DCD-RLS Adaptive Filters With Penalties for Sparse Identification

Yuriy V. Zakharov, *Senior Member, IEEE*, and Vítor H. Nascimento, *Senior Member, IEEE*

***Abstract*—In this paper, we propose a family of low-complexity adaptive filtering algorithms based on dichotomous coordinate descent (DCD) iterations for identification of sparse systems. The proposed algorithms are appealing for practical designs as they operate at the bit level, resulting in stable hardware implementations. We introduce a general approach for developing adaptive filters with different penalties and specify it for exponential and sliding window RLS. We then propose low-complexity DCD-based RLS adaptive filters with the lasso, ridge-regression, elastic net, and ℓ_0 penalties that attract sparsity. We also propose a simple recursive reweighting of the penalties and incorporate the reweighting into the proposed adaptive algorithms to further improve the performance. For general regressors, the proposed algorithms have a complexity of $\mathcal{O}(N^2)$ operations per sample, where N is the filter length. For transversal adaptive filters, the algorithms require only $\mathcal{O}(N)$ operations per sample. A unique feature of the proposed algorithms is that they are well suited for implementation in finite precision, e.g., on FPGAs. We demonstrate by simulation that the proposed algorithms have performance close to the oracle RLS performance.**

***Index Terms*—Adaptive filter, dichotomous coordinate descent (DCD), DCD algorithm, FPGA, penalty function, reweighting, RLS, sparse representation.**

I. Introduction

THERE is significant interest in developing adaptive filtering algorithms that can deal with sparse recovery problems (see [1]–[6] and references therein). They are often associated with adaptive identification of linear systems with sparse impulse responses [7]–[11], but can also be useful for other applications. Sparse adaptive filters often solve least squares (LS) optimization problems with sparsity-attracting penalties. E.g., adaptive algorithms in [1], [2], [5] use different techniques for solving LS problems with the ℓ_1-norm (lasso [12]) penalty. In [5], [13], an approximation to the ℓ_0-norm penalty is used.

Manuscript received May 28, 2012; revised October 05, 2012 and January 19, 2013; accepted March 15, 2013. Date of publication April 16, 2013; date of current version May 22, 2013. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Konstantinos Slavakis. The work of Y. V. Zakharov was supported in part by the FAPESP grants 2011/06994-2 and 2012/50565-1. The work of V. H. Nascimento was supported in part by CNPq and FAPESP grants 303921/2010-2, 2011/06994-2, and 2012/50565-1.

Y. V. Zakharov is with the Department of Electronics, University of York, Heslington, York Y010 5DD, U.K. (e–mail: yz1@ohm.york.ac.uk).

V. H. Nascimento is with the Department of Electronic Systems Engineering, University of São Paulo, São Paulo 05508–970, Brazil (e–mail: vitor@lps.usp.br).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2013.2258340

There is also interest in real-time implementation of sparse recovery techniques, particularly on FPGAs [14]–[18]. However, many algorithms capable of providing a high recovery performance are not well suited to such implementation due to high complexity and high numerical-precision requirements [18]. Therefore, only a few (nonadaptive) algorithms that mostly belong to the greedy family have been implemented in hardware on FPGAs [14], [16]–[19].

The coordinate descent (CD) search has an inherent property of having low complexity when used for sparse recovery [12], [20]–[23]. The CD search is used in [1] for sparse RLS adaptive filtering. The dichotomous CD (DCD) search is especially well suited to real-time implementation, e.g., using FPGAs [24]–[27], and it was intensively used for adaptive filtering [24], [27]–[30]. In [24], a general approach was proposed for developing RLS adaptive filters with exponential and sliding windows; when combined with DCD iterations, it resulted in algorithms with a performance close to the RLS performance and yet having as low complexity as $\mathcal{O}(N)$ operations per sample. Since the algorithms do not directly propagate the inverse of the regressor autocorrelation matrix, they are stable and well suited to implementation in finite precision [29], [31]. However, this approach has only been exploited in application to purely-RLS adaptive filters [24] or RLS with diagonal loading [29]; that is, algorithms based on the standard LS cost function with at most quadratic regularization. In this paper, we extend these results to include sparsity-promoting penalties, thereby obtaining fast, stable, and low-cost adaptive filters suitable for hardware implementation.

When dealing with sparse recovery, *a priori* information on the support can significantly improve the recovery performance. If the support is perfectly known, an algorithm achieves the so-called *oracle* performance. However, such knowledge is most often unavailable. Techniques have been previously proposed for estimating and further refining the information on the support in a set of *reweighting* iterations and incorporating these estimates in the cost function in the form of a weight vector for the penalty [32]–[35]. The penalty reweighting has also been used in sparse adaptive filtering [4], [36]. Proportionate adaptive filters are also based on reweighting and they demonstrate improved performance when identifying sparse systems [37]. We also take advantage of reweighting techniques to improve the performance of our algorithms.

The contributions of this paper are as follows:

1. We present a general framework for developing adaptive filtering algorithms with different LS criteria and different penalty functions, in particular, sparsity-attracting penalties, such as lasso, elastic net, and ℓ_0 penalties.
2. We specify this framework for the exponential and rectangular sliding windows. These two adaptive filter struc-

tures have previously been proposed for the RLS algorithm without regularization [24] and the exponential RLS with ℓ_2-regularization [29]. Here we extend these schemes to arbitrary separable penalty functions.

3. We propose a simple and yet efficient reweighting recursion for updating penalties in adaptive filtering.
4. We propose the use of DCD iterations for solving the LS problems with penalties and arrive at a universal DCD-solver that can efficiently exploit a solution found for the previous sample as a *warm-start* for the current sample. As a result, we arrive at numerically stable adaptive filters with as low complexity as $\mathcal{O}(N^2)$ operations per sample for general regressors and $\mathcal{O}(N)$ operations per sample for (transversal) regressors with time-shifted structure.
5. We investigate the proposed algorithms by simulation and present results that show that the algorithms outperform advanced sparse adaptive algorithms and perform close to the oracle RLS algorithm.

The paper is organized as follows. Section II presents the adaptive filtering setup. In Section III, we present the general framework for developing adaptive filters with different penalty functions and describe exponential and sliding window cases. In Section IV, we describe the DCD algorithm for solving LS problems with penalties. Section V introduces the ridge-regression, lasso, modified lasso, elastic-net, and ℓ_0 penalty functions. Section VI presents simulation results and Section VII gives conclusions.

Notations: We use capital and small bold fonts to denote matrices and vectors, respectively; e.g., $\mathbf{X}$ is a matrix and $\mathbf{x}$ a vector. Elements of the matrix and vector are denoted as $X_{n,p}$ and x_n, respectively. We also denote: $\mathbf{R}^{(q)}$ the qth column of $\mathbf{R}$; $\mathbf{X}^H$, Hermitian transpose of $\mathbf{X}$; $\mathbf{I}_M$, $M \times M$ identity matrix; $\mathbf{0}_{P\times Q}$, $P\times Q$ matrix with zero entries; $\mathbf{0}_N$ and $\mathbf{1}_N$ are N-length vectors of zeros and ones, respectively; $\Re\{\cdot\}$ and $\Im\{\cdot\}$, the real and imaginary part of a complex number, respectively; $(\cdot)^*$ denotes the complex conjugate.

II. Regularized Adaptive Filtering Setup

We will consider adaptive filters with the task of finding a complex-valued $N \times 1$ vector $\mathbf{h}(n)$ that, at every time instant n, minimizes the cost function

$$\bar{J}[\mathbf{h}(n)] = \bar{f}_{\rm LS}[\mathbf{h}(n)] + f_{\rm p}[\mathbf{h}(n)]. \tag{1}$$

The first term in (1) is the LS error of the solution $\mathbf{h}(n)$ and the second term is a penalty function that incorporates *a priori* information on the true solution. These two terms will be different for different scenarios. E.g., if the true solution is sparse, we may want to use the second term in the form $f_{\rm p}[\mathbf{h}(n)] = \tau\|\mathbf{h}(n)\|_1$ for some positive τ.

Let complex-valued $\mathbf{x}(n)$ and $d(n)$ be an $N \times 1$ regressor vector and desired signal, respectively, at time instant n. We denote

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^H(1) \\ \cdots \\ \mathbf{x}^H(n) \end{pmatrix} \text{ and } \mathbf{d} = \begin{pmatrix} d^*(1) \\ \cdots \\ d^*(n) \end{pmatrix} \tag{2}$$

the $n\times N$ matrix of the regressor data and $n\times 1$ vector of the desired signal, respectively. In many adaptive filtering scenarios, the first term of the cost function in (1) can be expressed in the following form [2], [4]:

$$\bar{f}_{\rm LS}[\mathbf{h}(n)] = \frac{1}{2}\|\mathbf{D}^{1/2}(n)[\mathbf{d}(n) - \mathbf{X}(n)\mathbf{h}(n)]\|_2^2, \tag{3}$$

where $\mathbf{D}^{1/2}(n)$ is an $n \times n$ matrix. We define $\mathbf{D}(n) = [\mathbf{D}^{1/2}(n)]^H\mathbf{D}^{1/2}(n)$ and obtain

$$\bar{f}_{\rm LS}[\mathbf{h}(n)] = \frac{1}{2}[\mathbf{d}(n) - \mathbf{X}(n)\mathbf{h}(n)]^H\mathbf{D}(n)[\mathbf{d}(n) - \mathbf{X}(n)\mathbf{h}(n)],$$

which can be represented as

$$\bar{f}_{\rm LS}[\mathbf{h}(n)] = \frac{1}{2}\mathbf{d}^H(n)\mathbf{D}(n)\mathbf{d}(n) + f_{\rm LS}[\mathbf{h}(n)], \tag{4}$$

where $f_{\rm LS}[\mathbf{h}(n)] = \frac{1}{2}\mathbf{h}^H(n)\mathbf{R}(n)\mathbf{h}(n) - \Re\{\mathbf{h}^H(n)\mathbf{b}(n)\}$, $\mathbf{R}(n) = \mathbf{X}^H(n)\mathbf{D}(n)\mathbf{X}(n)$ and $\mathbf{b}(n) = \mathbf{X}^H(n)\mathbf{D}(n)\mathbf{d}(n)$. As the first term in (4) does not depend on the unknown vector $\mathbf{h}(n)$, we will obtain the same result if minimizing the cost function

$$J[\mathbf{h}(n)] = f_{\rm LS}[\mathbf{h}(n)] + f_{\rm p}[\mathbf{h}(n)]. \tag{5}$$

There are two important cases of the matrix $\mathbf{D}(n)$. The first one is when an exponential window is used for computing the matrix $\mathbf{R}(n)$ and vector $\mathbf{b}(n)$, similarly to what is done in the classical RLS algorithm [38], [39]. In this case, we have

$$\mathbf{D}(n) = \text{diag}\,[\lambda^{n-1}, \lambda^{n-2}, \ldots, \lambda, 1], \tag{6}$$

where λ is the forgetting factor, $\lambda \in (0, 1]$. The other one is when a sliding window is used, in which case we have

$$\mathbf{D}(n) = \begin{pmatrix} \mathbf{0}_{(n-M)\times(n-M)} & \mathbf{0}_{(n-M)\times M} \\ \mathbf{0}_{M\times(n-M)} & \mathbf{I}_M \end{pmatrix}, \tag{7}$$

where M is the length of the sliding window.

III. Framework for Developing Adaptive Filters With General Penalty Functions

We are dealing with a sequence of optimization problems described as

$$\min_{\mathbf{h}(n)} J[\mathbf{h}(n)], \quad n > 0. \tag{8}$$

We can use a straightforward approach for solving this sequence of problems by treating each of them independently. This, however, would result in high complexity. Instead, we want to solve the n-th optimization problem using as much information from the $(n-1)$-th solution as possible to reduce the complexity. Let at time instant n a system of equations $\mathbf{R}(n)\mathbf{h}(n) = \mathbf{b}(n)$ be approximately solved and the approximate solution be $\hat{\mathbf{h}}(n)$. Denote

$$\mathbf{c}(n|m) = \mathbf{b}(n) - \mathbf{R}(n)\hat{\mathbf{h}}(m) \tag{9}$$

a residual vector for this solution. The notation $\mathbf{c}(n|m)$ indicates that the residual vector in (9) corresponds to the system matrix $\mathbf{R}(n)$ and the right-hand vector $\mathbf{b}(n)$ at time instant n, whereas the solution $\hat{\mathbf{h}}(m)$ corresponds to the system $\mathbf{R}(m)\mathbf{h}(m) =$

$\mathbf{b}(m)$ at time instant m. In iterative methods, such as line search methods [40], the residual vector $\mathbf{c}(n-1|n-1)$ is often available. We introduce the following notation:

$$\begin{cases} \Delta\mathbf{R}(n) = \mathbf{R}(n) - \mathbf{R}(n-1) \\ \Delta\mathbf{b}(n) = \mathbf{b}(n) - \mathbf{b}(n-1) \\ \Delta\mathbf{h}(n) = \mathbf{h}(n) - \hat{\mathbf{h}}(n-1) \end{cases} \tag{10}$$

Note that the last line in (10) shows the increment of the n-th solution, which we want to find, with respect to the solution obtained for the $(n-1)$-th problem.

For solving the n-th problem, it is possible to use the $(n-1)$-th solution as a *warm-start*. In order to exploit the previous solution $\hat{\mathbf{h}}(n-1)$ as a *warm-start*, it turns out that we also need to obtain a new residual vector, taking into account the variation of the matrix $\mathbf{R}(n)$ and vector $\mathbf{b}(n)$ (see Section IV and Table III). More specifically, we need the following residual vector

$$\mathbf{c}(n|n-1) = \mathbf{b}(n) - \mathbf{R}(n)\hat{\mathbf{h}}(n-1). \tag{11}$$

Using the notation from (10), we can rewrite (11) as

$$\mathbf{c}(n|n-1) = [\mathbf{b}(n-1) + \Delta\mathbf{b}(n)] - [\mathbf{R}(n-1) + \Delta\mathbf{R}(n)]\hat{\mathbf{h}}(n-1)$$

and obtain

$$\mathbf{c}(n|n-1) = \mathbf{c}(n-1|n-1) + \Delta\mathbf{b}(n) - \Delta\mathbf{R}(n)\hat{\mathbf{h}}(n-1). \tag{12}$$

If computation in (12) can be done with low complexity, we would obtain a good *warm-start* for solving the n-th problem, for which we need to find $\Delta\mathbf{h}(n)$.

Thus, we need to transform the cost function in (5) in another cost function that should be minimized at instant n:

$$\min_{\Delta\mathbf{h}(n)} J_{\Delta}[\Delta\mathbf{h}(n)]. \tag{13}$$

As a result, we will obtain an (approximate) solution $\Delta\hat{\mathbf{h}}(n)$ to (13), and the final solution to (5) will be given by $\hat{\mathbf{h}}(n) = \hat{\mathbf{h}}(n-1) + \Delta\hat{\mathbf{h}}(n)$. The cost function in (13) can be obtained from

$$J[\mathbf{h}(n)] = f_{\mathrm{LS}}[\hat{\mathbf{h}}(n-1) + \Delta\mathbf{h}(n)] + f_{\mathrm{p}}[\hat{\mathbf{h}}(n-1) + \Delta\mathbf{h}(n)] \tag{14}$$

that follows from (5) and (10). We assume that $\hat{\mathbf{h}}(n-1)$ is fixed and consider the cost function in (14) as a function of $\Delta\mathbf{h}(n)$. The first term in (14) can be written as

$$\begin{aligned} & f_{\mathrm{LS}}[\hat{\mathbf{h}}(n-1) + \Delta\mathbf{h}(n)] \\ &= \frac{1}{2}\hat{\mathbf{h}}^H(n-1)\mathbf{R}(n)\hat{\mathbf{h}}(n-1) \\ &\quad - \Re\{\hat{\mathbf{h}}^H(n-1)\mathbf{b}(n)\} \qquad (15) \\ &\quad + \frac{1}{2}\Delta\mathbf{h}^H(n)\mathbf{R}(n)\Delta\mathbf{h}(n) \\ &\quad - \Re\{\Delta\mathbf{h}^H(n)[\mathbf{b}(n) - \mathbf{R}(n)\hat{\mathbf{h}}(n-1)]\}. \qquad (16) \end{aligned}$$

Note that the two terms in (15) do not depend on $\Delta\mathbf{h}^H(n)$ and therefore they can be excluded from the minimization of the cost function over $\Delta\mathbf{h}(n)$. The term in (16), using (11), can be expressed as $\Re\{\Delta\mathbf{h}^H(n)\mathbf{c}(n|n-1)\}$. Therefore, for the cost function in (13) we obtain

$$J_{\Delta}[\Delta\mathbf{h}(n)] = \frac{1}{2}\Delta\mathbf{h}^H(n)\mathbf{R}(n)\Delta\mathbf{h}(n) - \Re\{\Delta\mathbf{h}^H(n)\mathbf{c}(n|n-1)\} + f_{\mathrm{p}}[\hat{\mathbf{h}}(n-1) + \Delta\mathbf{h}(n)], \tag{17}$$

where $\mathbf{c}(n|n-1)$ is given by (12).

We now show how $\mathbf{c}(n|n-1)$ in (12) can be computed with low complexity in two important cases, the exponential window and sliding window, and present two general structures for developing adaptive filters. Although this is similar to the derivation in [24], for completeness we briefly show the derivation here.

A. Adaptive Filters With Exponential Window

In the case of the exponential window, we have the following recursions for updating the matrix $\mathbf{R}(n)$ and vector $\mathbf{b}(n)$ [38], [39]:

$$\mathbf{R}(n) = \lambda\mathbf{R}(n-1) + \mathbf{x}(n)\mathbf{x}^H(n), \tag{18}$$

$$\mathbf{b}(n) = \lambda\mathbf{b}(n-1) + d^*(n)\mathbf{x}(n). \tag{19}$$

With these recursions, we have

$$\Delta\mathbf{R}(n) = (\lambda-1)\mathbf{R}(n-1) + \mathbf{x}(n)\mathbf{x}^H(n), \tag{20}$$

$$\Delta\mathbf{b}(n) = (\lambda-1)\mathbf{b}(n-1) + d^*(n)\mathbf{x}(n). \tag{21}$$

From (20) and using (9), we obtain

$$\Delta\mathbf{R}(n)\hat{\mathbf{h}}(n-1) = (\lambda-1)[\mathbf{b}(n-1) - \mathbf{c}(n-1|n-1)] + \mathbf{x}(n)y^*(n), \tag{22}$$

where $y(n) = \hat{\mathbf{h}}^H(n-1)\mathbf{x}(n)$ is the adaptive filter output at time instant n. Using (12), (21), and (22), we obtain

$$\mathbf{c}(n|n-1) = \lambda\mathbf{c}(n-1|n-1) + e^*(n)\mathbf{x}(n), \tag{23}$$

where $e(n) = d(n) - y(n)$.

With zero initialization of the solution, i.e., $\hat{\mathbf{h}}(0) = \mathbf{0}_N$ and $\mathbf{b}(0) = \mathbf{0}_N$, from (9) we obtain $\mathbf{c}(0|0) = \mathbf{0}_N$. The matrix $\mathbf{R}(n)$ is initialized as $\mathbf{R}(0) = \eta\mathbf{I}_N$, where $\eta > 0$ is a small number.

In the case of general (unstructured) regressors, the complexity of updating the matrix $\mathbf{R}(n)$ is $\mathcal{O}(N^2)$ operations per time instant. For shift-structured regressors $\mathbf{x}(n) = [x(n)\, x(n-1)\, \ldots\, x(n-N+1)]^T$, where $x(n)$ is a discrete-time signal, updating $\mathbf{R}(n)$ is simplified. The lower-right $(N-1)\times(N-1)$ block of $\mathbf{R}(n)$ can be obtained by copying the upper-left $(N-1)\times(N-1)$ block of $\mathbf{R}(n-1)$. The only part of the matrix $\mathbf{R}(n)$ that should be updated is the first row and first column. Due to the Hermitian symmetry of the matrix, it is enough to calculate the first column. The updating for the exponential window is then given by

$$\mathbf{R}^{(1)}(n) = \lambda\mathbf{R}^{(1)}(n-1) + x^*(n)\mathbf{x}(n). \tag{24}$$

As a result, for transversal adaptive filters, the complexity is reduced down to $\mathcal{O}(N)$ operations per time instant. Note that fast

TABLE I
ADAPTIVE FILTERS WITH EXPONENTIAL WINDOW

Step	Equation	×	+
	Initialization: $\hat{\mathbf{h}}(0)=\mathbf{0}_N, \mathbf{c}(0\|0)=\mathbf{0}_N, \mathbf{R}(0)=\eta\mathbf{I}_N$		
	for $n=1,2,\ldots$		
1	$\mathbf{R}(n)=\lambda\mathbf{R}(n-1)+\mathbf{x}(n)\mathbf{x}^H(n)$ or $\mathbf{R}^{(1)}(n)=\lambda\mathbf{R}^{(1)}(n-1)+x^*(n)\mathbf{x}(n)$	$3N^2$ $[6N]$	$2N^2$ $[4N]$
2	$y(n)=\hat{\mathbf{h}}^H(n-1)\mathbf{x}(n)$	$4N$	$4N$
3	$e(n)=d(n)-y(n)$	$-$	2
4	$\mathbf{c}(n\|n-1)=\lambda\mathbf{c}(n-1\|n-1)$ $+e^*(n)\mathbf{x}(n)$	$6N$	$4N$
5	Solve: $\min_{\Delta\mathbf{h}} J_\Delta(\Delta\mathbf{h})\rightarrow\Delta\hat{\mathbf{h}}$, $\mathbf{c}(n\|n)$ and update $\hat{\mathbf{h}}(n)=\hat{\mathbf{h}}(n-1)+\Delta\hat{\mathbf{h}}$	P_m	P_a
6	Update the weight vector $\mathbf{w}(n)$	$P_{m,w}$	$P_{a,w}$

RLS transversal adaptive filtering algorithms, such as the fixed order and lattice algorithms, also have the complexity $\mathcal{O}(N)$ operations per time instant [39].

An adaptive filter with exponential window can be implemented as shown in Table I, which also shows the complexity of the algorithm steps in terms of real-valued multiplications and additions. The complexity of step 5 depends on the method used for solving the optimization problem, which we present in Section IV; we denote P_m the number of multiplications and P_a the number of additions required by the method. Note that the complexity of calculating $\mathbf{c}(n|n)$ is involved in step 5. The complexity of updating the weight vector $\mathbf{w}(n)$ at step 6 depends on the reweighting method, which is discussed in Section V; $P_{m,w}$ and $P_{a,w}$ are numbers of multiplications and additions, respectively, in the method. E.g., the DCD-ℓ_0 algorithm introduced below does not require reweighting, thus $P_{m,w}=0$ and $P_{a,w}=0$.

B. Adaptive Filters With Sliding Window

In the case of the sliding window, the matrix $\mathbf{R}(n)$ and vector $\mathbf{b}(n)$ can be recursively updated as

$$\mathbf{R}(n)=\mathbf{R}(n-1)+\mathbf{x}(n)\mathbf{x}^H(n)-\mathbf{x}(n-M)\mathbf{x}^H(n-M) \quad (25)$$

and

$$\mathbf{b}(n)=\mathbf{b}(n-1)+d^*(n)\mathbf{x}(n)-d^*(n-M)\mathbf{x}(n-M). \quad (26)$$

To find the vector $\mathbf{c}(n|n-1)$, we notice that

$$\Delta\mathbf{b}(n)=d^*(n)\mathbf{x}(n)-d^*(n-M)\mathbf{x}(n-M) \quad (27)$$

an2d

$$\Delta\mathbf{R}(n)\hat{\mathbf{h}}(n-1)=y^*(n)\mathbf{x}(n)-y_M^*(n)\mathbf{x}(n-M), \quad (28)$$

where $y_M(n)=\hat{\mathbf{h}}^H(n-1)\mathbf{x}(n-M)$. From (27) and (28), we obtain

$$\mathbf{c}(n|n-1)=\mathbf{c}(n-1|n-1)+e^*(n)\mathbf{x}(n)-e_M^*(n)\mathbf{x}(n-M), \quad (29)$$

where $e_M(n)=d(n-M)-y_M(n)$.

TABLE II
ADAPTIVE FILTERS WITH SLIDING WINDOW

Step	Equation	×	+
	Initialization: for $n\le 0$: $\mathbf{x}(n)=\mathbf{0}_N$, $\hat{\mathbf{h}}(0)=\mathbf{0}_N, \mathbf{c}(0\|0)=\mathbf{0}_N, \mathbf{R}(0)=\eta\mathbf{I}_N$		
	for $n=1,2,\ldots$		
1	$\mathbf{R}(n)=\mathbf{R}(n-1)+\mathbf{x}(n)\mathbf{x}^H(n)-\mathbf{x}(n-M)\mathbf{x}^H(n-M)$ or $\mathbf{R}^{(1)}(n)=\mathbf{R}^{(1)}(n-1)+x^*(n)\mathbf{x}(n)-x^*(n-M)\mathbf{x}(n-M)$	$4N^2$ $[8N]$	$4N^2$ $[8N]$
2	$y(n)=\hat{\mathbf{h}}^H(n-1)\mathbf{x}(n)$	$4N$	$4N$
3	$e(n)=d(n)-y(n)$	$-$	2
4	$y_M(n)=\hat{\mathbf{h}}^H(n-1)\mathbf{x}(n-M)$	$4N$	$4N$
5	$e_M(n)=d(n-M)-y_M(n)$	$-$	2
6	$\mathbf{c}(n\|n-1)=\mathbf{c}(n-1\|n-1)+e^*(n)\mathbf{x}(n)-e_M^*(n)\mathbf{x}(n-M)$	$8N$	$8N$
7	Solve: $\min_{\Delta\mathbf{h}} J_\Delta(\Delta\mathbf{h})\rightarrow\Delta\hat{\mathbf{h}}$, $\mathbf{c}(n\|n)$ and update $\hat{\mathbf{h}}(n)=\hat{\mathbf{h}}(n-1)+\Delta\hat{\mathbf{h}}$	P_m	P_a
8	Update the weight vector $\mathbf{w}(n)$	$P_{m,w}$	$P_{a,w}$

For shift-structured input data and the sliding window, we obtain the following recursion for updating the matrix $\mathbf{R}$:

$$\mathbf{R}^{(1)}(n)=\mathbf{R}^{(1)}(n-1)+x^*(n)\mathbf{x}(n)-x^*(n-M)\mathbf{x}(n-M). \quad (30)$$

Again, for arbitrary regressors, the complexity is $\mathcal{O}(N^2)$, and, for regressors with the time-shifted structure, the complexity is reduced to $\mathcal{O}(N)$ operations per time instant. Adaptive filters with the sliding window can be summarized as shown in Table II.

The complexity of exponential window algorithms is lower than the complexity of sliding window algorithms [24]. However, for some applications, the finite memory of sliding window adaptive filters can be attractive.

IV. DCD ALGORITHM FOR LS OPTIMIZATION WITH PENALTY

We now consider minimization of the cost function given by (17). Omitting the time index n and denoting $\mathbf{c}=\mathbf{c}(n|n-1)$ and $\mathbf{h}=\hat{\mathbf{h}}(n-1)$, the function can be rewritten as

$$J_\Delta[\Delta\mathbf{h}]=\frac{1}{2}\Delta\mathbf{h}^H\mathbf{R}\Delta\mathbf{h}-\Re\{\Delta\mathbf{h}^H\mathbf{c}\}+f_{\rm p}(\mathbf{h}+\Delta\mathbf{h}). \quad (31)$$

In order to minimize it, we use a low-complexity version of CD algorithm, the DCD algorithm. In standard CD, at every iteration, only the p-th element of the solution vector $\Delta\mathbf{h}$ may be updated as $\Delta\mathbf{h}\leftarrow\Delta\mathbf{h}+\alpha\mathbf{e}_p$, where α is a complex-valued scalar and $\mathbf{e}_p$ is the p-th column of the identity matrix $\mathbf{I}_N$. The update should only be done if the cost function is reduced, i.e., if

$$\Delta J=J_\Delta(\Delta\mathbf{h}+\alpha\mathbf{e}_p)-J_\Delta(\Delta\mathbf{h})<0.$$

After some algebra, we obtain

$$\Delta J=\frac{1}{2}|\alpha|^2R_{p,p}-\Re\{\alpha^*c_p\}+f_{\rm p}(\mathbf{h}+\Delta\mathbf{h})-f_{\rm p}(\mathbf{h}). \quad (32)$$

When using DCD iterations, elements of the solution vector are represented in a fixed-point format with M_b bits within

an amplitude interval $[-H, H]$; H may be chosen as $H \geq \max_q\{|\Re[h_q]|, |\Im[h_q]|\}$. However, the choice is not very critical as soon as H is chosen as a power-of-two number. This allows multiplications and divisions in the DCD algorithm to be replaced by bit-shift operations; see discussion on the choice of H in [25]. The DCD iterations start updating the most significant bits of the solution, proceeding towards less significant bits. This is controlled by a step-size $\delta > 0$ that starts with $\delta = H$ and is reduced as $\delta \leftarrow \delta/2$ for less significant bits.

In CD iterations, there can be different strategies for selecting coordinates for updates. The most often used are cyclic and leading [24] (also called greedy [41]) selections. We will concentrate on this second option.

As the solution vector is complex-valued, we need to consider four possible directions on the complex plane for updating every coordinate: 1, -1, j and $-j$, where $j = \sqrt{-1}$. For every δ, there are four values by which a coordinate can be updated: $\boldsymbol{\alpha} = [\delta,\ -\delta,\ j\delta,\ -j\delta]$.

In the leading DCD iterations, we need to compute ΔJ for all $s = 1, \ldots, N$ and $q = 1, \ldots, 4$ and find the minimum

$$[p, k] = \arg\min_{s,q}[(\delta^2/2)R_{s,s} - \Re\{\alpha_q^* c_s\} + f_{\mathrm{p}}(\mathbf{h} + \alpha_q \mathbf{e}_s) - f_{\mathrm{p}}(\mathbf{h})]. \quad (33)$$

This will provide us with both the coordinate p to update and direction k in which the update should be done. The minimum is given by

$$\Delta J_{\min} = \frac{\delta^2}{2} R_{p,p} - \Re\{\alpha_k^* c_p\} + f_{\mathrm{p}}(\mathbf{h} + \alpha_k \mathbf{e}_p) - f_{\mathrm{p}}(\mathbf{h}). \quad (34)$$

If $\Delta J_{\min} < 0$, then the iteration is *successful*, that is the p-th element of the solution $\mathbf{h}$ and the residual vector $\mathbf{c}$ are updated. Note that the update $\hat{\mathbf{h}}(n) = \hat{\mathbf{h}}(n-1) + \Delta\hat{\mathbf{h}}$ in line 5 of Table I and in line 7 of Table II is incorporated in the DCD algorithm. If $\Delta J_{\min} \geq 0$, no update is necessary and the step size is halved ($\delta \leftarrow \delta/2$). The procedure is then repeated until the desired precision (number of bits in the solution) is obtained, or until the maximum number of *successful* iterations is met. This way, the complexity of the algorithm can be controlled. The DCD algorithm is shown in Table III. Here, M_b is the number of bits used for representation of entries in the solution vector and N_u is the limit to the number of the *successful* iterations. The parameter M_b defines the accuracy of the fixed-point representation, whereas the parameter N_u limits the complexity.

The complexity of the DCD algorithm has two main contributions. The first contribution is due to updating the residual vector $\mathbf{c}$ in the *successful* iterations (step 6). In general, this would involve $4N$ real multiplications and $4N$ real additions. However, choosing H as a power-of-two number and taking into account the structure of vector $\boldsymbol{\alpha}$, it follows that this step does not require multiplications and it only requires $2N$ real additions. The other contribution is due to computing the penalty function at step 2 and finding the maximum at step 3. Without the penalty term, i.e., for purely LS optimization, the minimization at step 3 becomes especially simple [25]:

$$[p, k] = \arg\min_{s,q}[(\delta^2/2)R_{s,s} - \Re\{\alpha_q^* c_s\}]. \quad (35)$$

TABLE III
DCD ALGORITHM FOR LS OPTIMIZATION WITH PENALTY

Step	Equation
	Input: $\mathbf{c} = \mathbf{c}(n\|n-1)$, $\mathbf{R} = \mathbf{R}(n)$, $\mathbf{h} = \hat{\mathbf{h}}(n-1)$, H, M_b, N_u
	Output: $\hat{\mathbf{h}}(n) = \mathbf{h}$, $\mathbf{c}(n\|n) = \mathbf{c}$
	Initialization: $\delta = H$, $\boldsymbol{\alpha} = \delta[1, -1, j, -j]$, $m = 0$, $u = 0$
1	While $m < M_b$ and $u < N_u$, repeat:
2	Compute $\Delta f(s, q) = f_{\mathrm{p}}(\mathbf{h} + \alpha_q \mathbf{e}_s) - f_{\mathrm{p}}(\mathbf{h})$
3	Find $[p, k] = \arg\min_{s,q}[(\delta^2/2)R_{s,s} - \Re\{\alpha_q^* c_s\} + \Delta f(s, q)]$ for $s = 1, \ldots, N$ and $q = 1, \ldots, 4$
4	$\Delta J_{\min} = (\delta^2/2)R_{p,p} - \Re\{\alpha_k^* c_p\} + \Delta f(p, k)$
5	If $\Delta J_{\min} < 0$
6	$h_p \leftarrow h_p + \alpha_k$, $\mathbf{c} \leftarrow \mathbf{c} - \alpha_k \mathbf{R}^{(p)}$, $u \leftarrow u+1$
	else
7	$\delta \leftarrow \delta/2$, $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}/2$, $m = m+1$

For finding the minimum, we only need to compare magnitudes of the real and imaginary parts of c_s with $(\delta^2/2)R_{s,s}$; this costs only two real-valued additions. However, in the general case, complexity of this part significantly depends on the penalty used and may contribute heavily to the whole algorithm complexity. We present the penalty functions and corresponding complexity in the next section.

It is important to note that, even though the DCD-RLS is a fast algorithm, with $\mathcal{O}(N)$ complexity, it is very stable even in finite-precision arithmetic, since the algorithm does not update $\mathbf{R}^{-1}(n)$ as other fast versions of RLS [38].

V. PENALTY FUNCTIONS

We will consider the lasso, modified lasso, ridge-regression, elastic net, and ℓ_0 penalty functions. All the penalties are separable, i.e., they can be represented in the form

$$f_{\mathrm{p}}(\mathbf{h}) = \sum_{k=1}^{N} f(h_k).$$

The separability makes the implementation of the adaptive filters with coordinate descent iterations especially simple.

A. Elastic-Net, Lasso and Ridge-Regression Penalties

The elastic-net penalty is given by [42]

$$f_{\mathrm{p}}(\mathbf{h}) = \tau \left[\frac{1}{2}(1-\beta)\|\mathbf{h}\|_2^2 + \beta\|\mathbf{h}\|_1\right], \quad (36)$$

where $\tau > 0$ is a regularization parameter. This penalty is a compromise between the ridge-regression penalty ($\beta = 0$) and the lasso penalty ($\beta = 1$) [12]. The complexity of using such penalty is approximately the sum of complexities for the lasso and ridge-regression penalties. More detailed analysis shows that this is $22N$ additions, $10N$ multiplications and $5N$ square root operations.

For the *ridge* regression, the regularization parameter τ is often chosen related to the noise variance. This is a very useful regularization as it can make finite-precision implementation of adaptive filters stable. The classical RLS algorithm does have such a regularization that, however, quickly decays in time with the time constant defined by the forgetting factor λ: $\tau = \tau(n) = \lambda^n$ and, consequently, $\tau(n) \to 0$ as $n \to \infty$. This

is one of the causes of instability of classical RLS adaptive filters when implemented in finite precision. Note that this regularization can be equally incorporated into the filters either using the $f_{\rm p}(\mathbf{h})$ function or using the diagonal loading of the matrix $\mathbf{R}(n)$ with the parameter τ: $\mathbf{R}(n) \leftarrow \mathbf{R}(n) + \tau \mathbf{I}_N$. The latter, however, may require redesigning the general algorithm structures in Tables I and II [29]. We will be using the general scheme of introducing the penalty (regularization) via the function $f_{\rm p}(\mathbf{h})$. In Appendix II, we show that the complexity of using the ridge regression penalty is only $4N$ real-valued additions and $2N$ real-valued multiplications.

With the lasso penalty, the adaptive filter solves the basis pursuit denoising [43] at every time instant n. The lasso penalty in one DCD iteration (see steps 2 to 5 in Table III) can be computed using $18N$ additions, $6N$ multiplications, and $5N$ square-root operations (see Appendix I). This is a significant computational load (compared to $2N$ additions for updating the residual vector in the DCD iteration at step 6). To reduce the complexity, we will also be using the modified lasso penalty as described below.

The penalties can be generalized using a weight vector $\mathbf{w}$:

$$f_{\rm p}(\mathbf{h}) = \tau \sum_{k=1}^{N} w_k \left[\frac{1}{2}(1-\beta)|h_k|^2 + \beta |h_k| \right]. \qquad (37)$$

If τ in (37) is very high and $w_k = 0$ for k within the true support of the unknown vector, and $w_k = 1$ for k outside the support, we arrive at an LS *oracle* algorithm providing the best LS solution. We will be using the performance of the oracle RLS adaptive filter as a benchmark when analyzing the proposed adaptive filters.

Of course, in practice, the true support is usually unknown. A practical algorithm for choosing the weights is discussed next. The weight vector can be updated in reweighting iterations [33], [44]. We will be using the following updating mechanism. At every time instant n, a weight support $\Gamma(n)$ is identified using the thresholding operation:

$$\Gamma(n) = \{p: \ |h_p| > \mu_{\rm d} \max_k \{|h_k|\}\}, \qquad (38)$$

where $\mu_{\rm d}$ is an adjusted parameter. Starting from $\mathbf{w}(0) = \mathbf{1}_N$, the weight vector is updated using the recursion:

$$\mathbf{w}(n) = (1-\mu_w)\mathbf{w}(n-1) + \mu_w \mathbf{g}(n), \qquad (39)$$

where $\mu_w \in [0,1]$ is another adjusted parameter, defining the memory of the recursion, and

$$g_k(n) = \begin{cases} 0, & k \in \Gamma(n) \\ 1, & \text{otherwise} \end{cases} \qquad (40)$$

Other recursions can also be used for updating the weight vector, e.g., see [4], [36]. The proposed weight updating recursion is simple for practical implementation. It does not use division operations that are often involved in reweighting. Moreover, choosing μ_w as a power-of-two number, all multiplications are replaced by bit-shifts that are simple for hardware implementation. For algorithms that automatically adjust τ, the reader is referred to [45], [46].

The modified lasso penalty is given by [2]

$$f_{\rm p}(\mathbf{h}) = \tau(\|\Re\{\mathbf{h}\}\|_1 + \|\Im\{\mathbf{h}\}\|_1). \qquad (41)$$

It can also be generalized by introducing the weighting:

$$f_{\rm p}(\mathbf{h}) = \tau \sum_{k=1}^{N} w_k(|\Re\{h_k\}| + |\Im\{h_k\}|). \qquad (42)$$

In this case, for computation of $\Delta J_{\min}$, $10N$ additions and $4N$ multiplications are required, which is about twice fewer than for the lasso penalty. Besides, most of the operations are additions. What is also important for practical implementation is that the modified lasso penalty does not require square root operations (as does the lasso penalty), that are more complicated for implementation than multiplications and additions.

B. ℓ_0 Penalty

The ℓ_0 penalty is given by

$$f_{\rm p}(\mathbf{h}) = \tau \|\mathbf{h}\|_0. \qquad (43)$$

This penalty results in a non-convex optimization problem that is NP-hard. When using CD iterations for solving this problem, there is no guarantee that we will arrive at the optimal solution. However, as can be seen from the numerical results in Section VI below, this penalty function provides high performance, close to the oracle performance. Besides, this is the simplest function for computing and is very well suited to implementation on hardware design platforms such as FPGAs.

From (43), we have

$$\Delta f(p,k) = \tau u_{p,k}, \qquad (44)$$

where, as explained below,

$$u_{p,k} = \begin{cases} -1, & h_p = -\alpha_k \\ +1, & h_p = 0 \\ 0, & \text{otherwise} \end{cases} \qquad (45)$$

The first equality implies that the p-th element is currently within the support, but after the update $h_p \leftarrow h_p + \alpha_k$ it will be removed from the support, i.e., the support size will decrease by one, thus $u_{p,k} = -1$. The second equality implies that the element is currently outside the support, but after the update it will enter the support, i.e., the support will increase by one, thus $u_{p,k} = +1$. The third equality implies that the element is within the support and after the update it will still stay in the support, i.e., the support size does not change and thus $u_{p,k} = 0$.

Note that the equalities in (45) are exactly achievable when using DCD iterations as they use the fixed-point representation of the vector $\mathbf{h}$ and the values 0 and α_k are within the feasible set of the solution. This would be more difficult to achieve with other versions of CD iterations or non-CD iterations that are not based on fixed-point representation of the solution.

It is seen that all operations for computing $u_{p,k}$ are logical ($h_p = -\alpha_k$ can be identified by checking that the bit corresponding to the step-size δ in the word representing h_p is set) which is very well suited to FPGA design. Apart from the logical operations, for computing $\Delta J_{\min}$, $8N$ real-valued additions are necessary. Thus this penalty allows especially simple implementation of the whole DCD algorithm as the algorithm now is multiplication-free.

VI. Numerical Results and Complexity Analysis

In this section, we present results of computer simulations. We compare the Mean Square Deviation (MSD) performance of

the proposed algorithms against the classical RLS algorithm, oracle RLS algorithm, and advanced sparse adaptive filters. Only scenarios with the time-shifted structure of input data, corresponding to transversal adaptive filters, are considered. We consider two cases of the input data: random signals and a speech signal.

The random input signals are generated as

$$d(n) = \mathbf{h}^H(n)\mathbf{x}(n) + \nu(n), \tag{46}$$

where $\nu(n)$ is the additive zero-mean complex-valued Gaussian random noise with variance σ^2; two cases are considered: $\sigma = 0.01$ (low-noise case) and $\sigma = 0.5$ (high-noise case). The vector $\mathbf{x}(n) = [x(n)\, x(n-1)\, \ldots\, x(n-N+1)]^T$ contains zero-mean complex-valued Gaussian random numbers of unit variance. In each simulation trial, new realizations of the input signal, impulse response $\mathbf{h}(n)$, and noise are generated. The impulse response $\mathbf{h}(n)$ is kept constant in the first half of each trial and then abruptly changed at the beginning of the second half. The change is in both positions and values of the non-zero taps. Positions of the K nonzero elements in $\mathbf{h}(n)$ are chosen randomly. In most of our simulations, we use $K = 5$ and $N = 100$. When investigating the MSD performance against the sparsity level, K varies from $K = 1$ to $K = N = 100$. The nonzero elements of $\mathbf{h}(n)$ are generated as independent complex-valued Gaussian zero-mean random numbers of unit variance and then $\mathbf{h}(n)$ is normalized to have unity norm. The MSD in a simulation trial is calculated as

$$\mathrm{MSD}(n) = \|\mathbf{h}(n) - \hat{\mathbf{h}}_\Gamma(n)\|_2^2, \tag{47}$$

where

$$\hat{h}_{\Gamma,k}(n) = \begin{cases} \hat{h}_k(n), & \text{if } k \in \Gamma(n) \\ 0, & \text{otherwise} \end{cases} \tag{48}$$

The MSDs obtained in 100 trials are averaged and plotted against the time index n.

Parameters of the proposed algorithms are chosen as follows. DCD parameters are set to $H = 1$ and $M_b = 15$, whereas N_u varies. The regularization parameter τ for every time instant is computed as [47], [48]

$$\tau = \mu_\tau \max_{s=1,\ldots,N} |b_s|,$$

where $\mu_\tau > 0$ and b_s are elements of the vector $\mathbf{b}(n)$ from (19) and (26) for the exponential and sliding window adaptive filters, respectively. If $\mu_\tau = 0$, we arrive at the LS optimization. For the lasso penalty without reweighting, the choice of μ_τ is limited to the interval $\mu_\tau \in [0, 1]$; if $\mu_\tau \geq 1$, no new element could enter the support and the solution is a zero vector [48]. The choice of μ_τ is defined by the noise variance σ^2. The higher is the noise variance the closer μ_τ should be to unity and typically μ_τ is chosen proportional to σ [1], [49]. For the ridge-regression penalty without reweighting, the regularization parameter μ_τ is typically chosen proportional to σ^2. In the adaptive algorithms with reweighting, the optimal choice of μ_τ is more complicated and investigated below by simulation.

A. Performance of the Exponential-Window DCD-Lasso Algorithm

Fig. 1 presents the MSD performance of the exponential-window DCD-lasso algorithm against the limit N_u to the number of *successful* DCD iterations. With a large N_u, the DCD-lasso algorithm provides a faster initial convergence and a lower steady-state MSD than the classical RLS algorithm. However, in practice, the initial convergence is not the most important feature. More important is the reaction of the adaptive filter to the variation of the impulse response $\mathbf{h}(n)$ to be estimated. This is characterized by the second part of the curves for $n > 500$ after the abrupt change of the impulse response. We can see that one *successful* update ($N_u = 1$) is already enough to outperform the classical RLS algorithm. Below, we will only show the performance curves for $n > 500$. Note however that even for small N_u, the initial convergence speed of the DCD-lasso algorithm in not much slower than that of RLS. This observation remains valid for all the other simulations presented here.

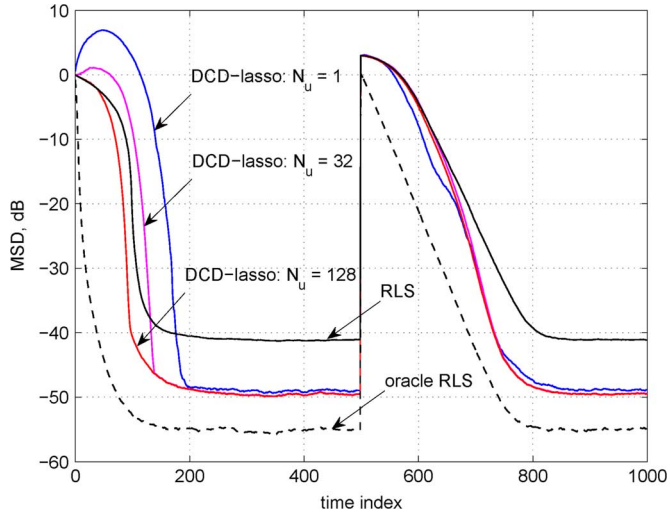


Fig. 1. MSD performance of the exponential-window DCD-lasso adaptive filter for different values of N_u. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$. Parameters of the algorithms: $\lambda = 0.975$, $\mu_\tau = 0.001$, $\mu_w = 0$ (no reweighting), $\mu_d = 0$, $\eta = 3$, $M_b = 15$, $H = 1$.

Fig. 2 shows the MSD performance of the DCD-lasso adaptive filter with different values of the regularization parameter μ_τ. It is seen that, when compared to the DCD adaptive filter without regularization, the use of the lasso penalty allows significant reduction in the steady-state MSD as well as speeding-up the convergence after the abrupt change of the impulse response of the identified system. The adaptive DCD-lasso algorithm also outperforms the classical RLS algorithm. It is seen that there is an optimal value of the regularization parameter μ_τ; at $\mu_\tau = 10^{-3}$ we obtain the lowest steady-state MSD. Note that the results in Fig. 2 are obtained for $N_u = 1$, i.e., for the lowest complexity of the DCD-lasso algorithm. Thus, with the lasso penalty, we need only one DCD iteration per sample in order to, in this scenario, achieve performance better than that of the classical RLS algorithm. However, even for the optimum value of μ_τ, there is a gap between the DCD-lasso performance and the performance of the oracle RLS algorithm; in this case, the difference in the steady-state MSD is about 6 dB. Fig. 3 shows that increasing N_u does not improve the steady-state performance.

Fig. 3 studies the dependence of the steady-state MSD performance of the exponential-window DCD-lasso algorithm against the regularization parameter μ_τ. This graph shows that indeed

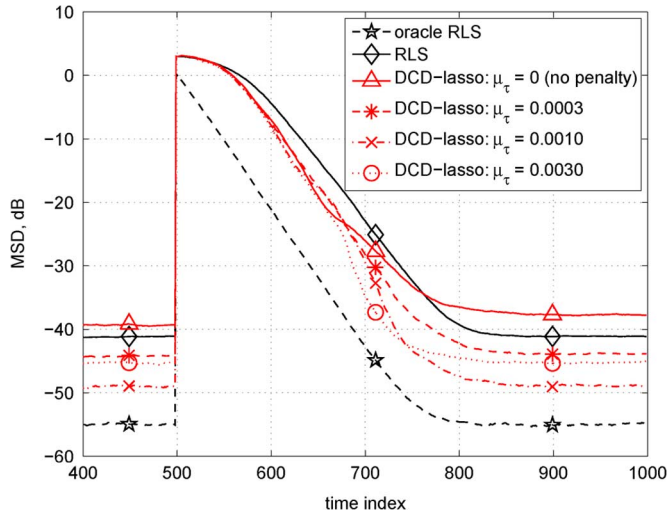


Fig. 2. MSD performance of the exponential-window DCD-lasso adaptive filter for different values of the regularization parameter μ_τ. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$. Parameters of the algorithms: $\lambda = 0.975$, $\mu_w = 0$ (no reweighting), $\mu_d = 0$, $\eta = 3$, $N_u = 1$, $M_b = 15$, $H = 1$.

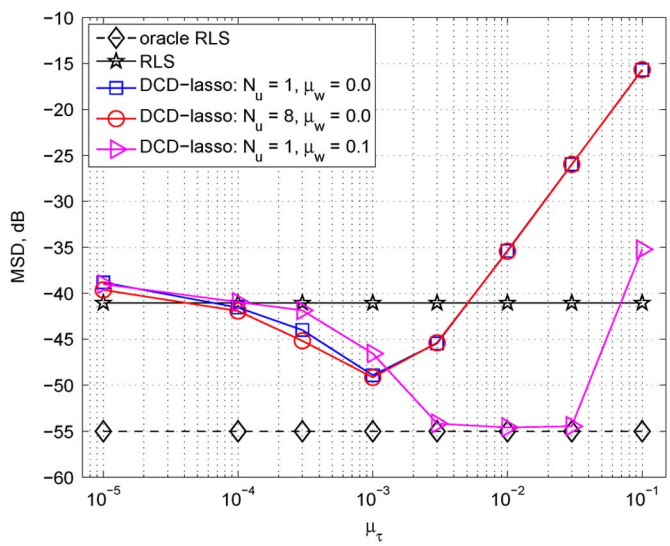


Fig. 3. Steady-state MSD performance of the exponential-window DCD-lasso algorithm against the regularization parameter μ_τ after 500 input samples. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$. Parameters of the algorithms: $\lambda = 0.975$, $\eta = 3$, $M_b = 15$, $H = 1$, $\mu_d = 0$.

the MSD achieves a minimum at $\mu_\tau = 10^{-3}$. When μ_τ is reduced, the performance approaches that of the DCD adaptive algorithm without penalty, which is close to that of the classical RLS algorithm. However, when μ_τ is increased too much, the performance starts to deteriorate and becomes significantly inferior to that of the classical RLS algorithm. This is due to the fact that a large μ_τ prevents new elements entering the support of the solution. At $\mu_\tau = 1$, any element entering the support would increase the cost function, and therefore the optimal solution will be zero, implying that the MSD will be 0 dB. It is seen that the increase in N_u does not improve much the steady-state performance of the DCD-lasso algorithm; thus, in this case, only $N_u = 1$ DCD iteration per sample is enough for the high performance. However, there is a gap of about 6 dB between the best

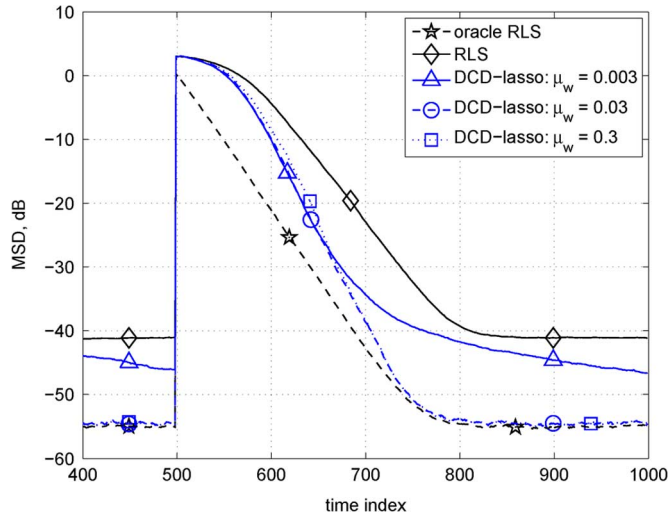


Fig. 4. MSD performance of the exponential-window DCD-lasso algorithm for different values of the reweighting parameter μ_w. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$. Parameters of the algorithms: $\lambda = 0.975$, $\eta = 3$, $\mu_\tau = 0.01$, $\mu_d = 0.005$, $N_u = 1$, $M_b = 15$, $H = 1$.

DCD-lasso MSD performance and the oracle RLS performance (see Fig. 3) that cannot be improved with the pure lasso penalty.

Fig. 3 also shows the performance of the DCD-lasso algorithm with the reweighting described by equations (38), (39), and (40). It can be seen that the reweighting greatly improves the performance which now closely approaches the oracle performance. It also significantly widens the range of μ_τ values for which the algorithm shows high performance and outperforms the classical RLS algorithm. This fact is useful in practice as it allows more freedom in choosing the regularization parameter.

Fig. 4 demonstrates that not only the steady-state performance is improved due to the reweighting, but the convergence speed also increases and the transition MSD curve becomes close to that of the oracle RLS algorithm. This figure also shows how the performance varies with the reweighting step size μ_w. Clearly, reducing μ_w, the time constant of the reweighting recursion increases and it takes longer to achieve the steady-state performance, which is however the same for all the values of μ_w. We have found that, in most cases, choosing μ_w in the interval $[0.03, 0.5]$ resulted in good performance. This choice is not affected by other parameters of the algorithm.

Fig. 5 shows the MSD performance of the exponential-window DCD-lasso algorithm for different sparsity levels K. The parameters of the algorithm for each K are adjusted to guarantee the best performance with a minimum N_u. It can be seen that, as K increases, the steady-state MSD and the convergence time also increase. However, for all K, the DCD-lasso algorithm provides a steady-state MSD close to that of the oracle RLS. Since in one DCD iteration only one element of the solution vector can be updated, higher values of K require a higher N_u to approach the oracle RLS performance. Smaller N_u than that shown in Fig. 5 result in (not shown here) a slower convergence speed, but the steady-state MSD reached after the convergence will still be close to that of the oracle RLS. Note that the forgetting factor $\lambda = 0.975$ chosen for our simulation is considered to be very low; typically, λ is chosen to satisfy $\lambda > 1 - 2/N$ [39], i.e., in our case, it should be $\lambda > 0.98$. The small λ is a worst case situation for assessment

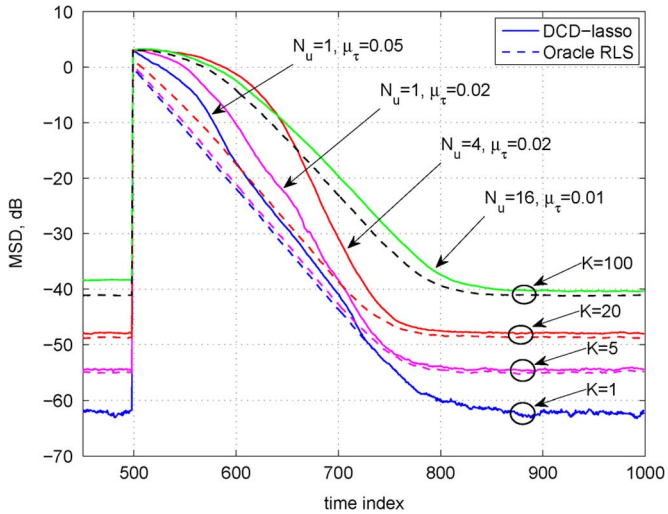


Fig. 5. MSD performance of the exponential-window DCD-lasso algorithm with reweighting for different levels of sparsity K. Parameters of the scenario: $N = 100$, $\sigma = 0.01$. Parameters of the algorithms: $\lambda = 0.975$, $\mu_w = 0.1$, $\mu_{\rm d} = 0.005$, $\eta = 3$, $M_b = 15$, $H = 1$.

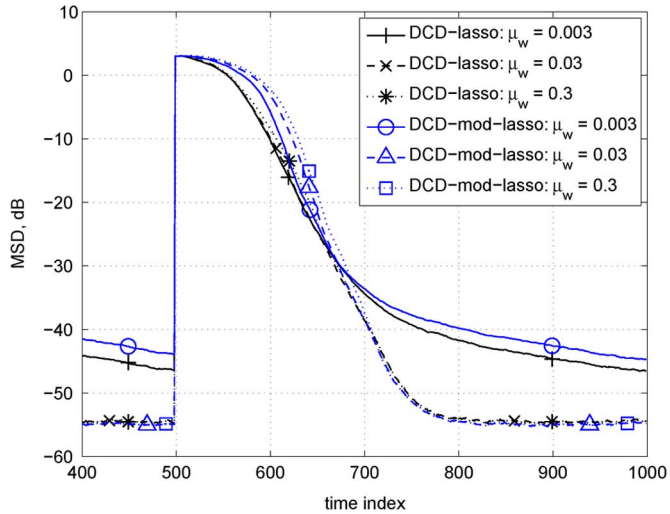


Fig. 6. MSD performance of the exponential-window algorithms with lasso and modified lasso penalty functions for different reweighting parameters μ_w. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$. Parameters of the algorithms: $\lambda = 0.975$, $\mu_{\rm d} = 0.005$, $\eta = 3$, $\mu_\tau = 0.01$, $N_u = 1$, $M_b = 15$, $H = 1$.

of our algorithms as the convergence speed depends on N_u, i.e., with higher λ the DCD based algorithms require smaller N_u to match the oracle performance. For sparser systems (e.g., $K = 1$), the regularization parameter μ_τ can be chosen higher (e.g., $\mu_\tau = 0.05$); for a constant K, a smaller μ_τ results in somewhat slower convergence.

Fig. 6 compares the performance of the exponential-window DCD adaptive filtering with the lasso and modified lasso penalties. It is seen that the modified lasso penalty results in somewhat inferior performance compared to the lasso penalty. Note that the small reweighting forgetting factor $\mu_w = 0.003$ slows down the convergence of both the algorithms, but the steady-state MSD reached by the algorithms is the same as for the higher μ_w. Taking into account that the difference in the performance of the two algorithms is small and that, with the modified lasso penalty, the complexity of the algorithm is reduced, it can be a good candidate for implementation.

B. Performance of the Sliding-Window DCD-Lasso Algorithm

We now investigate the behavior of the adaptive DCD-lasso algorithm with the sliding window. The length of the sliding window M is chosen equal to $M = 150$ to make the steady-state MSD performance of the sliding window RLS algorithm close to that of the earlier considered exponential window RLS algorithm. Comparing the convergence of the two RLS algorithms, it is seen that the sliding window provides a faster convergence to the steady-state (compare Figs. 4 and 7); the convergence time for the sliding window version is about twice smaller. The fact that sliding-window adaptive algorithms are not often considered in the literature and used in practice is probably due to the fact that until recently there was not computationally efficient implementation of such algorithms [24].

Fig. 7 shows that the DCD-lasso algorithm with reweighting and the sliding window approaches the steady-state MSD of the oracle sliding-window RLS algorithm and significantly outperforms the sliding-window RLS algorithm. However, the transition part of the MSD curve for $N_u = 1$ is almost twice longer than that of the oracle RLS. This can be reduced by using extra

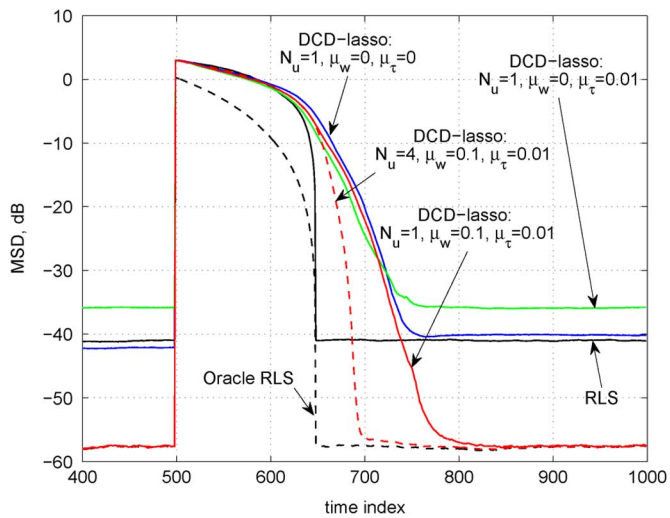


Fig. 7. MSD performance of the sliding-window DCD-lasso algorithm. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$. Parameters of the algorithms: $M = 150$, $\mu_d = 0.001$, $M_b = 15$, $H = 1$.

DCD iterations. With $N_u = 4$, the convergence time is only 30% longer than that of the oracle RLS.

C. Performance of the Exponential-Window DCD-Ridge Algorithm

We now investigate the performance of the exponential-window DCD adaptive filter with the ridge-regression penalty. Fig. 8 presents simulation results for the case of a low noise variance, $\sigma = 0.01$ (as was the case in the previous simulation scenarios). With $\mu_\tau = 0$, the DCD-ridge algorithm is equivalent to the DCD-RLS algorithm with no penalty. When μ_τ increases ($\mu_\tau = 0.01$), without the reweighting ($\mu_w = 0$), the steady-state MSD increases, i.e., the regularization makes the MSD performance worse. However, the reweighting greatly improves the performance and, with $\mu_\tau = 10$ and $\mu_w = 0.1$, the MSD performance is very close to that of the oracle RLS.

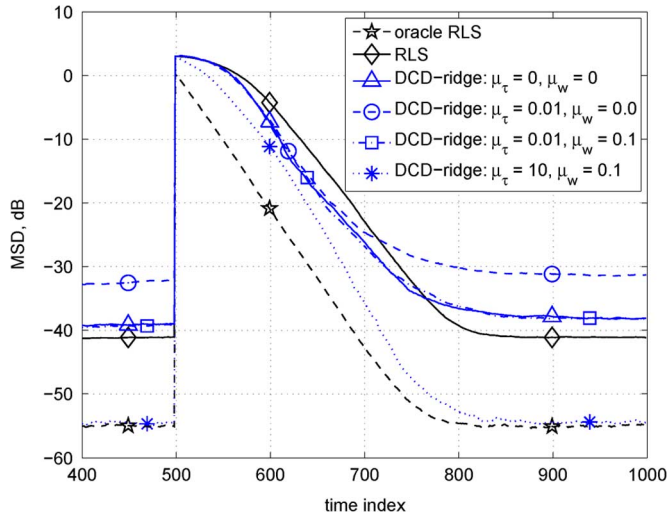


Fig. 8. MSD performance of the exponential-window DCD-ridge algorithm for different regularization μ_τ and reweighting μ_w parameters. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$. Parameters of the algorithms: $\lambda = 0.975$, $\mu_{\rm d} = 0.001$, $\eta = 3$, $N_u = 1$, $M_b = 15$, $H = 1$.

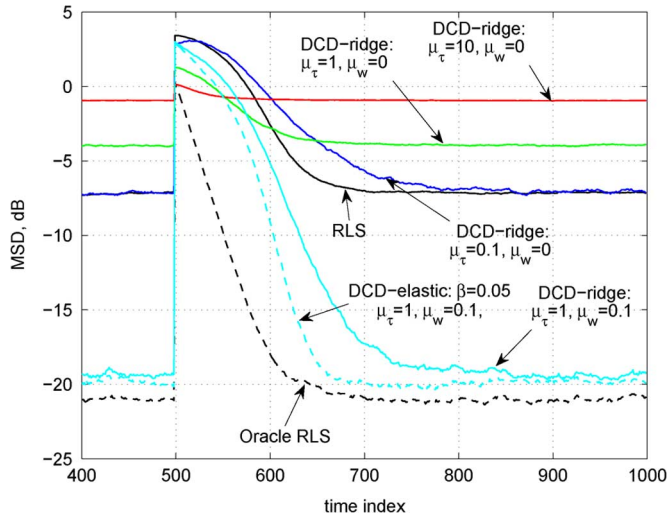


Fig. 9. MSD performance of the exponential-window DCD-ridge and DCD-elastic algorithms for different regularization μ_τ and reweighting μ_w parameters. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.5$. Parameters of the algorithms: $\lambda = 0.975$, $\mu_{\rm d} = 0.1$, $\eta = 3$, $N_u = 4$, $M_b = 15$, $H = 1$.

Thus, the use of the ridge-regression penalty, similarly to the lasso penalty (see Fig. 7), together with the reweighting has allowed achieving a close-to-oracle performance. Note that the ridge-regression penalty, as we mentioned before, is equivalent to applying regularization (diagonal loading) to $\mathbf{R}(n)$ and as such is useful to avoid numerical instability. What is surprising is that, with the use of reweighting, this penalty also turns out to provide good performance for sparse estimation.

Fig. 9 shows the MSD performance for the case of a significantly higher noise variance, $\sigma = 0.5$. It can be seen that, without the reweighting, the performance cannot approach the oracle RLS performance. However, with the reweighting, the steady-state MSD is quite close to the oracle performance with a gap of about 2 dB. The elastic-net penalty gives an extra opportunity to fill the gap. With the parameter $\beta = 0.05$, the DCD-elastic algorithm has reduced this gap by about 0.5 dB, but, more significantly, it reduced the convergence time.

D. Performance of the Exponential-Window DCD-ℓ_0 Algorithm

Fig. 10 shows the dependence of the steady-state MSD provided by the exponential-window DCD-ℓ_0 algorithm on the regularization parameter μ_τ for different values of $\mu_{\rm d}$ and μ_w. The case $\mu_w = 0$ implies that there is no reweighting, and it can be seen that, for this penalty, the reweighting does not make any impact on the performance. However, the threshold $\mu_{\rm d}$ does influence the performance at low μ_τ. The main conclusion here is that the ℓ_0-penalty achieves the oracle performance. Recall that, among the considered penalties, the ℓ_0-penalty results in the simplest implementation of the DCD adaptive filter. Combined with the good steady-state performance as indicated by Fig. 10, this variant of the adaptive filter is very attractive for practical implementation.

Fig. 11 shows the MSD performance of the exponential-window DCD-ℓ_0 algorithm for different sparsity levels K. The parameters of the algorithm for each K are adjusted to guarantee the best performance with a minimum N_u. The

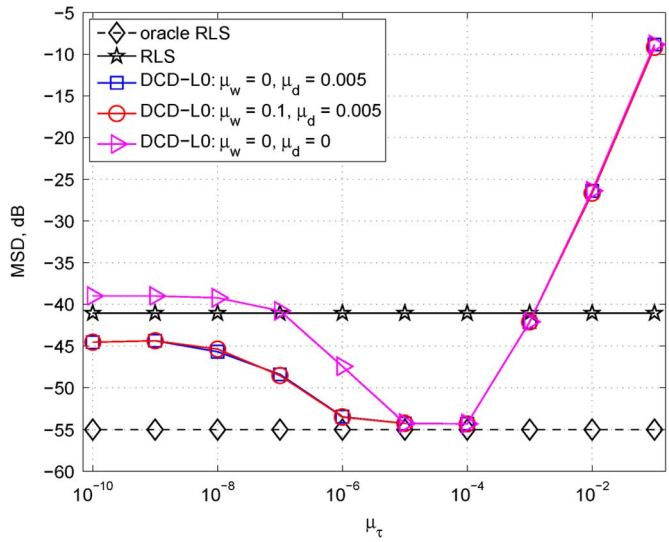


Fig. 10. Steady-state MSD performance of the exponential-window DCD-ℓ_0 algorithm against the regularization parameter μ_τ after 500 input samples. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$. Parameters of the algorithms: $\lambda = 0.975$, $\eta = 3$, $M_b = 15$, $H = 1$.

performance of the DCD-ℓ_0 algorithm is close to that of the oracle RLS algorithm for all K, similarly to the performance of the DCD-lasso algorithm (see Fig. 5). It is interesting that for both the algorithms, the same number of DCD iterations is required. The regularization parameter can vary considerably without significant influence on the performance. E.g., for $K = 1$, choosing μ_τ in the interval between 10^{-6} to 10^{-1} results in a steady-state MSD close to that of the oracle RLS. Smaller μ_τ somewhat increases the convergence time, which however in all the cases remains smaller than 300 samples after the change of the impulse response.

Fig. 12 compares the MSD performance of the exponential-window DCD adaptive filters with all the penalties. It is seen that all the penalties allow achieving a performance close to that of

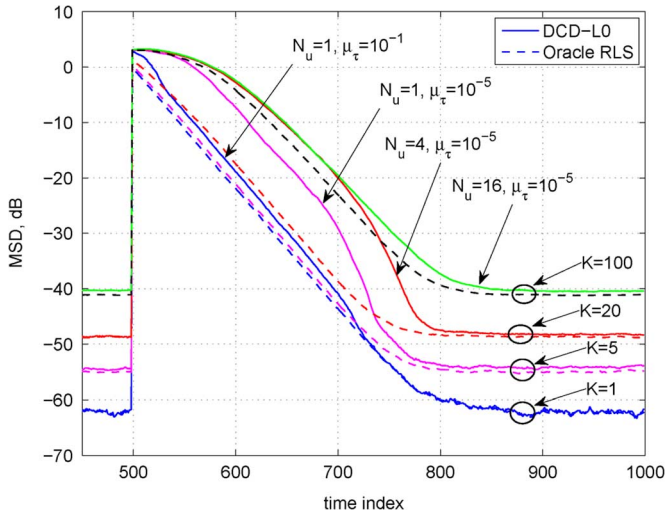


Fig. 11. MSD performance of the exponential-window DCD-ℓ_0 algorithm with reweighting for different levels of sparsity K. Parameters of the scenario: $N = 100$, $\sigma = 0.01$. Parameters of the algorithms: $\lambda = 0.975$, $\mu_{\mathrm{d}} = 0.005$, $\eta = 3$, $M_b = 15$, $H = 1$.

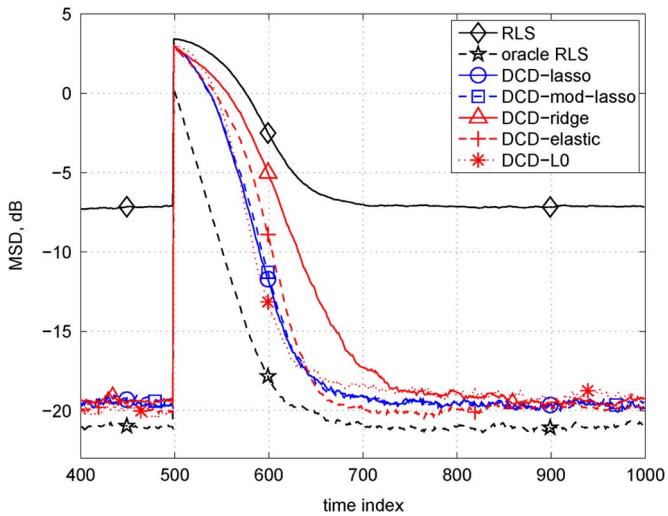


Fig. 12. Comparison of MSD performance of the exponential-window DCD adaptive filters with all the penalties. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.5$. Parameters of the algorithms: $\lambda = 0.975$, $\eta = 3$, $N_u = 1$, $M_b = 15$, $H = 1$, $\mu_{\mathrm{d}} = 0.1$, $\mu_w = 0.1$ (except for the DCD-ℓ_0 algorithm that does not use reweighting). Regularization parameters: $\mu_\tau = 0.1$ (lasso and modified lasso), $\mu_\tau = 1$ (ridge), $\mu_\tau = 1$ and $\beta = 0.05$ (elastic net), $\mu_\tau = 0.01$ (ℓ_0).

the oracle RLS. The ridge-regression penalty provides a slightly slower convergence speed than the other penalties.

E. Performance of the Sliding-Window DCD-ℓ_0 Algorithms

Fig. 13 compares the MSD performance of the sliding-window DCD adaptive filters with all the penalties. The window length $M = 100$ is chosen to match the steady-state performance of the sliding-window adaptive filters to the exponential-window adaptive filters (Fig. 12). It is seen again that all the penalties allow achieving a performance close to that of the oracle RLS. Similarly to the exponential case, the ridge-regression penalty provides a slightly worse performance than the other penalties. It is also seen that both the families

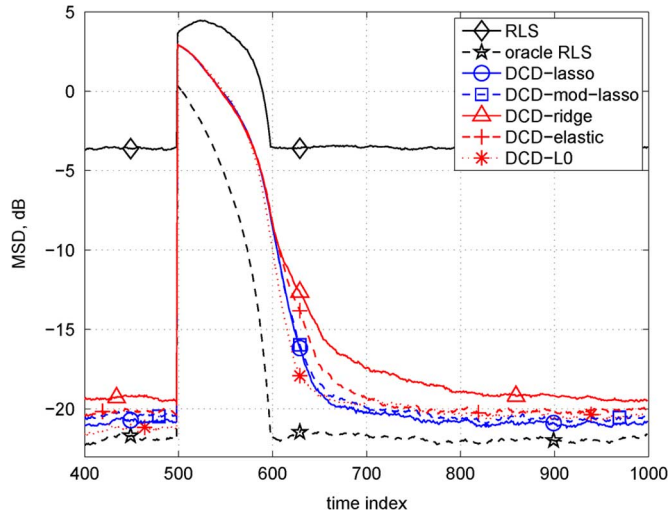


Fig. 13. Comparison of MSD performance of the sliding-window DCD adaptive filters with all the penalties. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.5$. Parameters of the algorithms: $M = 100$, $\eta = 3$, $N_u = 1$, $M_b = 15$, $H = 1$, $\mu_{\mathrm{d}} = 0.1$, $\mu_w = 0.1$ (except for the DCD-ℓ_0 algorithm that does not use reweighting). Regularization parameters: $\mu_\tau = 0.1$ (lasso and modified lasso), $\mu_\tau = 2$ (ridge), $\mu_\tau = 1$ and $\beta = 0.05$ (elastic net), $\mu_\tau = 0.01$ (ℓ_0).

of adaptive filters significantly outperform the classical RLS algorithms.

F. Comparison With Other Sparse Adaptive Filtering Algorithms

We now compare the proposed algorithms with other adaptive filtering algorithms. Note that the weight vector that we are using relates to the power delay profile of the unknown system. A similar idea of reweighting is exploited in proportionate adaptive filters for controlling step sizes involved in updating the filter taps [37]. Therefore, for comparison we consider the μ-law proportionate NLMS (MPNLMS) algorithm, which is an advanced version of the PNLMS algorithm, and the proportionate affine projection algorithm (PAPA) [37]. In the literature, there have also been proposed RLS-like sparse adaptive filtering algorithms [1], [2], [5], [6]; for comparison, we will be using the ℓ_1-RLS and ℓ_0-RLS algorithms from [5], the OSCD-TWL and OSCD-TNWL algorithms from [1] and the SPARLS algorithm from [2]. We will also be using the ℓ_0 LMS algorithm from [13] which is based on using an approximation to the ℓ_0 penalty (the same approximation is used in the ℓ_0-RLS algorithm [5]). For all the algorithms, parameters have been adjusted to achieve the best possible performance.

Fig. 14 compares the MSD performance of the algorithms for a high level of noise, $\sigma = 0.5$. It is seen that with the same forgetting factor $\lambda = 0.975$, the proposed DCD-lasso algorithm without reweighting outperforms the ℓ_1-RLS algorithm from [5], although both algorithms use the same lasso penalty. This can be explained by the approximations used in [5] for derivation of the ℓ_1-RLS algorithm (based on the assumption that the impulse response estimates do not change significantly from one sample to another). Reweighting significantly reduces the steady-state MSD of the DCD-lasso algorithm without compromising the convergence speed. For the ℓ_1-RLS algorithm, to achieve the same steady-state MSD, the forgetting factor has to

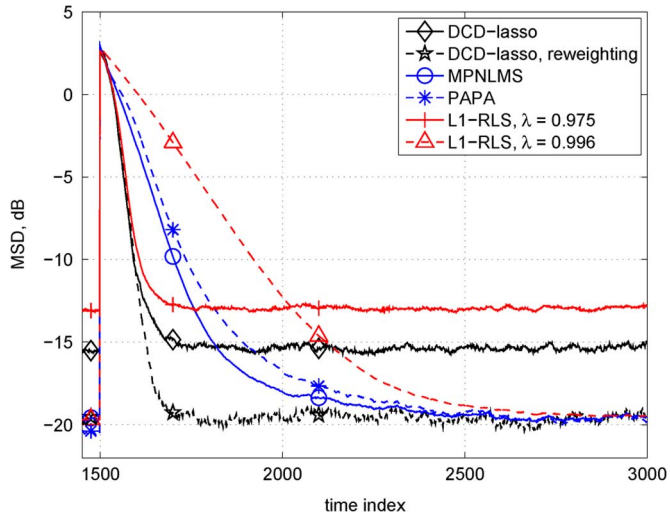


Fig. 14. Comparison of MSD performance of different adaptive filters: MPNLMS, PAPA, ℓ_1-RLS, and exponential-window DCD-lasso with and without reweighting. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.5$.

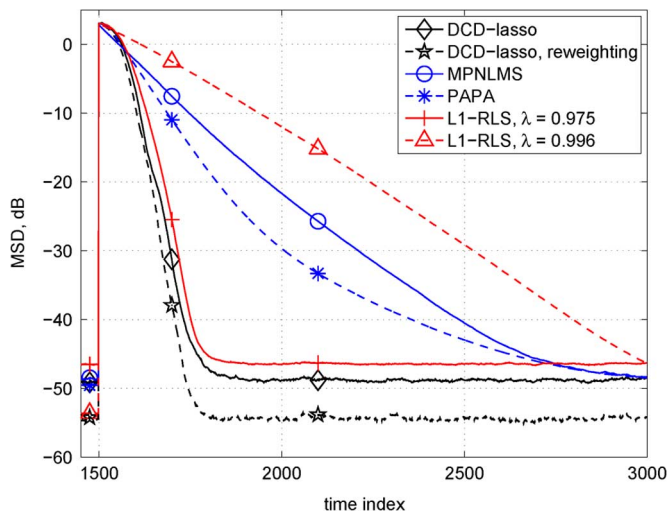


Fig. 15. Comparison of MSD performance of different adaptive filters: MPNLMS, PAPA, ℓ_1-RLS, and exponential-window DCD-lasso with and without reweighting. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$.

be increased to $\lambda = 0.996$, significantly slowing down the convergence.

Similar behavior is observed for a lower level of noise, $\sigma = 0.01$ (see Fig. 15). Here, the ℓ_1-RLS algorithm with $\lambda = 0.996$ provides a steady-state MSD the same as that of the DCD-lasso algorithm with reweighting and $\lambda = 0.975$. It is seen that, to achieve the same steady-state MSD level, the DCD-lasso algorithm with reweighting converges faster than the ℓ_1-RLS algorithm. Parameters of the MPNLMS and PAPA algorithms are chosen to match the steady-state MSD of the DCD-lasso algorithm without reweighting; as it is seen, the DCD-lasso algorithm provides faster convergence.

Fig. 16 shows the MSD performance of the adaptive filters with the same parameters as in Fig. 14 for a scenario where the system impulse response varies in time according to the autoregressive model:

$$\mathbf{h}(n) = (1 - \alpha_h)\mathbf{h}(n-1) + \boldsymbol{\chi}(n),$$

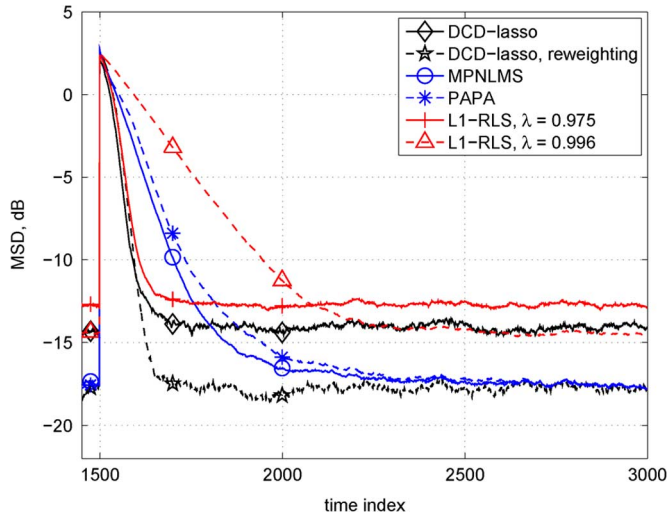


Fig. 16. Comparison of MSD performance of different adaptive filters: MPNLMS, PAPA, ℓ_1-RLS, and exponential-window DCD-lasso with and without reweighting for a system with an impulse response varying in time according to the autoregressive model with $\alpha_h = 10^{-4}$. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.5$.

where $\mathbf{h}(n)$ at $n = 0$ and $n = 1500$ are generated as described above, and $\boldsymbol{\chi}(n)$ is a vector whose elements are zero outside of the support of $\mathbf{h}(n)$ and nonzero within the support. Nonzero elements (K elements) of $\boldsymbol{\chi}(n)$ are independent zero-mean complex-valued random Gaussian numbers of variance $\alpha_h(2 - \alpha_h)/K$ that results in the variance of nonzero elements in $\mathbf{h}$ equal to $1/K$. The parameter α_h defines the speed of the $\mathbf{h}(n)$ time variation; for the simulation results shown in Fig. 16, we used $\alpha_h = 10^{-4}$. It is seen that, compared to the time-invariant impulse response (see Fig. 14), the steady-state MSD of the adaptive algorithms increases; however, the DCD-lasso algorithm still outperforms the other algorithms. It is seen that the ℓ_1-RLS algorithm with $\lambda = 0.996$ experiences the largest increase in the steady-state MSD. This is due to the large forgetting factor $\lambda = 0.996$, which corresponds to an efficient average time $t_{\text{av}} = 1/(1 - \lambda) = 250$ for computation of $\mathbf{R}(n)$ and $\mathbf{b}(n)$, comparing to the significantly smaller $t_{\text{av}} = 40$ for $\lambda = 0.975$ used in the other algorithms. The high t_{av} provides an excessive smoothing of the time variations and results in a higher MSD.

Fig. 17 compares the OSCD-TWL and OSCD-TNWL algorithms from [1], SPARLS algorithm from [2], and DCD-lasso algorithm in a scenario with real-valued signals (note that the OSCD-TWL and OSCD-TNWL algorithms are only available for the case of real-valued signals [1]). When implementing the DCD-lasso algorithm for the real-valued case, in the DCD algorithm (see Table III), the step-size vector $\boldsymbol{\alpha}$ contains only two elements, $\boldsymbol{\alpha} = [\delta, -\delta]$, and the index q at step 3 takes the values $q = 1, 2$. The SPARLS algorithm approximately solves the lasso problem with the modified penalty function. Parameters of the SPARLS algorithm (see details in [2]) are tuned to provide the oracle steady-state performance for $\lambda = 0.975$ and minimize the convergence time. The OSCD-TWL algorithm approximately solves the lasso problem using the coordinate descent iterations with the exact line search (we use one iteration per sample to match the DCD-lasso algorithm). The OSCD-TNWL algorithm exploits reweighting with a specific weight function (see more details in [1]). Parameters of the OSCD-TWL and

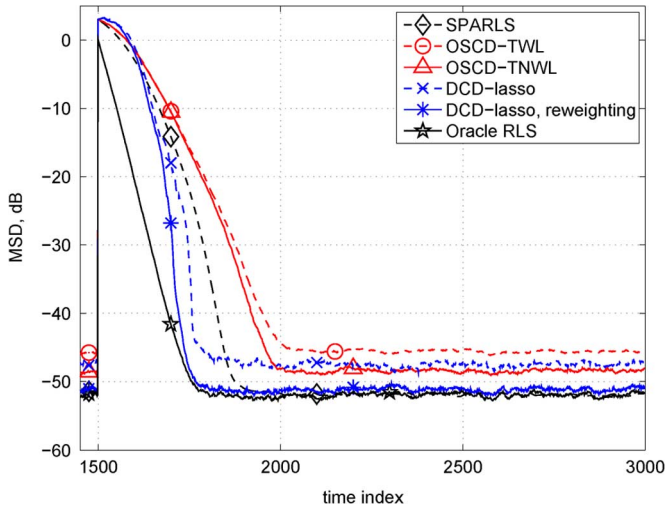


Fig. 17. Comparison of MSD performance of different adaptive filters: SPARLS, OSCD-TWL, OSCD-TNWL, and exponential-window DCD-lasso with and without reweighting. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$. Parameters of the DCD-lasso algorithm: $\lambda = 0.975$, $\eta = 3$, $N_u = 1$, $M_b = 15$, $H = 1$, $\mu_{\mathrm{d}} = 0.01$, $\mu_\tau = 0.001$, $\mu_w = 0.1$ (for the DCD-lasso algorithm with reweighting).

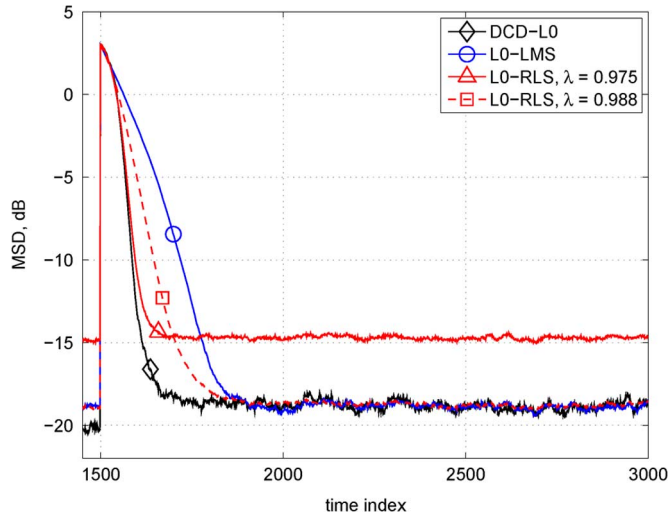


Fig. 18. Comparison of MSD performance of different adaptive filters: ℓ_0-LMS, ℓ_0-RLS, and exponential-window DCD-ℓ_0. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.5$.

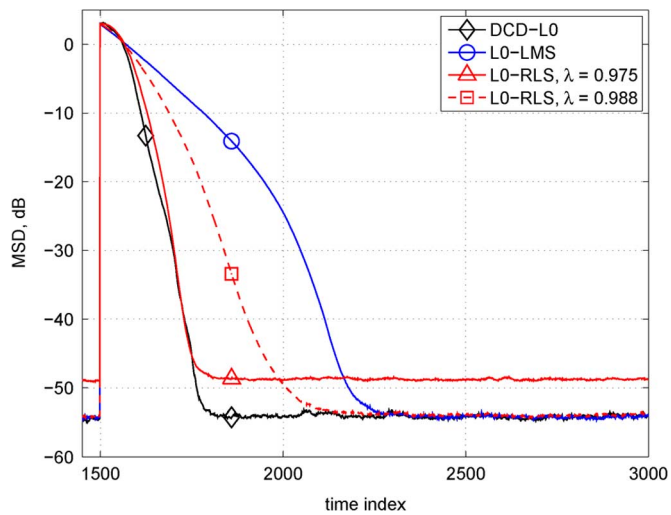


Fig. 19. Comparison of MSD performance of different adaptive filters: ℓ_0-LMS, ℓ_0-RLS, and exponential-window DCD-ℓ_0. Parameters of the scenario: $N = 100$, $K = 5$, $\sigma = 0.01$.

OSCD-TNWL algorithms are tuned to provide the best possible performance for the forgetting factor $\lambda = 0.975$. It is seen that the DCD-lasso algorithm with reweighting shows faster convergence than the SPARLS algorithm; besides, the DCD-lasso algorithm has a lower complexity ($\mathcal{O}(N)$ against $\mathcal{O}(N^2)$ for the SPARLS algorithm). The OSCD-TWL and OSCD-TNWL algorithms have a complexity comparable to that of the DCD-lasso algorithm. However, as seen from Fig. 17, the performance of OSCD-TWL and OSCD-TNWL algorithms is inferior to that of the DCD-lasso algorithm and DCD-lasso algorithm with reweighting, respectively.

The adaptive algorithms with the ℓ_0 penalty (or its approximation, in the case of the ℓ_0-RLS [5] and ℓ_0-LMS [13] algorithms) show, as seen in Figs. 18 and 19, better performance than the algorithms based on the ℓ_1 penalty. In both the cases of high and low noise levels, the proposed DCD-ℓ_0 algorithm demonstrates superior performance compared to the other algorithms.

Finally, Fig. 20 shows the MSD performance of the exponential-window DCD-lasso and DCD-ridge adaptive filters with reweighting against that of the MPNLMS, PAPA, RLS and oracle RLS adaptive filters in a scenario with a speech signal as the input to the adaptive filter. The filter length is $N = 512$, whereas only $K = 100$ filter taps are nonzero and generated as independent zero-mean real-valued random Gaussian numbers of unit variance and then $\mathbf{h}(n)$ is normalized to have unity norm. The impulse response $\mathbf{h}(n)$ is kept constant during the first 2000 samples and then changed in both positions and values of the nonzero taps. The proposed DCD based adaptive filters with lasso and ridge penalties significantly outperform the MPNLMS, PAPA, and RLS algorithms. The affine projection order in the PAPA algorithm was set to 32, which is considered to be a high projection order [50] (typically, a projection order 8 or lower is used) resulting in a high complexity; for lower projection orders, the performance of the PAPA algorithm quickly degrades. The other parameters of the algorithms are chosen to guarantee the best performance. For the DCD-lasso algorithm, the parameters are: $\lambda = 0.998$, $\eta = 10^6$, $\mu_\tau = 0.001$, $\mu_w = 0.3$, $\mu_{\mathrm{d}} = 0.05$. It is seen that with increase in N_u in the DCD-lasso algorithm, the convergence speeds up; however, the steady state performance (the tracking part of the curves) is almost the same irrespectively of N_u. Moreover, it is almost the same as that of the oracle RLS algorithm. With $N_u = 64$, the DCD-lasso adaptive algorithm is only about twice slower in convergence than the oracle RLS algorithm. For the DCD-ridge algorithm, the parameters are: $\lambda = 0.998$, $\eta = 10^6$, $\mu_\tau = 0.1$, $\mu_w = 0.1$, $\mu_{\mathrm{d}} = 0.03$, $N_u = 64$. The DCD-ridge adaptive algorithm with $N_u = 64$ also significantly outperforms the MPNLMS, PAPA, and RLS algorithms in the steady-state performance. However, it is somewhat inferior to the DCD-lasso algorithm in both the convergence speed and the steady-state performance.

G. Complexity of Proposed Algorithms

Tables IV and V summarize numbers of real-valued additions, multiplications and square-root operations required for implementation of the proposed adaptive filtering algorithms. For computing the complexity of a particular adaptive filter with

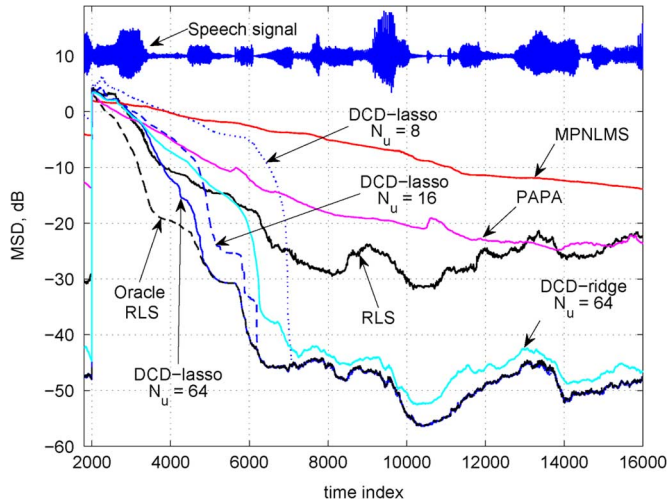


Fig. 20. MSD performance of MPNLMS, PAPA, exponential-window RLS, oracle RLS, DCD-lasso and DCD-ridge adaptive filters with speech as the input signal. Parameters of the scenario: $N = 512$, $K = 100$, $\sigma = 10$. Speech samples are obtained at a rate of 8 kHz and represented as 14-bit fixed-point numbers; the maximum signal magnitude is equal to 16260 at time instant 9933.

TABLE IV
COMPLEXITY OF DIFFERENT STRUCTURES OF RLS ADAPTIVE FILTERS

Algorithm structure:	$+$	$\times$
Exponential	$2N^2+8N$	$3N^2+10N$
Exponential transversal	$12N$	$16N$
Sliding window	$4N^2+16N$	$4N^2+16N$
Sliding window transversal	$24N$	$24N$

TABLE V
COMPLEXITY OF FINDING THE LEADING ELEMENT IN DCD ITERATIONS WITH DIFFERENT PENALTIES

Penalty:	$+$	$\times$	$\sqrt{\cdot}$
Lasso	$18N$	$6N$	$5N$
Modified lasso	$10N$	$4N$	-
Ridge regression	$4N$	$2N$	-
Elastic net	$22N$	$10N$	$5N$
ℓ_0	$8N$	-	-

a particular penalty function, we need to take the complexity in Table IV for the structure of the filter (exponential or sliding window, transversal or not) and add, for every DCD update, the complexity for a specific penalty from Table V, plus $2N$ additions for step 6 of the DCD algorithm in Table III, and the complexity of updating the weight vector $\mathbf{w}(n)$. When using the proposed reweighting method, described by (38) to (40), the complexity of the weight update can be made as low as $2N$ additions for all the penalties except the ℓ_0 penalty. For the latter case, $2N$ multiplications and $2N$ additions are required. However, as is seen from the simulation results in Section VI, the ℓ_0-DCD algorithm does not require the reweighting to achieve a high performance. Thus, the transversal version of all the proposed adaptive algorithms has a complexity of $\mathcal{O}(N)$ operations per sample. E.g., for the DCD-ℓ_0 transversal adaptive filter with exponential window and without reweighting, we obtain $14N$ real-valued multiplications; the number of additions can vary. In the worst-case scenario, when all *successful* updates are performed at the least significant bit $m = M_b$ and the number of successful updates is exactly equal to the upper limit N_u, the algorithm requires $8NM_b + 10NN_u$ real-valued additions.

VII. CONCLUSION

In this paper, we have proposed a general approach for developing low complexity adaptive algorithms for identification of sparse complex-valued systems and specified it for exponential and sliding window RLS. The proposed algorithms are based on DCD iterations. Using this approach, we have proposed DCD-based RLS adaptive filters with the lasso, modified lasso, ridge-regression, elastic net, and ℓ_0 penalties that attract sparsity of signals. We have proposed a simple recursive reweighting of the penalties to further improve the performance. For general regressors, the proposed algorithms have a complexity of $\mathcal{O}(N^2)$ operations per sample, where N is the filter length. For transversal adaptive filters, the algorithms have a low complexity, of $\mathcal{O}(N)$ operations per sample. The core of the proposed algorithms are DCD iterations that are known to be very well suited to implementation on FPGA platforms as was reported in [25], [51], [52]. Importantly, the proposed algorithms structurally and in the number and type of operations required are very close to the DCD-based RLS adaptive algorithms proposed in [24]. As indicated in [29], [31], [53], [54], the algorithms in [24] have been proved to be very well suited to implementation on FPGA platforms, providing a low chip area, high throughput, and numerical stability. Moreover, this approach has also been used to make other adaptive filters well suited to implementation on FPGAs [27], [55], [56]. Our proposed algorithms differ from the algorithm in [24] only in the computation of the penalty functions and reweighting, which do not represent a significant challenge for hardware implementation. Therefore, the adaptive algorithms proposed in this paper can be expected to be also well suited to implementation in finite precision, e.g., on FPGA platforms. We have demonstrated by simulation that the proposed adaptive algorithms outperform known advanced adaptive filtering algorithms in sparse identification scenarios and possess performance close to the oracle RLS performance with perfect knowledge of the support.

An important problem not addressed in this paper is the development of algorithms for online selection of the regularization parameters; for this purpose, the approaches from [5] and [57] can be used. Another important problem is the investigation of the convergence of the proposed algorithms. These will be directions for our further research.

APPENDIX I
COMPLEXITY OF COMPUTING THE LASSO PENALTY

We need to find the minimum:

$$[p,k] = \min_{s=1\ldots N, q=1\ldots 4} \left[\frac{\delta^2}{2} R_{s,s} - \Re\{\alpha_q^* c_s\} + \Delta f(s,q) \right], \quad (49)$$

where $\Delta f(s,q) = f_{\mathrm{p}}(\mathbf{h}+\alpha_q \mathbf{e}_s) - f_{\mathrm{p}}(\mathbf{h})$. For the lasso penalty, we have

$$\Delta f(s,q) = \tau w_s(|h_s + \alpha_q| - |h_s|). \quad (50)$$

Then, a direct (naive) computation of $[p,k]$ would require $76N$ real operations, including additions, multiplications, and square roots. We will show how this can be reduced based on the specific properties of the operations.

We will be doing computations separately in N groups, each one dealing with one (s-th) coordinate.

First, we notice that $|h_s|$ is the same for all 4 terms within the group, and therefore it is computed only once; this requires 2 multiplications (squaring), 1 addition, and 1 square root operation. When computing $|h_s|$, we also have $|h_s|^2$ as a by-pass product.

We now notice that α_q takes one of the four values: $[\delta, -\delta, j\delta, -j\delta]$ and δ is a power-of-two number, thus multiplications by δ are just bit-shifts. We need to compute $|h_s + \alpha_q|$ for all the four values of α_q. For $\alpha_q = \delta$, we have

$$\begin{aligned} |h_s + \alpha_q| &= |h_s + \delta| \\ &= \sqrt{(\Re\{h_s\} + \delta)^2 + (\Im\{h_s\})^2} \\ &= \sqrt{|h_s|^2 + \delta^2 + 2\delta\Re\{h_s\}}. \end{aligned} \tag{51}$$

Similarly, for the other three values, we obtain

$$\begin{aligned} |h_s - \delta| &= \sqrt{|h_s|^2 + \delta^2 - 2\delta\Re\{h_s\}}, \\ |h_s + j\delta| &= \sqrt{|h_s|^2 + \delta^2 + 2\delta\Im\{h_s\}}, \\ |h_s - j\delta| &= \sqrt{|h_s|^2 + \delta^2 - 2\delta\Im\{h_s\}}. \end{aligned} \tag{52}$$

As $|h_s|^2$ is available, we only need 1 addition to compute $|h_s|^2 + \delta^2$, 4 other additions, and 4 square root operations to obtain all the four quantities in (51) and (52). To obtain the four values in (50) for one s, we also need 4 additions and 4 multiplications. In total, computing (50) for all s and q results in $10N$ additions, $6N$ multiplications, and $5N$ square root operations.

For finding the maximum in (49), we take into account that $\Re\{\alpha_q^* c_s\}$ involves only selecting the real or imaginary part and a bit-shift. Then, with $\Delta f(s,q)$ available, we need $8N$ additions to find the maximum.

Thus, the use of the lasso penalty in the DCD algorithm requires in total $18N$ additions, $6N$ multiplications, and $5N$ square root operations, or $29N$ real-valued operations. This is almost three times fewer than when using the direct computations.

Appendix II
Complexity of Computing the Ridge-Regression Penalty

For applying the DCD algorithm, we need to find the minimum:

$$[p,k] = \min_{s=1\ldots N, q=1\ldots 4} \left[\frac{\delta^2}{2} R_{s,s} - \Re\{\alpha_q^* c_s\} + \Delta f(s,q) \right], \tag{53}$$

where, for the ridge-regression penalty, we have

$$\Delta f(s,q) = \tau w_s (|h_s + \alpha_q|^2 - |h_s|^2). \tag{54}$$

This maximization is the main contribution to the algorithm complexity. Although it has a complexity $\mathcal{O}(N)$, i.e., linear in the filter length N, the factor multiplying N can be quite large. E.g., a direct (naive) implementation of (53) would require 21 real-valued operations for every s and q and, thus, the total complexity would be $84N$ real-valued multiplications and additions. Below we show that this can be reduced down to as few as $6N$ real-valued operations.

We can write

$$|h_s + \alpha_q|^2 - |h_s|^2 = 2\Re\{\alpha_q^* h_s\} + \delta^2,$$

where we use the fact that $|\alpha_q|^2 = \delta^2$ for any q. Then (53) can be rewritten as

$$[p,k] = \arg\min_{s,q} \left[\frac{\delta^2}{2} R_{s,s} - \Re\{\alpha_q^* v_s\} \right], \tag{55}$$

where $v_s = c_s - 2\tau w_s h_s$. Computation of the vector $\mathbf{v}$ with elements v_s requires $2N$ additions and $2N$ multiplications. Denoting $\alpha_q = a_q \delta$, where $a_q \in [1,\ -1,\ j,\ -j]$, we can rewrite (55) as

$$[p,k] = \arg\min_{s,q} \left[\frac{\delta}{2} R_{s,s} - \Re\{a_q^* v_s\} \right]. \tag{56}$$

The operation $\max_q \Re\{a_q^* v_s\}$ requires searching for $\max[|\Re\{v_s\}|, |\Im\{v_s\}|]$; thus, computations in (56) require $2N$ additions, which together with computing $\mathbf{v}$ results in $4N$ additions and $2N$ multiplications.

Acknowledgment

The authors would like to thank the reviewers for the helpful comments that allowed them to improve the paper material and presentation. The authors are also thankful to B. Babadi for providing the SPARLS Matlab code.

References

[1] D. Angelosante, J. A. Bazerque, and G. B. Giannakis, "Online adaptive estimation of sparse signals: Where RLS meets the l_1-norm," *IEEE Trans. Signal Process.*, vol. 58, no. 7, pp. 3436–3447, 2010.

[2] B. Babadi, N. Kalouptsidis, and V. Tarokh, "SPARLS: The sparse RLS algorithm," *IEEE Trans. Signal Process.*, vol. 58, no. 8, pp. 4013–4025, 2010.

[3] G. Mileounis, B. Babadi, N. Kalouptsidis, and V. Tarokh, "An adaptive greedy algorithm with application to nonlinear communications," *IEEE Trans. Signal Process.*, vol. 58, no. 6, pp. 2998–3007, 2010.

[4] Y. Murakami, M. Yamagishi, M. Yukawa, and I. Yamada, "A sparse adaptive filtering using time-varying soft-thresholding techniques," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2010, pp. 3734–3737.

[5] E. M. Eksioglu and A. K. Tanc, "RLS algorithm with convex regularization," *IEEE Signal Process. Lett.*, vol. 18, no. 8, pp. 470–473, 2011.

[6] B. Dumitrescu, A. Onose, P. Helin, and I. Tabus, "Greedy sparse RLS," *IEEE Trans. Signal Process.*, vol. 60, no. 5, pp. 2194–2207, 2012.

[7] S. Cotter and B. Rao, "The adaptive matching pursuit algorithm for estimation and equalization of sparse time-varying channels," in *Proc. 34th Asilomar Conf. Signals Syst. Comput.*, 2000, vol. 2, pp. 1772–1776.

[8] S. F. Cotter and B. D. Rao, "Sparse channel estimation via matching pursuit with application to equalization," *IEEE Trans. Commun.*, vol. 50, no. 3, pp. 374–377, 2002.

[9] G. Z. Karabulut and A. Yongacoglu, "Sparse channel estimation using orthogonal matching pursuit algorithm," in *Proc. IEEE 60th Veh. Technol. Conf. (VTC2004-Fall)*, 2004, vol. 6, pp. 3880–3884.

[10] W. Li and J. C. Preisig, "Estimation of rapidly time-varying sparse channels," *IEEE J. Ocean. Eng.*, vol. 32, no. 4, pp. 927–939, 2007.

[11] C. R. Berger, Z. Wang, J. Huang, and S. Zhou, "Application of compressive sensing to sparse channel estimation," *IEEE Commun. Mag.*, vol. 48, no. 11, pp. 164–174, 2010.

[12] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *J. Stat. Softw.*, vol. 33, no. 1, pp. 1–22, 2010.

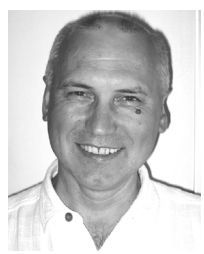
[13] Y. Gu, J. Jin, and S. Mei, "ℓ_0 norm constraint LMS algorithm for sparse system identification," *IEEE Signal Process. Lett.*, vol. 16, no. 9, pp. 774–777, 2009.

[14] Y. Meng, A. Brown, R. Iltis, T. Sherwood, H. Lee, and R. Kastner, "MP core: Algorithm and design techniques for efficient channel estimation in wireless applications," in *Proc. 42nd Design Autom. Conf.*, Jun. 2005, pp. 297–302.

[15] Y. Meng, W. Gong, R. Kastner, and T. Sherwood, "Algorithm/architecture co-exploration for designing energy efficient wireless channel estimator," *ASP J. Low Power Electron.*, vol. 1, no. 3, pp. 1–11, 2005.

[16] B. Benson, A. Irturk, J. Cho, and R. Kastner, "Survey of hardware platforms for an energy efficient implementation of matching pursuits algorithm for shallow water networks," in *Proc. 3rd ASM Int. Workshop Underwater Netw.*, 2008, pp. 83–86.

[17] J. Lu, H. Zhang, and H. Meng, "Novel hardware architecture of sparse recovery based on FPGAs," in *Proc. 2nd IEEE Int. Conf. Signal Process. Syst. (ICSPS)*, 2010, pp. V1-302–V1-306.
[18] P. Maechler, P. Greisen, N. Felber, and A. Burg, "Matching pursuit: Evaluation and implementation for LTE channel estimation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2010, pp. 589–592.
[19] D. Yang, H. Li, G. D. Peterson, and A. Fathy, "Compressed sensing based UWB receiver: Hardware compressing and FPGA reconstruction," in *43rd Ann. Conf. Inf. Sci. Syst. (CISS)*, 2009, pp. 198–201.
[20] Y. V. Zakharov and T. C. Tozer, "Multiplication-free iterative algorithm for LS problem," *Electron. Lett.*, vol. 40, no. 9, pp. 567–569, 2004.
[21] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, "Pathwise coordinate optimization," *Ann. Appl. Statist.*, vol. 1, no. 2, pp. 302–332, 2007.
[22] T. T. Wu and K. Lange, "Coordinate descent algorithms for Lasso penalized regression," *Ann. Appl. Statist.*, vol. 2, no. 1, pp. 224–244, 2008.
[23] M. Garcia-Magarinos, R. Cao, A. Antoniadis, and W. Gonzalez-Manteiga, "Lasso logistic regression, GSoft and the cyclic coordinate descent algorithm: Application to gene expression data," *Statist. Appl. Genetics Molec. Biol.*, vol. 9, no. 1, pp. 1–28, 2010, Article 30.
[24] Y. Zakharov, G. White, and J. Liu, "Low complexity RLS algorithms using dichotomous coordinate descent iterations," *IEEE Trans. Signal Process.*, vol. 56, no. 7, pp. 3150–3161, Jul. 2008.
[25] J. Liu, Y. V. Zakharov, and B. Weaver, "Architecture and FPGA design of dichotomous coordinate descent algorithms," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 56, no. 11, pp. 2425–2438, 2009.
[26] J. Liu, B. Weaver, Y. Zakharov, and G. White, "An FPGA-based MVDR beamformer using dichotomous coordinate descent iterations," in *Proc. Conf. ICC'2007*, Glasgow, UK, Jun. 24–28, 2007, pp. 2551–2556.
[27] J. Liu and Y. Zakharov, "FPGA implementation of affine projection adaptive filter using coordinate descent iterations," in *Proc. 7th IEEE Int. Symp. Wireless Commun. Syst. (ISWCS)*, 2010, pp. 374–378.
[28] Y. Zakharov and F. Albu, "Coordinate descent iterations in fast affine projection algorithm," *IEEE Signal Process. Lett.*, vol. 12, no. 5, pp. 353–356, May 2005.
[29] J. Liu and Y. Zakharov, "Low complexity dynamically regularised RLS algorithm," *Electron. Lett.*, vol. 44, no. 14, pp. 886–885, 2008.
[30] C. Stanciu, C. Anghel, C. Paleologu, J. Benesty, F. Albu, and S. Ciochina, "A proportionate affine projection algorithm using dichotomous coordinate descent iterations," in *Proc. 10th Int. Symp. Signals, Circuits Syst. (ISSCS)*, 2011, pp. 1–4.
[31] J. Liu and Y. Zakharov, "Dynamically regularized RLS-DCD algorithm and its FPGA implementation," in *Proc. 42nd Asilomar Conf. Signals, Syst. Comput.*, 2008, pp. 1876–1880.
[32] E. Candès, M. Wakin, and S. Boyd, "Enchancing sparsity by reweighted ℓ_1 minimization," *J. Fourier Anal. Appl.*, vol. 14, no. 5, pp. 877–905, 2008.
[33] Y. Wang and W. Yin, "Sparse signal reconstruction via iterative support detection," *SIAM J. Imag. Sci.*, vol. 3, no. 4, pp. 462–491, 2010.
[34] D. Wipf and S. Nagarajan, "Iterative reweighted ℓ_1 and ℓ_2 methods for finding sparse solutions," *IEEE J. Sel. Topics Signal Process.*, vol. 4, no. 2, pp. 317–329, 2010.
[35] X. Xu, X. Wei, and Z. Ye, "DOA estimation based on sparse signal recovery utilizing weighted ℓ_1 norm penalty," *IEEE Signal Process. Lett.*, vol. 19, no. 3, pp. 155–158, 2012.
[36] K. Slavakis, Y. Kopsinis, and S. Theodoridis, "Adaptive algorithm for sparse system identification using projections onto weighted ℓ_1 balls," in *Proc. IEEE ICASSP*, 2010, vol. 1, pp. 3742–3745.
[37] C. Paleologu, J. Benesty, and S. Ciochina, "Sparse adaptive filters for echo cancellation," *Synthesis Lectures Speech Audio Process.*, vol. 6, no. 1, pp. 1–124, 2010.
[38] S. Haykin, *Adaptive Filtering*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1991.
[39] A. H. Sayed, *Fundamentals of Adaptive Filtering*. Hoboken, NJ, USA: Wiley, 2003.
[40] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD, USA: The Johns Hopkins Univ. Press, 1996.
[41] Y. Li and S. Osher, "Coordinate descent optimization for ℓ_1 minimization with application to compressed sensing; a greedy algorithm," *Inverse Probl. Imag.*, vol. 3, no. 3, pp. 487–503, 2009.
[42] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *J. Royal Stat. Soc. B.*, vol. 67, no. 2, pp. 301–320, 2005.
[43] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, 1998.
[44] C. Qi, X. Wang, and L. Wu, "Underwater acoustic channel estimation based on sparse recovery algorithms," *IET Signal Process.*, vol. 5, no. 8, pp. 739–747, 2011.
[45] Z. Yang, R. C. de Lamare, and X. Li, "ℓ_1-Regularized STAP algorithm with a generalized sidelobe canceler architecture for airborne radar," *IEEE Trans. Signal Process.*, vol. 60, no. 2, pp. 674–686, 2012.
[46] A. Panahi and M. Viberg, "Fast candidate points selection in the LASSO path," *IEEE Signal Process. Lett.*, vol. 19, no. 2, pp. 79–82, 2012.
[47] S. J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky, "An interior-point method for large-scale ℓ_1-regularized least squares," *IEEE J. Sel. Topics Signal Process.*, vol. 1, no. 4, pp. 606–617, 2007.
[48] S. J. Wright, R. D. Nowak, and M. A. T. Figueiredo, "Sparse reconstruction by separable approximation," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2479–2493, 2009.
[49] J. J. Fuchs, "Recovery of exact sparse representations in the presence of bounded noise," *IEEE Trans. Inf. Theory*, vol. 51, no. 10, pp. 3601–3608, 2005.
[50] S. L. Gay and J. Benesty, *Acoustic Signal Processing for Telecommunication*. Boston, MA, USA: Kluwer Academic, 2000.
[51] J. Liu, B. Weaver, and G. White, "FPGA implementation of the DCD algorithm," in *Proc. Commun. Symp.*, London, U.K., Sep. 2006, pp. 1–4.
[52] J. Liu, Z. Quan, and Y. Zakharov, "Parallel FPGA implementation of DCD algorithm," in *Proc. Conf. DSP* , Cardiff, U.K., Jul. 1–4, 2007, pp. 331–334.
[53] Y. Zakharov, G. White, and J. Liu, "Fast RLS algorithm using dichotomous coordinate descent iterations," in *Proc. 41st Asilomar Conf. Signals, Syst. Comput.*, 2007, pp. 431–435.
[54] J. Liu and Y. Zakharov, "FPGA implementation of RLS adaptive filter using dichotomous coordinate descent iterations," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Dresden, Germany, Jun. 14–18, 2009, pp. 1–5.
[55] C. Stanciu, C. Anghel, C. Paleologu, J. Benesty, F. Albu, and S. Ciochina, "FPGA implementation of an efficient proportionate affine projection algorithm for echo cancellation," in *Proc. 19th Eur. Signal Process. Conf. (EUSIPCO)*, Barcelona, Spain, Aug. 29–Sep. 2, 2011, pp. 1284–1288.
[56] M. Algreer, M. Armstrong, and D. Giaouris, "Active online system identification of switch mode DC—DC power converter based on efficient recursive DCD-IIR adaptive filter," *IEEE Trans. Power Electron.*, vol. 27, no. 11, pp. 4425–4435, 2012.
[57] Y. Chen, Y. Gu, and A. O. Hero, "Regularized least-mean-square algorithms," *arXiv preprint arXiv:1012.5066*, 2010.



Yuriy V. Zakharov (M'01–SM'08) received the M.Sc. and Ph.D. degrees in electrical engineering from the Moscow Power Engineering Institute, Moscow, Russia, in 1977 and 1983, respectively.

From 1977 to 1983, he was an Engineer with the Special Design Agency, Moscow Power Engineering Institute. From 1983 to 1999, he was the Head of Laboratory at the N. N. Andreev Acoustics Institute, Moscow. From 1994 to 1999 he was with Nortel as a DSP Group Leader. Since 1999, he has been with the Communications Research Group, University of York, York, U.K., where he is currently a Reader. His interests include signal processing and communications.

Vítor H. Nascimento (M'01–SM'12) was born in São Paulo, Brazil. He received the B.S. and M.S. degrees in electrical engineering from the University of São Paulo in 1989 and 1992, respectively, and the Ph.D. degree from the University of California, Los Angeles, CA, USA, in 1999.

From 1990 to 1994, he was a Lecturer at the University of São Paulo, and in 1999, he joined the faculty at the same school, where he is now an Associate Professor. His research interests include signal processing theory and applications, robust and nonlinear estimation, and applied linear algebra.

Dr. Nascimento received the 2002 IEEE SPS Best Paper Award. He served as an Associate Editor of the IEEE SIGNAL PROCESSING LETTERS (2003 to 2005), the IEEE TRANSACTIONS ON SIGNAL PROCESSING (2005 to 2008), and the EURASIP *Journal on Advances in Signal Processing* (2006 to 2009). He was a member of the IEEE-SPS Signal Processing Theory and Methods Technical Committee from 2007 to 2012. Since 2010, he has been Chair of the São Paulo SPS Chapter.