\Psi} \leftarrow \mathbf{\Psi} + \mathbf{v}\mathbf{v}^T - \mathbf{R}(M, 1:N)\mathbf{R}(M, 1:N)^T + \mathbf{R}(\mu, 1:N)\mathbf{R}(\mu, 1:N)^T$
15. Delete row $\mu$ of $\mathbf{R}$ and $\mathbf{c}$

The most complex operation is the update of the scalar products, requiring about $7/2(N-M)^2$ operations.

### 3.4 Initialization

Adding a regularization term $\delta\lambda^t\|\mathbf{h}\|_2^2$ to the criterion (4) amounts to the initialization $\mathbf{A}_0 = \sqrt{\delta}\mathbf{I}_N$, $\mathbf{b}_0 = \mathbf{0}_{N\times 1}$. Hence, we initialize the diagonal elements of $\mathbf{R}$ with $\sqrt{\delta}$ and the lower $N-M$ diagonal elements of $\mathbf{\Psi}$ with $\delta$, while $\mathbf{c}$ and $\mathbf{s}$ are initialized with zero.

### 3.5 Review of the complete algorithm

The input parameters of the algorithm are the maximum degree of the filter $N$, the maximum number of nonzero coefficients $M$, the forgetting factor $\lambda$, the permutation update lag $\tau_0$ and the regularization constant $\delta$. The variables $\mathbf{R}$, $\mathbf{c}$, $\mathbf{\Psi}$ and $\mathbf{s}$ are initialized as above and $\mathbf{p} = [1\ 2\ \ldots\ N]$. The operations performed at time $t$ are the following.

First, form (8). If $t$ is not a multiple of $\tau_0$, no permutation is performed and hence algorithm *No_permutation_update* is run. If $t$ is a multiple of $\tau_0$, the algorithms *Neighbor_permutations* and *Last_active_column* are run in succession.

The number of operations is

$$\left(\frac{3}{2} + \frac{2}{\tau_0}\right)(N-M)^2 + O(MN).$$

For comparison, the standard RLS algorithm requires about $6N^2$ operations for a filter of length $N$.

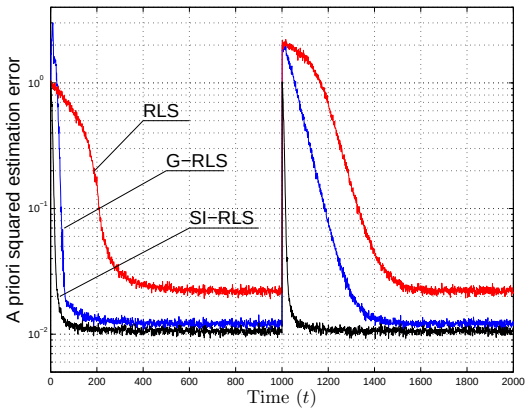


Figure 2: A priori squared estimation error for $M = 12$.

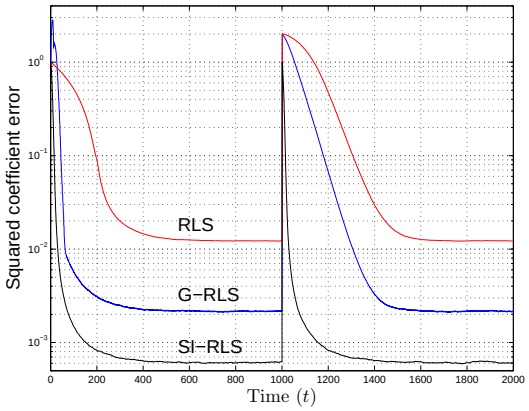


Figure 3: Squared coefficient error for $M = 12$.

### 3.6 Further algorithmic developments

Until now we have considered the number $M$ of nonzero coefficients to be fixed. However, it can be easily increased or decreased by 1 at each time $t$. Decreasing $M$ can be implemented by putting column $M$ in the inactive set, after the permutations of the active columns; row $M$ of $\mathbf{R}$ and $\mathbf{c}$ is used to update the scalar products $\mathbf{\Psi}$ and $\mathbf{s}$ and then deleted. Increasing $M$ is done by running an algorithm in the style of *Last_active_column* for the selection of the $(M+1)$-th column, which is then added to the active set; row $M+1$ is appended to $\mathbf{R}$ and $\mathbf{c}$. However, the difficulty lies in a method to decide when $M$ should be changed. The decision can be based on the norms of the residuals resulting from the different choices of $M$, which can be computed easily in the context of the proposed algorithm (although not detailed here). These issues are left for further research.

## 4. NUMERICAL RESULTS

To test the performance of the proposed algorithm, we have used the following setup. The input-output data from (3) are generated with $u(t) \in \mathcal{N}(0,1)$ and $\eta(t) \in \mathcal{N}(0,\sigma^2)$; we report for $\sigma^2 = 0.01$, but similar results were obtained for other values; the filter $\tilde{H}(z)$ has a length $N = 200$ and $\tilde{M} = 6$ nonzero coefficients, in randomly chosen (uniformly distributed) positions; the coefficients are generated from $\mathcal{N}(0,1)$, then their vector normed to 1. The first 1000 out-

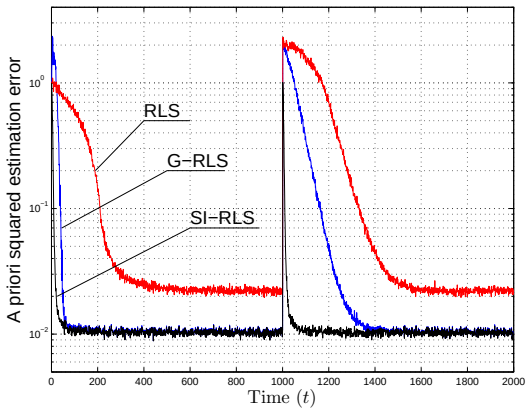


Figure 4: A priori squared estimation error for $M = 6$.

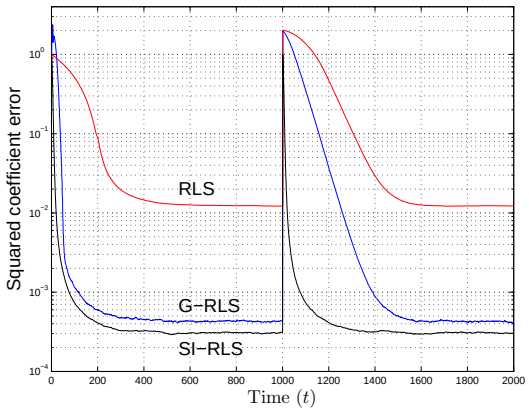


Figure 5: Squared coefficient error for $M = 6$.

put samples are generated with a filter, the next 1000 with another one; this corresponds to a sudden change in the channel. We report results for 1000 runs of the algorithms (different filters generated at each run). We compare three RLS algorithms: our greedy algorithm (G-RLS), the standard one with full support of size $N$ (unaware of sparsity, named simply RLS) and the standard one on a support of size $M$ containing the true $\tilde{M}$ positions and other $M - \tilde{M}$ random ones (named SI-RLS, *sparsity informed RLS*). The forgetting factor is $\lambda = 0.99$ for all algorithms.

We take first $M = 12$, i.e. a value that is larger than the true one. The a priori squared output estimation error (5) (averaged over the 1000 runs) is shown in Figure 2. It is visible that G-RLS converges more quickly than RLS and has a better stationary error (hence better misadjustment), closer to that of SI-RLS. The squared estimation error averaged over the last 100 samples is $2.22 \cdot 10^{-2}$ for RLS, $1.22 \cdot 10^{-2}$ for G-RLS and $1.07 \cdot 10^{-2}$ for SI-RLS; this translates into a gain of about 5.25 dB of G-RLS over RLS. G-RLS was run with $\delta = 0.5$; larger values tend to slow the convergence, while smaller values tend to increase the error in the first samples. The time lag $\tau_0$ for performing permutations was $\tau_0 = 2$; increasing $\tau_0$ to 5 only slightly slows the convergence; for $\tau_0 = 10$, the G-RLS algorithm still tracks faster than RLS. Figure 3 shows the mean square error of the coefficient vector $\|\mathbf{h} - \tilde{\mathbf{h}}\|^2$ (where $\tilde{\mathbf{h}} \in \mathbb{R}^N$ is the vector of true filter coefficients, with $\|\tilde{\mathbf{h}}\| = 1$, and $\mathbf{h}$ the estimated one, with zeros in the inactive $N - M$ positions for G-RLS and SI-RLS).

A second round of experiments was performed changing only the support size to $M = 6$, i.e. to the true value. The squared estimation and coefficient errors are shown in Figures 4 and 5. The squared estimation error averaged over the last 100 samples is $1.04 \cdot 10^{-2}$ for G-RLS and $1.03 \cdot 10^{-2}$ for SI-RLS. The performance of G-RLS is much closer to that of SI-RLS, compared to the case $M = 6$. Occasionally, G-RLS loses track of the true support, which explains the worse steady-state behavior compared with SI-RLS. This happens only if the filter has some small coefficients. Imposing a sufficiently large threshold on the coefficients, e.g. $|\tilde{h}_i| \geq 0.05$, leads to identical steady-state responses of G-RLS and SI-RLS.

## REFERENCES


[1] D. Angelosante and G.B. Giannakis. RLS-Weighted Lasso for Adaptive Estimation of Sparse Signals. In *Int. Conf. Acoustics, Speech, Signal Proc.*, pages 3245–3248, 2009.

[2] B. Babadi, N. Kalouptsidis, and V. Tarokh. Comparison of SPARLS and RLS Algorithms for Adaptive Filtering. In *IEEE Sarnoff Symp.*, 2009.

[3] S. Chen, S.A. Billings, and W. Luo. Orthogonal Least Squares Methods and Their Application to Non-Linear System Identification. *Int. J. Control*, 50(5):1873–1896, 1989.

[4] Y. Chen, Y. Gu, and A.O. Hero III. Sparse LMS for System Identification. In *Int. Conf. Acoustics, Speech, Signal Proc.*, pages 3125–3128, 2009.

[5] S.F. Cotter and B.D. Rao. The Adaptive Matching Pursuit Algorithm for Estimation and Equalization of Sparse Time-Varying Channels. In *34th Asilomar Conf. Sign. Syst. Comp.*, volume 2, pages 1772–1776, 2000.

[6] D.L. Duttweiler. Proportionate Normalized Least-Mean-Squares Adaptation in Echo Cancelers. *IEEE Trans. Speech Audio Proc.*, 8(5):508–518, Sept. 2000.

[7] Y. Gu, J. Jin, and S. Mei. $\ell_0$ Norm Constraint LMS Algorithm for Sparse System Identification. *IEEE Signal Proc. Letters*, 16(9):774–777, Sept. 2009.

[8] G.Z. Karabulut and A. Yongacoglu. Estimation of Time-Varying Channels with Orthogonal Matching Pursuit Algorithm. In *Symp. Adv. Wired Wireless Comm.*, pages 141–144, 2005.

[9] R.K. Martin, W.A. Sethares, R.C. Williamson, and C.R. Johnson, Jr. Exploiting Sparsity in Adaptive Filters. *IEEE Trans. Signal Proc.*, 50(8):1883–1894, Aug. 2002.

[10] P.A. Naylor, J. Cui, and M. Brookes. Adaptive Algorithms for Sparse Echo Cancellation. *Signal Proc.*, 86(6):1182–1192, 2006.

[11] L. Rebollo-Neira and D. Lowe. Optimized Orthogonal Matching Pursuit Approach. *IEEE Signal Proc. Letters*, 9(4):137–140, April 2002.

[12] L.R. Vega, H. Rey, J. Benesty, and S. Tressens. A Family of Robust Algorithms Exploiting Sparsity in Adaptive Filters. *IEEE Trans. Audio Speech Lang. Proc.*, 17(4):572–581, May 2009.